Network Perspectives on Open Source Sustainability and Novelty

Hongbo Fang

CMU-S3D-24-107

July 2024

Software and Societal Systems Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Dr. James Herbsleb, Co-chair
Dr. Bogdan Vasilescu, Co-chair
Dr. Patrick Park
Dr. Tom Zimmermann (Microsoft Research)
Dr. James Evans (University of Chicago)

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Societal Computing.

This work was supported by the following funding: NSF Awards 1546393, 1633083, 1717415, 190131, 2107298, 2120323, Alfred P.Sloan Foundation, Google Open Source Programs Office, Google Faculty Research Awards, and Google Award for Inclusion Research.

Keywords: open source software, social network analysis, sustainability, innovation

To my family.

Abstract

Open-source software (OSS) has become increasingly important, permeating diverse sectors and delivering substantial economic advantages alongside notable societal influence. Within the realm of open-source software (OSS), two fundamental questions revolve around the sustainability of software development and the creation of innovative software projects. The sustainability of OSS projects heavily relies on the active participation of volunteer developers, necessitating a comprehensive understanding of the mechanisms that allure and sustain their contributions. Similarly, the development of innovative software projects serves as a pivotal driver for developer engagement and organizational investment in the open-source domain. The emergence of innovative OSS projects attracts significant attention due to their potential to instigate transformative shifts in the current software landscape, resulting in substantial impact.

A wealth of empirical research has been devoted to exploring these themes. However, prior studies have predominantly focused on associating outcome variables solely with the inherent characteristics of individual projects or the developers involved. This approach often neglects the intricate web of relationships that bind projects and developers, encompassing dependencies among projects and the social networks that connect developers.

My thesis extends the existing body of research by employing a network-analysis framework to investigate the interplay between projects and developers. The first part of my thesis studies developers' interactions within Twitter networks and their collaborative networks established through joint contributions to projects. Through this investigation, I ascertain the impact of developer engagements on social media and the code hosting platforms on the allocation of developer attention and efforts, subsequently influencing the sustainability of projects.

In the second part, I introduce a novel metric for gauging project novelty in OSS, grounded in the analysis of projects' dependency networks. Additionally, I evaluate the networks of interactions among developers on code hosting platforms, exploring the relationship between access to knowledge through these interactions and its impact on project novelty.

In summary, my research contributes both theoretically to the OSS literature and practically to OSS practitioners. I offer actionable insights to developers on enhancing project sustainability and fostering innovation. Furthermore, by adopting a network perspective, I provide a nuanced understanding of project sustainability and novelty through an analysis of the intricate relationships and connections among projects and developers.

Acknowledgments

I am profoundly grateful for the invaluable guidance and support of my two exceptional advisors, Professor James Herbsleb and Bogdan Vasilscu, throughout my past four years at CMU. They have played an instrumental role in shaping me into a more proficient researcher. Moreover, they have been unwaveringly supportive, not only in helping me forge connections within the academic and professional communities but also in facilitating friendships and assisting with my career development. They have not just been exemplary advisors and mentors but also cherished friends. I I consider myself exceptionally fortunate to have had them as my advisors.

Throughout my doctoral journey, I had the privilege of completing an internship with Microsoft Research, where I received invaluable guidance and support from my mentors: Dr. Chris Bird, Dr. Nicole Forsgren, Jeremy Haubold, and, most notably, Dr. Tom Zimmermann. During my internship, they generously dedicated their time to discuss research ideas and provided guidance at every stage of my study. Their extensive knowledge and expertise have been a constant source of inspiration, and I have learned immensely from them. Equally noteworthy is their open-mindedness, which makes every conversation with them an exciting opportunity for new insights. I am very fortunate and grateful that Dr. Tom Zimmermann accepted my invitation to be part of my PhD committee, and I hope to continue the inspiring conversations.

I want to thank Prof. Patrick Park for his guidance and support during my PhD. In my view, he is an exceedingly rare asset within our department As I endeavored to integrate network analysis and social theories into my research, I found Prof. Park to be one of the few individuals within the School of Computer Science with a profound background in social science. His expertise has been instrumental in shaping my work. Furthermore, we collaborated to initiate the Computational Social Science Seminar at S3D, an endeavor that has introduced fresh and innovative perspectives to both me and our department. This accomplishment would not have been possible without Patrick's leadership as the faculty lead and his mentorship throughout the entire process.

I would like to extend my gratitude to Prof. James Evans. Our collaboration began during my undergraduate years, and he played a pivotal role in providing support during my PhD application process. Not only is Prof. Evans deeply knowledgeable in the field of social science, but he also possesses a remarkable capacity for creativity, effortlessly amalgamating perspectives from diverse fields. Our past collaborations and conversations have been truly enriching, and I eagerly anticipate his continued presence on my PhD committee. This ensures that our collaboration and insightful discussions will persist and flourish.

Finally, I want to express my gratitude to my family back in China and all my friends who have been a part of my journey at CMU. I genuinely believe that my PhD journey has been enriched and made truly joyful by your presence and companionship. Thank you for being an integral part of my happiness during this phase of my life.

Contents

1 Introduction **1**

1.1 Literature review and knowledge gap . . . 2

1.1.1 Social media and OSS . . . 3

1.1.2 OSS sustainability and the attraction of new developers . . . 3

1.1.3 OSS novelty . . . 4

1.2 Thesis . . . 4

1.2.1 The use of social media in OSS and the attraction of new stargazers/developers 4

1.2.2 Attraction of new developers and the project's labor pool . . . 5

1.2.3 Open-source novelty and atypical combination . . . 5

1.2.4 The origin of open-source innovation . . . 5

1.3 Proposed contributions . . . 6

2 Open-source Information Diffusion on Social Media and its Network Effects **7**

2.1 Introduction . . . 7

2.2 Related works . . . 8

2.3 Research questions . . . 10

2.4 Methods . . . 11

2.4.1 Study Overview . . . 11

2.4.2 Preprocessing . . . 13

2.4.3 Aggregating Tweets into Bursts . . . 14

2.4.4 Compiling a Control Group . . . 16

2.4.5 Estimating Tweet Burst Causal Effects and Their Moderators (**RQ1**, **RQ2**) 17

2.4.6 Identifying Characteristics of Developers Likely Attracted By Tweets (**RQ3**) . . . 18

2.5 Results . . . 19

2.5.1 Tweet Effects on Project Popularity and New Contributors (**RQ1**) . . . 19

2.5.2 Characteristics of Impactful Tweets (**RQ2**) . . . 21

2.5.3 Characteristics of New Contributors (**RQ3**) . . . 22

2.6 Implications . . . 23

2.6.1 Twitter can be an effective mechanism to popularize open source software projects . . . 23

2.6.2 Not all tweets are created equal . . . 24

2.6.3 Community engagement on Twitter is important . . . 24

2.6.4 Social connections can be leveraged for work-related tasks, especially short-term ones . . . 25
2.6.5 Attention can be a double edged sword . . . 25
2.6.6 The role of Twitter in open-source development . . . 25
2.6.7 Twitter can be more tightly integrated into code hosting platforms . . . 25
2.7 Conclusions . . . 26

3 Social-technical Networks and its Influence on OSS Sustainability 27
3.1 Introduction . . . 27
3.2 Related Works . . . 28
3.3 Theoretical Framework and Research Hypotheses . . . 29
3.4 Methods . . . 30
3.4.1 Key Study Design Decisions and Tradeoffs . . . 30
3.4.2 Overview of the Analysis . . . 32
3.4.3 Data Collection and Filtering . . . 33
3.4.4 Part I: Individual Models . . . 34
3.4.5 Part II: Project Models . . . 36
3.5 Results . . . 37
3.5.1 Models of Individual Joining Tendency . . . 37
3.5.2 Models of the Number of New Contributors . . . 38
3.6 Implications . . . 39
3.6.1 Labor Pool as an Important Factor for Project Sustainability . . . 40
3.6.2 The Size of the Labor Pool Is Important, but It Is Not All That Matters . . . 41
3.6.3 Towards Targeted Project Promotion . . . 41
3.6.4 The Relationship Between Projects and the Success of Open-Source Ecosystems . . . 41
3.6.5 Validations and Robustness Checks . . . 42
3.7 Conclusions . . . 42

4 Novelty of OSS Projects 45
4.1 Introduction . . . 45
4.2 Theoretical Framework and Hypothesis . . . 46
4.3 Methods . . . 48
4.3.1 Data Collection . . . 48
4.3.2 Pre-processing of package dependencies . . . 49
4.3.3 Quantifying Project Atypicality . . . 49
4.3.4 Studying the Association between Project Atypicality and Interest from Developers and Users . . . 52
4.3.5 Survival analysis on the sustainability of project and its relationship to atypicality . . . 52
4.4 Results . . . 53
4.4.1 Understanding the Atypicality Measurement . . . 53
4.4.2 Project Atypicality and the Size of Developer Team . . . 55
4.4.3 Project Atypicality and its Sustained Development . . . 55

4.5 Implications . . . 56
4.5.1 Atypical Projects Draw User Attentions . . . 56
4.5.2 Atypical Projects Face Difficulty in Development . . . 57
4.5.3 The Tension Between Popularity and Participation . . . 57
4.5.4 Future Work: The Context Surrounding Innovativeness . . . 58
4.5.5 Future Work: Atypicality and Measurements of Innovation . . . 58
4.6 Conclusions . . . 58

5 The Strength of Weak Ties in Open-Source Software Innovation 61
5.1 Introduction . . . 61
5.2 Theoretical Frameworks and Hypothesizes . . . 62
5.2.1 Social-Technical Interactions, Knowledge Diffusion and Innovation . . . 62
5.2.2 The Differed Tie Strength and their Effects on Innovation . . . 63
5.3 Related Works . . . 64
5.3.1 Innovation in open-source software . . . 64
5.3.2 Social Ties, Knowledge Flow and Innovation . . . 65
5.3.3 Social Network Analysis in Open Source Software . . . 65
5.4 Methods . . . 66
5.4.1 Dataset . . . 66
5.4.2 Data Pre-process . . . 66
5.4.3 Network Construction . . . 67
5.4.4 Variable Measurements . . . 68
5.4.5 Using Principal Component Analysis (PCA) to Decompose the Correlated Variables . . . 70
5.5 Results . . . 70
5.5.1 Empirical Evidence on the Strength of Different Ties . . . 71
5.5.2 PCA Analysis of Degree and Diversity Variables . . . 72
5.5.3 The Influence of Degree Centrality and Diversity of Ties on Project Novelty 73
5.5.4 The Influence of Degree and Diversity Variables Across Years . . . 75
5.6 Discussions . . . 76
5.6.1 Accessing Diverse Knowledge Drives the Development of Innovative Projects . . . 76
5.6.2 Learning through Weak Ties Boosts the Creation of Novel Projects . . . 76
5.7 Conclusions . . . 77

Bibliography 81

List of Figures

2.1 Overview of our data collection and analysis steps. 10
2.2 The number of new GITHUB stars attracted by the `axa-group/nlp.js` repository per day before, during, and after the Twitter burst in the Appendix. The inset shows the timeline of tweets (labeled) and retweets (unlabeled). 12
2.3 Operationalization of tweet *burst* and diff-in-diff data setup. TOP LEFT: Tweets mentioning the same repository within X days of each other are considered part of the same burst. BOTTOM LEFT: Two bursts mentioning the same repository must be at least Y days apart. RIGHT: Control group repositories must not have experienced any bursts of their own at least Y days after the end of the corresponding treatment group repository burst. 14
2.4 Tweets mentioning the `axa-group/nlp.js` repository as part of a burst of social media activity spanning 10 days. 15

3.1 The summary of data preparation and analysis process 32
3.2 Illustration of the project relative advantage 36
3.3 Illustration of the social-technical effects on developer joining tendency (y=2021). 38

4.1 Package combinations within a project 50
4.2 Switching package importing events 51
4.3 Typical combination of Python packages (until 2019) 54
4.4 Estimated coefficient of Z-score with different outcome variables across different period duration (Error bar represents 95% confidence interval) 55

5.1 The triads and triangles in networks 71
5.2 Knowledge access and its relationship to project novelty across years (error bar represents the 95% confidence interval) 74

List of Tables

2.1 Summaries of diff-in-diff regressions estimating the effect of tweets mentioning GITHUB repositories on attracting new stars. 19
2.2 Summaries of diff-in-diff regressions estimating the effect of tweets mentioning GITHUB repositories on attracting new committers. 20
2.3 Characterizing the developers attracted by tweets 23

3.1 Modeling the project-joining tendency of developers 39
3.2 Modeling the number of new developers a project will receive in the next year ($y = 2021$) 40

4.1 Results for regression analysis (X=5) 56
4.2 The effect of project atypicality on the sustained development (Y=2016) 57

5.1 The definitions of the variables in our models. 78
5.2 Empirical evaluation of the "strength" of social ties 79
5.3 Proportional of variance explained by each principal component for degree and diversity variables 79
5.4 Loadings of principal components on degree/diversity variables 79
5.5 Regression analysis on factors influencing the project atypicality 80

Chapter 1

Introduction

Open-source software (OSS) refers to the software released under a license that permits the inspection, use, modification and redistribution of the software's source code [59]. In the past decades, we have witnessed a drastic expansion of OSS. The number of projects being developed, and the amount of developers involved in OSS development keeps increasing over the years [125]. OSS projects play important roles in varied domains such as energy [1], healthcare [2], and finance [3]. Survey reports showed that over three quarters of US companies create software for customers which is built on OSS [72], and in 2020 alone, OSS is estimated to directly contribute £15.7 billion to UK's economy, with a big potential economic impact [4]. Nowadays, OSS has become the digital infrastructure which many of our businesses rely on, and influences varied aspects of our daily life [72].

