SUSTAINABILITY OF OPEN SOURCE SOFTWARE PROJECTS: ON THE INFLUENCE OF TECHNICAL INTERDEPENDENCIES IN SOFTWARE ECOSYSTEMS ON DEVELOPER PARTICIPATION

Inauguraldissertation
zur
Erlangung des Doktorgrades
der
Wirtschafts- und Sozialwissenschaftlichen Fakultät
der
Universtität zu Köln

2023

presented
by

M.Sc. Mario Müller

from

Bonn

First reviewer: Prof. Dr. Christoph Rosenkranz
Second reviewer: Prof. Dr. Markus Weinmann

Date of oral defense: August 9, 2023

ACKNOWLEDGEMENTS

At various points during this endeavor, I thought I might never reach the point of having to write the acknowledgment section of my finished dissertation. This time has been as much rewarding, insightful, and joyful as it was oftentimes challenging, frustrating, and hell on earth. Nevertheless, I am glad I was provided the opportunity and had faith in myself that I could do it.

However, I could not have done this without the support and guidance from numerous people along the way. First and foremost, I'd like to thank my supervisor team, starting with Christoph, who has continuously supported me through these years. Also, Markus and his training in data analytics and Bayesian statistics have made finishing this dissertation possible. Thank you both for your engagement and mentorship.

A special thanks goes out to Julian, Philipp, Karl, and Phil, who were always available to discuss ideas, listen to my complaints, or just have beers or tacos. To all the others that I didn't have enough space to explicitly mention: It was a pleasure to meet you all and thank you for this great time.

Next, I would like to extend a heartfelt thanks to my family. You have always supported and put up with me. Especially my love, Rebecca, who repeatedly built me up and provided me with much-needed diversion and plenty of coffee. Thank you for your patience and care.

I would also like to thank my friends, who provided the perfect distractions and excuses to procrastinate. Without you guys, I may have finished this dissertation a lot sooner.

To sum up, thanks to everyone who helped me get through this challenging and often confusing process. I couldn't have done it without you.

ABSTRACT

In the community-based model of open source software (OSS) development, OSS projects are built and maintained by developers that voluntarily contribute their skills, knowledge, and time, thus making them dependent on their continued participation. Therefore, the question of how projects can attract and retain developers is of major concern for their sustainability. OSS projects are embedded into a complex network of technical interdependent projects that emerges from building upon and reusing existing software components. In these so-called software ecosystems, the issue of sustained participation is not only a concern of a single project but also other dependent projects. However, the role and influence of these interdependencies between projects have so far been neglected by Information Systems researchers. This dissertation thus asks: *How do technical interdependencies in software ecosystems influence the sustainability of open source software projects?*

To answer this question, this dissertation consists of three independent empirical studies that focus on three aspects of how technical interdependencies influence developer participation and thus contribute to the sustainability of open source projects: (1) the ability to attract developers, (2) the influences on developers' participation decision, and (3) the retention of developers in a project. This dissertation finds that OSS projects attract more developers when depending on other projects and their ability to retain developers increases with the number of shared developers with other technical interrelated projects. Furthermore, the participation decisions of developers are also positively influenced by these technical relations.

Together, these studies contribute to the body of knowledge on developer participation by highlighting the role of technical interdependencies for the overall sustainability of open source projects.

CONTENTS

ABBREVIATIONS

CI Confidence Interval

ESS Effective Sample Size

GOF Goodness-of-Fit

IT Information Technology

LOO Leave-One-Out Cross-Validation

LOOIC Leave-One-Out Cross-Validation Information Criterion

MCMC Markov Chain Monte Carlo

OSS Open Source Software

PSIS Pareto-Smoothed Importance Sampling

SAOM Stochastic Actor-Oriented Model

VIF Variance Inflation Factor

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

DISSERTATION OVERVIEW

1.1 Introduction

This dissertation focuses on open source software (OSS) development in software ecosystems and the sustainability of OSS projects and their communities. OSS has become famous through the success and wide adoption of projects such as the Linux kernel, Apache web server, or Google's Android operating system. OSS projects have found their way into modern digital infrastructures and organizations, with recent estimates suggesting that OSS software functions as the foundation of 80-90% of any software today (Nagle, 2019; Nagle et al., 2020).

OSS projects have two main characteristics that differentiate them from traditional software development projects. The first characteristic of OSS projects is their license model that allows organizations and developers to use, review, change, and distribute the source code, and, second, their community-based development approach, which is contrary to traditional commercial software development (Raymond, 1999; von Krogh & Spaeth, 2007). This approach represents a "private-collective" model of innovation where developers are rewarded by collectively writing and sharing code (von Hippel & von Krogh, 2003). Software produced following this approach is often even superior to software developed in a traditional fashion (Mockus et al., 2002).

In the community-based model of software development, the success of OSS projects relates to the growth and sustainability of their communities (von Krogh et al., 2003). The sustainability of OSS projects, that is their ability to exhibit long-term development and maintenance activity (Chengalur-Smith et al., 2010), relies on the continuous participation of developers who are voluntarily contributing by committing code, reporting or fixing bugs, or helping other (Roberts et al., 2006; Setia et al., 2012). Hence,

attracting and keeping developers is the foundation of sustainable development (Butler, 2001; Chengalur-Smith et al., 2010), with most OSS projects failing because they are unable to leverage the necessary developer resources (Fang & Neufeld, 2009).

1.2 Problem Statement and Research Question

As developers and their voluntary participation are key resources in OSS development, research has investigated what motivates developers to contribute their time, skills, and knowledge to a project. Studies have thereby focused on individual developer factors, such as enjoyment (Shah, 2006), learning and skill development (von Krogh et al., 2003), career advancements (Lerner & Tirole, 2002), or attachment to the project (Maruping et al., 2019), and project characteristics, such as license restrictions (Stewart et al., 2006), organizational sponsorship (Shah, 2006; Stewart et al., 2006), or the modularity of the project's codebase (Baldwin & Clark, 2006). More recently, studies have also focused on network-related characteristics that emerge through social relations between developers and affiliations of developers with multiple projects. They found that, for example, previous collaborations influence a developer's participation decision (Hahn et al., 2008), and central projects in an affiliation network profit from increased developer attention (Hahn et al., 2008) and access to developer resources (Grewal et al., 2006; Sojer & Henkel, 2010).

However, these studies primarily focus on the individual developers or projects, thereby neglecting the environment of a project. This is important because a project's environment also affects its sustainability (Chengalur-Smith et al., 2010; Valiev et al., 2018). Modern OSS development is organized in so-called software ecosystems, that is interdependent projects and communities emerging on a common technology platform (e.g., programming languages), which allows them to create, share, and reuse software components (Bogart et al., 2021), which are reusable pieces of software code (Kikas et al., 2017). As the reuse of software components has been a common practice in OSS development (Haefliger et al., 2008; Sojer & Henkel, 2010),

modern programming languages introduced various tools that ease the process of integrating and publishing software components, such as dependency managers and online registries (Cox, 2019). The resulting dependencies have been identified as an issue in OSS development (Nagle et al., 2020), because they can lead to technical and organizational problems that are harmful to the quality and functioning of the individual project and the entire ecosystem (Cataldo et al., 2009; Cox, 2019; Decan et al., 2019).

Despite the importance of organizing OSS development in software ecosystems (Bogart et al., 2021) and the increasing emergence of technical interdependencies between projects through recombination and reuse of existing software components (Bogart et al., 2016; Cox, 2019; Howison & Crowston, 2014; Singh et al., 2011), Information Systems research integrating these technical interdependencies is, to the best of our knowledge, scarce. This is problematic as OSS projects are no longer independent units but dependent on others and therefore have to consider their role in the software ecosystem (Bogart et al., 2021; Jansen et al., 2009). This means that now concerns and issues related to sustaining participation are no longer limited to a single OSS project (Chengalur-Smith et al., 2010) but affect all other dependent projects within the software ecosystem (Valiev et al., 2018). Thus, these technical interdependencies should be considered to achieve a holistic socio-technical perspective on OSS projects, their communities, and their sustainability.

To close this gap, this dissertation investigates three core aspects of sustaining participation in OSS projects and the role that technical interdependencies play in software ecosystems: (1) the attraction of new developers, (2) the influence on a developer's participation choice, and (3) the retention of already contributing developers. To summarize, this dissertation addresses the following overarching research question:

> *How do technical interdependencies in open source software ecosystems influence developer participation?*

To answer this research question, I conducted three empirical studies each focusing

on a different aspect of developer participation. First, as OSS projects frequently fail to ever achieve traction and thus fail soon after their initiation (Chengalur-Smith et al., 2010; Fang & Neufeld, 2009; Stewart et al., 2006), the attraction of developers to join and contribute to the project has become a major issue in OSS projects (Butler, 2001; Crowston et al., 2003). Thus, OSS projects require access to the pool of developer resources in their environment (Butler, 2001; Wang et al., 2013) and compete for these resources with other projects (Setia et al., 2020; Wang et al., 2013). How a project's position in the software ecosystem reflected by its technical interdependencies increases its legitimacy and importance and enables access to the pool of developer resources is the focus of the first study.

Second, while prior research has shown that social relations and connections are important antecedents of developers' participation decisions, studies have overlooked the role of technical interdependencies as an influencing factor, which is the focus of the second study. Projects might profit from the number and nature of technical interdependencies in becoming more attractive by allowing developers to focus on more rewarding and challenging tasks (Haefliger et al., 2008) and providing opportunities for developers seeking reputation gain and visibility in the community (Hu et al., 2012). Furthermore, as developers tend to support other projects that depend on their projects (Bogart et al., 2016) or that they use themselves (Shah, 2006), technical interdependencies should reflect this behavior.

Third, besides attracting developers, keeping them involved over time is also important for the sustainability of an OSS project (Crowston et al., 2003; Markus et al., 2000). As OSS developers simultaneously work on multiple projects (Grewal et al., 2006; Singh et al., 2011), that results in developer overlap between projects competing for their time and attention. Previous studies showed that this overlap can lead to either competitive or mutualistic effects (Barnett & Carroll, 1987), which influence a project's ability to retain developers. Thus, the third study focuses on developer overlap between projects with technical interdependencies resulting in communities characterized by shared goals, challenges, and problems and its influence on sustained participation.

1.3 Structure of Dissertation

This dissertation is a cumulative work and addresses the overarching research question in three independent but interrelated empirical studies. These studies investigate (1) the influence of dependencies on developer attraction, (2) the influence of dependencies on developers' participation decisions, and (3) the influence of interdependencies in technological niches on developer retention. Table 1.1 summarizes the included studies and their current status. All data, analyses, and associated files are available in an open science repository[1].

In the following, I will briefly summarize each study and provide an outlook for the remaining chapters of this dissertation.

Table 1.1. Overview of Studies

	Study 1	Study 2	Study 3
Title	The Influence of Dependency Networks on Sustained Participation in Open Source Software Ecosystems	The Role of Dependency Networks in Developer Participation Decisions in Open Source Software Ecosystems: An Application of Stochastic-Actor Oriented Models	Sustaining Open Source Software Projects: An Ecological Perspective on Technological Interdependencies in Software Ecosystems
Addressed Aspect	Influence of Project's Position in Dependency Network on Developer Attraction	Influence of Project Dependencies and Network Effects on Developer Participation Decisions	Influence of Developer Overlap in Technological Interdependent Communities on Developer Retention
Research Design	Quantitative Network Analysis Using Longitudinal Digital Trace Data and Bayesian Multilevel Modeling	Quantitative Network Analysis using Longitudinal Digital Trace Data and Stochastic Actor-Oriented Modeling	Quantitative Network Analysis Using Longitudinal Digital Trace Data and Bayesian Multilevel Modeling

[1] https://osf.io/h5qud/?view_only=dde3e51dfcdb414fae62df6a8ffaeb21

	Study 1	Study 2	Study 3
Status of Research	Published in the Proceedings of the 43rd International Conference on Information Systems (ICIS) in 2022	Published in the Proceedings of the 56th Hawaii International Conference on System Sciences (HICSS) in 2023	Submitted to MIS Quarterly in April 2023

The first study, presented in Chapter 4, draws from a resource-based and ecological view on sustainable online communities to investigate how an OSS project's position in the software ecosystem's dependency network enables it to leverage these dependency relations to attract developers. OSS research has neglected these technical interdependencies so far, as studies adopting a network perspective on OSS development in large focused on the social relations between developers and projects (e.g., Grewal et al., 2006; Hahn et al., 2008; Maruping et al., 2019; Singh et al., 2011). This study closes this gap by focusing on the technical interdependencies between projects. It shows that projects profit from integrating other software components, resulting in upstream dependencies, but do not profit from their importance for the ecosystem reflected by their downstream dependencies.

The second study, presented in Chapter 5, aims at investigating the influence of technical interdependencies on the antecedents of developers' participation decisions. While previous studies identified individual (e.g., Hu et al., 2012; Lerner & Tirole, 2002; von Hippel & von Krogh, 2003), project (Baldwin & Clark, 2006; Shah, 2006; e.g., Stewart et al., 2006), and social factors (e.g., Grewal et al., 2006; Hahn et al., 2008; Oh & Jeon, 2007) as important antecedents of participation decisions, again, they overlooked technical relations and connections. This study adopts a dynamic network modeling approach utilizing stochastic actor-oriented models (SAOMs, Snijders, 1996) to test the influence of technical relations and connections on developer participation decisions. The findings show that developers tend to engage in a project if they worked on another technically interrelated project before.

The third and last study, presented in Chapter 6, draws from organizational ecology theory (Hannan & Freeman, 1989) and investigates OSS communities emerging around interdependent projects in technological niches (Podolny et al., 1996; Podolny & Stuart, 1995) and the effect of shared developers between these projects on developer retention, that is a project's ability to keep developers involved in the long-term. Previous studies have found ambiguous effects of member overlap (e.g., Wang et al., 2013; Zhu, Kraut, et al., 2014) resulting from either competitive or mutualistic dynamics (Barnett & Carroll, 1987). This study shows that in a project's technological niche, which we conceptualize as communities defined by their technical interdependencies, the effect of developer overlap is mutualistic, thereby increasing its ability to retain developers.

All studies are the result of collaboration with my supervisor team. Table 1.2 shows the contributions of each participating researcher to the respective study following the Contributor Roles Taxonomy (CRediT)[2].

Table 1.2. Contributor Roles

	Study 1	Study 2	Study 3
Mario Müller	Conceptualization, Methodology, Software, Validation, Formal Analysis, Investigation, Data Curation, Writing - Original Draft, Writing - Review & Editing Visualization, Project Administration	Conceptualization, Methodology, Software, Validation, Formal Analysis, Investigation, Data Curation, Writing - Original Draft, Writing - Review & Editing, Visualization, Project Administration	Conceptualization, Methodology, Software, Validation, Formal Analysis, Investigation, Data Curation, Writing - Original Draft, Writing - Review & Editing, Visualization, Project Administration
Christoph Rosenkranz	Writing - Review & Editing, Supervision	Writing - Review & Editing, Supervision	Writing - Review & Editing, Supervision
Markus Weinmann	Validation, Supervision	Supervision	Validation, Supervision

[2]https://www.elsevier.com/authors/policies-and-guidelines/credit-author-statement

The remainder of this dissertation is structured as follows. Chapter 2 presents and summarizes related work on OSS sustainability and technical interdependencies in software ecosystems. Chapter 3 describes the research design of the three empirical studies. Chapters 4, 5, and 6 contain the studies of this dissertation. Chapter 7 summarizes the findings and contributions and discusses this dissertation's limitations and future research directions. Chapter 8 closes this dissertation with a brief conclusion.

CHAPTER 2

RESEARCH BACKGROUND

2.1 Sustainability of Open Source Software Projects

OSS development is a community-based model of software development where software is produced by a decentralized community of distributed developers who collaborate via online platforms without traditional hierarchies (Fang & Neufeld, 2009; Lindberg et al., 2016). The community consists mostly of unpaid developers who voluntarily contribute to a project (Crowston, 2011; Roberts et al., 2006), even though more recently commercial organizations started to assign employees to help in OSS projects (von Krogh et al., 2012). To ensure sustained development, OSS projects rely on the continuous participation of their communities (Gamalielsson & Lundell, 2014; Mockus et al., 2002; Roberts et al., 2006; Shah, 2006).

Despite the success stories of popular OSS projects such as the Linux kernel, most projects fail to keep their community engaged and thus cannot sustain development activity over time and are often abandoned soon after their initiation (Chengalur-Smith et al., 2010; Fang & Neufeld, 2009; Stewart et al., 2006). To succeed, OSS projects need to keep their communities healthy by continuously attracting new and retaining already participating developers (Butler, 2001; Crowston et al., 2003). The sustainability of their communities is therefore closely related to the success of OSS projects (von Krogh et al., 2003), as without the community, sustained development of the project is not possible (Roberts et al., 2006). Without sustained development, the project also risks losing the interest of users (Subramaniam et al., 2009), which in turn negatively impacts the development activity (Stewart et al., 2006). Hence, the issue of how OSS projects can sustain the participation of their developers has become one major theme in research on the sustainability of OSS projects (Chengalur-Smith et

al., 2010; Curto-Millet & Corsín Jiménez, 2022; Fang & Neufeld, 2009; Gamalielsson & Lundell, 2014) as well as online communities (Bock et al., 2015; Butler, 2001; Oh et al., 2016; Ridings & Wasko, 2010; Sun et al., 2012).

In the existing literature, three streams emerge that investigate the antecedents of developer participation. The first stream focuses on the individual's motivation that drives participation. This includes fun or enjoyment (Shah, 2006), learning and developing skills (von Hippel & von Krogh, 2003), or increasing reputation (Hu et al., 2012; Wasko & Faraj, 2005) and career advancements (Lerner & Tirole, 2002) leading to intrinsic and extrinsic motivation (von Krogh et al., 2012).

The second stream focuses on project characteristics, such as organizational sponsorship (Spaeth et al., 2015; Stewart et al., 2006), license choice (Gamalielsson & Lundell, 2014; Stewart et al., 2006), governance structures (Markus, 2007; Shah, 2006; West & O'mahony, 2008), quality controls (Ho & Rai, 2017), task complexity (Sun et al., 2012), or a modular codebase (Baldwin & Clark, 2006), that influence a developer's motivation to join a community. Furthermore, a project needs to signal its legitimacy to become attractive to developers. As the number of capable and available developers is limited, OSS projects are competing with other projects for these developer resources (Setia et al., 2012; Wang et al., 2013). Therefore, the project needs to be attractive for new developers to join and existing developers to stay (von Krogh et al., 2003). Important indicators of a project's legitimacy are, for example, the size of its community (Butler, 2001; Chengalur-Smith et al., 2010) and its community activity (Butler, 2001; Setia et al., 2020).

The third stream focuses on social dynamics and relational effects that emerge through developer collaborations and affiliations within and between projects. Studies found that a project's position in affiliation networks influences its success and ability to attract developers (Grewal et al., 2006). Also, it reduces uncertainty about how to contribute and thereby helps in attracting developers (Maruping et al., 2019). Furthermore, a project's position allows it to better access available resources

in the ecosystem (Sojer & Henkel, 2010) and signals higher quality (Grewal et al., 2006; Setia et al., 2020), thereby increasing a project's attractiveness. In addition, research has found that previous collaborations with a project's initiator increase the likelihood of joining (Hahn et al., 2008). A developer's decision to remain involved is also influenced by other developers in the community as well as the leadership style (Oh & Jeon, 2007) and the strength of emotional ties between members (Bock et al., 2015). Research also found that the response from the community to developers' contributions influences their continued participation (Zhang et al., 2013). Another important aspect is that situated learning and identity construction leads to sustained participation (Fang & Neufeld, 2009). In addition, studies found that social capital plays an important role in sustained contributions (Wasko & Faraj, 2005).

In sum, although various antecedents of a developer's participation decision have been studied, existing research falls short in incorporating the technical interdependencies that exist between OSS projects in software ecosystems and the resulting dynamics and effects. To solve this shortcoming, this dissertation focuses on these technical interdependencies and their influence on developer participation.

2.2 Software Ecosystems and the Emergence of Dependency Networks

A popular phenomenon within OSS development and software engineering is the emergence of *software ecosystems* (Bosch, 2009; Messerschmitt & Szyperski, 2003). Software ecosystems, analogous to biological ecosystems, involve a network of interdependent software components maintained by geographically dispersed communities of collaborating contributors (Decan et al., 2019). These communities are built around a common technology platform (such as a programming language) and allow developers to collaborate, share, and leverage existing software components to create new software more efficiently (Bogart et al., 2021). An example of a software ecosystem is the JavaScript programming language, which has a large and active

community of developers who contribute to a growing library of *software components* (also called packages), which are reusable pieces of code that can be included in other projects and applications (Kikas et al., 2017). Software components are therefore closely related to the concept of a module in modular system design (Baldwin & Clark, 2000).

Through the availability of online collaboration tools, version control, dependency managers, and software component registries, organizing work in software ecosystems has become increasingly popular in OSS development (Bogart et al., 2021; Cox, 2019; Decan et al., 2019). The trend towards organizing OSS development in software ecosystems highlights a shift away from the individual project towards its environment and inherent relations (Hanssen, 2012) and allows the development of functionality outside the focal project (Bosch, 2009). By adopting principles from modular system design (Baldwin & Clark, 2000) and making the development process and activities transparent (Cataldo & Herbsleb, 2010), software ecosystems thereby enable the coordination of large numbers of developers that contribute interdependent software components to the software ecosystem (Jacobides et al., 2018). Thus, software ecosystems are critical for innovation (Jacobides et al., 2018), reducing development time and costs (Bosch, 2009; von Hippel & von Krogh, 2003), and ensuring the sustainability and longevity of software projects (Valiev et al., 2018).

However, the software ecosystem's stability depends on the balance between its software components and their *dependencies*, with changes to one software component potentially resulting in cascading effects through the entire ecosystem (Bogart et al., 2016). Dependencies emerge dynamically through the common practice of reusing software components in other projects during the development process (Cataldo et al., 2008; Haefliger et al., 2008), resulting in the project being dependent on the software component to function properly (Valiev et al., 2018). For instance, a web application might depend on a database management system to store and retrieve data, which is available as a separate software component in the ecosystem. Thus, software ecosystems can be seen as self-organizing systems that emerge and evolve through the

addition and combination of already available software components (Arthur, 2009).

To understand the emergence and evolution of software ecosystems, the technical interdependencies and their changes can be conceptualized as networks of complementary and substitutable software components (Funk & Owen-Smith, 2017). These *dependency networks* consist of the software components and their dependency relations (Kikas et al., 2017). Furthermore, the dependency relations can be further distinguished between *upstream dependencies* and *downstream dependencies* (Valiev et al., 2018). From the perspective of a focal component, upstream dependencies emerge as the result of reuse by the focal component, whereas downstream dependencies result from other components in the software ecosystem depending on the focal component (Valiev et al., 2018).

Research on dependency networks has investigated how they affect a project's survival (Valiev et al., 2018), how to deal with breaking changes in the ecosystem (Bogart et al., 2016; Bogart et al., 2021), how the size of software ecosystems steadily grows (German et al., 2013), which factors lead to upgrading dependencies and how developers discuss the need and risk to do so (Bavota et al., 2015), and why developers use software components that perform trivial tasks (Abdalkareem et al., 2020). Overall, research highlights the need to carefully manage and analyze dependencies. Thus, research on software ecosystems and emerging dependency networks has, to the best of our knowledge, mostly focused on the technical and engineering-related aspects, thereby neglecting the social aspects of software development in general and the role of dependency networks in influencing developer participation in particular.

To summarize, the concept of software ecosystems reflects the complexities and challenges involved in their development, management, and evolution and highlights the importance of a holistic approach when investigating OSS development by recognizing the technical interdependencies between software components and projects. Thus, this dissertation integrates the concept of software ecosystems to investigate their influence on developer participation and the sustainability of OSS projects.

CHAPTER 3

RESEARCH DESIGN

This dissertation consists of three empirical studies on, as of today, one of the largest software ecosystems bounded by the JavaScript programming language. The overarching theme of these studies is the focus on the technical interdependencies between OSS projects in software ecosystems and their influence on developer participation. Specifically, the three studies address the following research questions:

> **RQ1:** What is the influence of a project's technical dependencies in the software ecosystem on its ability to attract developers?
>
> **RQ2:** How do technical interdependencies influence a developer's participation decision in open source software ecosystems?
>
> **RQ3:** What is the effect of developer overlap in technological niches on the sustainability of open source projects in software ecosystems?

Following, this dissertation outlines the research approach and design chosen to answer these research questions. Specifically, it describes the research methods and collected data and explains the theoretical foundations of the adopted network perspective.

3.1 Strategy of Inquiry and Empirical Material

As each research question addresses various factors of technical interdependencies and the resulting dependency network on an outcome of developer participation, each study adopts a quantitative research approach to test the derived hypotheses by examining the relationship among different variables (Creswell, 2009). In study 1 and 3, I used Bayesian multilevel regression models to estimate the hypothesized effects. I chose multilevel models because they account for clustered data structures resulting

from repeated measures (Gelman & Hill, 2007). In study 2, I estimated the effects of the dependency network on developers' participation decisions by applying SAOMs. SAOMs are predictive models for dynamics networks enabling researchers to test various mechanisms that influence the evolution of a network (Cornwell, 2015). I chose these models because they allow the representation and statistical inference of network dynamics using observed longitudinal data (Snijders et al., 2010).

All three studies focus on the JavaScript programming language software ecosystem. This empirical setting is suitable for three main reasons. First, choosing a common programming language as the boundary condition is in line with my definition of software ecosystems, which emerge around a common technology platform such as programming languages. Second, in doing so, it also controls for knowledge-related preferences of developers who specialize in a specific programming language. Third, a common programming language is a suitable characteristic for a definitional focus for network construction (Laumann et al., 1989), and defining boundary conditions based on the programming language is common practice in OSS research adopting a network perspective (e.g., Grewal et al., 2006; Singh et al., 2011).

Table 3.1 provides a summary of the studies' research designs.

Table 3.1. Research Approaches

	Study 1	Study 2	Study 3
Research Approach	Deductive hypotheses testing via Bayesian multilevel regression model	Deductive hypotheses testing via stochastic actor-oriented model	Deductive hypotheses testing via Bayesian multilevel regression model
Empirical Setting	Open source software ecosystem based on JavaScript programming language	Open source software ecosystem based on JavaScript programming language	Open source software ecosystem based on JavaScript programming language
Role of Theory	Theory testing through hypothesis validation	Theory testing through hypothesis validation	Theory testing through hypothesis validation

	Study 1	Study 2	Study 3
Data Sources	Longitudinal panel data for 1,832 projects between October 2020 until January 2022; 7,328 observations and 5 snapshots of dependency network state	Longitudinal panel data for 250 projects and 1,172 developers between February and May 2021; 4 snapshots of affiliation and dependency network state	Longitudinal panel data for 2,283 projects between January 2017 and January 2019; 37,502 observations and 24 snapshots of affiliation and dependency network state
Contribution	Identification of positive effect of upstream dependencies on developer attraction and no effect of downstream dependencies	Identification of positive effect of past participation in interdependent project on likelihood to participate in focal project	Identification of mutualistic effect of developer overlap in technological niches on developer retention that increases with project age

All three studies use digital trace data, that is digital records of activities or events involving information technologies (Berente et al., 2019). Digital trace data eliminates various researcher-induced biases (Lindberg, 2020) and thereby leads to increasing the validity of findings (Grover et al., 2020). Furthermore, digital trace data is suitable to integrate a temporal dimension into research (Lindberg, 2020). As networks grow and change over time and thus change the underlying mechanisms that drive participation (Bock et al., 2015; Faraj & Johnson, 2011; Wang et al., 2013), a longitudinal approach was in order.

I created the dataset by collecting and merging data from two main sources. First, I collected data on available software components in the software ecosystem from the npm registry, which is the online database for JavaScript packages provided and maintained by npm Inc.[3]. It provides information such as the related project repository, version history, used software license, and dependencies to other projects for each project. Second, I collected data on the development process and the involved devel-

[3]https://www.npmjs.com/

opers from GitHub[4], which is the most popular collaborative development platform to date and is a rich source of data for OSS development (Kalliamvakou et al., 2016). Afterward, I combined both data sources by matching the provided repository information from the npm registry with the related software repository on GitHub.

For studies 1 and 2, I crawled the npm registry manually using automated data collection processes. For study 3, I used the dataset from Libraries.io[5] (Katz, 2020) because of the limited availability of historical data and difficulties in identifying all potentially relevant projects in the software ecosystem. Similarly, data from GitHub was not directly collected from the system due to API restrictions and limited access to historical data. In studies 1 and 2, I used archival data from GHArchive[6] whereas I used the dataset from GHTorrent[7] (Gousios, 2013) in study 3.

3.2 Network Perspective on Open Source Software Development and Software Ecosystems

To answer the three research questions, this dissertation adopts a network perspective, which is common for studies investigating the OSS phenomenon (e.g., Grewal et al., 2006; Peng et al., 2013; Singh et al., 2011; Tang et al., 2020). From this theoretical perspective, OSS development can be understood as a distributed system of developers and projects that are interconnected through relationships emerging from collaborations, knowledge exchange, and technical interdependencies.

In network theory, networks are represented as *graphs* that contain a set of *nodes*, which can represent various types of actors (e.g., individuals, organizations, or projects), and a set of *edges*, which can represent different types of relationships (e.g., friendship, collaboration, or employment) between the nodes (Borgatti & Halgin, 2011). Further, these graphs can be directed where the direction of the edge is

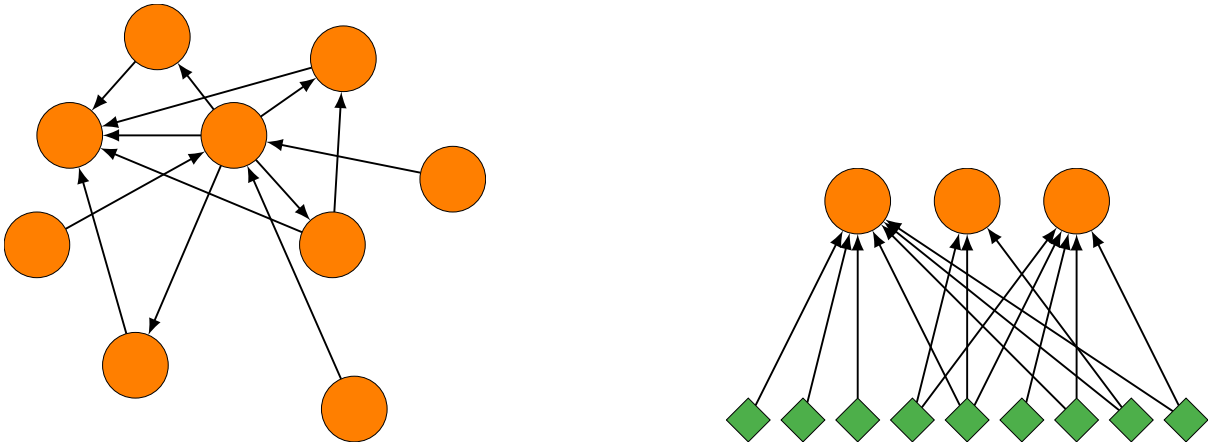
[4]https://www.github.com/
[5]https://www.libraries.io/
[6]https://www.gharchive.org/
[7]https://www.ghtorrent.org

specified and represents the directionality of the relationship from the sender to the receiver (Newman, 2018).

Another dimension in which networks differ is their *mode*, that is the number of different sets of actors represented in the network (Wasserman & Faust, 1994). The predominant types of networks are *one-mode networks*, consisting of a single set of actors, and *two-mode networks*, consisting of two sets of actors or one set of actors and one set of events (Faust, 1997; Wasserman & Faust, 1994). In two-mode networks, edges are only allowed between two distinct node types (Koskinen & Edling, 2012). Two-mode networks are often also referred to as affiliation or bipartite networks (Wasserman & Faust, 1994).

In this dissertation, I utilized directed one- and two-mode networks to represent OSS projects and their technical interdependencies, and developer affiliations. To represent the dependency network between projects, I used directed one-mode networks where nodes represented the software components or projects and edges reflected the dependencies directed from the depending towards the dependent project. This allowed me to differentiate a project's interdependent projects as either *upstream* or *downstream* dependencies (Valiev et al., 2018). To represent the relationship between developers and projects, I used two-mode networks. Here, the networks consisted of two node sets for developers and projects with edges representing the participation of a developer in a specific project. Figure 3.1 illustrates the two types of networks applied in this dissertation, where orange circles represent projects and green diamonds represent developers.

The following section briefly summarizes the data collection and analysis process for each study in this dissertation. For detailed explanations, I refer to the Method section of the related study.



(a) Dependency Network **(b)** Affiliation Network

Figure 3.1. Types of Networks

3.3 Empirical Studies on Technical Interdependencies

3.3.1 Influence on Developer Attraction

The first study investigates the first aspect of sustained participation and thereby investigates the effect of a project's position in the dependency network on its ability to *attract* resources in the form of developers by asking *what is the influence of a project's technical dependencies in the software ecosystem on its ability to attract developers*?

The theoretical argument draws from studies and theories that suggest that actors with more connections in a network are perceived as more prestigious, have increased access to information and resources, and signal a higher quality (Borgatti & Foster, 2003; Grewal et al., 2006; Newman, 2018; Setia et al., 2020). Thus, the study derives two hypotheses and argues that the number of upstream dependencies, representing the number of other projects the focal project depends on, and downstream dependencies, representing the number of other projects depending on the focal project, in the software ecosystem have similar effects on a project's perceived importance and attractiveness and grants access to developers.

In the first step, the network construction started with the identification of software components and related projects in the software ecosystem. In doing so, we identified

an initial sample of 3000 software components, followed by adding additional components using the expanding selection approach by Doreian & Woodard (1992). This approach is related to snowball sampling and starts with a fixed set of nodes and adds further nodes that are connected to the initial sample (Marsden, 2005). Here, the technical interdependencies between the components were utilized to collect connected components and their projects. We repeated this process until no further projects were identified.

The second step involved constructing the dependency network based on the final set of components. The dependency network was operationalized as a directed one-mode network, where the nodes reflected the software components and the edges reflected the technical dependencies between them. The state of the network was inferred using the provided timestamped version history of each component which also included the specific dependencies of each component version. Following this approach, we created snapshots of the network state at the beginning of each quarter between January 2021 and January 2022, which resulted in a total of 5 observations. In this study, the concept of a node's degree centrality was used to measure the position of the component in the network, which has been used in OSS research before (Daniel & Stewart, 2016; e.g. Grewal et al., 2006; Maruping et al., 2019; Singh, 2010). Furthermore, the component's degree centrality was differentiated between in- and out-degree, reflecting a component's up- and downstream dependencies.

In the third step, we accumulated the data generated from the network snapshots with the development activities derived from the related components' repositories. We applied various exclusion criteria to the sample to ensure that the components were still under development and that development activities could be related to a particular component because often components share the same repository. The final dataset consisted of 1,832 components and 7,328 component-quarter observations.

In the fourth and final step, we specified a series of Bayesian multilevel regression models to test the effects of interest. In doing so, we started with a baseline model

and subsequently added the variables of interest. We estimated each model using `R` and the package `brms` (Bürkner, 2017, 2018).

3.3.2 Influence on Developers' Participation Decisions

The second study investigates the dynamics in an affiliation network of developers and projects and asks *how do technical interdependencies influence a developer's participation decision in open source software ecosystems*? Thus, it focuses on the formation of the affiliation network between developers and projects by focusing on the individual developer's participation decision.

The study derives four hypotheses from the literature on developer participation. The derived hypotheses are then tested using SAOMs. We argue that these technical connections play an important role in developers' participation decisions.

In the first step, based on the data collected in study 1, 250 projects were randomly selected from the dataset and the participating developers were identified. Here, only developers with at least 5 activities during the observation from February until May 2021 were selected, which resulted in 1,172 developers. The reduction of the dataset was necessary to make the amount of data feasible for the application of SAOMs. The observation period was selected because it reduced time heterogeneity and kept the amount of change between two subsequent periods at an acceptable level.

In the second step, we constructed a two-mode affiliation network with projects and developers as two distinct node types and edges between them reflecting a developer's participation in the project. Again, network snapshots were created at the beginning of each month during the observation resulting in a total of 4 observations. We chose a monthly period duration because the statistical models require the network change to be within a certain range. The amount of network change between each period was checked using Jaccard coefficients for tie changes, which were between 0.45 and 0.51 and therefore suitable for the model (Conaldi et al., 2012). The total observation period of four months also corresponds to a typical release cycle in OSS development (Hahn

et al., 2008)

In the third step, SAOMs (Snijders, 1996, 2001) were applied to analyze the network change using the `R` package `RSiena` (Ripley et al., 2022). For a complete introduction to this type of model and detailed explanations of the statistics and assumptions, please see Snijders et al. (2010). In short, SAOMs are predictive models for dynamic networks that enable researchers to test endogenous and exogenous effects that influence the network's evolution (Cornwell, 2015). Model development was conducted via a forward selection of effects following guidelines by Snijders et al. (2010) and Ripley et al. (2022). During this process, we checked the resulting models for convergence. Afterward, the models were validated by performing a backward selection process.

3.3.3 Influence on Developer Retention

The third study focuses on the competition of projects for developers in software ecosystems and its effect on their sustainability by asking: *what is the effect of developer overlap in technological niches on the sustainability of open source projects in software ecosystems*?

This study draws from theories of organizational ecology (Hannan & Freeman, 1989) and adopts the concept of technological niches (Podolny et al., 1996; Podolny & Stuart, 1995) defined as communities in the ecosystem bounded by their technical relationships. The study argues that the number of shared developers (i.e., developer overlap) with other projects in the technological niche has a mutualistic effect (Barnett & Carroll, 1987) and therefore leads to increasing developer retention.

In this study, we used secondary data from the datasets from Libraries.io (Katz, 2020) and GHTorrent (Gousios, 2013). First, the dataset from Libraries.io includes data from different software ecosystems' package managers (e.g., npm for JavaScript, Maven for Java, or pip for Python) and holds information about the published software components, their release histories, and interdependencies. Second, GHTorrent archives the event histories of OSS projects hosted on GitHub by monitoring GitHub's public event

timeline.

In the first step, we identified all projects from the Libraries.io dataset that were part of the JavaScript ecosystem. For these projects, we collected all relevant data monthly from January 2017 until January 2019. As many projects in the dataset had a low activity or were inactive (Kalliamvakou et al., 2016), we applied various filters to only select active and legit projects. After this process, the final data set consisted of 2,403 projects and a total of 41,909 project-month observations.

Secondly, we created a one-mode network reflecting the technical interdependencies between the projects and a two-mode affiliation network representing the relationship between developers and the projects for each period. In the affiliation network, we created an edge between a developer and a project when the developer had participated in the project during that specific period. This step resulted in a total of 24 snapshots for each of the two networks.

The third step involved the calculation of the respective measurements by utilizing both networks as well as the version and dependency history, and development activities of each project.

The fourth and final step included the model specification for the Bayesian multilevel regression. Again, we used `R` and the `brms` package to estimate the models and tested each model for convergence and robustness.

Chapter 4

THE INFLUENCE OF DEPENDENCY NETWORKS ON DEVELOPER ATTRACTION IN OPEN SOURCE SOFTWARE ECOSYSTEMS

Mario Müller
University of Cologne

Christoph Rosenkranz
University of Cologne

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.

Published in the *Proceedings of the 43rd International Conference on Information Systems (ICIS)* in 2022

4.1 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 (Aksulu & 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 be-

tween 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; 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.

4.2 Related Work and Theoretical Background

4.2.1 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; 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; 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; 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.

4.2.2 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.

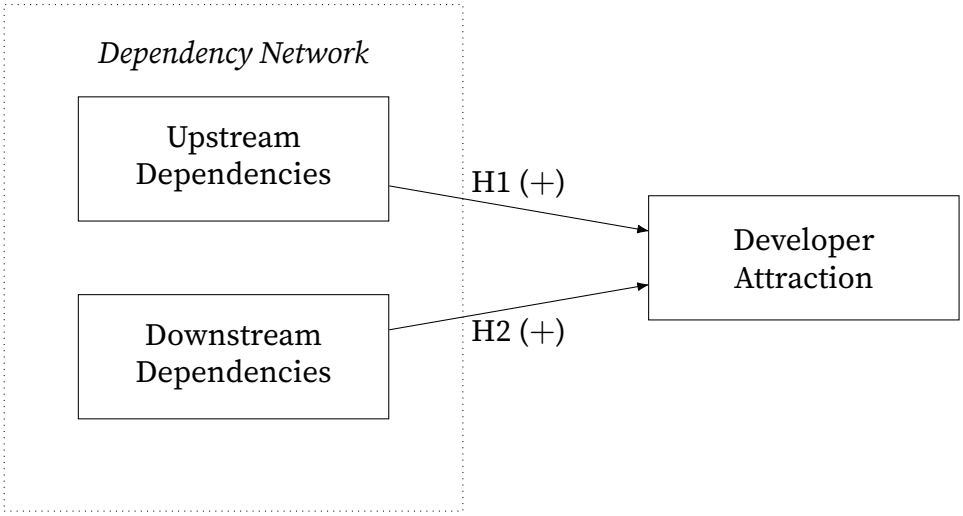


Figure 4.1. Research Model

4.3 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 4.1 summarizes our 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 (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.*

4.4 Research Methodology

4.4.1 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[8]. 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[9]. 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

[8]https://registry.npmjs.org/
[9]https://www.gharchive.org/

network feasible for the analysis, we started the network construction by identifying 3000 projects from Libraries.io's list of top ranked projects[10] in March 2022. As our boundary specification and sampling strategy, we followed the expanding selection approach by Doreian & 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 cri-

[10] https://libraries.io/search?order=desc&platforms=npm&sort=rank

teria. 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.

4.4.2 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 4.1 provides a summary of our used variables including a brief description of their operationalization and Table 4.2 reports the related descriptive statistics.

Table 4.1. Variable Definitions

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).
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 4.2. Descriptive Statistics for Selected Variables (N = 1,832 × 4)

Variable	Mean	Median	St. Dev.	Min	Max
Developer Attraction	16.40	4	54.38	0	1073
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	1290
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

4.5 Results

4.5.1 Dependency Network Evolution

Table 4.3 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.