The factors making OSS projects sustainable is an important question, as many OSS projects rely on the contribution from volunteer developers [58] and even the widely-used projects may suffer from the inadequacy of developers [69]. Another important research question is the team composition of OSS projects and what factors influence the team performance. Due to the lack of formal coordination mechanisms that are traditionally used in industry software development, it is an interesting question that how developers identify new projects to work on and how do they coordinate with other team members [56]. Existing works try to answer those questions from multiple perspectives. For example, it is reported that developers rely on signals such as the project popularity [29], or the quality of project readmes [160] to identify the project to work on, and the project governance mechanism [153], age, the number of current developers, and the size of the project's target audience [47] all influence its sustainability in the long term.

Most existing works study OSS projects and developers in a standalone fashion, that they focus on the characteristics of projects or developers being studied itself, but overlook the connections between each other and their influence to the individual project, or developer under study. Indeed, the interactions and connections among projects, among developers, and between projects and developers are becoming increasingly complicated. The growing number of software packages

[1] https://github.com/OpenEMS/openems

[2] https://github.com/openemr/openemr

[3] https://github.com/KDE/kmymoney

[4] https://openuk.uk/press-releases-posts/open-source-software-contributed-an-estimated-46-5bn-to-uk-business-in-2020-according-to-openuk/

avoids reinventing the wheel, but at the same time creates more dependencies between projects that require developers' attentions [98]. Meanwhile, open-source developers are getting increasingly "social" [190], and the interactions between developers go beyond the code hosting platforms and extend to online forums, social media, and many online and offline channels [192, 193].

To understand the relationships among projects and developers, I adopt network analysis, which is a technique widely used in social science [94], management science [169, 224], computer science [218] and many other domains. This approach typically constructs a topological graph between entities of interest, and analyze the property of individual entities in the network or the network structure as a whole to extract meaning information from the relationships formed. In the context of OSS research, while not being the most commonly used statistical approaches, there has been studies adopting the idea of network analysis. For example, Grewal et al.found that the network position of projects and their managers influence project success [95], Qiu and her collaborators reported the developers' social capital, measured by the developers' network position, influence the sustainability of their contributions [161].

In my thesis, I extend the previous works on OSS network analysis. I go beyond the networks of developers within the code hosting platforms itself (*e.g.*, GitHub or SourceForge), but also study the interactions on social media space. Similarly, I analyze the networks of project dependencies which captures different information from the existing works, which often focus on projects sharing the same developers [95, 131]. In addition to study the projects' or developers' network position, my research also applies more complicated network analysis techniques such as the random reshuffle of network links, and incorporating the characteristics of network nodes into analysis. It enables us to extract more information from the networks, for example the overall network properties and the node level attributes, into our analysis.

The inclusion of new networks and new techniques provide new answers to important OSS questions. In my thesis, I find that the sustainability of OSS projects can be characterized and explained by the social-technical relationships between the project and the developers around in the network, and using social media to promote projects can be effective, but influence the number of project stars and developers at different scales. In addition, analysis on dependency networks create opportunities to understand the novelty of open-source projects, which is a traditionally difficult concept to measure despite its significance for OSS community [72], as existing works talk about how open-source creates opportunity for innovation [216, 217] and the importance of software novelty to industry and other fields [71].

1.1 Literature review and knowledge gap

In my thesis, I adopt network analysis approach to provide new answers to established OSS questions, and create new directions on large-scale empirical analysis over open-source novelty. In this section, I summarize the existing works for each topic I touch and introduce social network analysis in general.

1.1.1 Social media and OSS

Social media, defined as channels which support many-to-many communication, are widely used in OSS community. Social media is used for multiple purposes, developers use social media to connect with each other, collect information about interesting projects and news in the community, and promote their projects to a large community [20, 182, 192, 193], just to name a few.

Different media platforms are used for different purposes, and there exists an ecosystem of social media which supports the various aspects of open-source software development [9, 193].

Despite being widely used, the practical influence of social media on open-source projects, particularly to those mentioned in social media space is unclear. My research address this gap by first using a difference in differences model to make causal inference about social media's attraction to new developers and stargazers, and then report the varied influence different media platforms have on the developer community at different network distances.

1.1.2 OSS sustainability and the attraction of new developers

The sustainability of open-source software is an important research question. Lacking traditional coordination and work assignment mechanisms, OSS developers identify projects and tasks to work on themselves [58]. It can be problematic as software projects may need constant maintenance, and OSS developers may disengage from the development and maintenance tasks for many reasons [140]. Indeed, project abandonment is not uncommon in OSS [13] and it even applies to widely used projects which cause tremendous damage to services depending on them [69].

At a high level, existing works look at the OSS sustainability as attraction and retention problems [187]. Attraction is the early-stage interactions developers have with the projects, Because of the self-organizing nature of open-source contributions, prior research has found that developers tend to promote their projects in multiple platforms [31, 182], and developers rely on "signals", such as the number of project stars (popularity) [29], the quality of project's READMEs [160], to decide whether they will contribute. Research on the retention of project developers focuses on the developer interaction within the project team and project governance, and study how developers evolve to a sustained contributor and the reasons of developer disengagement. Von Krogh et al [212] proposed a "joining script" which described the list of activities that leads a new developer to a project member, and researchers found that the developers' relative sociality [230], their attitudes [231], and the project factors such as the availability of work opportunities, project popularity [231, 232], and the internal governance mechnisms [153] all play a role to influence the likelihood of long-term contributions.

The existing works has been largely focused on the characteristics of individual developer or project and their influence to sustainability. By characterizing the networks of developers around a project, I extend our understanding on project sustainability by suggesting the network position of projects matter to their sustainability, and projects may compete for the attention of developers which influence their sustainability as well.

1.1.3 OSS novelty

It has long been recognized that open-source software development is an avenue for innovation and creative expression — "how creative a person feels when working on the project is the strongest and most pervasive driver" of participation in open source [120], and the easy access of open-source software by different stakeholders facilitate the communications and encourage "open-innovation" [216].

Innovation has long been a highly regarded yet challenging concept across various domains [4, 71, 194]. Despite the widespread recognition and expectation for open-source software (OSS) to generate innovative projects, there has been a notable absence of prior studies, to the best of our knowledge, that systematically analyze the level and process of OSS innovation on a large scale.

In my thesis, I contribute one of the initial computational measurements of OSS project innovation by adopting a Schumpeterian [176] view of innovation, which perceives innovation as a novel recombination of existing bits of knowledge. This approach involves computing project novelty based on the combination of software packages that are typically not used together. Additionally, I delve into exploring the origins of innovation by analyzing the relationship between the social networks where open-source software developers acquire knowledge and the subsequent novelty of the projects they develop.

1.2 Thesis

In my thesis, I use large-scale social network analysis to provide new perspectives about open-source sustainability, the attraction of new developers to OSS projects, and the influence of social media on OSS. My work also creates new directions about the novelty of OSS projects and propose metrics to analyze OSS novelty and its association with other project outcome of interest at a large scale. I briefly summarize the several projects consisting of my thesis below, and I use "we" when describing works that I collaborated with other researchers.

1.2.1 The use of social media in OSS and the attraction of new stargazers/developers

This study consists of two papers. In the first work, we analyze the posts on Twitter (*i.e.*, tweets) mentioning open-source projects on GitHub, and use difference in differences framework [65] to make causal inference about the effects of tweets on the attraction of new stargazers and developers to the mentioned projects. An analysis over 44,544 tweets mentioning 2,370 open-source projects found out that tweets can help attract both new stargazers and new developers to the mentioned projects, and the attraction effects are stronger for stargazers compared to new developers. In addition, we identify the features of tweets that influence the attraction effects.

The second paper takes one step further to understand the use of different social media platforms and the difference in their attraction effects. Particularly the different stargazers or developers they attract. For all stargazers and developers to a project, we measure their network distances to the project owner, and conduct regression analysis to study the association between the number of times that this project being mentioned on different media platforms, and the

number of stargazers/developers from each network distance. We found that different media platforms seem to attract stargazers/developers from different network distances, indicating varied attraction effects.

Chapter 2 provides more details about these two works.

1.2.2 Attraction of new developers and the project's labor pool

In the study, we view the sustainable development of OSS projects from a supply-chain perspective, where we have the projects under development or maintenance at the demand side that are in need of contribution, and the software developers who are willing, and capable to contribute to those projects in the supply side. For each open-source projects, the number of potential developers who are likely to contribute is limited, and we describe them as the "labor pool" of the project in this study. We operationzalition a project's labor pool based on the collaboration networks of their existing developers, and we show that the characteristics of their developers, particularly about their social connections and technical similarities to the focal project (and its developers) is predictive of the number of new developers that the project can receive in the near future. In addition, our results indicate that projects may compete for developers' attentions and being exposed to many projects may decrease the likelihood of joining each one of them. This study is conducted over 516,893 Python projects recorded in the World of Code dataset [132], and more details can be found in Chapter 3.

1.2.3 Open-source novelty and atypical combination

In this work, we adopt the Schumpeterian [176] view of innovation as a novel recombination of existing bits of knowledge, and draw inspiration from studies of innovation in research publications and its relationship to scientific impact. We propose to measure the novelty of open-source projects based on the atypical combination of dependency packages. A package that a software project imports imply its functions, and some packages are typically used together in a project because they serve related purposes. Oppositely, software projects importing packages that are commonly not imported together may imply its unique functions or applications from other projects and we consider it to be "novel". We first construct a project-package dependency graph, and compare it against a random null graph, where all the edges in the graph are randomly reshuffled with the degree of each node (*i.e.*, projects and packages) holds the same. We compute the atypicality measurement of package co-dependencies based this comparison and aggregate the package level atypically at the project level by taking its average. Our result suggests that atypical projects tend to be more popular and receive more stars, but they are more likely to have a sustainability problem because each developer have a higher workload. Chapter 4 provides more details about this work.

1.2.4 The origin of open-source innovation

In this study, we scrutinize developers' knowledge access through three distinct social networks: code commits, issue posting, and project stars. We empirically assess both the volume and diversity of interactions within these networks and examine their impact on the novelty of projects

developed by developers in subsequent periods. Our findings indicate that developers who access a more diverse set of knowledge tend to produce more novel projects, although the volume of knowledge accessed shows minimal effect. Notably, the diversity of knowledge accessed through weak ties emerges as a particularly significant factor in enhancing developer innovation. Chapter 5 elaborates further on these findings.

1.3 Proposed contributions

The proposed contributions of this thesis include the following:

- Use causal analysis to verify the effect of social media posts to attract stargazers and developers to OSS projects.
- Describe the varied influence different media platforms have to OSS projects by suggesting they attract stargazers and developers from different network distance.
- Provide a network perspective to better understand and predict the number of new developers to a project.
- Propose arguably the first measurement of OSS project novelty at a large scale and conduct preliminary analysis to understand the association between novelty and other project outcomes.
- Perform an empirical analysis aimed at comprehending the access to knowledge through social networks and its impact on the novelty of projects.
- Introduce novel approaches to OSS research community, including the difference in differences framework to make causal analysis over observational data, and social network analysis to study the relationships among OSS projects and developers.

Chapter 2

Open-source Information Diffusion on Social Media and its Network Effects

2.1 Introduction

In open-source software (OSS) development, attention can be a double edged sword. Sometimes, OSS projects receive too much attention, and maintainers have to deal with overwhelming volumes of requests and demands from users [73]; in these cases, maintainers might rather fend off new attention coming their way. Other times, even successful OSS projects are unable to attract more than a few contributors, and occasionally OSS projects are maintained by no one at all [13, 50]; in these cases, more sustained involvement from users and contributors would be welcome. Yet, for many OSS projects, gaining attention from the community, *e.g.*, to increase adoption and attract more contributors, remains a challenge.

Several mechanisms through which OSS projects can gain attention [31, 133, 201] and attract new contributors [27, 133, 160] have been studied in the past. The literature is especially rich in recent years, in the context of social coding platforms like GitHub , because of the high level of transparency and many opportunities for project maintainers to *signal*, explicitly and implicitly, about their work [60]. For example, prior studies of OSS projects hosted on GitHub have found that how projects organize their repository homepages and README files [160], whether projects get *featured* by the hosting platform [133], whether projects have public releases [29], and how maintainers use prominent repository badges to indicate less observable project qualities [201], all have an impact on how the project is perceived by its audience and even the actions that some audience members take, *e.g.*, joining the project.

However, prior work has, by and large, focused only on *endogenous* or "in-network" attention eliciting mechanisms, *i.e.*, taking actions or displaying signals afforded by the code hosting platform itself. This leaves an important gap—little is known about attention eliciting mechanisms that can be considered *exogenous* from the perspective of OSS projects hosted on GitHub or similar platforms. Here we focus on one such mechanism, social media. Social media platforms, widely used by software developers [191], enable OSS maintainers to share their work with a potentially larger audience, that exists beyond their immediate connections on any code hosting platform; *e.g.*, social media posts about an OSS project may be amplified by the authors' social

networks, influential social media users, or the platform itself. Social media platforms also tend to have low barrier to participation and high viewership, which makes them actionable and potentially impactful for OSS maintainers, admirers, and evangelists looking to attract attention to projects in need. A better understanding of the effectiveness of using social media to attract attention to OSS projects could directly impact the projects' success and sustainability.

Yet, little is known about how much social media activity can contribute to OSS sustainability, if at all. The evidence from other contexts suggests that actions taken on social media platforms can have spillover, out-of-network effects; *e.g.*, researchers have found that tweets can predict movie ratings [154] and increase citations to academic papers [129]. Can similar effects be expected for OSS?

To address this gap, in this paper, we compile a large dataset of 44,544 tweets containing links to open source GitHub repositories,[1] spanning 6 months of history, and with cross-links between user profiles on both platforms. We then apply statistical causal inference techniques to: (a) estimate the causal effect of tweets on the number of new GitHub repository stars and new committers; (b) characterize the tweets with the highest impact; and (c) characterize the OSS contributors attracted by these tweets.

Among other results, we find that:

- Tweets have a statistically significant and sizable effect on attracting new *stars* to OSS GitHub projects, estimated at around 11% increase in stars on average for every set of tweets mentioning a repository around the same time.
- The effect of tweets on attracting new *committers* is present but small, around 2% more commit authors on average.
- The effect of tweets on attracting both stars and committers is moderated by multiple factors, including tweet purpose, size of tweet 'burst' (number of tweets mentioning the same repository around the same time), and tweet author affiliation with the OSS projects.
- Newly attracted contributors tend to be more active in the OSS GitHub projects when they have had prior Twitter interaction with the tweet authors.

2.2 Related works

Like all software, OSS also needs a steady supply of development and maintenance effort to remain relevant, of high quality, and secure. In community-driven OSS projects, this effort comes largely from volunteers [81, 180]. And even though OSS as a whole plays important infrastructure roles in our digital economy [72], OSS maintainers' ability to attract, onboard, and retain contributors has generally not kept up with this success. For example, prior work describes how many popular OSS projects are maintained by only one or two developers [5, 12, 51, 223], how project newcomers face a swath of barriers that hinder their first contributions [188, 189], and how many of these newcomer barriers are accentuated by gender [137], which further reduces the available contributor pool. Researchers have also found that high turnover in OSS projects can have negative effects, including knowledge loss [167], longer time to fix issues [82], and decreased software quality [85]. More generally, researchers have studied the internal and external

[1]The title quote was part of one such tweet; see O5 in Figure 2.4 in the Appendix.

factors that contribute to the OSS projects' risk of becoming 'dormant,' 'inactive,' 'unmaintained,' or 'abandoned' [13, 50, 51, 52, 113, 204].

Although sometimes OSS maintainers are overwhelmed with the high volume of requests and demands on their time from users and contributors [73, 164], increased OSS project popularity is generally associated with desirable outcomes [146]. For example, prior work found that popular OSS projects are perceived as having higher quality and better community support [15, 60], tend to be more attractive to new contributors [29, 86, 160], and tend to be more successful at fundraising [155]. That is, they tend to be more sustainable.

Besides the intrinsic quality of the projects or the reputation and influence of their maintainers [27], which can affect project popularity and attractiveness to new contributors, various interventions have also been attempted. Some, like the *signals* providing transparency into otherwise less observable attributes, are relatively subtle, or implicit. Yet they can be effective and are abundant, with many instances being a standard part of the platform UI on social coding platforms like GitHub , or being customizable by project maintainers. For example, prior work has found that NPM packages displaying quality assurance badges on their GitHub READMEs tend to be downloaded more than packages without badges [201]; moreover, adding badges to READMEs seems to encourage projects to update their dependencies [141, 201]. Similarly, the daily activity streak counters that used to be part of the GitHub user profile page UI seemed to steer users towards long, uninterrupted streaks of activity, including arguably unhealthy activity on the weekends, as a recent natural experiment has shown [145]. More generally, prior work has found that developers make rich inferences about each other and the quality of their work based on the signals available on individual profile and repository homepages on the GitHub platform [60, 135] and respond to nudges based on such signals [35, 160].

Other interventions are explicit. For example, GitHub uses an algorithm[2] to identify *trending repositories* for the day/week/month based on their recent growth in activity and popularity metrics, and features the resulting projects on a dedicated page. This can cause the OSS projects to face an "attention shock" [133] with notable effects, including a surge in new contributors. Within their control, maintainers can also actively promote their projects. [31] found in a sample of 96 highly popular OSS projects that being mentioned in highly upvoted posts on the Hacker News aggregator site is associated with a statistically significant increase in the number of GitHub stars in the first three days after the publication date on Hacker News compared to the three days before.

More generally, prior work identified Twitter , blogs, in-person meetings and events, and RSS feeds as the most popular promotion channels for OSS [31]. In particular, Twitter is widely used in the software engineering community [26, 193] for a variety of purposes, including learning about new technologies, staying updated about interesting repositories, and OSS project promotion [32, 76, 182]. However, there are no studies quantitatively evaluating the effectiveness of OSS project promotion on social media, with the exception of the Hacker News study [31] above. Yet, there is evidence outside of OSS that activity on Twitter predicts popularity of offline events and other types of online content, including the online news cycle [44], gross earnings of movies [228], and popularity of academic research [115]. This effect has also been observed on other social media platforms besides Twitter , *e.g.*, Reddit [70] and YouTube [168, 170].

[2] `https://github.blog/2013-08-13-explore-what-is-trending-on-github/`

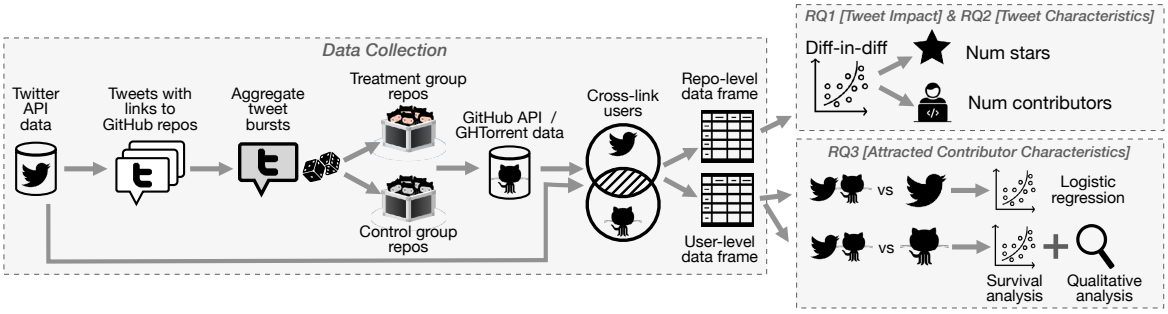


Figure 2.1: Overview of our data collection and analysis steps.

2.3 Research questions

The main goal of this study is to evaluate the effectiveness of tweeting about OSS projects. Focusing on GitHub , the most popular platform for hosting OSS development, we expect that tweets mentioning OSS projects could expand the projects' audience and reach beyond what they already have on GitHub through their watchers [181] or through their maintainers' direct followers [27, 123]. We seek to estimate how much of this extended audience, if any, such tweets are able to attract and convert into stargazers or project contributors, both outcomes which can drive project success and sustainability, as discussed above. We ask:

RQ1. How do tweets mentioning open-source projects impact their popularity and attractiveness to new contributors?

However, social media content, including tweets, are hardly perennial in any user's momentary view of the platform, since they are typically organized as a stream ("news feed"). On Twitter , one's timeline displays a mix of tweets from accounts they follow plus content suggested by the platform based on a variety of signals, *e.g.*, whether someone one follows has interacted with that content. In addition, one's level of engagement with social media, attention span, and ability to absorb a typically overwhelming volume of content are all very limited. It follows that in order to even stand a chance at being effective, tweets mentioning open source projects should at the very least be noticed. This may depend on many factors, including their content and virality, and their authors' Twitter network span. For instance, popular tweets on Twitter are liked by many and may have stronger promotional effects, and tweets with certain hashtags, of different lengths, and with different types of content may also have different effects. It is important to understand the characteristics of successful tweets, if this OSS project promotional mechanism is to be used effectively. We ask:

RQ2. How does the impact of tweets mentioning open-source projects vary with different tweet characteristics?

Finally, we seek to better understand *who* is being attracted by these tweets. Knowing what type of audience and contributors can be attracted via Twitter is important for developers' decision to tweet or not. Prior social connections are known to impact developer engagement and retention [43, 161, 202] in open source. Twitter offers an additional modality for developers to socialize and form connections. In turn, these connections may help explain developers' engagement with the open source projects they discovered via Twitter, *e.g.*, they may motivate

people to contribute more and for a longer period. By cross-linking user accounts across the two platforms (GitHub and Twitter), we can investigate the characteristics of the users who were likely attracted by the tweets mentioning GitHub repositories, both relative to other users on Twitter who were likely exposed to those same tweets, as well as to other GitHub contributors to those same projects. In short, we ask:

RQ3. What are the characteristics of the contributors likely attracted via tweets mentioning open-source projects?

The following section gives an overview of our study design to answer these research questions.

2.4 Methods

2.4.1 Study Overview

We designed and carried out a mixed-methods empirical study, analyzing a novel dataset that integrates data across Twitter and GitHub . Starting from a set of tweets containing links to GitHub repositories, we collect data about two outcomes of interest—project popularity as indicated by the growth in the number of stars and project success in attracting new contributors as indicated by the growth in commit authors. We also collect characteristics of those tweets and their authors, including the tweet authors' ego networks. Finally, we use public information to cross-link user accounts across Twitter and GitHub , and collect additional data about the tweet authors' relationship to the repositories mentioned in their tweets. At a high level (Figure 2.1), our study consists of two main parts. We give a brief overview here and discuss details below.

Part 1: Diff-in-Diff Analysis of the Causal Impact of Tweets (RQ1, RQ2) In the first part, we *mimic an experimental research design* using the observational data we collected, by modeling the differential effect of an intervention on a 'treatment group' versus a 'control group' in a *natural experiment*.

In a true experiment, the random assignment of subjects to one of the two conditions ('treatment' and 'control'), together with the pre-test manipulation of the independent variable under study, is what enables researchers to make causal claims about the nature of the association between the independent and dependent variables, if present. Our study is observational and, therefore, more limited in its ability to support causal claims, compared to a true experimental design. Consider project popularity, one of our two main outcomes of interest, as an example. While we expect that Twitter mentions may help increase the number of GitHub stars projects receive on average, such an increasing star count trend may have already started *before* the Twitter mentions, and for different reasons. Figure 2.2 illustrates this point—the number of new GitHub stars received per day seems to start increasing before the project was first mentioned on Twitter on March 9th. One of the tweets mentioning the repository shortly thereafter (O8 in the Appendix) offers a clue as to why, suggesting a possible in-person event where the repository first started being promoted. Therefore, we are not sure if it is the event itself that *caused* the increase in stars, or if those tweets also played a role in the star increase. Similarly, as discussed above, being featured on the GitHub *trending* page or mentioned on platforms like Reddit, Medium, and Hacker News may have also *caused* the observed increase in stars.