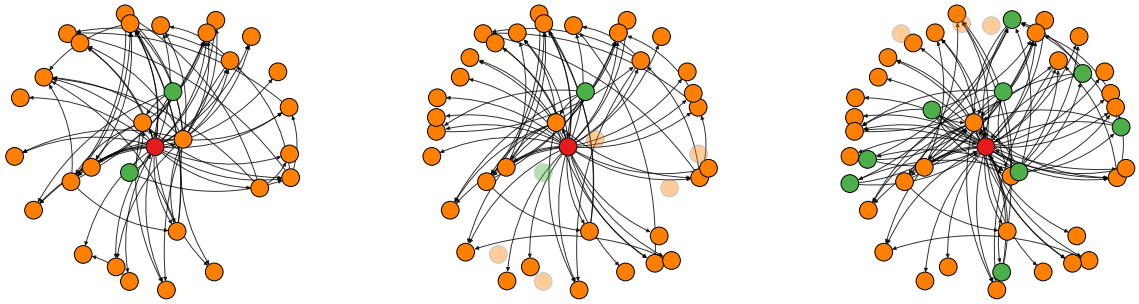
Table 4.3. Descriptive Statistics for Largest Component in Dependency Network

	Observ. Date	Nodes	Edges	Density	Network Diameter	Avg. Path Length	Avg. Degree
t_0	2021-01	10779	35082	0.000302	14	3.332	6.509
t_1	2021-04	10900	34997	0.000294	13	3.283	6.418
t_2	2021-07	10983	35119	0.000291	13	3.204	6.395
t_3	2021-10	11048	35133	0.000288	13	3.202	6.360
t_4	2022-01	11186	35505	0.000284	13	3.192	6.348

Figure 4.2 exemplarily shows the ego-network's evolution of the project "ember-auto-import"[11] 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

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

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.



(a) $t = 0$ **(b)** $t = 2$ **(c)** $t = 4$

Figure 4.2. Exemplary Evolution of an Ego-Network

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.

4.5.2 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 in-

fluences 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 ($\widehat{R}$ = 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.

4.5.3 Regression Results

Table 4.4 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).

Table 4.4. Correlation Matrix

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	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

In Table 4.5 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 (β = .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% 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 (β = .63***; 95%-CI: [.58, .67]). We also found positive effects for community interest (β = .26***; 95-CI%: [.24, .29]). Interestingly, compared to previous findings, we found a negative effect of release activity in previous periods (β = -.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 (β = .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 4.3 plots the posterior uncertainty intervals for each variable.

4.6 Discussion

4.6.1 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 legit-

Table 4.5. Bayesian Multilevel Model Results

	DV: Developer Attraction$_t$					
	Model 1		Model 2		Model 3	
Fixed Effects						
Intercept	-0.58^{***}	(0.03)	-0.67^{***}	(0.03)	-0.64^{***}	(0.03)
Developer Pool Size$_{t-1}$	0.71^{***}	(0.02)	0.65^{***}	(0.02)	0.63^{***}	(0.02)
Community Interest$_{t-1}$	0.24^{***}	(0.01)	0.26^{***}	(0.01)	0.26^{***}	(0.01)
Release Activity$_{t-1}$	-0.09^{***}	(0.01)	-0.10^{***}	(0.01)	-0.09^{***}	(0.01)
Project Age$_t$	-0.00	(0.00)	-0.00	(0.00)	-0.00	(0.00)
Organizational Ownership	0.17^{***}	(0.02)	0.14^{***}	(0.02)	0.17^{***}	(0.03)
Upstream Dependencies$_{t-1}$			0.12^{***}	(0.01)	0.12^{***}	(0.01)
Downstream Dependencies$_{t-1}$			-0.00	(0.01)	0.00	(0.01)
Random Effects						
σ_{Package}	0.28^{***}	(0.02)	0.31^{***}	(0.02)		
$\sigma_{\text{Repository Owner}}$					0.26^{***}	(0.02)
$\sigma_{\text{Repository Owner/Package}}$					0.22^{***}	(0.02)
$N_{\text{Observations}}$	7328		7328		7328	
N_{Packages}	1832		1832		1832	
$N_{\text{Repository Owner}}$					1016	
LOOIC	39707.33		39518.62		39338.04	

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

imacy 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

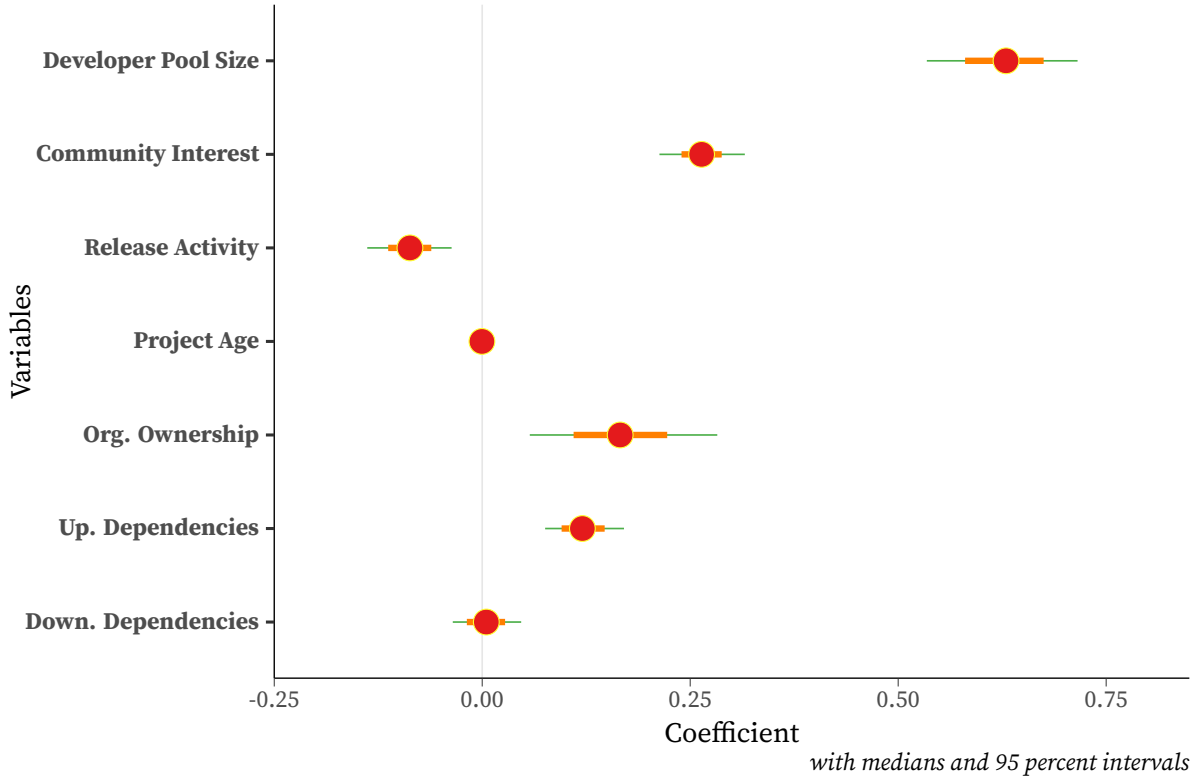


Figure 4.3. Posterior Uncertainty Intervals

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.

4.6.2 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.

4.6.3 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.

4.7 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.

4.8 Acknowledgements

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

Chapter 5

THE ROLE OF DEPENDENCY NETWORKS IN DEVELOPER PARTICIPATION DECISIONS IN OPEN SOURCE SOFTWARE ECOSYSTEMS: AN APPLICATION OF STOCHASTIC-ACTOR ORIENTED MODELS

Mario Müller
University of Cologne

Christoph Rosenkranz
University of Cologne

Abstract

Open source software relies on the contributions of developers who participate voluntarily in projects. While prior research has investigated social characteristics, relations, and connections that influence a developer's participation, we argue that the technical relations and connections of projects, which emerge through dependencies between packages in software ecosystems, play a focal role in that decision as well. We empirically test these assertions by applying stochastic actor-oriented models to an affiliation network in the JavaScript software ecosystem. Our results show that while the number of dependencies of a project does not influence participation, developers are more likely to participate in projects to which their own projects have dependency relations. This study thereby contributes to the understanding of antecedents that influence developers' participation decisions by highlighting the importance of project interdependencies in software ecosystems.

Published in the *Proceedings of the 56th Hawaii International Conference on System Sciences (HICSS)* in 2023

5.1 Introduction

Unlike software development in organizations, *open source software (OSS)* projects are usually undertaken by a decentralized community of developers who collaborate via development platforms to produce the software (Fang & Neufeld, 2009; Lindberg et al., 2016). Often, these developers are not paid (Crowston, 2011; Roberts et al., 2006), although a fraction is employed by companies specifically to help in OSS development (von Krogh et al., 2012). Thereby, OSS projects depend on the continuous, voluntary participation of distributed developers (Mockus et al., 2002; Roberts et al., 2006).

Previous research has therefore investigated what factors influence a developer's decision to participate in a project. In doing so, studies have focused on individual-related factors that lead to intrinsic and extrinsic motivations (von Krogh et al., 2012), as well as project-related factors, such as organizational sponsorship or license restrictions (Stewart et al., 2006). Researchers also have taken the social structure of projects and their community into account by applying techniques from social network analysis (e.g., Grewal et al., 2006; Hahn et al., 2008; Oh & Jeon, 2007).

While prior research has shown that social relations and connections are important antecedents of participation, existing studies have ignored another essential component: *technical relations and connections*, that is, dependencies that arise in software ecosystems through the reuse of software packages (Decan et al., 2019; Haefliger et al., 2008). We argue that these technical connections play an important role in developers' participation decision due to four key reasons.

First, packages that are reused extensively by others are important for the health and stability of the entire ecosystem, making them more attractive for developers who want to gain reputation and become visible in the community (Hu et al., 2012). Second, by reusing packages, developers can work on tasks they actually enjoy working on (Haefliger et al., 2008), therefore making packages with more reuse more attractive for potential participants. Third, developers tend to support other projects that depend on their provided package (Bogart et al., 2016). In some cases, the provided

packages are even spun off from larger projects, which are then maintained by the same developers (Valiev et al., 2018). Fourth, developers tend to participate in projects that they use themselves (Shah, 2006), which is reflected in a dependency towards the used package.

The aim of this study is to empirically test these assertions. To do so, we theorize the role of dependency networks for developers' decisions based on prior literature. Then, we adopt a dynamic network modeling approach and analyze the affiliation network of packages in a large OSS ecosystem over a period of four months. We apply stochastic actor-oriented models (Snijders, 1996) to investigate the effect of software dependencies on the evolution of affiliation networks between developers and OSS projects. We find that while the number of up- and downstream dependencies of a project does not affect its ability to attract participants, developers tend to contribute to projects that are either up- or downstream dependencies of that particular project. These findings contribute to the literature on developer participation in OSS projects by focusing on the technical connections in the form of package interdependencies of OSS projects in software ecosystems. This offers a more complete view than the prevailing focus on social relations and connections only.

The remainder of this paper is structured as follows. In Section 2, we provide an overview of related work on developer participation and dependency networks in OSS ecosystems, and we develop our hypotheses. In Section 3, we describe our data collection process and network construction, and give a brief overview about stochastic actor-oriented models. In Section 4, we report the results of our analysis. Finally, in Section 5, we discuss our results, implications, and limitations.

5.2 Theoretical Background

5.2.1 Developer Participation in Open Source Software Projects

OSS development is driven by a decentralized community of developers that mostly contribute voluntarily to projects and collaborate via online development platforms and management software such as GitHub (Fang & Neufeld, 2009; Roberts et al., 2006). OSS projects rely on the continuous participation of their community members (Roberts et al., 2006; Shah, 2006) and need to attract and retain developers (Butler, 2001; Crowston et al., 2003) in order to stay viable. Therefore, the question of what motivates developers to participate in a particular OSS project has been central to OSS research (Roberts et al., 2006).

Previous research has focused on individual characteristics of developers as well as project-related aspects that influence developers' participation decisions, and various project- and individual-related factors have been identified. For example, project factors include license restrictions (Stewart et al., 2006), organizational sponsorship (Shah, 2006; Stewart et al., 2006), and the modularity of a project's codebase (Baldwin & Clark, 2006). Individual factors that drive participation include fun or enjoyment (Shah, 2006), learning and developing skills (von Hippel & von Krogh, 2003), or increasing reputation (Hu et al., 2012) and career advancements (Lerner & Tirole, 2002).

Moreover, due to the community-based model of developing OSS and the importance of social relations, connections, and structures (Grewal et al., 2006), researchers early on have adopted a network perspective on OSS and have investigated the effect of network structures on participation. Related to the social network of developers, for example, previous collaborations with the project initiators increase the likelihood of joining a project (Hahn et al., 2008), and the decision to remain involved in a project is influenced by other neighboring developers (Oh & Jeon, 2007).

While these network studies highlight the importance of social interactions of developers, they largely neglect the technical relations and connections in the form of package

interdependencies between projects. In the following, we focus on these interdependencies that arise in projects embedded in software ecosystems.

5.2.2 Dependency Networks in Software Ecosystems

OSS is not built from scratch but relies on reuse of code and already implemented functionality (Haefliger et al., 2008; Sojer & Henkel, 2010). This functionality is typically provided via *packages*, which are "reusable code or set of components that can be included in other applications by using dependency management tools" (Kikas et al., 2017). Modern programming languages ease the process of reuse by providing package managers that allow developers to publish and use packaged software components (Cox, 2019). Adding a package to a project creates a dependency relationship, making the project dependent on the package to function (Bogart et al., 2016). This practice results in so-called *software ecosystems*, "large collections of interdependent software components that are maintained by large and geographically distributed communities of collaborating contributors" (Decan et al., 2019).

From a package's perspective, dependency relationships exist in two directions. In the ecosystem, the package has other packages depending on it, so-called *downstream dependencies*, and it might also depend on other packages itself, which results in *upstream dependencies* (Valiev et al., 2018).

The reuse of packages allows developers to save time and to implement proven solutions to their software (Haefliger et al., 2008; von Hippel & von Krogh, 2003). The functionality provided by the reused package enables them to focus on tasks they actually like to work on (Haefliger et al., 2008). Therefore, projects with extensive reuse of packages, reflected in the number of upstream dependencies, should become more attractive for developers, which leads to our first hypothesis:

> *H1a: Developers tend to participate in packages with more upstream dependencies.*

Furthermore, the number of downstream dependencies reflects the importance and value of a package in the software ecosystem. This is because the more other packages depend on the focal package, the higher the number of downstream dependencies, which also makes it a proxy for a package's user base (Valiev et al., 2018). Since one driving factor of developer participation is the potential gain in reputation (Hu et al., 2012), important and valued packages in the software ecosystem should become more attractive to participate in. Hence, we propose:

> *H1b: Developers tend to participate in packages with more downstream dependencies.*

However, dependencies can create issues in case of breaking changes (i.e., a change in a package that potentially causes other packages to fail). In order to counter breaking changes, package providers usually support and coordinate with their dependents by announcing changes or helping to migrate to another version (Bogart et al., 2016). Moreover, smaller packages are often split off from larger projects, which are then maintained by the same developers (Valiev et al., 2018). Therefore, developers also become affiliated with a package's upstream dependencies. Hence, we propose:

> *H2a: Developers tend to participate in a package when they are also affiliated with its upstream dependencies.*

Furthermore, existing studies have shown that developers tend to participate in projects that they use themselves (Shah, 2006). These developers, often referred to as "peripheral developers", are thereby motivated to improve the package for their own use and are important for the quality assessment and enhancement of a project (Setia et al., 2012). For example, developers participate by submitting bug reports or pull requests to the dependent package (Setia et al., 2012). Hence, we propose that this also holds true for the reuse of packaged software components:

> *H2b: Developers tend to participate in a package when they are also affiliated with its downstream dependencies.*

5.3 Research Method

5.3.1 Data and Network Construction

In order to test our hypotheses, we focus on the JavaScript ecosystem, which is one of the largest software ecosystems in the world (Decan et al., 2019). Data was collected from two data sources to construct the dependency network between packages and capture the development activities. Meta and dependency data was collected from the npm registry[12], whereas development activities were collected from GitHub for the package-related repositories. Due to API restrictions, we used the event archives provided by GHArchive[13]. Both datasets were linked by matching repository URLs provided in the packages' metadata.

In network studies, setting boundary conditions is important (Marsden, 2005). For our boundary specification and sampling strategy, we followed the "expanding selection" approach (Doreian & Woodard, 1992), which starts with a fixed set of nodes and adds further nodes linked to the initial set. Thus, we started our data collection with the selection of 3,000 packages identified from Libraries.io's list of top ranked packages[14] in March 2022. Based on that initial set, we added packages to the set that were listed as a runtime dependency between 2019 and 2022. This time restriction helped in avoiding adding abandoned or by now irrelevant packages. The process was repeated for every newly identified package which allowed us to identify all relevant packages further down the dependency tree. In sum, this resulted in a total of 12,678 packages.

To reduce the number of nodes and thereby make the size of the network feasible for the analysis, we performed additional steps for the selection of our final sample of packages and related developers. First, we excluded all packages that shared a repository with other packages to make sure that developer activities were specifically targeted at a particular package. Second, we checked for the development activity during

[12] https://registry.npmjs.org/
[13] https://www.gharchive.org/
[14] https://libraries.io/search?order=desc&platforms=npm&sort=rank

the observation and only included packages with activity in every period. From the resulting set of packages, we randomly selected 250 packages. Third, we collected all developers participating in the sampled packages and only included developers with at least 5 activities (i.e., comments, commits, actions related to issues and pull requests) during the observation. This resulted in a set of 1,172 developers.

Based on the selected packages and identified developers, we constructed the affiliation network between developers and packages. An affiliation network, also referred to as two-mode or bipartite network, is a graph with two distinct node types (i.e., developers and packages), where edges are only allowed between different types of nodes (Koskinen & Edling, 2012). To construct the network, we collected all activities of the selected developers towards the sampled packages for each observation period and created edges between developer and package in case there existed an activity in the particular period.[15]

We observed the affiliation network from February until May 2021. We opted for the four-month window to reduce time heterogeneity and keep the amount of change between periods at a sufficient level for analysis, which is also consistent with previous research (e.g., Hahn et al., 2008; Tang et al., 2020). Thereby, we created snapshots of the network state at the beginning of each month. This resulted in a total of four observations. Figure 5.1 shows the state of the affiliation network at the last observation point. The layout was generated using the Fruchterman-Reingold algorithm (Fruchterman & Reingold, 1991).

5.3.2 Data Analysis

We applied stochastic actor-oriented modeling (SAOM) (Snijders, 1996, 2001, 2005) by using the R package RSiena (Ripley et al., 2022). In the following, we briefly summarize the underlying assumptions behind SAOMs. For a detailed description, we refer to Snijders (1996, 2001) and Snijders et al. (2010).

[15]All data and scripts to construct the data as well as the analysis results can be found here: https://tinyurl.com/2p9avyrp

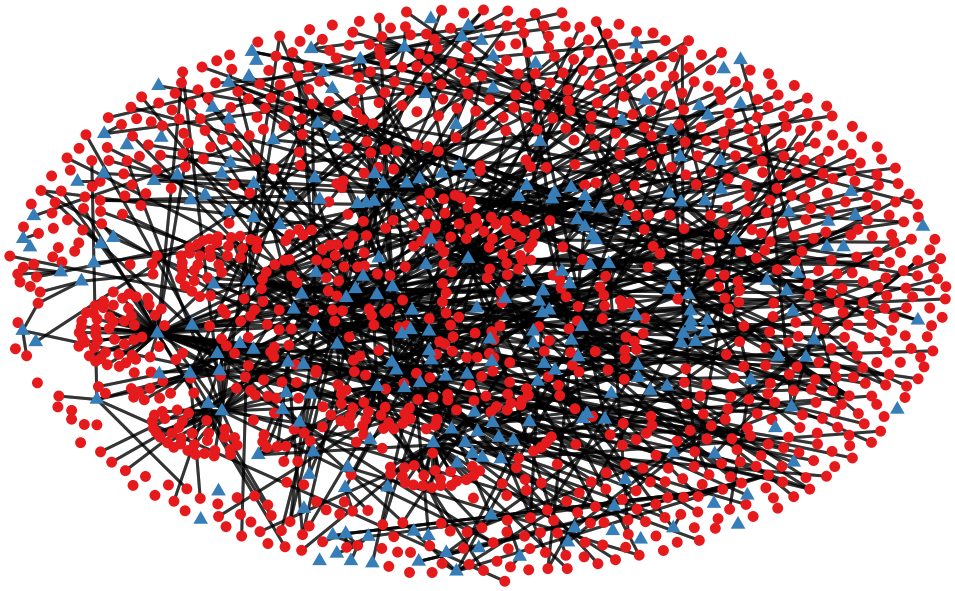


Figure 5.1. Affiliation network at observation t_4 (developers as red circles; packages as blue squares)

SAOMs are the most advanced predictive models for dynamic networks that allow the testing of various mechanisms influencing the evolution of a network (Cornwell, 2015). SAOMs assume a continuous change process of the network structure, which is represented by Markov chain models (Holland & Leinhardt, 1977) with a continuous-time parameter, although the network is observed at discrete points in time (Snijders, 1996, 2001). The change process consists of two sub-processes: the (1) change opportunity process and the (2) change determination process (Snijders et al., 2010). Thereby, when an actor has the opportunity to change ties, determined by the rate function, the probabilities of change are determined by the evaluation (or objective) function (Snijders et al., 2010). The evaluation function thereby represents the relative attractiveness of establishing a tie (Conaldi et al., 2012). The evaluation function includes effects related to the structural properties of the network (endogenous effects) and effects based on the attributes of an actor in the network (exogenous effects) (Snijders et al., 2010).

5.3.3 Model Development

We followed the guidelines for model development provided by Snijders et al. (2010) and Ripley et al. (2022). Thereby, we specified the model for the dynamics in the affiliation network via forward selection of theoretically grounded effects and tested these effects using the score-type test proposed by Schweinberger (2012). During the selection process, we checked the t-ratios for convergence for each effect, which indicate the stability of parameter estimates across simulations and should be below an absolute value of 0.1 (Kalish, 2020; Snijders et al., 2010). Furthermore, we checked the overall maximum convergence of the estimated models during the selection process, which should be below the threshold of 0.25 (Ripley et al., 2022). We started with endogenous effects, followed by exogenous effects. Results were then validated by performing a backward selection. We also tested for time heterogeneity (Lospinoso et al., 2011) and accounted for the composition change of developers (Huisman & Snijders, 2003).

In terms of endogenous effects, by default, an effect for *outdegree* (density), which represents the tendency of an actor to have ties at all, is included in the model (Snijders et al., 2010). Also, an effect for the tendency toward *transitivity* should be included in the model (Snijders et al., 2010). In two-mode networks, transitivity is expressed by the number of four-cycles (Robins & Alexander, 2004). This effect reflects the extent to which actors make the same choices as their peers (Ripley et al., 2022). Another set of effects that should be accounted for during the model selection process are degree-related effects (Snijders et al., 2010). Both in- and out-degree are important positional characteristics of nodes that drive network dynamics (Snijders et al., 2010) and should be included when high dispersion in in- and out-degrees is present (Ripley et al., 2022). Based on our theoretical foundation, we included the *in-degree popularity* effect, which accounts for the tendency of dispersion in in-degrees of packages (i.e., number of participating developers). The *out-degree activity* effect, which reflects the tendency of dispersion in out-degree of developers (i.e., the number of packages a developer participates in), was also significant and contributed to the convergence of the overall

model and was therefore also included.

Furthermore, we included several exogenous actor-specific effects. We started by including control effects. Related to packages, we controlled for *community interest* (measured as the number of GitHub stars given to a package during a period), *release activity* (measured as the number of version releases of a package during a period), *license restrictiveness* (by adopting the categorization of Lerner & Tirole (2005) into (1) permissive, (2) restrictive, and (3) highly restrictive licenses), and *age* (measured as the number of months from the package's creation date until the end of a period). These effects were modeled as receiver effects, which means that actors with higher values of the covariate tend to have higher in-degrees (Snijders et al., 2010). Related to developers, we controlled for the overall *activity of a developer* by measuring the number of activities a developer performed during a period. These include comments made on issues or pull requests, status changes of issues or pull requests (e.g., opening or closing), and pushing commits to the repository. This effect was modeled as a sender effect, which means that actors with higher covariate values tend to have higher out-degrees (Snijders et al., 2010).

Finally, we included exogenous actor-specific receiver effects as well as dyadic effects for the influence of the dependency network. First, we measured the number of *up- and downstream dependencies* of a package in a period. Second, we included dyadic effects accounting for the relation between a developer and a package. The dyadic covariate thereby reflects if the developer also *participates in an upstream or downstream dependency of the targeted package* in the period. Therefore, the effect expresses the extent to which participation becomes more likely if a developer also participates in an upstream or downstream dependency.

Because the estimation operations of RSiena are not scale-independent, it is advised to scale covariates to achieve standard deviations between 0.1 and 10 (Ripley et al., 2022). Therefore, we log-transformed the values of our actor-specific covariates. Table 5.1 summarizes the relevant effects used in this study.

Table 5.1. Summary of Endogenous Network Effects and Exogenous Actor-Specific Covariates

Parameter	Description
Outdegree (Density)	Tendency of developers to participate in packages.
Transitivity	Tendency of developer pairs to participate in the same package.
Package Popularity	Tendency of popular packages to attract more developers.
Developer Participation	Tendency of developers that participate in more packages to engage in extra packages.
Package License	Tendency of developers to participate in packages with specific license restrictiveness.
Developer Activity	Tendency of developers with a higher level of activity to participate in more packages.
Community Interest	Tendency of packages with higher community interest to attract more developers.
Release Activity	Tendency of packages with more releases to attract more developers.
Package Age	Tendency of packages with higher age to attract more developers.
Package Upstream Dependencies	Tendency of packages with more upstream dependencies to attract more developers.
Package Downstream Dependencies	Tendency of packages with more downstream dependencies to attract more developers.
Participation in Upstream Dependency	Tendency of developers to participate in a package if they also participate in an upstream dependency of that package.
Participation in Downstream Dependency	Tendency of developers to participate in a package if they also participate in a downstream dependency of that package.

5.4 Results

First, Table 5.2 summarizes network descriptives. Over all periods, the network's density is relatively low. During all four periods, the network's density, and its average degree declines. This is a result of the joining actors in periods 2 to 4. Therefore, the number of possible ties increases, but the number of ties that are actually established does not increase accordingly.

Second, we report the tie changes of the networks in subsequent observations (Table

Table 5.2. Network Descriptives

Period	1	2	3	4
Density	0.005	0.004	0.003	0.003
Avg. Degree	1.211	0.909	0.736	0.647
No. Ties	758	807	788	758
Miss. Fraction	0.466	0.242	0.087	0.000
Joined Actors	–	262	182	102

5.3). For example, between observation 1 and 2 (1 → 2), 88 developers newly participated in packages (0 → 1), 325 developers discontinued their participation (1 → 0), and 433 developers continued to participate in the related packages. The Jaccard index represents the amount of change between two observations (Snijders et al., 2010). For SAOMs, the suggested value is between 0.2 and 0.9 (Conaldi et al., 2012). In our case, the values of the Jaccard coefficients for tie changes between observations are between 0.45 and 0.51.

Table 5.3. Network Tie Changes between Periods

	0 → 0	0 → 1	1 → 0	1 → 1	Distance	Jaccard
1 → 2	155654	88	325	433	413	0.51
2 → 3	221043	150	362	445	512	0.47
3 → 4	266517	195	345	443	540	0.45

5.4.1 Model Estimates

We estimated the models using the procedure of method of moments (Snijders, 1996). Table 5.4 presents the results of the models. We report parameter estimates and standard errors for rate effects, endogenous network effects, actor-specific and dyadic covariates. Model 1 includes the effects of the endogenous network structure. Model 2 adds to Model 1 the effects of exogenous actor-specific covariates for our control variables. Model 3 adds to Model 2 the effects of the dependency network related covariates (i.e., our hypotheses). All models were run for 3,000 iterations in phase 3. In all models, t-ratios for convergence for each effect are below the suggested threshold of

|0.1| (Kalish, 2020; Snijders et al., 2010) and the overall maximum convergence ratios are below the suggested value of 0.25 (Ripley et al., 2022).

In the following, we report the estimates for Model 3 in more detail. In general, the rate function indicates the expected number of opportunities that developers have to change their affiliation with a project (Conaldi et al., 2012). Hence, the parameter estimates can be interpreted as the number of changes developers make regarding their affiliations over time. For example, developers make on average 0.8 changes in the last period. This rate remains relatively stable over time and indicates that developers are reluctant to change their affiliation with a project. Furthermore, the developer's activity level has a positive and significant effect on the number of change opportunities (0.05; $p < 0.01$), indicating that more active developers tend to change their affiliation more often.

The evaluation function controls for the subjective utility for developers when changing their affiliation (Conaldi et al., 2012). In terms of endogenous network effects, we observe that the estimate for the out-degree of the developers is negative and highly significant (-5.52; $p < 0.001$). This indicates that developers show a lower tendency to participate in new packages over time. Also, package popularity has a small but positive and significant effect (0.04; $p < 0.001$), which indicates that already popular packages are more likely to attract additional developers. Transitivity is positive but not statistically significant (0.52; $p < 0.1$), which does not indicate a significant tendency towards clustering in the network. Thus, developers do not seem to follow their previous collaborators to new packages in the future. The parameter estimate for developer participation is not significant.

In terms of exogenous effects of actor-specific covariates, we first focus on the effects of interest related to the effect of a package's dependency network on developer participation. We find that the number of up- and downstream dependencies of a package does not influence its ability to attract developers, with both estimates being not significant (both *H1a* and *H1b* are not supported). However, the estimates for both dyadic

effects of up- (1.93; $p < 0.001$) and downstream dependencies (1.34; $p < 0.01$) are both positive and significant. This indicates that developers are more likely to participate in a package if they also participate in another package that is a down- or upstream dependency of that specific package. Hence, we observe support for both *H2a* and *H2b*.

We conclude by reporting estimates for our control variables. For packages, the estimates for community interest (0.20; $p < 0.001$) and package age (-0.20; $p < 0.01$) are significant. Furthermore, the estimate for developer activity is positive and highly significant (0.47; $p < 0.001$), which indicates that developers with a higher level of activity tend to participate in more packages. In contrast to prior findings, both release activity (0.13; $p < 0.1$) and license restrictiveness (0.31; $p > 0.1$) of a package are not significantly influencing a package's ability to attract developers.

5.4.2 Goodness-of-Fit

The simulation-based goodness-of-Fit (GOF) test for the estimated models tests the hypothesis that the model which generated the observed data is equal to the fitted model (Lospinoso & Snijders, 2019). The approach implemented in RSiena takes an auxiliary statistic, that is, a feature of the data not included in the model and therefore not a function of the estimation and compares it with the observed data and their distribution (Lospinoso & Snijders, 2019). We tested auxiliary statistics for outdegree and indegree distributions. Both Model 1 and 2 performed poorly for both in- and outdegree distributions, but the fit for Model 3 meets the criteria of $p > 0.05$ for the Mahalanobis distance-combination, indicating a good model fit (Kalish, 2020; Lospinoso & Snijders, 2019).

Figure 5.2 shows the results of the GOF tests for Model 3. Observed values are indicated by the number connected by the red line. The simulated statistics are represented by the violin plots. The dotted lines represent the 95th percentile bands. Wald-type tests

Table 5.4. Estimated Stochastic Actor-Oriented Models for Affiliation Networks

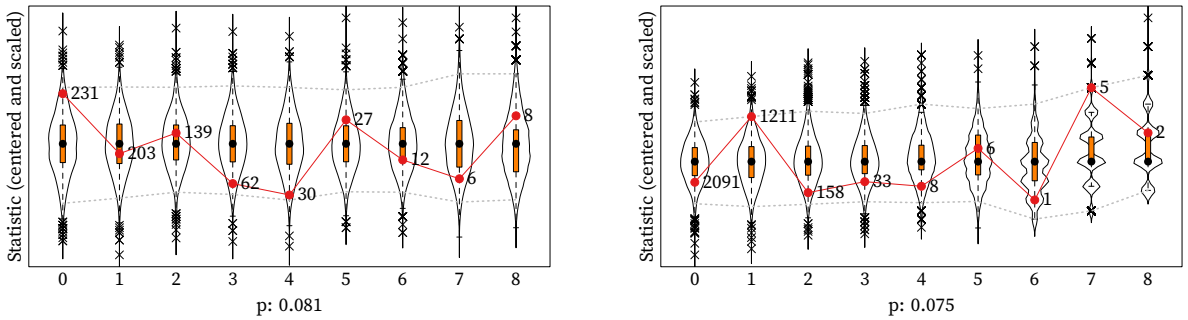
	Model 1		Model 2		Model 3	
Effects	Estim.	S.E.	Estim.	S.E.	Estim.	S.E.
Rate Function						
Rate of Network Change 1	0.72^{***}	(0.04)	0.77^{***}	(0.04)	0.77^{***}	(0.04)
Rate of Network Change 2	0.74^{***}	(0.04)	0.81^{***}	(0.04)	0.81^{***}	(0.04)
Rate of Network Change 3	0.70^{***}	(0.04)	0.80^{***}	(0.04)	0.80^{***}	(0.04)
Effect of Dev. Activity on Rate	0.10^{***}	(0.02)	0.05^{*}	(0.02)	0.05^{**}	(0.02)
Evaluation Function						
Endogenous Network Effects						
Outdegree (Density)	-5.27^{***}	(0.14)	-5.48^{***}	(0.12)	-5.52^{***}	(0.13)
Transitivity (Four-Cycles)	0.50	(0.38)	0.54^{*}	(0.25)	$0.52^{\dagger}$	(0.29)
Package Popularity	0.05^{***}	(0.00)	0.04^{***}	(0.00)	0.04^{***}	(0.01)
Developer Participation	0.27^{***}	(0.04)	0.05	(0.04)	0.04	(0.04)
Exogenous Actor-Specific Covariates						
Developer Activity			0.49^{***}	(0.04)	0.47^{***}	(0.04)
Community Interest			0.19^{***}	(0.04)	0.20^{***}	(0.05)
Release Activity			0.12	(0.08)	$0.13^{\dagger}$	(0.08)
License Restrictiveness			0.31	(0.33)	0.31	(0.33)
Package Age			-0.18^{*}	(0.08)	-0.20^{**}	(0.08)
Upstream Dependencies					0.05	(0.05)
Downstream Dependencies					0.08	(0.06)
Dyadic Covariates						
Participation in Upstream Dependency					1.93^{***}	(0.56)
Participation in Downstream Dependency					1.34^{**}	(0.46)
Wald χ^2 Statistics (df)	320.19^{***}	(4)	231.63^{***}	(5)	14.64^{***}	(4)
Gen. score χ^2 Statistics (df)	394.84^{***}	(4)	198.98^{***}	(5)	19.20^{***}	(4)

$^{***}p < 0.001$; $^{**}p < 0.01$; $^{*}p < 0.05$; $^{\dagger}p < 0.1$; Convergence t-ratios for all effects $< |0.1|$.
Overall maximum convergence ratios < 0.13

and score-type tests for the joint significance of the added effects as reported in Table 5.4 also indicate an improvement of model fit and strong significance of the added effects (p < 0.001).

5.5 Discussion

In this study, we developed and estimated a dynamic network model for the analysis of the evolution of an affiliation network in a large OSS ecosystem. With this model and the test of associated hypotheses, we contribute to the literature on the participation decisions of developers by focusing on the role of technical relations and connections in the form of package interdependencies, thus introducing (package-based) project



(a) Indegree Distribution **(b)** Outdegree Distribution

Figure 5.2. Goodness of Fit of Model 3 for Indegree and Outdegree Distributions

dependencies as important antecedents for developer participation decisions.

Our results show that developers not only contribute to packages they use themselves (*H2b*), but also to packages that make use of their own packages (*H2a*). This shows that projects benefit from their dependencies in both directions through contributions made by developers of interdependent projects. While previous research already mentioned need-driven motivation as one antecedent for participation (Shah, 2006), this empirically shows for the first time that users of a package do not free-ride but also contribute back. Moreover, our findings show that package providers contribute and provide help to their dependent packages. Even though we find that the likelihood of contribution increases, the actual type of the contribution remains an open question and provides opportunities for future research. However, the number of dependencies by themselves do not influence a developer's decision; a large number of up- or downstream dependencies does not equal more attractiveness, thus we did not find support for *H1a* and *H1b*.

Furthermore, our results show that developers only rarely change affiliations, as reflected in the rate effects. Given that the participation in a new project comes with associated costs related to required knowledge, skill, and necessary time to get involved and familiar with a project (von Krogh et al., 2003), this is not surprising.

In comparison to prior studies, our results support previous findings related to the influence of community interest on a project's attractiveness (Subramaniam et al., 2009)

and its decreasing ability to attract developers with growing age (Chengalur-Smith et al., 2010). Interestingly, we did not find an effect of license restrictiveness. This might be related to the fact that most of the analyzed packages are released under the MIT license and, in general, we did not see a great variety of used licenses in the overall JavaScript ecosystem.

From a research perspective, our study demonstrates the benefits and potential insights that can be gained by applying dynamic network models to affiliation networks in OSS projects. From a practical perspective, our results highlight that community efforts should be directed not only towards a project itself, but also to interdependent projects that build upon or are used by the focal project. This may also help to counter negative effects such as breaking changes.

As with all research, this study has several limitations. First, we did not include all available packages in the JavaScript ecosystem. However, by following the selected sampling approach, we were able to identify and analyze the most important and used packages during our observation. Furthermore, we only focused on one specific ecosystem. Hence, future research could analyze if the shown mechanisms are also present and influential in other software ecosystems.

Second, we focused only on effects driving the structural evolution and formation of the affiliation network and neglected the co-evolutionary aspect of its structure on potential outcomes, such as a project's sustainability and success. Hence, future research should build upon this study by including project-related outcomes and their interplay with both social and technical network structures.

Third, the dependency network has only partially been included in our analysis by projecting it as actor and tie variables. Future research could therefore explicitly include its structure by investigating the co-evolution of the dependency and affiliation network.

Fourth, we used digital trace data, which entails potential validity problems (Howison et al., 2011). Even though we performed several checks to increase our data's confiden-

tiality, we cannot ensure complete accuracy due to the secondary nature of our data sources.

5.6 Conclusion

In sum, our study theoretically and practically contributes to our understanding of antecedents of developer participation in OSS by introducing and highlighting the role of technical interdependencies of projects in a software ecosystem. Thereby, we underline the importance of a socio-technical lens on the OSS phenomena that considers the social as well as technical structures and provide several opportunities and directions for future research.

Chapter 6

SUSTAINING OPEN SOURCE SOFTWARE PROJECTS: AN ECOLOGICAL PERSPECTIVE ON TECHNOLOGICAL INTERDEPENDENCIES IN SOFTWARE ECOSYSTEMS

Mario Müller
University of Cologne

Christoph Rosenkranz
University of Cologne

Abstract

The sustainability of open source software projects relies on the participation of developers in their community. This study draws on organizational ecology theory to investigate how interdependencies between projects in a technological niche in software ecosystems influences a project's ability to retain its developers. We argue that interdependencies through shared developer resources lead to mutualistic dynamics in a project's technological niche that favor sustained participation. We empirically tested our hypotheses by collecting and analyzing longitudinal data from the JavaScript ecosystem and testing them using a Bayesian multilevel approach. We find that developer overlap in technological niches increases a project's ability to retain developers. This effect becomes stronger with increasing project age. Furthermore, we highlight the role of a project's environment by adopting the concept of a technological niche to software ecosystems. With these results, we contribute to the literature streams on open source sustainability and participation of developers.

Submitted to *MIS Quarterly* in April 2023

6.1 Introduction

The community-based model of *open source software* (OSS) development, and the software components and applications that have resulted from it, form a considerable part of contemporary organizations' IT infrastructures (Nagle, 2019). Estimates suggest that OSS is the basis for up to 90% of any current software (Nagle et al., 2020).

Fundamentally, OSS projects rely on the voluntary participation of developers (Fang & Neufeld, 2009; Roberts et al., 2006). However, volunteering developers can withdraw their participation at any time; therefore, they represent an unstable resource (von Krogh et al., 2012). Thus, many OSS projects at one point are no longer sustainable or completely fail due to insufficient participation of developers (Fang & Neufeld, 2009). Consequently, OSS projects must not only continuously attract new contributors but also retain their active developers (Butler, 2001; Crowston et al., 2003), that is, keep them involved in the project for a longer period.

Previous research has investigated individual-level and project-level characteristics that influence developers' initial and continuous participation decisions, for example, intrinsic and extrinsic motivation (Roberts et al., 2006), social relations with project participants (Hahn et al., 2008), firm sponsoring (Spaeth et al., 2015), license choices (Stewart et al., 2006), or code architecture (Baldwin & Clark, 2006). However, these studies predominantly focus on individual projects and their communities, neglecting the inter-project relationships of modern OSS: OSS communities no are no longer limited to a single project, but extend across interdependent projects through interdependencies emerging through the reuse of software components (Bogart et al., 2021).

These networks of interdependent OSS projects create so-called *software ecosystems*, referring to communities that form on a common technology platform (e.g., programming language) and allow developers to create, share, and reuse software components that build upon each other's functionality (Bogart et al., 2021). Software ecosystems have emerged as an important way of organizing software development (Bogart et al., 2021), which has helped considerably to ease the reuse of software code and compo-

nents (Cox, 2019). However, the resulting dependencies make individual projects dependent on the maintenance and survival of other projects in the software ecosystem, which can lead to potentially disastrous problems if a project is neglected or no longer sustained. For example, the "leftpad incident", where a popular project was removed from the software ecosystem, resulted in hundreds of JavaScripts projects failing due to broken dependencies (Gallagher, 2016). Another more recent example is the vulnerability discovered in the log4j framework, the de-facto standard for logging in Java-based applications, which has affected nearly every major software company and has been described as the "most serious security breach ever" (Hunter & De Vynck, 2021). These incidents triggered by software dependencies illustrate that the sustained participation of developers is not only crucial for individual OSS projects, but also critical for other dependent OSS projects in the software ecosystem.

In this paper, we use the concept of *technological niches* to theorize about the relationship between developers and project interdependencies within a software ecosystem. Technological niches are communities defined by their technological relationships in a common technology space (Podolny et al., 1996; Podolny & Stuart, 1995). We draw on organizational ecology theories and previous studies on online communities (e.g., Wang et al., 2013; Zhu, Kraut, et al., 2014; Zhu, Chen, et al., 2014) and argue that developer overlap, that is, the degree of shared developers with other projects (Wang et al., 2013), considerably impacts the growth and survival of OSS projects. Therefore, we ask the following research question:

> *What is the effect of developer overlap in technological niches on the sustainability of open source projects in software ecosystems?*

To answer our research question, we propose that developer overlap positively affects developer retention due to mutualistic effects (Barnett & Carroll, 1987). We conducted an empirical investigation of the JavaScript ecosystem, which is one of the largest software ecosystems today and extensively incentivizes code sharing (Cox, 2019). We collected projects and their dependencies as well as historical development activities

covering two years from January 2017 until December 2018. We analyzed this data with regard to the effect of developer overlap in technological niches inside the software ecosystem on the projects' ability to retain developers. We found that developer overlap has a positive effect on developer retention, indicating that the relationship between OSS projects in technological niches is mutualistic. Furthermore, we found no significant interaction between the effect of developer overlap density and project size, but the effect of developer overlap density increases with increasing project age.