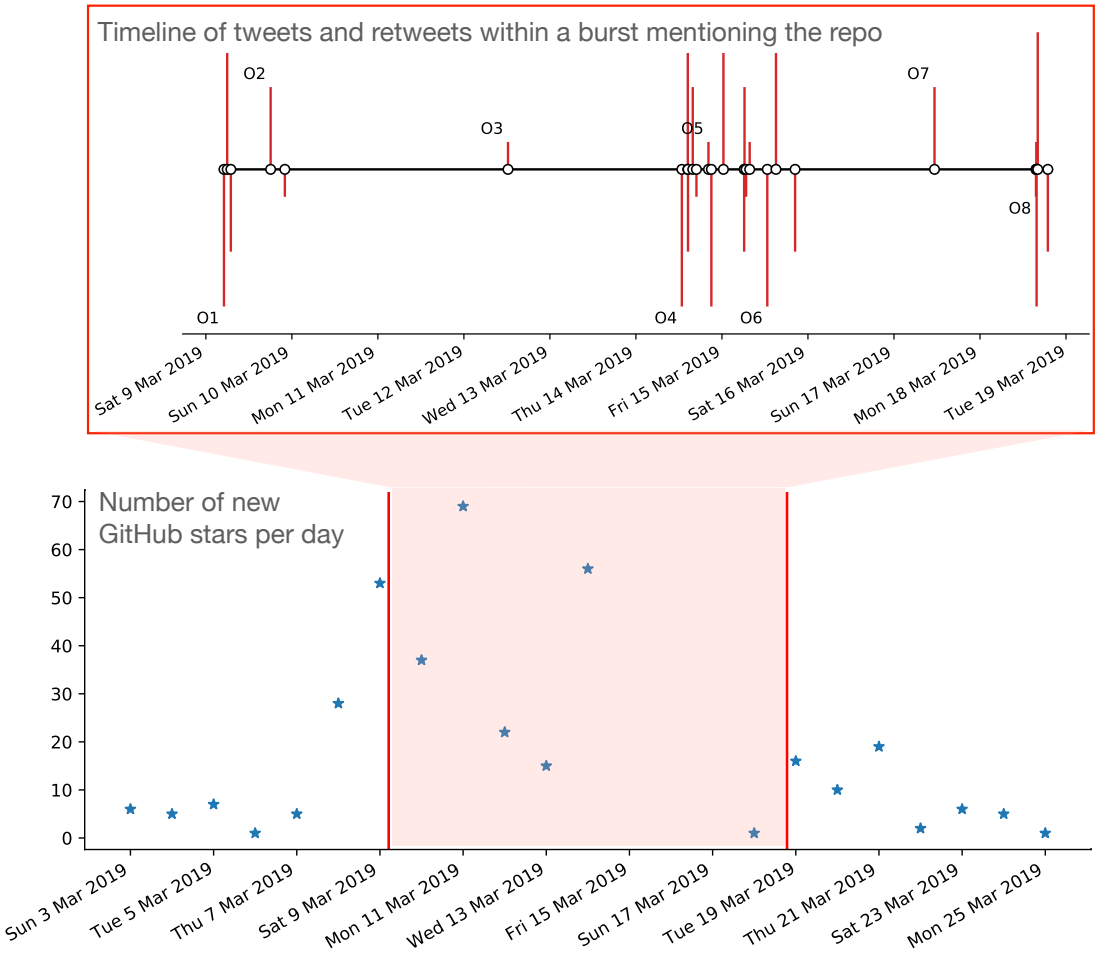


Figure 2.2: The number of new GITHUB stars attracted by the `axa-group/nlp.js` repository per day before, during, and after the Twitter burst in the Appendix. The inset shows the timeline of tweets (labeled) and retweets (unlabeled).

To be able to make causal inferences, the key idea behind our design is to compare not the historical changes in outcome measures before and after the intervention among mentioned repositories, but rather the *difference* in these changes between a group of treated (mentioned) repositories and a carefully selected group of untreated repositories, that acts as a control. The latter is chosen such that the pre-treatment trends in outcomes are similar between the treatment and control groups. That is, the confounding factors before the treatment (*e.g.*, offline in-person event), if present, would have on average affected both treated and control repositories similarly since the pre-treatment trends between the two groups are parallel. Under this assumption, the difference between the observed outcome and the "normal" outcome, *i.e.*, the difference that would still exist if neither group experienced the treatment given the same trend over time in both groups, can be seen as the true effect of the tweets assuming there were no concurrent treatments. In addition, our estimated causal effect is not subject to the influence of any other confounding variables as long as they apply to both treated and control repositories at the similar

level. For example, switching accounts across commits will cause a change in the number of new committers but will not affect the estimated treatment effect if we assume developers' tendency to switch accounts is similar in both treated and control repositories. This design is known as *difference-in-differences* (diff-in-diff) estimation [220].

We use the former analysis to address **RQ1**. To address **RQ2**, we include relevant tweet characteristics as predictors in the same diff-in-diff model, which allows us to estimate their effects.

Part 2: Mixed-Methods Analysis of Who Is Attracted By Tweets (RQ3) In the second part, we report on a mixed-methods qualitative and quantitative case study of a sample of new committers to GitHub projects that were likely attracted by tweets, to better understand when this mechanism can be effective. Quantitatively, we compare developers likely attracted by tweets both to other GitHub project contributors and to others likely exposed to the same tweets but who did not start committing to the GitHub projects. Qualitatively, we analyze instances of past Twitter interaction (*e.g.*, reply to each other's tweets) and GitHub collaboration (*e.g.*, commit to the same repository) to better understand the reasons why those developers may have been attracted.

2.4.2 Preprocessing

We start from the convenience sample of 70,427 GitHub users cross-linked with their Twitter accounts, published by [76]. The authors used two heuristics to identify the GitHub users' likely Twitter profiles, reportedly with 85% accuracy: "(1) mining explicit links to Twitter accounts from [all] GitHub user profile pages; (2) crawling personal websites linked from GitHub user profile pages and mining links to Twitter accounts therein" [76]. We then use the Twitter API to mine all these users' tweets, and identify a total of 331,627 tweets among these that contain links to GitHub artifacts (*e.g.*, issue thread, repository homepage).

Next, we apply a series of filters to de-noise this starting dataset, excluding tweets that mention: (i) more than one repository (more ambiguous effects), (ii) forks rather than main repositories (confounded repository activity metrics), (iii) repositories not recorded in GHTorrent [92] (one of our main sources of data), (iv) repositories with multiple entries in GHTorrent (unclear which entry to choose), (v) repositories with a later recorded creation time in GHTorrent than the tweet itself, (vi) repositories deleted from GitHub (or made private) since the tweet, (vii) repositories without explicit open source licenses.[3]

We then limit our sample to tweets posted between November 1st 2018 and April 30th 2019 because: a) the copy of GHTorrent we had access to ended in May 2019; and b) the Twitter API limits the number of tweets we can obtain from any single user, therefore tweets from highly active users posted further back in the past are less likely to be retrievable (this might bias the sample towards an over-representation of tweets from less active users). The chosen period, six months, is long enough to yield a large dataset for analysis: after this and all of the above filters, we were left with 10,837 tweets mentioning 7,816 distinct repositories.

[3]We used the Open Source Initiative list `https://opensource.org/licenses/alphabetical`

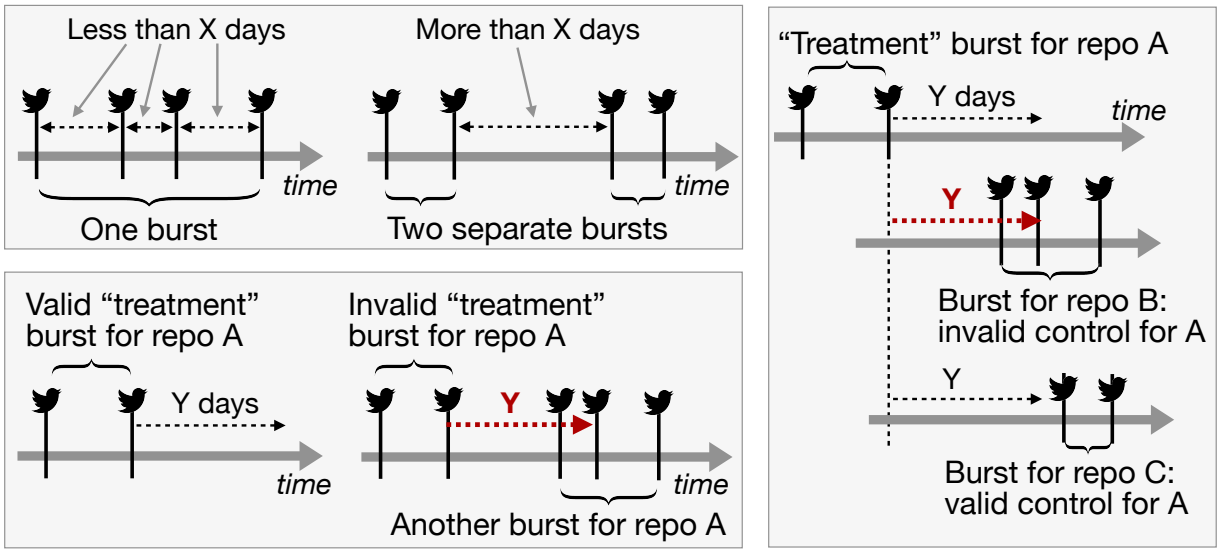


Figure 2.3: Operationalization of tweet *burst* and diff-in-diff data setup. TOP LEFT: Tweets mentioning the same repository within X days of each other are considered part of the same burst. BOTTOM LEFT: Two bursts mentioning the same repository must be at least Y days apart. RIGHT: Control group repositories must not have experienced any bursts of their own at least Y days after the end of the corresponding treatment group repository burst.

However, these 10,837 tweets are only from (a subset of) the 70,427 developers in cross-linked GitHub -Twitter public dataset we started from, while potentially many other people could have tweeted about those same repositories; their tweets would go undetected if they were not part of the cross-linked dataset we started from. To capture all other tweets mentioning those repositories during our observation period, posted by people that were not part of our starting cross-linked user dataset, we further query the Twitter API for all tweets containing links with the format `https://github.com/owner/repo_name`. Note that we exclude replies, *i.e.*, tweets posted explicitly "in reply to" other tweets, as they are generally less visible in one's timeline and therefore expected to have less attraction effect. We do, however, include retweets, for both these and all of the earlier tweets in our dataset.

Finally, we expand the set of heuristics used by [76] to link users across the two platforms. Specifically, we further cross-link users based on: (i) similarity of their display names and usernames / logins, as [24] did originally for commit logs and email archives; (ii) similarity of their profile pictures on GitHub and Twitter —we use average per-channel (RGB) histogram distance between two images for comparison. For validation, we manually checked random samples of GitHub -Twitter user pairs suggested by the heuristics against other public information online. See our replication package for scripts and more details.

2.4.3 Aggregating Tweets into Bursts

Twitter mentions of GitHub repositories may occur closely together in time, as part of coordinated or coincidental *bursts* of social media activity. Taking the example in Figure 2.2, note how

eight tweets by different authors (O1–O8) and many retweets of these, all mentioning the same repository, occurred within a short period.