By taking the environmental interdependencies of OSS projects in software ecosystems into account, our study thereby makes three key contributions. First, we add to the literature on OSS sustainability by highlighting the role of a project's technical environment, in the form of its technological niche, on its ability to keep developers invested. Second, we add to the growing OSS literature adopting a network perspective by combining it with organizational ecology theory. Third, we introduce a novel lens to IS research on OSS communities by studying communities using the concept of technology niches.

The remainder of this paper is structured as follows. In the next section, we focus on the interdependencies between OSS projects in software ecosystems and describe how we use an ecological perspective to conceptualize OSS projects environments through technological niches. Followingly, we review the literature on OSS sustainability and dynamics in organizational environments, which we apply to develop our hypotheses. Next, we describe our model specification and report on our results. In the final section, we discuss our study's theoretical and practical implications, its limitations, and future research directions. We close this paper with conclusions.

6.2 Related Work and Theoretical Background

6.2.1 Sustained Participation in Open Source Software Projects

The community-based model of OSS development is only possible through the "open, voluntary, and collaborative efforts" by its developers (Shah, 2006). These decentralized communities of developers collaborate via online development platforms such as GitHub and voluntarily contribute to the projects (Fang & Neufeld, 2009; Roberts et al., 2006), making OSS projects dependent on the continuous participation of their community (Roberts et al., 2006; Shah, 2006). Hence, failing to sustain enough developer participation and creating a community that is willing to engage with the project over the long run is one of the major reasons for project failure (Bateman et al., 2011; Crowston et al., 2003; Markus et al., 2000; Ren et al., 2012).

Therefore, the issue of developer participation and its relation to the sustainability of OSS projects has emerged as a major topic in OSS studies (Curto-Millet & Corsín Jiménez, 2022). Previous research has thereby investigated several individual- and project-related factors and characteristics that influence developers' participation behavior. Individual factors comprise, for example, the enjoyment of participation (Shah, 2006), the willingness to learn and develop skills (von Krogh et al., 2003), career advancements (Lerner & Tirole, 2002), or emotional attachment with the project (Maruping et al., 2019). Project-related factors influencing developer participation include, for example, the project's license restrictions (Stewart et al., 2006), organizational sponsorship (Shah, 2006; Stewart et al., 2006), or the modularity of its codebase (Baldwin & Clark, 2006).

More recently, social relations between developers and projects have become the focus of OSS research, with several studies using a network perspective to investigate the influence of ties between developers themselves and project affiliations. For example, developers are more likely to join a project if they have previous collaborations with the project initiator (Hahn et al., 2008), and their decision to remain involved in the

project is influenced by other developers (Oh & Jeon, 2007).

6.2.2 An Ecological Perspective on Software Ecosystems

In this study, we introduce an *ecological perspective* on OSS communities in software ecosystems to examine the competition for developers between OSS projects. Modern OSS communities emerge not only around a single project but around a collection of interdependent software components that result in the formation of software ecosystems, that is, communities that emerge on a shared technology platform (e.g., programming language) allowing developers to share and build upon packaged software components. In modern programming languages, the practice of reusing software components has become even more practical, and therefore more extensive, through the integration of dependency management tools such as component managers (e.g., npm) and online databases of available software components (e.g., the npm registry)[16], which ease the process of downloading and installing software components (Cox, 2019). Adding an external software component to the codebase creates a dependency relationship, making the integrating project dependent on the providing project to properly function (Bogart et al., 2016).

This has proven to be a viable means of organizing OSS development work (Bogart et al., 2021). In software ecosystems, interdependencies between projects emerge and evolve through the recombination and reuse of the existing software components (Bogart et al., 2016; Howison & Crowston, 2014; Singh et al., 2011), which is common practice in OSS development (Haefliger et al., 2008; Sojer & Henkel, 2010). A software ecosystem therefore can be seen as a self-organizing system that emerges from the combination of available technology (Arthur, 2009). This is enabled through incorporating the principles of modular system design (Baldwin & Clark, 2000) and transparency of the development process and activities (Cataldo & Herbsleb, 2010).

Even though projects benefit from reusing software components through saving their

[16] https://www.npmjs.com/

developers' time and implementing already proven solutions (von Hippel & von Krogh, 2003), it can lead to various technical as well as organizational problems such as failure of the system and increasing coordination efforts (Cataldo et al., 2009). Thus, despite reducing complexity on a project level by creating a modular architecture that is more attractive for developers (Baldwin & Clark, 2006), using existing components creates a complex structure of interdependencies on the ecosystem level (Kapoor & Agarwal, 2017). These interdependencies can become harmful for the quality of the entire ecosystem if they are not maintained and properly managed (Cox, 2019; Decan et al., 2019). Consequently, the development of software in ecosystems involves a large and heterogeneous group of projects and related developers that need to effectively collaborate to produce a functioning system (Cataldo & Herbsleb, 2010).

A prominent theoretical lens to study interdependencies in an ecosystem of interdependent entities is the theory of organizational ecology (Hannan & Freeman, 1989), putting emphasis on the population and community levels of organizational analysis (Baum, 1996). Organizational ecology accounts for the effects of environmental changes and shared resource requirements, which emerge in *populations* of interconnected organizations, on their performance (Hannan & Freeman, 1989). A population is usually defined as a set of organizations having "unit character" (Hannan & Freeman, 1977). Research on online groups defined, for example, the shared technological platform as the unit character to identify populations (Wang et al., 2013). We follow a similar approach by defining the population of OSS projects based on their shared programming language.

More recently, researchers have recognized that organizations in a population do not have equal influence on each other and therefore have started to focus on internal structures within organizational populations (Freeman & Audia, 2006). The unit character of these so-called organizational *communities* stems from patterns of interdependencies among organizational actors (Freeman & Audia, 2006). These interdependent relations have been operationalized in different ways, for example, by focusing on the organizational members and their characteristics (Baum & Singh, 1994b; McPherson,

1983), or through a shared identity (Ruef, 2000).

6.2.3 Technological Niches and Developer Overlap

In this study, we adopt Podolny & Stuart (1995)'s concept of a *technological niche*, which is a community that is based on a common technology space defined by relationships between technological innovations. In their original work, Podolny & Stuart (1995) defined the technology space of an organization using patent citation data to identify the build upon innovations. They operationalize for each organization its egocentric niche that includes (1) the focal organization, (2) those organizations the focal organization builds upon by using their innovations, and (3) those organizations that build upon the focal organization's innovations.

In software ecosystems, when a project reuses another project's provided functionality, thus building upon its work, this is reflected in their dependency relationships between its software components. Larger projects often consist of multiple software components, each with its own potential dependencies on other components; either by belonging to the same project or from a different project. These dependencies on the component level can be projected to the project level. By adopting Podolny & Stuart (1995), we conceptualize a focal project's egocentric technological niche in software ecosystems consisting of (1) the focal project, (2) those projects that the focal project's components depend on, and (3) those projects that have components depending on components of the focal project.

Figure 6.1 illustrates our ecological view on software ecosystems represented by the dependency network and technological niches. The bottom layer represents the component level of the software ecosystem. The directed relations between two components reflects a dependency relationship. The top layer represents the project level. OSS projects are often divided into multiple software components. Thus, the related software components and their dependency relationships are aggregated on the project level. On the project level, the striped rectangles represent the respective technologi-

cal niches of three exemplary focal projects, which are highlighted in red. Notice that projects can belong to multiple technological niches, as indicated at the top where a project belongs to two technological niches shown by the overlap.

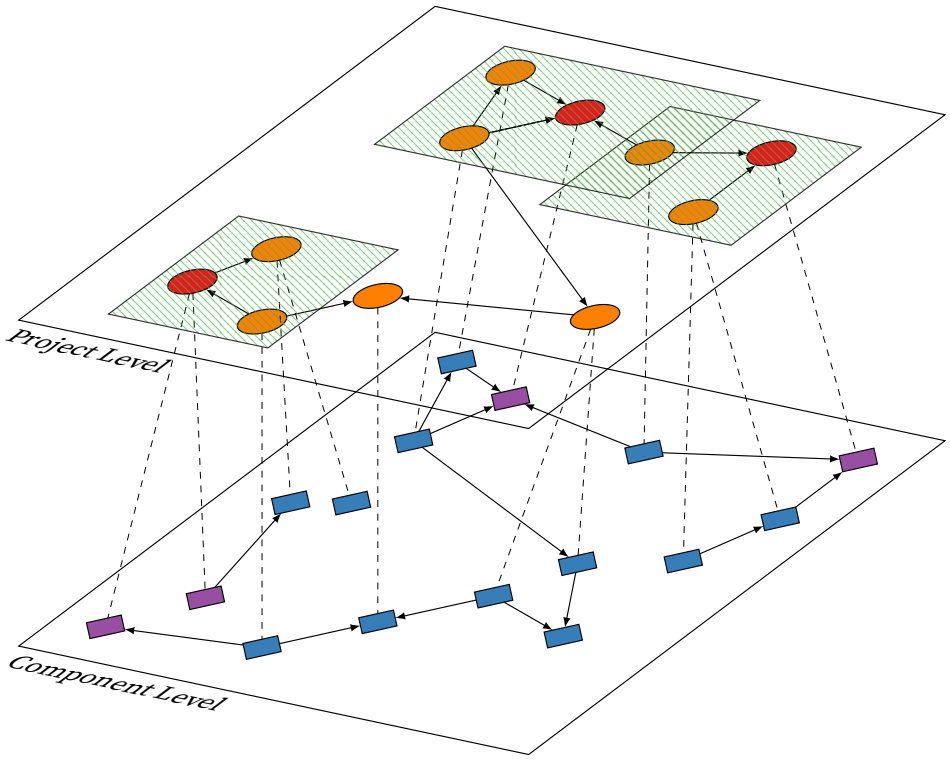


Figure 6.1. Software Dependency Network and Technological Niches

Consequently, not all projects in the software ecosystem are equally interdependent. Software ecosystems tend to form technological niches. Technological niches in OSS development are important because they provide a space for developers to specialize and thereby reduce the barriers of contribution (von Krogh et al., 2003). This allows them to accumulate more specific knowledge and strengthen their identification with the community, which increases their motivation to sustain their contribution to the community (Fang & Neufeld, 2009; von Krogh et al., 2012). In addition to the set of common technologies, these communities are characterized by shared goals, challenges, and problems that the involved projects aim to solve. For example, a technological niche could be formed around a specific type of application or tool (e.g., a content management system or a web-development framework), or a specific domain (e.g., scientific computing or machine learning).

Within a technological niche, OSS developers usually work on multiple projects simultaneously (Grewal et al., 2006; Singh et al., 2011) and hence must allocate their time and attention among different projects. From the perspective of a project, this results in *developer overlap*, that is, the degree of shared developers with other projects (Wang et al., 2013). Developer overlap reflects the degree of interdependencies between projects resulting from shared developer resources. Previous research on online groups and communities has recently drawn from organizational ecology theories to investigate the effect of these interdependencies that arise through overlapping members (e.g., Wang et al., 2013; Zhu, Kraut, et al., 2014; Zhu, Chen, et al., 2014). Along with studies on traditional organizations, the findings of these studies are ambiguous, with some showing positive and others negative effects of overlap on the growth and survival of online groups and organizations.

However, these studies so far have neglected the projects' environment and its effect on competition for developers' participation. As OSS projects exist in a larger software ecosystem of interdependent projects, they must acquire and compete for developer resources available in the pool of potential participants in the environment (Wang et al., 2013). Furthermore, to the best of our knowledge, the relations that emerge through the technical interdependencies in software ecosystems have not been considered. Thus, we focus on these relations and the resulting interdependencies between projects in a technological niche and their effect on sustained participation.

6.3 Theory and Hypotheses

6.3.1 Ecological Interdependencies in Technological Niches

OSS development is a collaborative process that involves a community of developers who work together to create software. In effect, they form an organization. In organizational ecology theory, organizations are considered interdependent when they influence each other's performance and success (Barnett & Carroll, 1987). Studies have

thereby focused on the influence on potential outcomes such as organizations' growth and decline (McPherson, 1983), their founding (Baum & Singh, 1994a), or mortality (Baum & Singh, 1994b). Interdependencies between organizations emerge on different levels, for instance, between individual organizations, populations, or communities of organizations (Barnett & Carroll, 1987).

In OSS communities, there are many interdependencies that exist between developers, users, and the project itself. Developers depend on each other to create and maintain software, while users depend on developers to provide software that meets their needs. Projects itself can also have interdependencies with other projects, as it may rely on other project's software components to function correctly. In this study, we focus on the interdependencies between communities of OSS projects, whose boundaries are defined by their technological niche. These interdependencies can lead to either competitive or mutualistic effects and therefore have negative or positive effects on the organizations in the environment (Barnett & Carroll, 1987).

In organizational ecology literature, the density dependence model has been used to capture the dynamics of competition and mutualism (Baum & Singh, 1994b). Early studies investigated competitive and mutualistic effects as a function of population density, that is, the number of organizations in a population (Carroll & Hannan, 1989). More recently, studies have used the resource overlap model to investigate competitive and mutualistic effects (Baum & Singh, 1994b, 1994a). Here, the intensity of interdependencies between organizations is driven by the degree of similarity in resource requirements (Hannan & Freeman, 1977, 1989).

6.3.2 Developer Overlap Density

In IS research, the density dependence model has been adopted to study the effect of member overlap on the growth of online communities (Wang et al., 2013). Thus, the resource of a community is defined through its shared members. As has been argued for voluntary organizations (McPherson, 1983) and online communities (Wang et al.,

2013), we consider developers in OSS projects as their most important resource. *Retention* of developers, that is, retaining newcomers and lessening turnover of longtimers, is thus an important objective of every OSS community.

We focus on the overlap between developers in the community to investigate the ecological interdependencies and their effect on retention. We argue for a mutualistic effect of developer overlap on their sustainability in OSS communities defined by their technological niche.

First, for OSS developers, it is common to work on multiple projects at the same time (Grewal et al., 2006; Singh et al., 2011), whereas employees in traditional organizations are usually contractually tied to a single organization. Therefore, the employee becomes unavailable for other organizations, with a decreasing focus of developers who participate in multiple projects (Kim et al., 2018; Wang et al., 2013) or a decrease in community engagement through a lack of diversity in the community (Daniel et al., 2013). This results in characteristics of a rival resource (Benkler, 2006). However, due to the observed nature of OSS developers to be affiliated with multiple projects (Grewal et al., 2006; Singh et al., 2011), we argue that they rather possess characteristics of a non-rival resource that is not fully exhausted once associated with a specific project.

Second, developer overlap indicates collaboration of developers in other projects. Previous research has shown that increased collaboration between developers not only increases knowledge sharing (Peng et al., 2013) and leads to positive spillover-effects that lead to better quality and greater technical expertise in technological niches (Grewal et al., 2006) as well as growth (Zhu, Kraut, et al., 2014), but also that collaborative ties increase the probability that developers stay involved in a project (Uzzi & Spiro, 2005), or even make new developers join the project (Hahn et al., 2008). Hence, we propose:

> *H1: Developer overlap density within a project's technological niche has a positive effect on developer retention.*

6.3.3 Developer Overlap Density and Project Size

The legitimacy of organizations plays an important role in demographic and ecological processes (Hannan & Freeman, 1988). Legitimacy is thereby influenced by two key demographic factors: size and age. In terms of size, larger organizations are usually perceived as being more successful and dependable in the future, which leads to disadvantages for smaller organizations (Baum, 1996). In ecological studies, this phenomenon is often referred to as the "liability of smallness" (Aldrich & Auster, 1986). Previous research on voluntary organizations has shown that large organizations, where members are more loosely connected, are more likely to experience membership turnover (McPherson et al., 1992). Thus, we argue that larger projects are more likely to lose developers when these developers are shared with other projects. Hence, we propose:

> *H2: Developer overlap density within a project's technological niche has smaller positive effects on developer retention in larger projects than in smaller projects.*

6.3.4 Developer Overlap Density and Project Age

In terms of age, younger organizations are usually seen as having a higher chance of failure and lower legitimacy (Baum, 1996). This phenomenon is referred to as the "liability of newness" (Stinchcombe, 1965). As has been shown in research on online groups, we similarly argue that older projects have more committed members that also have a more established network among other developers, which positively influences their intention to stay involved in the project (Wang et al., 2013). Hence, we propose:

> *H3: Developer overlap density within a project's technological niche has greater positive effects on developer retention in older projects than in younger projects.*

In summary, we propose that developer overlap with other projects in the technological niche has a positive effect on developer retention. Further, we assume that the project's size will negatively influence this effect whereas the project's age positively influences this effect. Figure 6.2 illustrates our proposed research model.

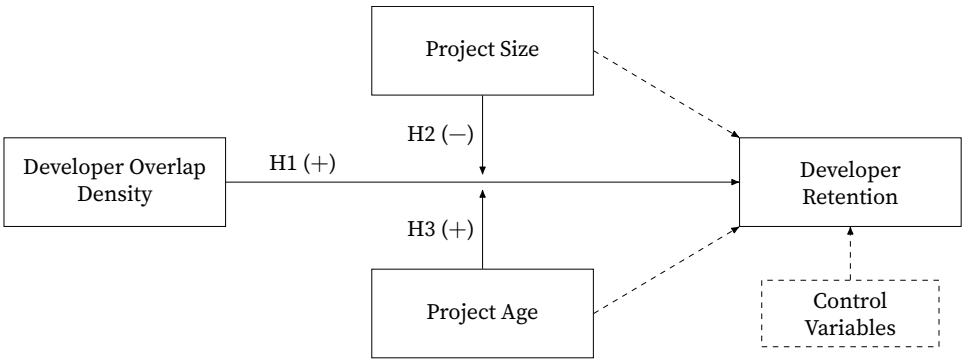


Figure 6.2. Research Model

6.4 Methods

6.4.1 Data

We tested our hypotheses in the JavaScript ecosystem, which is one of the largest OSS ecosystems with extensive technological interdependencies between projects (Decan et al., 2019). We used secondary data from the Libraries.io (Katz, 2020) and GHTorrent datasets (Gousios, 2013). First, Libraries.io[17] gathers data from different dependency managers and tracks open source projects, their published components, and the interdependencies between them. Since we focused on the JavaScript ecosystem, we first gathered all projects related to npm, which is the dependency manager of the JavaScript programming language maintained by npm, Inc.[18] Second, we added the related development activities from GHTorrent[19], which archives repository activity on GitHub by monitoring its public event timeline, by matching both datasets based on the repository information. We collected all relevant data from January 2017 until January 2019 on a monthly basis. We chose this time span to have sufficient observations and because of the restricted availability of historical event data[20].

Several criteria were applied to guarantee that only active and legit projects were included in our analysis. First, we included projects with releases and code contribu-

[17]https://www.libraries.io/
[18]https://www.npmjs.com/
[19]https://www.ghtorrent.org
[20]All data, code, and results are provided in an OSF repository here: https://tinyurl.com/4ckx8drk

tions after the beginning of our observation in the beginning of 2017 and with at least one participant in each period. Second, we removed all forks from the dataset, because in a development model driven by pull requests, that is the process of cloning a project, making changes, and requesting integration of the changes into the codebase, forks are automatically created once a pull request is issued. Third, we removed all activities related to bots from our data. These bots are used to coordinate work and automate tasks (e.g., 'dependabot'[21] automatically updates technical dependencies) and thereby they perform various activities in the repository (Hukal et al., 2019). Bots were in parts automatically identified through a high number of affiliated projects during a period, and in parts manually by scanning the related account login names. Fourth, we observed only projects that had at least one interdependency with another project during our observation. Therefore, projects that were not engaged in a dependency relationship with another project at any time during our observation were excluded. Lastly, we removed projects without information on their license. This resulted in a final data set of 2,403 projects. Some projects were created after January 2017 and have therefore fewer than 24 observations in the dataset. The final data set includes 41,909 project-month observations.

6.4.2 Network Construction

The JavaScript ecosystem functioned as the boundary condition for our network. We followed a whole network approach (Marsden, 2005) for our network construction. By focusing on the JavaScript ecosystem, we concentrate on a specific programming language as the underlying foundational technology, which is a common strategy when defining network boundaries in OSS studies (e.g., Grewal et al., 2006; Singh et al., 2011). Therefore, developers in that network do not differ in their capability to develop in a specific language. Also, because our main interest is on the effect of technology niches, these interdependencies only exist in the context of a specific language.

[21] https://github.com/dependabot

To construct the network, we first identified all projects related to the npm registry available in the Libraries.io dataset. For each project, we identified all associated components, their versions, and related dependencies. We created snapshots for the state of the network for each observation. We only included projects and their respective components in the network that had a release one year prior to the observation date because a large proportion of the projects in the data set showed no release activity due to potential feature completion or termination. Dependency relations of a component were gathered for all recent versions six months prior to the observation date because including only dependencies of the recent version would have excluded version branches that are still used and in development. Furthermore, in the JavaScript ecosystem, dependencies are distinguished between runtime dependencies and development dependencies. Runtime dependencies are required for the software to function, whereas development dependencies are only used during the development process. Thus, we focused only on runtime dependencies. Based on the dependency network, we created the project network projection. Based on the project network, we conceptualize a project's technological niche by following Podolny & Stuart (1995). Each project therefore occupies an egocentric niche that includes the (1) focal project, (2) the projects that the focal project uses as a dependency (upstream dependencies), (3) the projects that use the focal project as a dependency (downstream dependencies), and (4) the dependencies between these projects.