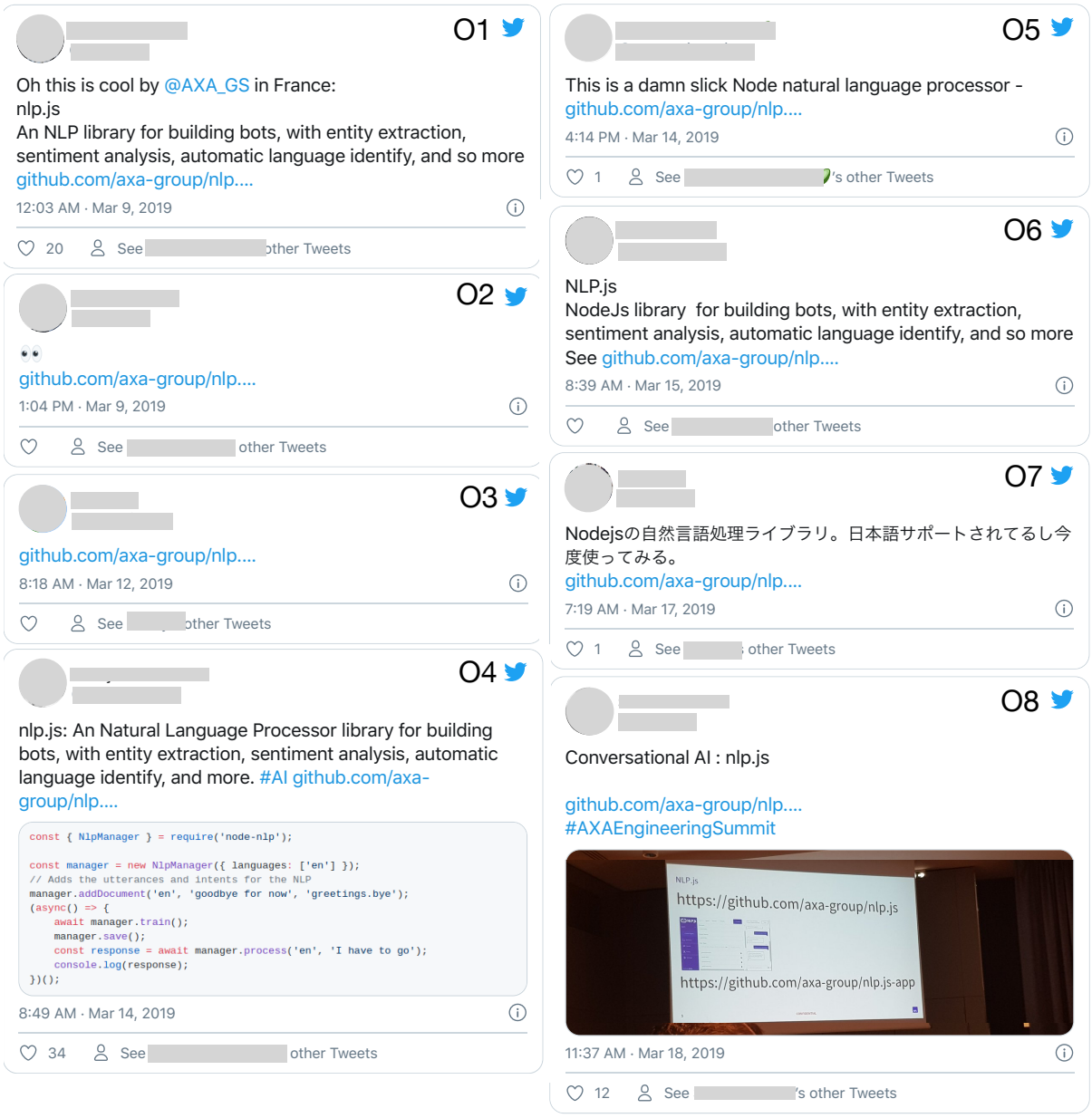


Figure 2.4: Tweets mentioning the `axa-group/nlp.js` repository as part of a burst of social media activity spanning 10 days.

In this case, it is unclear how to reason about which tweet caused a possible increase in GitHub stars shortly thereafter—it could be any subset of those tweets. Rather than reasoning about tweets separately, we therefore first aggregate tweets into *bursts* and then reason about the effect, if any, of a burst as a whole.

There are two important operationalization decisions here. First is *identifying the start and*

end of a burst (top left in Figure 2.3). To this end, we defined a hyper-parameter X representing the maximum allowable time gap (measured in days) between any consecutive two tweets or retweets mentioning a given repository, before they are considered to be part of different bursts.

Second is *dealing with neighboring bursts* (bottom left in Figure 2.3). It may take some time before the effects of a burst, if any, become visible. During this time, it is possible for another burst mentioning the same repository to have started, creating ambiguity about which burst caused those effects. To avoid this, we define a hyper-parameter Y representing the observable effects period after the end of a burst, and conservatively discard all bursts that are "too close" to neighboring bursts, *i.e.*, if there is another burst mentioning the same repository, either started less than Y days after the end, or ended less than Y days before the start of this burst.

This decision has implications also for the control group repositories (more details about how we identified them below). A key assumption behind our diff-in-diff causal inference analysis is that the repositories acting as controls have not been treated, *i.e.*, mentioned in Twitter bursts, around the same time. This implies that we necessarily also require that a potential burst mentioning a control group repository must not have started less than Y days after the end of a burst mentioning the corresponding treatment-group repository (right side of Figure 2.3).

Our results below are computed with $X = 3$ and $Y = 3$ days.[4] We performed sensitivity analyses for $X, Y \in \{3, 7\}$ and found the conclusions after regression analysis to be consistent. Given the chosen values, our resulting dataset contains 6,981 bursts in total (15,975 original tweets and 28,569 retweets), mentioning 2,370 unique repositories.

2.4.4 Compiling a Control Group

As discussed in Section 2.4.1, to test the causal relationship between tweets mentioning GitHub repositories and the number of GitHub stars and the number of new committers, we adopt a diff-in-diff design [1]. Specifically, we consider a burst of tweets mentioning a particular GitHub repository as an *intervention* and contrast the change in the two outcome variables post intervention between the treated repositories and an appropriate control group.

To compile our control group, we adopt a *stacked diff-in-diff* design [91], *i.e.*, we identify suitable control group repositories also *from among the set of repositories mentioned in tweets*. This is possible because even if all repositories considered are mentioned in at least one Twitter burst, *they are not all mentioned at the same time*; thus, for any repository treated at time t, other repositories treated at different times can serve as controls for time t. This approach has several advantages, including an implicit way to control for some confounding factors (potential ways in which the repositories ever mentioned in tweets are different from those never mentioned) and computational efficiency (some variables that need to be computed for the control group will have been computed anyway).

Specifically, we use propensity score matching [40] to sample as controls, for every treated repository, up to five other repositories that were not mentioned in tweets around the same time,[5] but which show *parallel trends* in the outcome variables leading up to the intervention

[4]The attentive reader may notice that in this case the bottom-left condition in Figure 2.3 becomes superfluous, unlike instances when $Y \neq X$.

[5]Recall also the constraints in Section 2.4.3 and Figure 2.3.

time, compared to the treated repository. We chose the 1:5 ratio (as opposed to 1:1) to increase robustness in our statistical conclusions while keeping the modeling sample relatively balanced.

2.4.5 Estimating Tweet Burst Causal Effects and Their Moderators (RQ1, RQ2)

As per Section 2.4.3, our unit of analysis is a tweet burst mentioning a particular GitHub repository. For every such burst, we record its time and mentioned GitHub repository, variables capturing different characteristics of the burst and tweet authors, variables capturing different characteristics of the repository, and measurements of the two dependent variables — number of GitHub stars and number of committers — computed immediately before and Y days after the burst; We also record analogous measures for the up to five control group bursts (repositories) assigned to every treatment group burst (recall Section 2.4.4). We refer to all the pre- and post-treatment observations for a given treatment and corresponding control group bursts as a *cohort*.

Model Specification. To answer **RQ1**, we estimate the regression:

$$O_{itc} = \beta_0 p_{tc} * t_{ic} + \beta_1 p_{tc} * S_{ic} + \beta_2 p_{tc} + \beta_3 t_{ic} + \delta_{tc} + \alpha_{ic} \tag{2.1}$$

Here, O_{itc} represents the outcome variable within the period starting at time t, for a given repository i in cohort c, where a cohort refers to a treated repository and all its corresponding controls. p_{tc} is a flag indicating whether time t is in the pre- or post-treatment period of cohort c, and t_{ic} is a flag indicating whether repository i is in the treatment or control group for cohort c. S_{ic} denotes concurrent non-tweet treatments potentially experienced by the same repository (recall the discussion of the GitHub trending page and similar 'treatments' in Section 2.4.1). Finally, δ_{tc} and α_{ic} are "time-cohort" and "repository-cohort" random effects, necessary given the inherent nested structure of our data—*e.g.*, the same repository may appear as part of different cohorts at different times, violating the independence assumption expected in regression analyses [159]. This specification gives an intuitive interpretation of the estimated coefficient β_0—it is *the average treatment effect of the tweet burst*, since p_{tc} and t_{ic} capture the average difference between the treated and control groups, and between the pre- and post-treatment periods.

To answer **RQ2**, we extend the model to incorporate X_{ic}, the characteristics of the tweets mentioning a given GitHub repository in a given cohort, following a model specification by [68]:

$$O_{itc} = \beta_0 p_{tc} * t_{ic} * X_{ic} + \beta_1 p_{tc} * S_{ic} + \beta_2 p_{tc} + \beta_3 t_{ic} + \delta_{tc} + \alpha_{ic} \tag{2.2}$$

The interaction term $p_{tc} * t_{ic} * X_{ic}$ ensures that the tweet burst characteristics will only affect the value of fitted outcome variables post-treatment. The estimated β_0 should be interpreted as the *moderating effect of those characteristics on the outcome variable*.

Model Estimation. When estimating the regressions, we take the standard precautions (see replication package), including filtering out outliers (top 1% most extreme values) [156], log-transforming variables with skewed distributions to reduce heteroscedasticity [89], and checking for multicollinearity using the variance inflation factor [199]. As indicators of goodness of fit, we report a marginal (R^2_m—the variance explained by the fixed effects alone) and a conditional (R^2_c—fixed and random effects together) coefficient of determination for generalized mixed-effects models [110, 149].

2.4.6 Identifying Characteristics of Developers Likely Attracted By Tweets (RQ3)

For the second part of our study, answering **RQ3**, we focus on developers placing their first contributions to the different GitHub projects soon after the repositories were mentioned in tweets, and who were plausibly attracted by those tweets.

To better understand the underlying mechanisms, we start by qualitatively studying the relationship between attracted developers and Twitter authors. For every repository and tweet burst we extract the authors of commits recorded in the main branch within 30 days of the end of the burst, that had not committed to the project before; both direct push commits and indirect merged pull request commits are captured this way. We then label those new committers as *likely attracted by tweets* if they (i) are part of our Twitter -GitHub cross-linked dataset, (ii) retweeted one of the tweets mentioning the GitHub repository within 3 days after the tweet was posted; and (iii) starred the mentioned repository in the same period. Retweeting ensures the user is exposed to the tweet, while starring the repository after the tweet increases the likelihood that the committer was attracted by the tweet.

With a set of 81 such committers likely attracted by tweets (and corresponding tweet authors who plausibly attracted them), one author coded and analyzed all instances of Twitter interaction and GitHub activity between the tweet authors and attracted committers using thematic analysis. This was an iterative process that involved discussing with another author the different codes and examples thereof, resolving disagreements and recoding, where needed. In each case, we coded on three dimensions: i) the frequency and directionality of past Twitter interactions; ii) the apparent intent of the current tweet and interaction; iii) the frequency of past GitHub collaboration and each other's roles in the respective repositories. The coded interaction is then used to infer the relationship between tweet authors and attracted committers, and try to understand the reason developers are attracted.

Following the qualitative analysis we estimate three regressions. First, we run a logistic regression model to **compare developers who contributed to the GITHUB projects to others who did not**, conditioning on both plausibly having been exposed to the tweets. Here we use past Twitter interaction between the new committers and the tweet authors to identify users exposed to tweets, as users with frequent past interaction are more likely to be exposed to tweets posted by each other.[6] We define interaction as explicit @-mentioning the tweet author and we use at least three past Twitter "interactions" as the threshold of "tweet exposure."

With the outcome variable being whether the exposed developer is attracted as committer (*i.e.*, made a first commit within 30 days after the tweet burst), GitHub collaboration and other developer-level co-variates are included as independent variables and the estimated coefficient reflects the difference between developers attracted versus not. Because of the low number of committers attracted and high volume of "exposed users," we randomly down-sample the exposed users to make the data frame relatively balanced with respect to the outcome variable.

Finally, we estimate regression and survival models to test how the new committers likely attracted by tweets differ from other new contributors during the same period in terms of (i) their

[6]https://help.twitter.com/en/using-twitter/twitter-timeline

Table 2.1: Summaries of diff-in-diff regressions estimating the effect of tweets mentioning GITHUB repositories on attracting new stars.

	Number of new stars (log)		
	Model I	Model II	Model III
Main Treatment Effect			
Is post-treatment	$-0.19(0.01)^{***}$	$-0.19(0.01)^{***}$	$-0.18(0.01)^{***}$
Is treated group?	$0.35(0.01)^{***}$	$0.34(0.01)^{***}$	$0.23(0.02)^{***}$
Is treated group? : Is post-treatment?	$0.11(0.01)^{***}$		
Other Treatments and Controls			
Number of Google search results (log)	$0.27(0.01)^{***}$	$0.25(0.01)^{***}$	$0.26(0.01)^{***}$
Had official release	$0.11(0.02)^{***}$	$0.08(0.02)^{***}$	$0.09(0.03)^{**}$
Is GitHub trending	$0.78(0.05)^{***}$	$0.71(0.05)^{***}$	$0.63(0.07)^{***}$
Burst duration (log)	$0.13(0.00)^{***}$	$0.13(0.00)^{***}$	$0.12(0.00)^{***}$
Tweet Burst Characteristics[1]			
Has hashtags		$-0.04(0.02)^{*}$	$-0.05(0.03)^{*}$
Is promotional		$0.00(0.01)$	$-0.01(0.02)$
Number of likes (log)		$0.05(0.01)^{***}$	$0.08(0.02)^{***}$
Number of original tweets (log)		$0.08(0.02)^{***}$	$0.09(0.03)^{***}$
Number of retweets (log)		$0.08(0.02)^{***}$	$0.07(0.02)^{**}$
Is from committers			$-0.20(0.03)^{***}$
Num. obs.	$65,354$	$65,354$	$32,050$
R^2_m (R^2_c)	$0.08(0.83)$	$0.09(0.83)$	$0.07(0.83)$

Note: $^{*}p < 0.05$; $^{**}p < 0.01$; $^{***}p < 0.001$

total number of commits 30 days after their first contribution (linear regression);[7] and (ii) their **total length of engagement with the project** (Cox proportional-hazards regression)—we follow prior work [140, 161] to detect disengagement as the start of 12 months of inactivity.