6.4.3 Measures

Dependent Variable

Our study focuses on the sustained participation of developers in OSS projects. Therefore, we capture the continuous participation by measuring *developer retention* as the number of developers that continued their participation in the project in the following month. This is based on existing measures to compare participants from one period to the next (e.g., Butler, 2001; Chengalur-Smith et al., 2010; Wang et al., 2013). Partic-

ipation in OSS projects is more than just committing code. Developers report issues, provide feedback and help, and manage tasks (Setia et al., 2012). Therefore, we follow Hartwick & Barki (1994) and operationalize participation in a project as the "behaviors, assignments, and activities" during the development process. This includes writing comments in issues or pull requests, opening or closing issues or pull requests, or pushing commits to the repository. To measure developer retention of a project, we first collected all developers that participated in each period. Afterwards, we compared the list of developers participating in subsequent periods and counted developers that participated in both periods.

Independent Variables

We follow Wang et al. (2013) by using a weighted measure of *developer overlap density* that accounts for both the number of projects in the technological niche that have shared developers with the focal project and the degree of overlap between the focal project and other projects in the niche. Two projects were considered sharing a developer if the developer participated in both projects in the same month. To arrive at our measure of developer overlap density, we calculated the degree of overlap between the focal project i with another project j in the focal project's technological niche as the number of shared developers divided by the total number of participating developers in focal project i in month t. The developer overlap for a focal project i in month t was then calculated as the sum of the degree of overlap between the focal project and each project in its technological niche:

$$\text{Developer Overlap Density}_{it} = \sum_{j=1}^{J} \frac{\text{Shared Developers}_{ijt}}{\text{Developers}_{it}} \tag{6.1}$$

For our interaction effects, we measured *project size* by counting all participating developers in a month. Here, the definition of participation used for calculating developer retention applies as well. *Project age* was measured as the difference in months between the creation data of the project's repository and the end date of the respective

month.

Control Variables

We controlled for the *population size* measured as the total number of projects in the software ecosystems in each month. *Technological niche size* was calculated as the number of projects in the focal project's technological niche in each month. *Average activity* in a project was calculated as the total number of participation activities divided by the total number of participating developers in a month. Furthermore, we created dummy variables to account for the restrictiveness of the project's license. In doing so, we followed Singh et al. (2011) and coded a project's *license restrictiveness* as either 'permissive', 'copyleft', if the license included the respective clause, or 'others'. In case license information was not provided on the project-level, we collected the licenses of the individual components and assigned the most restrictive license. A second dummy variable reflected the organizational ownership of the project based on the account type of the repository's owner ('USR' or 'ORG').

6.5 Model Specification

We specified a Bayesian multilevel regression model to estimate the effect of developer overlap density on developer retention. A multilevel model accounts for our clustered data structure due to the repeated measures of each project. Therefore, we included a random effect that allowed the intercept to vary across projects. An initial Hausman test suggested that fixed effects are sufficient. Thus, we modeled the effects of our independent and control variables as fixed effects.

Our dependent variable, *developer retention*, is a count and initial analysis showed overdispersion. Therefore, we choose a negative binomial posterior distribution, which takes additional variation due to unobserved influences into account (McElreath, 2020). We used a log-link function, which is the typical link function used for

count data (Gelman et al., 2014), and used weakly informative priors as suggested by McElreath (2020). We fitted the model using *R* and the *brms* package(Bürkner, 2017). Markov chain Monte Carlo (MCMC) simulations were used to draw samples from the posterior distribution. We used four chains and ran them for 4,000 iterations (2,000 warmup; 2,000 sampling) each. All chains converged, were well-mixed ($\hat{R}$< 1.01) and reached sufficient effective sample size for our variables of interest (ESS > 1,000). 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) with Pareto-smoothed importance sampling (PSIS), which is recommended in case of weak priors and influential observations (Vehtari et al., 2017).

Furthermore, initial analysis of our variables indicated that they were not normally distributed, which can lead to biased parameter estimates (Gelman & Hill, 2007). Thus, we applied a logarithmic transformation to all independent variables. In addition, to reduce biases due to multicollinearity, we mean-centered variables before creating the interaction terms to reduce correlations and keep variance inflation at an acceptable level. All variables except project age and developer retention were lagged by 1 month to account for reverse causality. Initial analysis also revealed outliers regarding our variable for developer overlap density in our data that might bias the regression results. Therefore, we excluded these observations from the dataset.

6.6 Results

Figure 6.3 exemplarily shows the evolution of the technological niche of the project *vue*[22] from the first snapshot in January 2017 until the last snapshot in December 2018. We refrain from showing the entire software ecosystem due to its large size. Both networks show the projects at the bottom layer with edges representing interdependen-

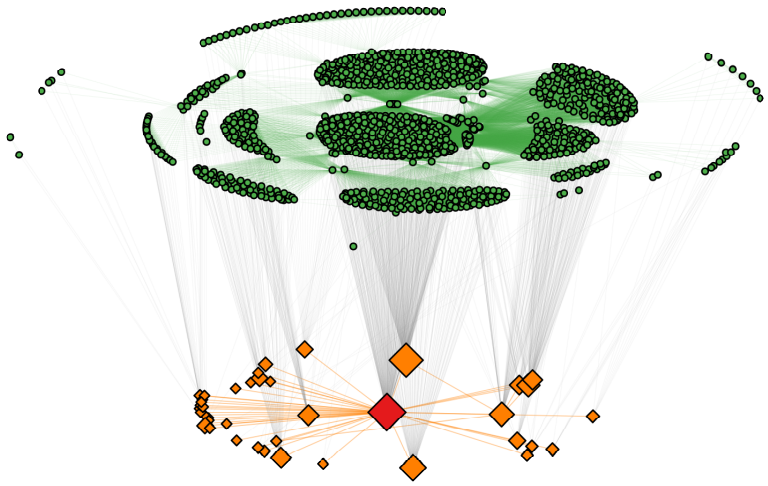
[22]https://github.com/vuejs

cies between them with the focal project highlighted in the center. The size of the nodes is relative to the size of the projects based on the number of participating developers in the period. The top layer represents the participating developers with edges representing participation in the same project. Both layers are connected through edges representing the affiliation of a developer to a project. The technological niche consists of 41 projects and 1096 developers in January 2017 and grows to 223 projects and 2870 projects in December 2018.

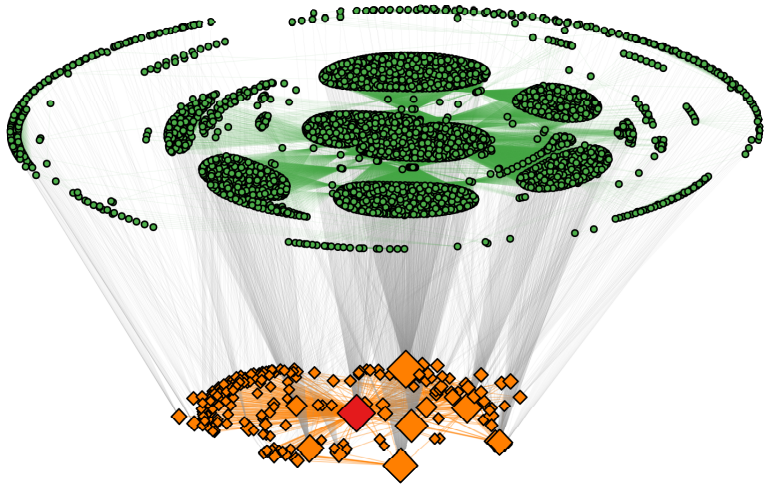
In January 2017, 49.2% of the projects had only one participating developer. 15.8% of the projects had two and 35% had three or more participating developers. In December 2018, 53.7% of the projects had only one participating developer. 17.1% of the projects had two and 29.2% had three or more participating developers. In January 2017, 29.5% of all projects in the ecosystem had no technical interdependency with another project in the ecosystem. 21,7% of the projects had only one interdependent project, 14,6% had two, and 34,2% had three or more. In December 2018, 28.3% of all projects in the ecosystem had no technical interdependency with another project in the ecosystem. 21,6% of the projects had only one interdependent project, 14,3% had two, and 35,8% had three or more.

Table 6.1 displays the descriptive statistics for our untransformed variables. Most of the variables are highly skewed. Thus, we also report the median values for each variable. In addition, our dummy variable for license restrictiveness shows that 80.3% of the projects used a permissive license, with the MIT license being the dominant choice with 60.8%. 2.57% of the projects chose a license including the copyleft clause, and 17.2% licenses were labeled as other. Regarding the project's ownership, 65.2% were owned by an organization and 34.8% by a user account.

Table 6.2 reports the correlations between our transformed variables and their variance-inflation factors (VIF) to test for multicollinearity. All VIF scores are < 3, indicating that multicollinearity is not an issue (James et al., 2021). We observed a high correlation between developer retention and project size, which is due to



(a) January 2017



(b) December 2018

Figure 6.3. Exemplary Technological Niche and Affiliated Developers of the "vue" Project

the operationalization of both variables. Furthermore, high correlations between developer overlap density and its interaction terms with project size and project age are rather structurally in nature and hence are no reason for concern.

In Table 6.3, we report the results of our regression analysis. The dependent variable

Table 6.1. Descriptive Statistics (n = 2,283; N = 37,520)

	Variable	Mean	St. Dev.	Min	Median	Max
1	Developer Retention	8.62	21.81	0	4	614
2	Population Size	47,658.92	13,765.46	25,854	47,494	69,431
3	Technological Niche Size	28.91	115.01	1	7	3087
4	Avg. Activity	9.35	24.41	0.11	3.79	1653.00
5	Project Size	34.20	94.94	2	14	3157
6	Project Age	34.14	23.15	0.93	29.43	128.69
7	Dev. Overlap Density	0.06	0.11	0.00	0.00	0.47
8	Dev. Overlap Density^2	0.02	0.04	0.00	0.00	0.22
9	Dev. Overlap Dens. × Project Size	2.76	12.48	0	0	403
10	Dev. Overlap Dens. × Project Age	2.38	4.80	0.00	0.00	50.11

Table 6.2. Correlation Matrix

	Variable	VIF	1	2	3	4	5	6	7	8	9	10
1	Developer Retention	–	1									
2	Population Size	2.541	−.028	1								
3	Technological Niche Size	1.567	.150	.129	1							
4	Avg. Activity	1.072	.079	.001	.076	1						
5	Project Size	1.520	.595	−.049	.309	−.081	1					
6	Project Age	2.863	.053	.085	.343	−.292	.253	1				
7	Dev. Overlap Density	1.155	.059	.015	.406	.067	.114	.077	1			
8	Dev. Overlap Density^2	1.041	−.077	−.014	−.192	.046	−.200	−.095	.040	1		
9	Dev. Overlap Dens. × Project Size	1.541	−.296	.036	−.023	.073	−.602	−.185	−.037	.130	1	
10	Dev. Overlap Dens. × Project Age	1.343	−.032	−.047	−.138	.205	−.176	−.658	−.007	.058	.257	1

developer retention leads the independent and control variables, except project age, by one month. Model 1 represents the baseline model with only the control variables. Notice that here we also control for the direct effects of project size and project age on developer retention. Model 2 adds the effect of developer overlap density to Model 1, and Model 3 adds the squared and interaction terms to Model 2. We report parameter estimates and the confidence intervals (95%) for both fixed and random effects. LOOIC scores indicate an improvement in the predictive accuracy from Model 1 to Model 3. Also, we calculated the intraclass correlation for each model which expresses the information provided by the grouping in multilevel models with values ranging from 0 if no information is provided to 1 if the members of a group are identical (Gelman & Hill, 2007). For all three models, the ICC score is around 0.47. Following, we refer to the full specification (Model 3) to discuss the results.

Table 6.3. Results of Bayesian Multilevel Models

	DV: Developer Retention								
	Model 1: Control			Model 2: Main Effect			Model 3: Interactions		
		95%-CI			95%-CI			95%-CI	
	Est.	Low	High	Est.	Low	High	Est.	Low	High
Fixed Effects									
Intercept	1.098^{***}	0.854	1.354	1.064^{***}	0.817	1.308	1.042^{***}	0.789	1.289
Population Size	-0.194^{***}	−0.218	−0.169	-0.191^{***}	−0.214	−0.167	-0.193^{***}	−0.217	−0.168
Technological Niche Size	-0.023^{***}	−0.030	−0.015	-0.030^{***}	−0.038	−0.022	-0.029^{***}	−0.037	−0.021
Avg. Activity	0.135^{***}	0.127	0.143	0.133^{***}	0.125	0.141	0.133^{***}	0.125	0.141
Project Size	0.874^{***}	0.866	0.883	0.875^{***}	0.868	0.883	0.878^{***}	0.869	0.887
Project Age	-0.123^{***}	−0.134	−0.111	-0.123^{***}	−0.134	−0.111	-0.114^{***}	−0.127	−0.101
Dev. Overlap Density				0.220^{***}	0.158	0.282	0.198^{***}	0.130	0.267
Dev. Overlap Density^2							0.474^{*}	0.015	0.933
Dev. Overlap Dens. × Project Size							0.028	−0.019	0.079
Dev. Overlap Dens. × Project Age							0.096^{**}	0.028	0.164
Random Effects									
σ_{Project}	0.306^{***}	0.294	0.317	0.306^{***}	0.294	0.318	0.306^{***}	0.294	0.317
Dummy Variables									
License Restrictiveness		Yes			Yes			Yes	
Year		Yes			Yes			Yes	
Ownership		Yes			Yes			Yes	
N_{Projects}		2,283			2,283			2,283	
$N_{\text{Observations}}$		37,520			37,520			37,520	
ICC		0.473			0.475			0.471	
LOOIC		154,993.906			154,958.271			154,952.583	

Note: *** indicates 99% confidence interval not including zero. ** indicates 95% confidence interval not including zero.
* indicates 90% confidence interval not including zero. LOOIC was computed following Vehtari et al. (2017).

6.6.1 Effect of Developer Overlap Density

We proposed that developer overlap in technological niches has a positive effect on developer retention (H1). We find support for this hypothesis with a positive coefficient for developer overlap density ($\beta = 0.198^{***}$; 95%-CI: [0.130, 0.267]). We also added a square term to test for a potential curvilinear relationship of developer overlap density. The square term was also positive ($\beta = 0.474^{**}$; 95%-CI: [0.015, 0.933]), indicating that developer overlap density improves developer retention at an increasing rate. When interpreting the coefficients and to understand their effects, it is important to consider that the variable has been log-transformed and we used a log-link function in our model, which means that the dependent variable is also log-transformed. Thus, the coefficient reflects the percent increase in the dependent variable for every 1% increase in the independent variable. Hence, for every 1% increase in developer overlap density, developer retention increases by about 0.2%. Our second hypothesis (H2) proposed a negative effect of project size on the effect of developer overlap density. The coefficient for the interaction term is positive but not significant ($\beta = 0.028$; 95%-

CI: [-0.019, 0.079]), therefore hypothesis H2 is not supported. However, consistent with hypothesis H3, we found a positive and significant interaction between developer overlap density and project age (β = 0.096**; 95%-CI: [0.028, 0.164]).

Figure 6.4 plots the conditional effects of developer overlap density on developer retention on the left and visualizes the interaction effect for three levels of project age from low, medium, to high on the right.

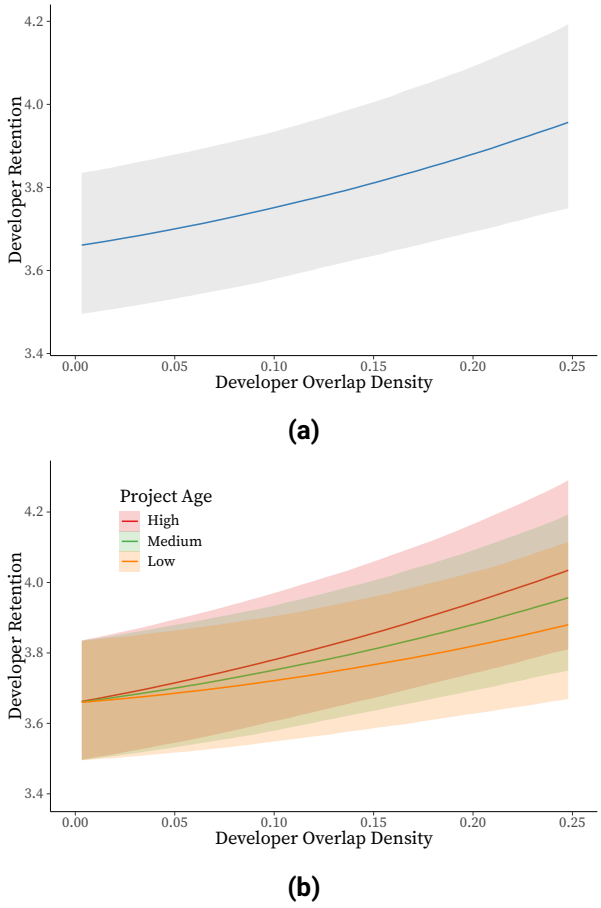


Figure 6.4. Effect of Developer Overlap Density on Developer Retention

6.6.2 Control Variables

For the effects of our control variables, we find that the coefficient for population size is negative and significant (β = -0.193***; 95%-CI: [-0.217, -0.168]). Moreover, the coefficient for the technological niche size is negative and significant (β = -0.029***; 95%-CI: [-0.037, -0.021]). The coefficient for the average amount of activity by a developer is

positive and significant ($\beta = 0.133^{***}$; 95%-CI: [0.125, 0.141]). The largest effect on developer retention has the project's size with a positive and significant coefficient ($\beta = 0.878^{***}$; 95%-CI: [0.869, 0.887]). This is not surprising because of the operationalization of both variables. The coefficient for project age is negative and significant ($\beta = -0.114^{***}$; 95%-CI: [-0.127, -0.101]).

6.6.3 Robustness Checks

We performed several robustness checks to ensure the robustness of our results (see Appendix A). During our data collection, we used various selection criteria and defined time spans that might influence the results of our analysis. Therefore, we reconstructed our entire sample and the associated affiliation, dependency, and project networks by changing each of these criteria and re-estimated the final model (Model 3).

First, we changed the observation frequency from monthly to quarterly. This time span is usually associated with the length of a typical release cycle in OSS projects (Hahn et al., 2008). In doing so, the period-dependent variables changed as well. For example, the measure for developer retention now compared developers participating in two subsequent quarters. The results are mostly consistent with Model 3, however, the interaction effect between developer overlap density and project size became significant. This might indicate that larger projects have a higher probability of retaining developers over a longer period compared to smaller projects.

Second, we changed the parameters for the dependency network construction by changing the time span for a project to be labeled as inactive from 1 year to 6 months as well as the release cycle lengths from 6 months to 3 months independently. Here, the squared term of developer overlap becomes insignificant when decreasing the time after which a project is marked as inactive to 6 months.

Third, we increased the depth of the technological niche by 1, meaning that we also included projects that were 2 hops away from the focal project in its niche. Here, the

results changed significantly with the squared term of developer overlap density and the interaction term with project age now being insignificant. Also, the direct effect of the technological niche size is now insignificant. This indicates that with an increasing depth of a project's technological niche, the community's cohesion diminishes and project's further away from the focal project might have a decreasing influence.

Fourth, we included only project size with at least 5 participating developers for all observations in our sample and re-estimated the model. Again, the results remained qualitatively the same.

Fifth, as our initial data set is unbalanced because projects are newly created during the observation period, we re-estimated the model using a balanced panel by only including projects that existed at the start of our observation in January 2017. The linear effect of developer overlap density remained qualitatively the same but the squared and interaction terms became insignificant. These changes might be explained by the fact that those projects that were already created at the beginning of our observation were already quite established and therefore the project's age has no influence on the effect of developer overlap density on developer retention.

Lastly, we re-estimated Model 3 by using different distributions for our dependent variable that are also suitable for count data. We tested poisson, zero-inflated poisson and zero-inflated negative binomial distributions. Only for the zero-inflated negative binomial distribution, the squared term of overlap density became insignificant. All other effects remained qualitatively the same.

6.7 Discussion

6.7.1 Theoretical Implications

In this study, we investigated the effect of developer overlap in a project's technological niche on developer retention. We developed and tested various hypotheses using

longitudinal data from 2,283 projects in the JavaScript software ecosystem. Our results indicate partial support for our hypotheses and are robust to numerous controls and model specifications.

Our study has several theoretical and practical implications. Theoretically, we argued that, because of mutualistic dynamics between OSS projects in technological niches, developer overlap would have a positive effect on developer retention. We found support for this hypothesis. The literature on online communities has so far produced mixed results on the effect of membership overlap. For instance, studies showed that membership overlap has negative effects on a community's growth (Wang et al., 2013), survival (Zhu, Kraut, et al., 2014), and attention and time spent by its members (Kim et al., 2018). However, sharing members has also been shown to have positive effects on knowledge collaboration (Zhu, Kraut, et al., 2014) and the access to external knowledge (Kim et al., 2018). Our study adds another positive effect of membership overlap to the discussion by focusing on developer retention as an important factor for an OSS project's sustainability. As we measure overlap within a project's technological niche, we posit that the resulting interdependencies lead to developers staying involved with the project due the resulting complementary relationships. Even though developers participate in other projects in a focal project's technological niche and therefore spend time externally, they collaborate in the larger community due to technological dependencies. This indicates that developers do not see projects in a technological niche as competing, individual projects but rather as a coherent stack of projects that function together. Therefore, participating in and sharing developers between technical interdependent projects leads to mutualistic rather than competitive effects.