2.5 Results

2.5.1 Tweet Effects on Project Popularity and New Contributors (RQ1)

We present a series of nested diff-in-diff regression results answering our first two research questions in Table 2.1 and 2.2.

We begin here by presenting the results for **RQ1** and the number of new stars outcome variable. Model I estimates the average causal effect of tweet bursts mentioning GitHub repositories on the number of new GitHub stars gained within 3 days after the end of the burst. Interpreting the regression results, we first note statistically significant effects for all control variables: having official releases, being featured on the GitHub trending page, and otherwise showing up in Google search results, are all associated with an increase in the number of GitHub stars gained,

[7] See replication package for results with 90, 180, and 360 days.

Table 2.2: Summaries of diff-in-diff regressions estimating the effect of tweets mentioning GITHUB repositories on attracting new committers.

	Number of new committers (log)		
	Model IV	Model V	Model VI
Main Treatment Effect			
Is post-treatment	$-0.08(0.00)^{***}$	$-0.08(0.00)^{***}$	$-0.07(0.01)^{***}$
Is treated group?	$0.01(0.00)^{***}$	$0.01(0.00)^{***}$	$0.01(0.00)$
Is treated group? : Is post-treatment?	$0.02(0.00)^{***}$		
Other Treatments and Controls			
Number of Google search results (log)	$0.10(0.00)^{***}$	$0.09(0.00)^{***}$	$0.10(0.01)^{***}$
Had official release	$0.11(0.01)^{***}$	$0.10(0.01)^{***}$	$0.10(0.01)^{***}$
Is GitHub trending	$0.07(0.02)^{***}$	$0.06(0.02)^{**}$	$0.04(0.03)$
Burst duration (log)	$0.02(0.00)^{***}$	$0.02(0.00)^{***}$	$0.03(0.00)^{***}$
Tweet Burst Characteristics[1]			
Has hashtags		$0.01(0.01)$	$-0.01(0.01)$
Is promotional		$-0.07(0.01)^{***}$	$-0.07(0.01)^{***}$
Number of likes (log)		$0.00(0.00)$	$0.01(0.01)$
Number of original tweets (log)		$0.06(0.01)^{***}$	$0.06(0.01)^{***}$
Number of retweets (log)		$0.01(0.01)$	$0.03(0.01)^{*}$
Is from committers			$-0.00(0.01)$
Num. obs.	$65,354$	$65,354$	$32,050$
R^2_m (R^2_c)	$0.04(0.43)$	$0.05(0.43)$	$0.05(0.44)$

Note: $^{*}p < 0.05$; $^{**}p < 0.01$; $^{***}p < 0.001$

as expected.

Turning to the main treatment effect, captured by the interaction term **Is treated group? * Is post-treatment** as per the model specification in Section 2.4.5, we find a statistically significant positive effect of the tweet burst on the number of GitHub stars gained: on average each tweet burst mentioning the repository corresponds to approximately 11% increase in stars (note the dependent variable is log-scaled, coefficient should be interpretted as the percentage of increase). Considering that the average number of stars gained for treated repositories in the pre-treatment period in our sample is 16.53 and the median is 9, a 11% increase corresponds to around two stars gained via tweets on average, or one star gained via tweets in the majority of cases, *for every tweet burst*. The average effect size of tweet promotion is only 15% of that of GitHub trending page appearance, 43% of Google search appearance (estimated effect being 0.27 if using *Number of Google search results larger than 0?* as independent variable), and similar as having an official project release. Moreover, we note a positive effect of the **Burst duration**—the longer the bursts (and thus the exposure of those tweets), the more stars are gained.

Next we turn to Model IV, which estimates the tweet bursts' effect on the **Number of new committers** to the GitHub projects. Similarly to Model I, we observe a statistically significant positive effect of tweeting about a repository on the number of new committers to the project in the following 3 days post burst. However, compared to the number of stars, the effect on new

committers is considerably smaller—the estimated coefficient of 0.02 for the interaction **Is treated group? * Is post-treatment** corresponds to an average of 2% increase in new committers attracted per tweet burst, which is much less than the effect size of trending page appearance, Google search result appearance and having official release. In absolute terms, considering that the mean number of new committers gained by treated repositories in the pre-burst period is 0.21 (median 0), given the size of our dataset one can expect this effect to translate to only approximately one out of every 250 repositories gaining a new committer as a result of the tweet burst, on average.

2.5.2 Characteristics of Impactful Tweets (RQ2)

Now we turn to Models II-III and V-VI, which add interaction terms between the various tweet burst characteristics and the previous **Is treated group? * Is post-treatment** effect on the **Number of new stars** (Models II-III) and the **Number of new committers** (Models V-VI). The estimated coefficients for these variables can be interpreted as the moderating effect of tweet burst characteristics on attracting new GitHub stars and new committers, since the tweet burst characteristics are only defined for repositories in the treatment group. Recall, since the addition of the **Is from committers** variable requires cross-linking of user accounts across the two platforms, we run Models III and VI on subsets of our dataset, where that information was available (cf. Section 2.4.2).

Interpreting the estimated coefficients, we make the following observations. First, the three variables related to burst popularity in Model II (III) all have statistically significant positive interaction effects with the treatment: the more original tweets, retweets, or likes in a burst, the stronger the effect of the burst on the **Number of new stars**. One can expect that doubling the number of tweets in the burst, either original or retweets, will lead to an average increase of 1.3 stars (*i.e.*, 8%). Model V (VI) confirms the overall moderating effect of tweet burst popularity on the **Number of new committers**, but only in terms of **Number of original tweets**—one can expect that doubling the number of original tweets in a burst can lead to an average of approximately 0.013 new committers per project, or one new committer for every 80 repositories.

Second, we turn to the effect of the **Is promotional** variable, capturing the tweet intent; recall, following [76] we consider tweets pointing to a GitHub repository homepage, as opposed to other targets like issue discussions, as promotional. We observe that the tweet intent has a statistically significant moderating effect on the **Number of new committers** per Model V (VI), but no moderating effect on the **Number of new stars** per Model II (III). That is, while the effect of tweet bursts on attracting new stars does not vary with tweet intent on average, tweet bursts that typically point to issue threads or pull requests (non-promotional per our operationalization) are expected to attract more new committers compared with bursts pointing to repository homepages.

Finally, the affiliation of tweet authors with the respective projects being mentioned in the tweets, *i.e.*, whether the tweet burst **Is from committers**, has a statistically significant moderating effect for the **Number of new stars**—Model III reveals that tweet bursts from project contributors can be expected to attract 20% fewer new stars on average, controlling for the number of tweets in the burst. The effect of the tweet burst on the **Number of new committers** does not vary with the affiliation of tweet authors with the project, per Model VI. The effect of tweet bursts on both outcomes also does not vary considerably with the presence of **Hashtags**.

2.5.3 Characteristics of New Contributors (RQ3)

Recall, we used thematic analysis to characterize the types of relationships between the new committers to the repositories mentioned in tweets and authors of those tweets (see Section 2.4.6). We stopped after qualitatively analyzing a sample of 19 such developers' GitHub and Twitter activity histories since we did not observe any new themes. Across our sample, we observe three main themes:

- **Repeated past Twitter interaction & GitHub collaboration** (6 instances, 32%): The new contributors and the tweet authors appear close socially and they share a long history of collaboration. They have had intense two-way Twitter interactions (*i.e.*, retweeting and replying to each other's tweets), and they usually have also committed to the same other GitHub repositories before as well.
- **Following community leaders or influential developers** (7 instances, 36%): The new contributors appear either interested in a specific project or the work of an influential developer. They posted many tweets or retweets about the project or from the influential developer. In one case, the influential Twitter account was a bot.
- **Weak ties** (6 instances, 32%): The new contributors have little or no past Twitter interaction with the tweet authors and no traces of past collaboration on GitHub , but they tend to follow the tweet authors on Twitter or are in the same broad software community on Twitter , which gives them exposure to each other's tweets.

The qualitative analysis suggests the existence of past social ties plays an important role to attract new committers, but promotion on GitHub repositories can also diffuse through weak ties and reach developers with little prior interaction.

To further characterize the attracted developers, we estimate the three regressions in Table 2.3. Model VII is a logistic regression testing the association between the presence of past GitHub interaction and the likelihood of a developer placing their first commit to a GitHub project mentioned in tweets (*i.e.*,**is attracted**), among developers in the cross-linked dataset who were possibly exposed to those tweets given their past Twitter interactions with the tweet authors. The model shows that developers who collaborated with focal project member in the past are more likely to be attracted (indicated by the positive effect of **Has GitHub collab**). The attracted developers tend to be newer to the GitHub platform (indicated by the negative effect of **GitHub tenure**) but not any more or less experienced otherwise outside of the focal project (no effect of **GitHub commits**); and they tend to be less active on Twitter (negative effect of **Num. tweets**), with more of their activity being original tweets than retweets (positive effect of **Ratio original tweets**). We fit another model with nominal variables representing individual developers, entered as a random effect, to assess the relative effect of individual-level variables. The associations reported above are statistically insignificant in this model, showing that individual-level random effects explain the majority of the variance, indicating whether developers being attracted are more affected by user-level characteristics not included in the model. We suggest including more detailed user-level variables (*e.g.*, the kind of project users have committed to in the past) in future research.

Model VIII summarizes the estimated linear regression testing the association between the short-term activity levels of new project contributors (**30-day commit counts** after their first

Table 2.3: Characterizing the developers attracted by tweets

	Model VII (is attracted)	**Model VIII** (30-day commits)	**Model IX** (disengagement)
Project controls			
Project age (log)		$-0.03\ (0.01)^{**}$	$0.13\ (0.03)^{***}$
Project contribs (log)		$-0.08\ (0.01)^{***}$	$0.11\ (0.03)^{***}$
Project commits (log)		$0.07\ (0.01)^{***}$	$-0.17\ (0.02)^{***}$
Developer Characteristics			
GitHub tenure (log)	$-0.36\ (0.13)^{**}$	$-0.00\ (0.01)$	$-0.02\ (0.04)$
GitHub commits (log)	$0.01\ (0.05)$	$-0.02\ (0.01)^{**}$	$0.01\ (0.01)$
Has GitHub collab.	$1.47\ (0.13)^{***}$	$0.06\ (0.02)^{**}$	$-0.34\ (0.05)^{***}$
Has Twitter interact.		$0.07\ (0.03)^{*}$	$0.01\ (0.07)$
Twitter tenure (log)	$0.15\ (0.10)$		
Num. tweets (log)	$-0.44\ (0.07)^{***}$		
Ratio original tweets	$0.84\ (0.28)^{**}$		
Num. obs.	$1,914$	$2,192$	$2,281$

$^{*}p < 0.05$; $^{**}p < 0.01$; $^{***}p < 0.001$

project commit) and the presence of past Twitter and GitHub interaction and collaboration. Controlling for project age, project size, and overall amount of past GitHub activity outside of the focal project, we observe statistically significant positive effects for both **Has GitHub collaboration** and **Has Twitter interaction**—on average, past connections are associated with higher levels of contribution in the first 30 days.

Finally, Model IX summarizes the Cox proportional hazards survival regression testing how past Twitter and GitHub interaction and collaboration associate with the risk of disengagement from the project (note the reverse coding of the outcome variable—negative estimated coefficients imply a lower risk of disengagement). The model shows that controlling for the same variables as above, prior **GitHub collaboration** with focal project member is associated with lower disengagement risk. However, the model does not reveal any statistically significant effect of past **Twitter interaction**.

2.6 Implications

We now summarize our main results and discuss their implications.