Furthermore, our findings suggest that the effect of developer overlap rises with increasing age. This indicates that it is less difficult for more mature and established projects to sustain the participation of shared developers. As mature projects benefit from higher perceived legitimacy and decreased risk of abandonment the uncertainty for developers also decreases and therefore their intention to stay involved increases

(Setia et al., 2012). Moreover, mature projects provide better opportunities for developers to build networks and establish social ties that encourage long-term participation (Chengalur-Smith et al., 2010; Uzzi & Spiro, 2005). However, we also found a negative direct effect of a project's age on developer retention, which supports the argument that as newer projects provide more interesting development tasks compared to maintenance-heavy work in older projects, they are more attractive for developers and thus are also able to retain their participation over the time (Chengalur-Smith et al., 2010). Taking the direct and indirect effects of a project's age together, our results indicate that older projects might be able to counter the negative direct effect through the amount of shared developers in their technological niche.

In addition, while previous studies showed that the negative effect of member overlap on growth decreases with increasing community size (Wang et al., 2013), we did not find a significant effect of project size on the influence of developer overlap on retention. This indicates that a shared developer's motivation to stay involved with a project is driven by the nature of the technological interdependence between the projects rather than their size (and therefore perceived legitimacy). Moreover, our results related to the positive effect of a project's size and developer retention differ from previous findings that showed that larger online communities are associated with larger member loss (Butler, 2001). In contrast, we suggest that increasing member size may counter the negative effects.

Our study highlights the importance of a project's environment (i.e., its software ecosystem) for its sustainability. Previous research on OSS projects' sustainability has largely focused on factors on the project-, individual-, and social-relational-level, thereby neglecting the technological interdependencies between projects in software ecosystems. As OSS projects and their communities do not exist in isolation but are part of a larger software ecosystem of technological interdependent software components and projects, our study demonstrates that these interdependencies need to be considered when investigating the evolution of a project and its community. By using technological niches to capture and analyze a project's environment in software

ecosystems, we introduce a useful concept from organizational ecology to define a project's environment.

6.7.2 Practical Implications

From a practical perspective, our research offers several implications for OSS project managers and community leaders. In showing the positive effect of developer overlap inside the technological niche, managers and leaders should try to integrate and collaborate actively with other projects in their technological niche. They may even encourage their members to participate in adjacent projects, instead of fearing to lose them to competitors. Therefore, they would not only profit from a higher probability of retaining their existing developers but could also benefit from knowledge sharing and increasing collaboration. As these projects are technically interdependent, a higher level of collaboration could thereby help in better understanding the needs of dependent projects and their developers, thus improving the quality and satisfaction with the project altogether.

6.7.3 Limitations and Future Research

This study has several limitations that provide directions for future research. First, we studied a single software ecosystem based on the shared JavaScript programming language. Each programming language and its related software ecosystem has its own characteristics. For instance, many projects in the JavaScript ecosystem are targeted at web developers. Furthermore, provided software components tend to be small with some projects only consisting of a few lines of code. Therefore, future studies should attempt to replicate our results in other software ecosystems (e.g., Python, Rust, or Go, to name just a few popular contemporary ones). Due to our transparent approach, our data collection, transformation, and extraction processes can be used for this purpose and even the used datasets provide the required data.

However, measure of developer retention does not account for the amount of activity

in the project, but rather if developers continue to participate altogether. Therefore, it might be the case that in case of increasing overlap density, the activity level of the individual developer or the project's community as a whole decreases.

Second, we did not account for the developers' role in the project or the quality of their contributions. Whereas some developers participate in more technical roles by revolving issues in the software or extending its functionality, others have more managerial responsibilities (Hann et al., 2013). We believe that for OSS sustainability, both roles are equally important, but we did not investigate which type of participant is more likely to retain in a project if there are overlapping affiliations in the technological niche. Furthermore, we did not account for other sociodemographic characteristics of the participating developers, such as level of experience, tenure, or age. Thus, in addition to the environmental effects in technological niches, future studies could include these developer characteristics into the analysis, specifically because ecological studies have also found that competitive effects between organizations increase if they target the same members based on their socio-demographics (Popielarz & McPherson, 1995).

Third, future research could further dive into the unique characteristics of technological niches. In our study, we only focused on the overlap between developers and included the niche's size as an additional control. However, previous research has proposed several other niche characteristics, such as status of the individual projects and indirect ties between them that are not related to the focal project (Podolny et al., 1996; Podolny & Stuart, 1995), competitive crowding through similarities in technological use (Podolny et al., 1996; Stuart & Podolny, 1996) that might influence dynamics in technological niches. Investigating these characteristics and their effect on OSS sustainability would contribute to our understanding of software ecosystems.

Fourth, using trace data comes with various promises but also perils (Grover et al., 2020; Kalliamvakou et al., 2014, 2016), especially when using it for network analysis (Howison et al., 2011). By applying various checks and filters during our data collec-

tion and adjusting the operationalization of our variables during initial analysis, we made sure to account for the most common perils such as low project activity, inactive projects, project-specific usage of GitHub features (Kalliamvakou et al., 2016). Thereby, we also ensured the validity of our constructs by account for potential temporal mismatch caused by our data aggregation (Howison et al., 2011), for instance, by varying the chosen observation periods and time intervals. Nevertheless, future studies should further validate our findings by either using different data sources, other observation periods, or different operationalization of our constructs.

6.8 Conclusion

The sustainability of OSS projects as the foundation of our digital applications and infrastructures becomes increasingly important. By highlighting the role of technical interdependencies in software ecosystems and the influence of environmental dynamics on developer retention, this study contributes to the growing literature on OSS sustainability and introduces an ecological perspective on software ecosystems and shows that project interdependencies in technological niches through shared developers have a mutualistic effect and lead to increasing developer retention in OSS projects. In general, we hope that this study also draws attention to the problems that stem from software component reuse in modern OSS development. Even though the failure of such dependencies has not been the focus of this study, by highlighting the level of interdependencies on a technical level, we hope to engage other IS researchers to further investigate this phenomenon.

CHAPTER 7

DISCUSSION

This dissertation presents three empirical studies on the influence of technical interdependencies in software ecosystems on the sustained participation of developers in OSS projects. By focusing on the technical interdependencies, this dissertation extends the body of knowledge by showing how they influence (1) developer attraction, (2) developers' participation decisions, and (3) developer retention. Thereby, these studies theoretically contribute to the literature on developer participation and OSS sustainability and hold several practical implications for OSS project managers and community leaders. Following, I summarize the contributions of each study, their practical implications, and limitations and discuss future research directions.

7.1 Summary of Contributions

7.1.1 Contributions of Study One

The first study contributes to the literature on OSS project characteristics that lead to attracting developers. Thereby, the study extends the body of knowledge by drawing attention to the technical interdependencies that emerge in software ecosystems. Furthermore, the study adds to OSS research that investigates the role of networks on sustained participation. Three main contributions can be derived from this study.

First, the study shows a positive effect of upstream dependencies on a project's ability to attract developers. As previous studies have found mixed effects of upstream dependencies on a project's survival (Valiev et al., 2018), this study provides evidence that the negative effects of more upstream dependencies, such as an increase in potential points of failure, might be offset by an increase in developer attraction.

Second, the study highlights the overarching problem in software ecosystems emerging from the fact that highly used projects that build the foundation of others, do not sufficiently profit from these dependencies. This study empirically found support for this phenomenon as the downstream dependencies of a project had no effect on its ability to attract developers. Thus, our findings support the assumption that developers reuse software components so that they do not have to maintain or build them themselves (Haefliger et al., 2008).

Third, the study contributes to network-based studies on OSS projects (e.g., Hahn et al., 2008; Maruping et al., 2019; Oh & Jeon, 2007; Peng, 2019) by introducing and focusing on the role of technical interdependencies in OSS ecosystems. Thus, this study highlights the importance of these technical networks and adopts a holistic socio-technical perspective on OSS projects and software ecosystems.

7.1.2 Contributions of Study Two

The second study contributes to the literature on developers' participation decisions by introducing technical interdependencies between OSS projects as important antecedents. The two main contributions of this study are as follows.

First, by leveraging the dependency relations between OSS projects, this study empirically shows that developers contribute to projects they use themselves, but also to other projects that build upon their projects. While previous research has already suggested that developers are often motivated by their own needs (Shah, 2006), this study empirically verifies this assumption.

Second, the study demonstrates the benefits and potential applications of advanced dynamic network models, such as SAOMs, in the context of OSS research. Even though SAOMs have been used in OSS research before (e.g., Conaldi et al., 2012), this study is the first to combine the affiliation and dependency networks between projects in software ecosystems.

7.1.3 Contributions of Study Three

The third and final study of this dissertation focuses on sustaining OSS projects through developer retention, that is, keeping existing developers engaged in the long term. The study draws from organizational ecology theory (Hannan & Freeman, 1989) and adopts the concept of a technological niche (Podolny et al., 1996; Podolny & Stuart, 1995) to investigate the effect of developer overlap on a project's ability to retain developers. Thereby, the study contributes to OSS research in two ways.

First, this study shows that the effect of developer overlap in technological niches is mutualistic instead of competitive, which leads to increased developer retention. Thus, it contributes to the literature on online communities that has produced mixed results. These studies have shown that membership overlap in communities has negative effects on their ability to grow (Wang et al., 2013) and their survival (Zhu, Kraut, et al., 2014) as members' attention and available time is shared between the overlapping communities (Kim et al., 2018). We show that due to technical interdependencies between the developer-sharing projects, these developers stay involved with the project over the long run and thereby increase its sustainability. Furthermore, this study shows that this effect even increases with increasing project age.

Second, this study highlights the importance of a project's environment in a software ecosystem for its sustainability. By introducing the concept of the technological niche, it demonstrates that OSS projects and their communities do not exist in isolation but are part of a larger community around technical interrelated projects.

7.2 Synopsis

Taken together, this dissertation contributes to the literature on the sustainability of OSS projects that focuses on the continuous participation of its developers (e.g., Chengalur-Smith et al., 2010; Fang & Neufeld, 2009; Zhang et al., 2013) and to the ecological perspective on online communities (e.g., Butler, 2001; Wang et al., 2013;

Zhu, Chen, et al., 2014) in two main ways.

First, by focusing on the technical interdependencies between OSS projects, this dissertation adds a missing piece for a holistic socio-technical view on OSS development. The investigation of their influence on developer participation revealed that technical interdependencies affect a project's ability to attract and retain developers and play a role in a developer's participation decision. Thus, this dissertation highlights the importance of the technical environment of OSS projects that emerge and evolve in software ecosystems.

Second, this dissertation introduces the concept of a technological niche to the ecological studies on online communities. This concept provides a way to define an OSS project's environment more precisely, it also makes for a suitable vehicle to investigate dynamics in specific parts of a software ecosystem. As I showed that the dynamics in technological niches in OSS projects are contrary to findings in online communities, other effects might be different in technological niches in the context of OSS projects as well.

In summary, this dissertation draws attention to the technical interdependencies and their importance for a holistic view of OSS projects and their sustainability. It shows that three major factors of sustainable OSS development, developer attraction, retention, and their participation choice, are all influenced by technical interdependencies. Therefore, this dissertation suggests that when researching the OSS phenomenon, these technical interdependencies and the resulting relations and dynamics should be included and encourages research to consider this technical aspect in future studies.

7.3 Practical Implications

This dissertation provides several insights and implications for OSS developers and managers. First, as the first study indicates that the reuse of software components leads to an increased ability to attract developers, project leads and managers should evaluate which parts of the required functionality could be provided by other projects, which could be implemented as an upstream dependency. This would allow developers to focus on the project's core functionality. However, potential upstream dependencies should be selected with care because their implementation leads to increasing costs for dependency management, the risk of breaking changes, and vulnerabilities.

Second, the first study also highlights the overall problem of software reuse in software ecosystems with highly important projects, indicated by their number of downstream dependencies, not benefiting from their importance by becoming more attractive for developers. Even though the second study indicates that developers tend to participate in the projects they use, this might not translate into growing a sustainable community.

Third, this dissertation indicates that community-building efforts should also be directed towards a project's interdependent projects. As the results of study 2 indicate that developers tend to participate in projects they use themselves as upstream dependencies and help other projects that make use of their projects resulting in downstream dependencies, OSS project leads and community managers should actively try to engage with those developers. Also, study 3 highlights that sharing developers who participate in other projects in the project's technological niche leads to an increase in retaining developers. Thus, to establish a sustainable community and thereby increase a project's chance of survival and success, projects need to create a greater community beyond the boundaries of their own scope and engage with their technological niche emerging from technical interdependencies.

7.4 Limitations and Future Research

Following, I will highlight the main limitations of this dissertation and derive potential avenues for future research.

7.4.1 Definition and Measurements of OSS Sustainability

The goal of this dissertation was to investigate the influence of technical interdependencies on the sustainability of OSS projects. In doing so, it used various measures that all reflect the *quantity of participation*. As shown by Curto-Millet & Corsín Jiménez (2022), the quantity of participation is only one aspect that emerges in research on sustainability. For instance, besides the quantity of participation, also the *type of participation* plays a key role in OSS sustainability. In this dissertation, participation was generally defined as any contribution in the form of code commits, issue or pull request creation, or engaging in discussions. Even though all these contributions matter and are important for a project to survive and evolve, their values and required efforts might differ and lead to different impacts on the overall sustainability of a project. Hence, future studies could further distinguish between these types of contributions and analyze if the effect of technical interdependencies differs depending on the type of contribution.

Other important factors for OSS sustainability are the characteristics and roles of the individual developers (Curto-Millet & Corsín Jiménez, 2022). In this dissertation, for example, I did not distinguish between core or peripheral developers (Setia et al., 2012), organizational employment, or other socio-demographic characteristics (i.e., nationality, age, gender). By taking these characteristics into account, future studies could try to investigate if the effects of technical interdependencies change based on the composition of a project's participants.

7.4.2 Beyond the JavaScript Ecosystem

Second, this dissertation focused on a single software ecosystem, the JavaScript ecosystem. However, there exist various other software ecosystems for different programming languages, such as Python, Java, and Rust. Each of these programming languages and their related ecosystems focus on different domains and applications and thus have different characteristics, features, communities, and audiences. For instance, the JavaScript ecosystem's main domain lies in web development and is characterized as an ecosystem with intensive reuse of software components (Cox, 2019). Due to this specific domain and resulting audience as well as the culture of heavily reusing, even trivial, software components, our results might differ in other ecosystems.

Hence, future research could replicate and extend the findings to different ecosystems. For example, the Java ecosystem is popular in the domain of enterprise software and is more mature compared to the JavaScript ecosystem. Therefore, the composition and characteristics of the community might be different, especially due to the potentially greater participation of organizations.

7.4.3 Inside Dependencies and Configurations

Third, this dissertation demonstrates the importance of technical interdependencies in the consideration of the OSS phenomenon. However, the observations and operationalization of technical interdependencies take place on a high level. This dissertation looked mainly at the presence and amount of these dependencies, and the resulting environment of an OSS project, thereby ignoring other properties and dynamics. For instance, the size of the dependency in terms of its functional scope or the number of participating developers might influence its effect on the focal project. Further research could consider the dependencies' properties when investigating their influence on developer participation.

Furthermore, the configuration of the used dependencies might result in different outcomes. For example, the usage of well-established and common projects as de-

pendencies might have a different impact on a project's attractiveness and ability to leverage additional developer resources as potentially more developers are familiar and knowledgeable with the implementation of these projects. In addition, the influence of a particular dependency might change when combined with another and there might exist specific configurations that are superior compared to others. Thus, future research should dive deeper into the specific dependency configurations of a project.

7.4.4 Structure of the Technological Niche

Fourth, I believe that the concept of the technological niche (Podolny et al., 1996; Podolny & Stuart, 1995), introduced in study 3, represents a promising analytical tool to look at the environment of an OSS project within a software ecosystem. In this dissertation, specifically in study 3, I used this concept to define the boundary of a project's environment. However, each technological niche has its distinctive structure that could be considered in future studies. For example, Podolny & Stuart (1995) investigate the structure of the technological niche by analyzing the effect of indirect ties between organizations in the niche that do not involve the focal organization on the intensity of competition. Also, in a following study, Podolny et al. (1996) looked at the technological overlap between organizations in the niche represented by their shared use of antecedent innovations, which they refer to as *crowding*.

Future research could apply these ideas to further study technological niches in OSS ecosystems. For instance, the mutualistic effect of shared developers might be stronger in niches where many projects share the same software components as their foundation. Also, this dissertation only looked at the impact of a project's technological niche on its sustainability, but influencing effects on other outcomes in OSS development might be possible. For example, the structure of the technological niche might influence the success of a project or the intensity of knowledge collaboration between niche occupants. Thus, I believe that by including an OSS project's environment defined by its technological niche will yield some further insights into

how OSS projects and their communities interact, and which dynamics emerge in these interdependent software ecosystems.

7.4.5 Evolution of the Software Ecosystem

Fifth, the studies of this dissertation mostly focused on the influence of a project's technical dependencies on sustained participation as the outcome. Accordingly, this dissertation explores the variation in sustainability as a function of the technical interdependencies of a project. When applying the topology proposed by Borgatti & Foster (2003), this dissertation thereby treats the outcome as a *consequence* of the network structure. Thus, the question of how the overall software ecosystem evolves and changes remains unanswered in this dissertation.

Therefore, future studies could adopt the other perspective by focusing on the *cause* of the network structure emerging from technical interdependencies. How do social and technical relations influence its evolution, growth, and shape? Do technical interdependencies of a project influence the social relations and collaborations between developers or is it the other way around? How does the introduction of a novel innovation through a project influence the evolution of the software ecosystem and what influences the adoption of projects? These questions might shed light on the dynamics and evolution of software ecosystems and thereby lead to a better understanding of the socio-technical structures in modern OSS development.

7.4.6 Digital Trace Data and Network Analysis

Lastly, this dissertation uses digital trace data, which implies several potential validity issues (Howison et al., 2011; Kalliamvakou et al., 2016). This data was generated from IT systems and was not designed and originally intended for research purposes. Furthermore, most of the data comes from secondary sources, GHArchive, GHTorrent, and Libraries.io. Therefore, even though I conducted various reliability checks and followed recommendations to ensure construct validity and avoid temporal mismatch

(Howison et al., 2011), I cannot guarantee that the data is completely accurate. Thus, future research should try to replicate our findings with data from different sources or periods.

CHAPTER 8

CONCLUSION

With the increasing dependence of organizations on OSS in the digital age, the question of how to sustain OSS projects becomes increasingly important. This dissertation investigated developer participation as one of the major factors influencing a project's sustainability. In doing so, it focused on the technical interdependencies between OSS projects that emerge in software ecosystems as a modern way of organizing work in OSS development. In three empirical studies situated in the JavaScript ecosystem, I analyzed how technical interdependencies influence (1) developer attraction, (2) developers' participation decisions, and (3) developer retention in technological niches.

These studies offer novel insights into OSS projects by highlighting the role of technical interdependencies in software ecosystems. The results from the empirical studies help to achieve a true socio-technical perspective on the OSS phenomenon. The study presented in Chapter 4 shows how OSS projects benefit from software reuse by becoming more attractive for developers. The following study presented in Chapter 5 provides evidence on how a developer's participation decision is influenced by technical interdependencies between projects. Finally, the last study presented in Chapter 6 adopts an ecological perspective and shows how projects in a technological niche benefit from sharing developers which leads to an increased ability to retain existing developers.

This dissertation contributes to the literature on OSS sustainability and developer participation by introducing technical interdependencies in software ecosystems as an important antecedent and influencing factor. As the importance of OSS for our digital world steadily increases and software ecosystems remain one of the most common ways of organizing work in OSS development, these insights are timely and relevant to sustain these projects in the future.

APPENDIX A

ROBUSTNESS CHECKS

Table A.1. Robustness Checks (Study 3)

	Quarterly	Inactivity	Release Cycle	Niche Depth	Project Size	Balanced Panel	Poisson	Zero-infl. Poisson	Zero-infl. Neg. Bin.
	Model 4	Model 5	Model 6	Model 7	Model 8	Model 9	Model 10	Model 11	Model 12
Fixed Effects									
Intercept	0.826^{***}	1.213^{***}	1.044^{***}	1.214^{***}	1.062^{***}	1.192^{***}	1.105^{***}	1.111^{***}	1.036^{***}
Population Size	-0.170^{***}	-0.212^{***}	-0.193^{***}	-0.204^{***}	-0.192^{***}	-0.212^{***}	-0.198^{***}	-0.199^{***}	-0.192^{***}
Technological Niche Size	-0.021^{***}	-0.023^{***}	-0.030^{***}	-0.001	-0.035^{***}	-0.031^{***}	-0.028^{***}	-0.028^{***}	-0.029^{***}
Avg. Activity	0.153^{***}	0.133^{***}	0.133^{***}	0.130^{***}	0.111^{***}	0.126^{***}	0.127^{***}	0.127^{***}	0.133^{***}
Project Size	0.849^{***}	0.878^{***}	0.878^{***}	0.868^{***}	0.898^{***}	0.904^{***}	0.882^{***}	0.882^{***}	0.878^{***}
Project Age	-0.135^{***}	-0.121^{***}	-0.114^{***}	-0.134^{***}	-0.126^{***}	-0.123^{***}	-0.116^{***}	-0.116^{***}	-0.115^{***}
Dev. Overlap Density	0.072^{**}	0.191^{***}	0.197^{***}	0.038^{**}	0.258^{***}	0.261^{***}	0.190^{***}	0.192^{***}	0.197^{***}
Dev. Overlap Density^2	0.818^{***}	0.469	0.748^{**}	0.034	0.332	0.619	0.576^{**}	0.566^{**}	0.473
Dev. Overlap Dens. $\times$ Project Size	0.051^{**}	0.021	0.026	-0.013	-0.004	0.012	0.039	0.037	0.028
Dev. Overlap Dens. $\times$ Project Age	0.060^{**}	0.092^{**}	0.098^{**}	-0.002	0.105^{**}	0.055	0.082^{**}	0.083^{**}	0.096^{**}
Random Effects									
σ_{Project}	0.280^{***}	0.306^{***}	0.306^{***}	0.311^{***}	0.332^{***}	0.330^{***}	0.309^{***}	0.309^{***}	0.306^{***}
N_{Projects}	6,695	2,283	2,281	2,342	955	1,116	2,283	2,283	2,283
$N_{\text{Observations}}$	34,969	37,374	37,400	39,763	19,207	25,668	37,520	37,520	37,520

Note: *** indicates 99% confidence interval not including zero. ** indicates 95% confidence interval not including zero.