2.6.1 Twitter can be an effective mechanism to popularize open source software projects

Our study provides robust empirical evidence that tweets mentioning GitHub repositories likely *lead to* an increase in the repositories' number of GitHub stars beyond what can be explained by other observable promotional mechanisms such as being featured on the GitHub trending

page or otherwise online. These results suggest that Twitter can be a useful tool for open source maintainers to promote projects, and perhaps even better than other promotional mechanisms, since anyone can tweet at any time, unlike *e.g.*, the GitHub trending page, which maintainers have no control over and which requires the project to be popular already before being featured.

2.6.2 Not all tweets are created equal

Our models suggest that the popularizing effect of tweets varies with different tweet characteristics. First, we find that the more tweets there are in a burst, the stronger the effect of that burst on increasing a project's start count. This reflects, intuitively, the increased exposure and attention the repositories get when mentioned by multiple tweets around the same time. However, we do not find any evidence that the purpose of the tweets matters. Unlike prior work [76] suggesting that tweets that point to issue discussions or pull requests when mentioning the GitHub repositories may have less promotional effect, we find that all tweets help attract attention to the project similarly, irrespective of intent, and some of this attention translates into new GitHub stars. We do, however, find evidence that the effect varies with the tweet authors' affiliation with the GitHub projects—tweets by existing project contributors or maintainers, *i.e.*, project 'insiders,' are seemingly less impactful than tweets by others. One possible explanation is that tweets from project insiders may be considered less objective (self-promotion) and may not be taken to reflect the true value or quality of the project. Another possible explanation is that tweets posted by others may bring this project to a different, wider audience, compared to tweets from project insiders whose Twitter followers may already know this project well. This result suggests that obtaining an 'endorsement' from a trustworthy third-party on Twitter may further benefit the project's popularity, and it also suggests the importance to promote outside one's own social circle. Multiple tweets about a repository by the same set of users may have decreasing value to attract new stars, because most of their audience have already considered the project before.

Tweets can help to attract new committers, but only under certain condition. The effect of tweets mentioning GitHub repositories on attracting new contributors is weaker than for stars. However, it is still statistically significant and *causal* given our study design. Comparing the stars and committers models, the latter indicate that more focused attention is needed. In particular, it seems important that tweets not be generic but rather point to specific repository elements, typically specific issue or pull request discussions, and that tweet bursts contain more original tweets than retweets to increase the magnitude of the effect.

2.6.3 Community engagement on Twitter is important

Our qualitative analysis revealed that of those repositories that do attract new committers, many of them succeeded through the bond of a community or strong interpersonal connections. Either the project itself has a vibrant community on Twitter , with many developers following the activity of the project, and major contributors and administrators of the project managing the online conversation and maintaining connection with other peer developers; or the project's maintainers are highly active and maintain intense communication with other developers on Twitter . Both suggest the importance of maintaining an active Twitter presence as a GitHub

developer and managing the interaction with a set of potential collaborators on Twitter , where the latest information about developers' project can be diffused to them swiftly.

2.6.4 Social connections can be leveraged for work-related tasks, especially short-term ones

Our models of how the new project contributors plausibly attracted by tweets differ from other contributors, on average, confirmed the importance of maintaining ties with past collaborators on GitHub and, in addition, the added value of Twitter social ties. However, comparing the models of short- (commits in the first 30 days) and long-term outcomes (length of engagement with the project), we see weakening impact of the past Twitter connections, long-term. This suggests that open-source maintainers may tap into their Twitter social connections for on-demand, perhaps for help with specific issues, but that these resources are not necessarily sustainable.

2.6.5 Attention can be a double edged sword

The fact that tweets have strong effect to attract user attention, but comparatively lower numbers of new committers also raises concerns to developers tweeting about GitHub repositories. As we mentioned in Section 2.1, while using Twitter as a promotion platform may increase the popularity of a project, it may also simultaneously bring about requests and demands from new users without a proportional amount of contribution input. Future research can further investigate other project outcomes that may be affected by increased attention, such as the number of issue reports. Developers should also consider this factor when they decide to start a promotion campaign for their project on Twitter .

2.6.6 The role of Twitter in open-source development

Comparing with developers who are not evidently attracted by tweets, the plausibly attracted ones are relatively new to GitHub , and they don't post much on Twitter . We hypothesize that they are new developers looking for open-source projects to contribute to so they may not have routine collaborators or the energy and motivation to maintain a strong social media presence. We argue that Twitter is especially important for such developers, since it provides them an opportunity to receive updates or information about GitHub projects at a low cost.

2.6.7 Twitter can be more tightly integrated into code hosting platforms

GitHub is one of the most popular platforms for hosting open-source repositories, and it has various initiatives to promote projects (*e.g.*, trending pages, project spotlights) which can lead to higher popularity of the promoted projects. Given that Twitter seems to be a valuable *exogenous* attention eliciting platform, we suggest that integrating Twitter access into the code hosting platform could help with broadcasting information about those projects more effectively (note that GitHub recently added an explicit Twitter field in user profile pages too[8]). This could be done,

[8] `https://rb.gy/l0bis`

e.g., by adding a 'tweet' button on each project homepage and issue thread page. Providing ready access to Twitter could lead to easy, immediate action for developers to promote their project. This can also be helpful to other non-developer users of the project to discuss issues or promote a certain feature of the repository to their broader social circle. In addition to this, keeping in mind the moderators of the effects of tweets we uncovered, platforms could also provide tweet templates that can incorporate some of these suggestions as soon as a user tries to tweet from the project homepage.

Diff-in-diff can be a useful design in software engineering research. At a higher level, we note that our causal inference research design, while well-established in the social sciences, has hardly ever been used in software engineering research. We hope that our current work will motivate empirical software engineering researchers to consider this and similar causal inference designs more frequently in the future.

2.7 Conclusions

In this paper, we empirically demonstrated that mentioning open-source GitHub repositories on Twitter can *lead to* an increase in project popularity and help to attract new developers. However, the mechanism is not equally effective for the two outcomes: while the effect of tweets on gaining new stars seems to apply to most repositories and kinds of tweets, even less popular ones, tweeting to attract new developers is considerably less effective on average, reflective of the relatively higher bar to placing a technical contribution in an open-source project compared to simply expressing interest in the project by starring it. Still, we argue, there is hope for open-source maintainers, community managers, and evangelists, since *the effect of tweeting on both outcomes is moderated by many factors within one's control*. We conclude, optimistically, that tweeting about open source can contribute to improving open source sustainability.

Chapter 3

Social-technical Networks and its Influence on OSS Sustainability

3.1 Introduction

Much like any software [34], open source also requires considerable effort to develop and maintain, *e.g.*, to respond to evolving user needs [14], unexpected bugs or issues [196], and ever-changing dependencies [28]. However, since much of open source is still being developed and maintained by volunteers [19, 166], for projects, being able to attract and retain new contributors is essential [162]. Attracting and onboarding new contributors preserves the continuity of project development and maintenance, and can even save the project from being abandoned when core developers leave [13]. The arrival of new contributors may also bring new knowledge and perspectives to the project team, and thus help to produce software of higher quality [165]. Yet, open-source projects often struggle to find the right contributors [13] and this is aggravated by the voluntary nature of many open-source contributions — turnover rates are high [85], disengagement of core project developers is common [82], and the reasons people drop out are often unavoidable, *e.g.*, a change of job or life status [140].

There has been considerable prior research to understand the characteristics of projects that succeed in attracting and onboarding new contributors, as well as the barriers faced by people placing their first contributions in open-source projects [188]. For example, there is empirical evidence that project popularity [29], age, size in terms of the number of contributors [187], license [173], the presence and quality of documentation [160], and even activity on social media [77], are all associated with a project's likelihood of attracting new contributors. However, much less is known about how factors external to the project, related to its position and role in the overall open-source ecosystem, impact the process of attracting and retaining new contributors.

Meanwhile, there is mounting evidence that open-source contributors, especially volunteers, have ample freedom to choose which projects to contribute to [90], typically join existing communities over starting new projects [90], and often jump around between projects and programming language ecosystems [53, 106, 207], bringing with them knowledge and skills. Therefore, it is insufficient to consider open-source projects as independent — their success or failure to attract and retain new contributors is likely influenced by the broader context they are part of and their

relationships to other projects in the overall open-source ecosystem.

With this holistic view, in this paper we take a step towards better understanding an open-source project's **labor pool**, defined as *the set of contributors active in the overall open-source ecosystem that a given project could plausibly attempt to recruit from at a given time*. Using longitudinal data from a large sample of 516,893 Python projects, and holding constant project-level characteristics previously reported in the literature, we show that 1) the level of technical skill match between potential contributors and a focal project, 2) the strength of their connection to existing team members from past collaborations, and 3) the amount of 'competition' from other similar projects in the overall ecosystem, are all statistically associated with a project's attractiveness to new contributors.

3.2 Related Works

The sustainability of open-source projects is one of the key questions emerging since the early days of open-source development. Software development is a complex process and takes much effort [34, 144], and the maintenance of such products can be no less costly [22]. Because open-source teams often consist of volunteer developers, whether it is able to produce sustainable contributions becomes especially challenging. Multiple studies have looked at the process of developer involvement in open-source projects. Krogh et al.proposed a "join script" of developer participation in the open-source mailing list and described the practice it took for a newcomer to become established in the community [212]. Crowston and Howison proposed an onion-like structural model of open-source teams, with the transition between different layers being a result of team interaction and developer contribution [55]. Steinmacher and others described the developer contribution to a project as a multi-stage process that started from an initial motivation and attraction phase, then moved to the retention phase in the later stage [187]. The developer's evolvements towards long-term contributors were also studied. Researchers found that the developers' relative sociality [230], their attitudes [231], and the project environmental factors such as the availability of work opportunities and project polarities [231, 232] all play a role to influence the likelihood of long-term contributions. Those works helped to support more sustainable open-source development by providing guidance on tool design [17, 97, 177, 178, 195] and project or task recommendations [174].

The attraction of new developers to contribute to an open-source project is an important part of the project's sustainability and the early stage of developer involvement in the project. Researchers have identified several factors which lead to better developer attraction and facilitate new developer onboarding. For example, Hahn et al.reported that developers tend to join projects of which they have past collaboration experience with the existing developers [99, 100], with a similar result found by [43] as well. Tan and others found that beginner-friendly tasks and their characteristics make a difference on developer onboarding [197]. More recently, multiple studies adopted the signal theory and reported that better README [160], the adoption of badges [201], the project popularity metrics (*e.g.*, the number of stars [29, 86]), and the time to review pull requests [86] all help to make the project more attractive to new contributors.

Our work provides an alternative perspective to understanding the new contributors to a project by shifting the focus from the project itself to the characteristics of potential contributors. Our

result also connects to the research on social media promotion [31, 77] for open-source projects and open-source project diffusion in general [121], as it specifies that the characteristics of the community that the promotion reaches matter to the attraction of new contributors. We further identify several such characteristics that open-source promoters may pay attention to.

3.3 Theoretical Framework and Research Hypotheses

Our study investigates the relationship between characteristics of a project's labor pool and its ability to attract new contributors. As mentioned in the Introduction, we define *labor pool* as the set of developers active in the overall open-source ecosystem around the same time, that a given project could *plausibly* attempt to recruit new contributors from. That is, in our definition labor pools are always *tied to specific projects*. For example, we expect that a text processing project's labor pool is different from a bioinformatics project's labor pool, although the two may overlap if they involve applications of machine learning. In addition, we require a notion of *plausible awareness* of the focal project from developers in the labor pool. Indeed, at the very least, one would need to be aware of a project and possible opportunities to contribute before deciding to do so. We discuss our operationalization later, in Section 3.4.1.

Under this definition, we hypothesize about differences in tendencies to join (*i.e.*, start contributing changes to) a focal project for developers in the labor pool, and the relationships (we theorize the direction of influence as well) between such factors external to a project and the project's success at attracting new contributors. We will operationalize and more formally test these hypotheses below, in Section 4.4. Note also that we use the terms *developer* and *contributor* interchangeably, and inclusively of all types of contribution, not just code.[1]

To begin with, the size of the project's labor pool at some point in time should be an important predictor of the number of new contributors the project engages in the near future. A larger labor pool size should indicate that more developers are aware of the project, thus the possibility that some of those developers will be interested to contribute increases.

Therefore, we hypothesize:

H$_1$. *The number of developers in a project's labor pool is positively associated with the number of new contributors the project receives in the near future.*

For potential contributors in the labor pool, a social connection with current project developers should reduce their uncertainty about the project, as they may trust the information about the project provided by their 'friends' and may be in a better position to evaluate the outcome of contributing [124]. In addition, familiarity with the current project members should help potential developers to understand the working norms and the way of collaboration better [75]. Finally, the focal project's current developers are generally more likely to accept contributions from people they know about [202]. We thus hypothesize:

H$_2$. *The strength of social connections between existing project members and others in the labor pool is positively associated with the number of new developers the project receives in the near future.*

[1]Although in our operationalization below we consider only contributions in the form of commits (including pull request commits), to keep our analysis tractable. Commits can still touch non-code parts of a project, *e.g.*, documentation files.