* indicates 90% confidence interval not including zero. All models include dummy variables for license restrictiveness, year, and sponsorship.

REFERENCES

Abdalkareem, R., Oda, V., Mujahid, S., & Shihab, E. (2020). On the impact of using trivial packages: an empirical case study on npm and PyPI. *Empirical Software Engineering*, *25*(2), 1168–1204. https://doi.org/10.1007/s10664-019-09792-9

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

Aldrich, H., & Auster, E. R. (1986). Even dwarfs started small: Liabilities of age and size and their strategic implications. *Research in Organizational Behavior*, *8*, 165–198.

Arthur, W. B. (2009). *The Nature of Technology: What It Is and How It Evolves*. Simon and Schuster.

Baldwin, C. Y., & Clark, K. B. (2000). *Design Rules: The Power of Modularity*. MIT Press.

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

Barnett, W. P., & Carroll, G. R. (1987). Competition and Mutualism among Early Telephone Companies. *Administrative Science Quarterly*, *32*(3), 400–421. https://doi.org/10.2307/2392912

Bateman, P. J., Gray, P. H., & Butler, B. S. (2011). Research Note—The Impact of Community Commitment on Participation in Online Communities. *Information Systems Research*, *22*(4), 841–854. https://doi.org/10.1287/isre.1090.0265

Baum, J. A. C. (1996). Organizational Ecology. In S. R. Clegg, C. Hardy, & W. R. Nord (Eds.), *Handbook of Organization Studies* (pp. 77–114). SAGE Publications Ltd.

Baum, J. A. C., & Singh, J. V. (1994a). Organizational Niches and the Dynamics of Organizational Founding. *Organization Science*, *5*(4), 483–501. https://doi.org/10.1287/orsc.5.4.483

Baum, J. A. C., & Singh, J. V. (1994b). Organizational Niches and the Dynamics of Organizational Mortality. *American Journal of Sociology*, *100*(2), 346–380. https://doi.org/10.1086/230540

Bavota, G., Canfora, G., Di Penta, M., Oliveto, R., & Panichella, S. (2015). How the Apache community upgrades dependencies: an evolutionary study. *Empirical Software Engineering*, *20*(5), 1275–1317. https://doi.org/10.1007/s10664-014-9325-9

Benkler, Y. (2006). *The Wealth of Networks: How Social Production Transforms Markets and Freedom*. Yale University Press.

Berente, N., Seidel, S., & Safadi, H. (2019). Research Commentary—Data-Driven Computationally Intensive Theory Development. *Information Systems Research*, *30*(1), 50–64. https://doi.org/10.1287/isre.2018.0774

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

Bogart, C., Kästner, C., Herbsleb, J., & Thung, F. (2021). When and How to Make Breaking Changes: Policies and Practices in 18 Open Source Software Ecosystems. *ACM Transactions on Software Engineering and Methodology*, *30*(4), 1–56. https://doi.org/10.1145/3447245

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

Borgatti, S. P., & Halgin, D. S. (2011). On Network Theory. *Organization Science*, *22*(5), 1168–1181. https://doi.org/10.1287/orsc.1100.0641

Bosch, J. (2009). From Software Product Lines to Software Ecosystems. *Proceedings of the Thirteenth International Software Product Line Conference (SPLC)*, 1–10.

Bürkner, P.-C. (2017). brms: An R Package for Bayesian Multilevel Models Using Stan. *Journal of Statistical Software*, *80*(1). https://doi.org/10.18637/jss.v080.i01

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

Carroll, G. R., & Hannan, M. T. (1989). Density Dependence in the Evolution of Populations of Newspaper Organizations. *American Sociological Review*, *54*(4), 524–541. https://doi.org/10.2307/2095875

Cataldo, M., & Herbsleb, J. D. (2010). Architecting in software ecosystems: interface translucence as an enabler for scalable collaboration. *Proceedings of the Fourth European Conference on Software Architecture*. https://doi.org/10.1145/1842752.1842772

Cataldo, M., Herbsleb, J. D., & Carley, K. M. (2008). Socio-technical congruence: a framework for assessing the impact of technical and work dependencies on software development productivity. *Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. https://doi.org/10.1145/1414004.1414008

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

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

Conaldi, G., Lomi, A., & Tonellato, M. (2012). Dynamic Models of Affiliation and the Network Structure of Problem Solving in an Open Source Software Project. *Organizational Research Methods*, *15*(3), 385–412. https://doi.org/10.1177/1094428111430541

Cornwell, B. (2015). *Social Sequence Analysis: Methods and Applications*. Cambridge University Press.

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

Creswell, J. W. (2009). *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches* (3rd ed.). SAGE Publications.

Crowston, K. (2011). Lessons from Volunteering and Free/Libre Open Source Software Development for the Future of Work. In M. Chiasson, O. Henfridsson, H. Karsten, & J. I. DeGross (Eds.), *Researching the Future in Information Systems* (Vol. 356, pp. 215–229). Springer. https://doi.org/10.1007/978-3-642-21364-9_14

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

Daniel, S., Agarwal, R., & Stewart, K. J. (2013). The Effects of Diversity in Global, Distributed Collectives: A Study of Open Source Project Success. *Information Systems Research*, *24*(2), 312–333. https://doi.org/10.1287/isre.1120.0435

Daniel, S., & Stewart, K. (2016). Open source project success: Resource access, flow, and integration. *The Journal of Strategic Information Systems*, *25*(3), 159–176. https://doi.org/10.1016/j.jsis.2016.02.006

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

Doreian, P., & Woodard, K. L. (1992). Fixed list versus snowball selection of social networks. *Social Science Research*, *21*(2), 216–233. https://doi.org/10.1016/0049-089X(92)90016-A

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

Faust, K. (1997). Centrality in affiliation networks. *Social Networks*, *19*(2), 157–191. https://doi.org/10.1016/S0378-8733(96)00300-0

Freeman, J. H., & Audia, P. G. (2006). Community Ecology and the Sociology of Organizations. *Annual Review of Sociology*, *32*(1), 145–169. https://doi.org/10.1146/annurev.soc.32.061604.123135

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

Fruchterman, T. M. J., & Reingold, E. M. (1991). Graph drawing by force-directed placement. *Software: Practice and Experience*, *21*(11), 1129–1164. https://doi.org/10.1002/spe.4380211102

Funk, R. J., & Owen-Smith, J. (2017). A Dynamic Network Measure of Technological Change. *Management Science*, *63*(3), 791–817. https://doi.org/10.1287/mnsc.2015.2366

Gallagher, S. (2016, March 25). *Rage-quit: Coder unpublished 17 lines of JavaScript and "broke the Internet"*. Ars Technica. https://arstechnica.com/information-technology/2016/03/rage-quit-coder-unpublished-17-lines-of-javascript-and-broke-the-internet/

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.

German, D. M., Adams, B., & Hassan, A. E. (2013). The Evolution of the R Software Ecosystem. *Proceedings of the 17th European Conference on Software Maintenance and Reengineering*, 243–252. https://doi.org/10.1109/CSMR.2013.33

Gousios, G. (2013). The GHTorrent dataset and tool suite. *Proceedings of the 10th Working Conference on Mining Software Repositories (MSR '13)*, 233–236.

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

Grover, V., Lindberg, A., Benbasat, I., & Lyytinen, K. (2020). The Perils and Promises of Big Data Research in Information Systems. *Journal of the Association for Information Systems*, *21*(2), 268–293. https://doi.org/10.17705/1jais.00601

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

Hann, I.-H., Roberts, J. A., & Slaughter, S. A. (2013). All Are Not Equal: An Examination of the Economic Returns to Different Forms of Participation in Open Source Software Communities. *Information Systems Research*, *24*(3), 520–538. https://doi.org/10.1287/isre.2013.0474

Hannan, M. T., & Freeman, J. (1977). The Population Ecology of Organizations. *American Journal of Sociology*, *82*(5), 929–964. https://doi.org/10.1086/226424

Hannan, M. T., & Freeman, J. (1988). The Ecology of Organizational Mortality: American Labor Unions, 1836-1985. *American Journal of Sociology*, *94*(1), 25–52. https://doi.org/10.1086/228950

Hannan, M. T., & Freeman, J. (1989). *Organizational Ecology* (1st ed.). Harvard University Press.

Hanssen, G. K. (2012). A longitudinal case study of an emerging software ecosystem: Implications for practice and theory. *Journal of Systems and Software*, *85*(7), 1455–1466. https://doi.org/10.1016/j.jss.2011.04.020

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

Ho, S. Y., & Rai, A. (2017). Continued Voluntary Participation Intention in Firm-Participating Open Source Software Projects. *Information Systems Research*, *28*(3), 603–625. https://doi.org/10.1287/isre.2016.0687

Holland, P. W., & Leinhardt, S. (1977). A dynamic model for social networks. *The Journal of Mathematical Sociology*, *5*(1), 5–20. https://doi.org/10.1080/0022250X.1977.9989862

Howison, J., & Crowston, K. (2014). Collaboration Through Open Superposition: A Theory of the Open Source Way. *MIS Quarterly*, *38*(1), 29–50.

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

Hu, D., Zhao, J. L., & Cheng, J. (2012). Reputation management in an open source developer social network: An empirical study on determinants of positive evaluations. *Decision Support Systems*, *53*(3), 526–533. https://doi.org/10.1016/j.dss.2012.02.005

Huisman, M., & Snijders, T. A. B. (2003). Statistical Analysis of Longitudinal Network Data with Changing Composition. *Sociological Methods & Research*, *32*(2), 253–287. https://doi.org/10.1177/0049124103256096

Hukal, P., Berente, N., Germonprez, M., & Schecter, A. (2019). Bots Coordinating Work in Open Source Software Projects. *Computer*, *52*(9), 52–60. https://doi.org/10.1109/MC.2018.2885970

Hunter, T., & De Vynck, G. (2021, December 20). *The "most serious" security breach ever is unfolding right now. Here's what you need to know.* Washington Post. https://www.washingtonpost.com/technology/2021/12/20/log4j-hack-vulnerability-java/

Jacobides, M. G., Cennamo, C., & Gawer, A. (2018). Towards a theory of ecosystems. *Strategic Management Journal*, *39*(8), 2255–2276. https://doi.org/10.1002/smj.2904

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

Jansen, S., Finkelstein, A., & Brinkkemper, S. (2009). A Sense of Community: A Research Agenda for Software Ecosystems. *2009 31st International Conference on Software Engineering - Companion Volume*, 187–190. https://doi.org/10.1109/ICSE-COMPANION.2009.5070978

Kalish, Y. (2020). Stochastic Actor-Oriented Models for the Co-Evolution of Networks and Behavior: An Introduction and Tutorial. *Organizational Research Methods*, *23*(3), 511–534. https://doi.org/10.1177/1094428118825300

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

Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., German, D. M., & Damian, D. (2016). An in-depth study of the promises and perils of mining GitHub. *Empirical Software Engineering*, *21*, 2035–2071. https://doi.org/10.1007/s10664-015-9393-5

Kapoor, R., & Agarwal, S. (2017). Sustaining Superior Performance in Business Ecosystems: Evidence from Application Software Developers in the iOS and Android Smartphone Ecosystems. *Organization Science*, *28*(3), 531–551. https://doi.org/10.1287/orsc.2017.1122

Katz, J. (2020). *Libraries.io Open Source Repository and Dependency Metadata* (Version 1.6.0) [Data set]. Zenodo. https://doi.org/10.5281/ZENODO.3626071

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

Kim, Y., Jarvenpaa, S. L., & Gu, B. (2018). External Bridging and Internal Bonding: Unlocking the Generative Resources of Member Time and Attention Spent in Online Communities. *MIS Quarterly*, *42*(1), 265–283. https://doi.org/10.25300/MISQ/2018/13278

Koskinen, J., & Edling, C. (2012). Modelling the evolution of a bipartite network—Peer referral in interlocking directorates. *Social Networks*, *34*(3), 309–322. https://doi.org/10.1016/j.socnet.2010.03.001

Laumann, E. O., Marsden, P. V., & Prensky, D. (1989). The Boundary Specification Problem in Network Analysis. In L. C. Freeman, D. R. White, & A. K. Romney (Eds.), *Research Methods in Social Network Analysis* (pp. 61–87). George Mason University Press.

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. (2020). Developing Theory Through Integrating Human & Machine Pattern Recognition. *Journal of the Association for Information Systems*, *21*(1), 90–116. https://doi.org/10.17705/1jais.00593

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

Lospinoso, J. A., Schweinberger, M., Snijders, T. A. B., & Ripley, R. M. (2011). Assessing and accounting for time heterogeneity in stochastic actor oriented models. *Advances in Data Analysis and Classification*, *5*, 147–176. https://doi.org/10.1007/s11634-010-0076-1

Lospinoso, J., & Snijders, T. A. B. (2019). Goodness of fit for stochastic actor-oriented models. *Methodological Innovations*, *12*(3). https://doi.org/10.1177/2059799119884282

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

Markus, M. L. (2007). The governance of free/open source software projects: monolithic, multidimensional, or configurational? *Journal of Management & Governance*, *11*(2), 151–163. https://doi.org/10.1007/s10997-007-9021-x

Markus, M. L., Agres, C. E., & Manville, B. (2000). What Makes a Virtual Organization Work? *MIT Sloan Management Review*, *42*(1), 13–26.

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.

McPherson, J. M., Popielarz, P. A., & Drobnic, S. (1992). Social Networks and Organizational Dynamics. *American Sociological Review*, *57*(2), 153–170. https://doi.org/10.2307/2096202

McPherson, M. (1983). An Ecology of Affiliation. *American Sociological Review*, *48*(4), 519–532. https://doi.org/10.2307/2117719

Messerschmitt, D. G., & Szyperski, C. (2003). *Software Ecosystem: Understanding an Indispensable Technology and Industry*. MIT Press.

Mockus, A., Fielding, R. T., & Herbsleb, J. D. (2002). Two case studies of open source software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology*, *11*(3), 309–346. https://doi.org/10.1145/567793.567795

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

Oh, W., Moon, J. Y., Hahn, J., & Kim, T. (2016). Research Note—Leader Influence on Sustained Participation in Online Collaborative Work Communities: A Simulation-Based Approach. *Information Systems Research*, *27*(2), 383–402. https://doi.org/10.1287/isre.2016.0632

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

Peng, G., Wan, Y., & Woodlock, P. (2013). Network ties and the success of open source software development. *The Journal of Strategic Information Systems*, *22*(4), 269–281. https://doi.org/10.1016/j.jsis.2013.05.001

Podolny, J. M., & Stuart, T. E. (1995). A Role-Based Ecology of Technological Change. *American Journal of Sociology*, *100*(5), 1224–1260. https://doi.org/10.1086/230637

Podolny, J. M., Stuart, T. E., & Hannan, M. T. (1996). Networks, Knowledge, and Niches: Competition in the Worldwide Semiconductor Industry, 1984-1991. *American Journal of Sociology*, *102*(3), 659–689. https://doi.org/10.1086/230994

Popielarz, P. A., & McPherson, J. M. (1995). On the Edge or In Between: Niche Position, Niche Overlap, and the Duration of Voluntary Association Memberships. *American Journal of Sociology*, *101*(3), 698–720. https://doi.org/10.1086/230757

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

Ren, Y., Harper, F. M., Drenner, S., Terveen, L., Kiesler, S., Riedl, J., & Kraut, R. E. (2012). Building Member Attachment in Online Communities: Applying Theories of Group Identity and Interpersonal Bonds. *MIS Quarterly*, *36*(3), 841–864. https://doi.org/10.2307/41703483

Ridings, C., & Wasko, M. (2010). Online discussion group sustainability: Investigating the interplay between structural dynamics and social dynamics over time. *Journal of the Association for Information Systems*, *11*(2), 95–121. https://doi.org/10.17705/1jais.00220

Ripley, R. M., Snijders, T. A. B., Boda, Z., Vörös, A., & Preciado, P. (2022). *Manual for RSiena*. University of Oxford, Department of Statistics; Nuffield College. http://www.stats.ox.ac.uk/~snijders/siena/RSiena_Manual.pdf

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

Robins, G., & Alexander, M. (2004). Small Worlds Among Interlocking Directors: Network Structure and Distance in Bipartite Graphs. *Computational & Mathematical Organization Theory*, *10*(1), 69–94. https://doi.org/10.1023/B:CMOT.0000032580.12184.c0

Ruef, M. (2000). The Emergence of Organizational Forms: A Community Ecology Approach. *American Journal of Sociology*, *106*(3), 658–714. https://doi.org/10.1086/318963

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

Schweinberger, M. (2012). Statistical modelling of network panel data: Goodness of fit. *British Journal of Mathematical and Statistical Psychology*, *65*(2), 263–281. https://doi.org/10.1111/j.2044-8317.2011.02022.x

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. (2010). The small-world effect: The influence of macro-level properties of developer collaboration networks on open-source project success. *ACM Transactions on Software Engineering and Methodology*, *20*(2), 1–27. https://doi.org/10.1145/1824760.1824763

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

Snijders, T. A. B. (1996). Stochastic actor-oriented models for network change. *The Journal of Mathematical Sociology*, *21*(1-2), 149–172. https://doi.org/10.1080/0022250X.1996.9990178

Snijders, T. A. B. (2001). The Statistical Evaluation of Social Network Dynamics. *Sociological Methodology*, *31*(1), 361–395. https://doi.org/10.1111/0081-1750.00099

Snijders, T. A. B. (2005). Models for Longitudinal Network Data. In P. J. Carrington, J. Scott, & S. Wasserman (Eds.), *Models and Methods in Social Network Analysis* (pp. 215–247). Cambridge University Press.

Snijders, T. A. B., van de Bunt, G. G., & Steglich, C. E. G. (2010). Introduction to stochastic actor-based models for network dynamics. *Social Networks*, *32*(1), 44–60. https://doi.org/10.1016/j.socnet.2009.02.004

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

Stinchcombe, A. L. (1965). Social Structure and Organizations. In J. G. March (Ed.), *Handbook of Organizations* (pp. 142–193). Rand McNally.

Stuart, T. E., & Podolny, J. M. (1996). Local search and the evolution of technological capabilities. *Strategic Management Journal*, *17*(S1), 21–38. https://doi.org/10.1002/smj.4250171004

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

Sun, Y., Fang, Y., & Lim, K. H. (2012). Understanding sustained participation in transactional virtual communities. *Decision Support Systems*, *53*(1), 12–22. https://doi.org/10.1016/j.dss.2011.10.006

Tang, T. (Ya)., Fang, E. (Er)., & Qualls, W. J. (2020). More Is Not Necessarily Better: An Absorptive Capacity Perspective on Network Effects in Open Source Software Development Communities. *MIS Quarterly*, *44*(4), 1651–1678. https://doi.org/10.25300/MISQ/2020/13991

Uzzi, B., & Spiro, J. (2005). Collaboration and Creativity: The Small World Problem. *American Journal of Sociology*, *111*(2), 447–504. https://doi.org/10.1086/432782

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

von Krogh, G., Haefliger, S., Spaeth, S., & Wallin, M. W. (2012). Carrots and Rainbows: Motivation and Social Practice in Open Source Software Development. *MIS Quarterly*, *36*(2), 649–676.

von Krogh, G., & Spaeth, S. (2007). The open source software phenomenon: Characteristics that promote research. *The Journal of Strategic Information Systems*, *16*(3), 236–253. https://doi.org/10.1016/j.jsis.2007.06.001

von Krogh, G., Spaeth, S., & Lakhani, K. R. (2003). Community, joining, and specialization in open source software innovation: a case study. *Research Policy*, *32*(7), 1217–1241. https://doi.org/10.1016/S0048-7333(03)00050-7

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

Wasko, M., & Faraj, S. (2005). Why Should I Share? Examining Social Capital and Knowledge Contribution in Electronic Networks of Practice. *MIS Quarterly*, *29*(1), 35–57.

Wasserman, S., & Faust, K. (1994). *Social Network Analysis: Methods and Applications*. Cambridge University Press.

West, J., & O'mahony, S. (2008). The Role of Participation Architecture in Growing Sponsored Open Source Communities. *Industry and Innovation*, *15*(2), 145–168. https://doi.org/10.1080/13662710801970142

Zhang, C., Hahn, J., & De, P. (2013). Research Note—Continued Participation in Online Innovation Communities: Does Community Response Matter Equally for Everyone? *Information Systems Research*, *24*(4), 1112–1130. https://doi.org/10.1287/isre.2013.0485

Zhu, H., Chen, J., Matthews, T., Pal, A., Badenes, H., & Kraut, R. E. (2014). Selecting an Effective Niche: An Ecological View of the Success of Online Communities. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 301–310. https://doi.org/10.1145/2556288.2557348

Zhu, H., Kraut, R. E., & Kittur, A. (2014). The Impact of Membership Overlap on the Survival of Online Communities. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 281–290. https://doi.org/10.1145/2556288.2557213