The development of open-source software is skilled work. To provide valuable contributions to the project, developers need to be familiar with the programming languages, technologies, and coding style in the project [41], and align with the project's goals and other such constraints. The requirement for technical programming skills is often a barrier for new contributors to join a project [189], and the contributions of people with less of a track record of activity in open source are also less likely to be accepted [202]. Going one step further, we hypothesize that it's not enough to have open source contribution experience, but rather that the experience should be technically relevant to the focal project:

H$_3$. *The degree of similarity (or fit) between the technologies used in a project and the technical background of the developers in the project's labor pool is positively associated with the number of new developers the project receives in the future.*

Finally, contributing to open-source projects takes a lot of effort [34, 144]. Given that the amount of time anyone can invest in contributing to open source is unavoidably limited (at the very least by the current laws of physics), one can expect that the total number of projects one can join should be limited as well. And while it is common for people to contribute to multiple open-source projects even during the same day [209], and one's capacity for 'multitasking' across projects increases when the projects share the same programming languages [209], there can still be more projects available than one has capacity for. Therefore, a developer part of the labor pool of multiple projects may be forced to choose from among them.

Stated differently, a developer's tendency to join a project may not only be influenced by the characteristics of the focal project, but also by the amount of other 'competing' projects the developer is exposed to.

As competition for new contributors can exist between projects with overlapping labor pools, we hypothesize:

H$_4$. *A project will engage fewer new contributors in the near future the more overlap there is between its labor pool and labor pools of other projects.*

H$_5$. *The project will receive more new contributors if it is relatively attractive compared to other projects that developers in its labor pool are also possibly exposed to.*

3.4 Methods

3.4.1 Key Study Design Decisions and Tradeoffs

Testing the hypotheses above requires a macro, ecosystem-level analysis. Given a focal project at a given time, we need to operationalize its labor pool of potential contributors from among those people active across the entire open-source 'universe' at that time (**H1**, **H2**). Then, for every such person, we need to make inferences about their past experience with different technologies across the open-source 'universe.' We then need to use these estimates to compute the people's technical fit with the focal project (**H3**), as well as to compute, pairwise, their technical fit with all other available projects in the open-source 'universe', to identify the ones which might be competing for their attention (**H4**, **H5**).

Given the obvious computational complexity of such analysis, we made several tradeoffs between computational feasibility, on the one hand, and realism and statistical power, on the other

hand. This resulted in the following three key study design decisions.

Labor Pool Sampling Frame: The Co-commit Network

We focus on Python projects, and operationalize the labor pool of a focal project at a given time based on relationships between nodes in the project's *collaboration network*. Nodes in the network represent individual developers, and edges indicate that those two developers made commits to a same project in the same time period, *i.e.*, a co-commit relationship. For simplicity, we compute the collaboration network in yearly snapshots, with each snapshot capturing the activity in the respective full calendar year y, *i.e.*, from 1st of January, y to 31st of December, y. We construct the network starting from the set of all contributors to the focal project in year y. Then we expand outwards, to include all their 'collaborators' from all other projects each contributed to in year y. Then, transitively, we collect all *their* 'collaborators' in the same year y and so forth, up to three hops away from the starting set of the focal project's contributors.[2]

The goal here is to approximate the number of developers who might be aware of the focal project at a given time — awareness is a first step towards deciding to contribute. Clearly, there are myriad online and offline ways to learn about new projects, including social media [30, 182], recommendation systems [136], directly following other developers on social coding platforms like GitHub [27], meetings [31], and many other channels [157].

Identifying the exact set of developers who are aware of the project at a given time is impossible. Instead, we use co-committing relationships as a loose proxy for awareness of the focal project. Prior research suggests that the collaboration between developers creates opportunities for communication, and the information about a project is likely to diffuse as a result of developer interactions [6]. Co-committing to a common project does not guarantee direct collaboration or interaction [108], but is a prerequisite for both for contributors who make changes to a project's codebase — at the very least, it provides an *opportunity* for interaction and direct collaboration. Moreover, not all such co-committing relationships will result in information about the focal project being diffused, especially as the number of hops from the focal project's developers increases. However, we expect that, on average, the more such opportunities for interaction exist, the more likely it becomes for information about the focal project to be diffused.

This operationalization also has the advantage of being agnostic to the platform where the source code repository is being hosted, thereby allowing us to more easily scale the analysis beyond any single platform where additional, platform-specific forms of interaction could also be considered, *e.g.*, interactions via issue tracker comments. While currently the dominant one, GitHub is not the only platform for hosting open source.

Labor Pool Operationalization: People One-Hop Away From Current Project Contributors in the Co-commit Network

To further reduce the computational complexity of the analysis, we operationalize the labor pool of a focal project at a given time only as the set of developers *within one hop away* from current project contributors in the collaboration network, who have never contributed to the project before.

[2]Note that all subsequent outcome measures in our models are computed in year $y + 1$, to avoid soundness issues caused by the possibly reversed chronology of the 'collaboration' and focal project joining events.

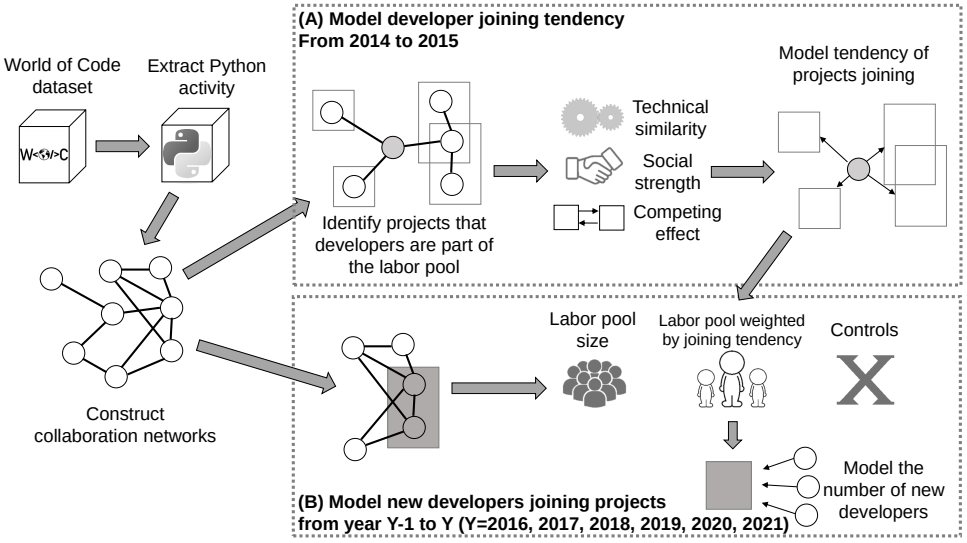


Figure 3.1: The summary of data preparation and analysis process

Information diffusion theory suggests that the closer nodes are to the information source in a network, the more likely they are to receive the information [172, 227].Thus, one can expect that awareness of the focal project should decrease substantially the more hops away one is from the focal project's current developers. Prior software engineering research on the impact of social media posts on open-source project popularity [77], arguably a distant connection, also showed a relatively low tendency to join the focal project in the future for people who saw it mentioned on Twitter.

3.4.2 Overview of the Analysis

Given these key design decisions, we structure our study in two parts, as summarized in Figure 3.1. Both parts involve regression models explaining the tendency and number of new developers joining projects in a next time period as a function of the sets of factors we formulate explicit hypotheses about, via their corresponding variables computed in a current time period. In the first part we take a *developer-centric view* — from the perspective of an individual developer, they typically have a choice of projects they could contribute to and a range of projects they're in the labor pool for, based on past collaborations. In the second part we take a *project-centric view*, aggregating individual-level effects to the level of the whole ecosystem, to reason about project labor pool characteristics and competition effects.

First, we estimate the relative importance of the three sets of factors we formulate explicit hypotheses about[3] — the strength of social connections to existing project members (**H2**), the fit between one's technical background and the focal project (**H3**), and the amount of competition (or choice one has) between available projects with similar technical fit (**H4**, **H5**) — at the individual level. To do this, we start by computing a data frame of (labor-pool-developer, focal-project) pairs, with measurements of the relevant variables (details below) for every developer in a given

[3]Excluding **H1**, which refers to the labor pool size, rather than its composition.

project's labor pool, across all projects in our sample; we also record a binary outcome variable indicating whether or not that developer joined the project in the next period. Using this data, we then construct a logistic regression model explaining the developers' tendency to join a focal project in the next year as a function of the variables of interest; the labor pool is operationalized as described above, *i.e.*, people one-hop away from the focal project's developers. We refer to variations of this logistic regression model (under different specifications) as **individual models**.

Note that the goal here is not to make individual predictions about any one developer's tendency to join a given project in the next time period. Rather, the goal is to estimate the *relative importance* of the three sets of factors of interest, on average, across a large sample, such that we can reuse these 'weights,' *i.e.*, the estimated β coefficients from the logistic regression model, in the second part of our analysis. For example, we estimate how much the technical background fit explains the joining tendency of an *average* developer, compared to the strength of social connections and the amount of competition from other projects, over a large sample. Because we estimate the logistic regression over a very large sample, we can assume that these coefficients are stable,[4] so we estimate only one set of individual models[5] to be used as input for the second part.

Next, we lift[6] the individual-level analysis to the project level by estimating regressions that explain the number of new developers joining projects in a next year as a function of their labor pool characteristics (and control variables) in the current year. We refer to these models as **project models** and we use them to formally test all our hypotheses **H1**–**H5**.

To ensure the robustness of our conclusions, we repeat this analysis for all the complete pairs of consecutive years in our data, from 2015–2016 to 2020–2021. In the end, we quantify the amount of variance that the labor pool characteristics explain when modeling the number of new contributors a project will receive, interpret the results, and discuss the implications of our findings.

3.4.3 Data Collection and Filtering

We mine our data from the World of Code dataset [132], which contains the git commit traces for all public projects hosted on GitHub, Gitlab, Bitbucket, SourceForge, and many other smaller ones. We expect that World of Code should give better coverage of open-source development compared to other datasets typically used in prior research.

To begin with, we define the open-source Python ecosystem as containing all repositories with over 50% of their files written in the Python language. We then apply several filters to de-noise the data, as typical with mining software repositories research [111].

First, we filter out repositories with fewer than 10 commits that involve changes to library import statements, *i.e.*, adding or removing dependencies. This step is needed because we later use this dependency information to characterize the technical needs of projects, *i.e.*, we assume that a project using certain libraries requires contributors with experience in those libraries. We chose the threshold arbitrarily, balancing a desire to retain a large sample, on the one hand, with an attempt to filter out trivial projects (code dumps, homework solutions, etc) and a need for 'enough'

[4]We discuss robustness checks for this assumption below.

[5]We computed all independent variables in 2014 and the outcome variable in 2015.

[6]The estimated β coefficients from the individual models enable this aggregation.

data for the subsequent embeddings-based approach to work. Similarly, we filter out developers from labor pools if they authored fewer than 10 commits that involve changes to library import statements, for analogous reasons. As a robustness validation, we run the same analysis over datasets where projects and authors with less than 100 commits that involve change of packages are removed, and the results are qualitatively similar (See the replication package for validation study).

Second, we made sure to use the de-aliased activity records from the World of Code dataset, which provides both raw data on commit authors as well as data on de-aliased commit authors, after merging developer identities when they use different aliases; see [87] for details on the random forest model used to merge developer aliases based on their user IDs. It is important to use the de-aliased activity records because the volume of developer aliases in such data may skew our measurements of project contributors and experience with Python libraries [8].

Finally, we also make a best effort attempt to filter out bots and unidentifiable accounts together with their associated commit activities, for similar reasons [214]. Specifically, we use three heuristics to identify bot-like accounts. First, we reuse a list of 13,169 bot accounts in the World of Code dataset compiled by [62] after developing a machine learning classifier for this purpose, based on author names, commit messages, files, and projects modified by the suspected bot account. Second, we convert all account usernames into lowercase characters and use string matching to flag as bots those with the last part of their username being *-bot* or *-robot*. Third, we order all developer accounts in our dataset based on the number of commits they made, and we manually evaluate the top 100 accounts. This revealed a few additional bot-like and unidentifiable accounts such as *GitHub Merge Button ¡merge-button@github.com¿*. Overall, all these commit authors are excluded from our analysis.

3.4.4 Part I: Individual Models

As discussed briefly above, we use logistic regression to model the factors associated with the individual tendency to join a focal project, across all (labor-pool-developer, project) pairs in our sample. The full model is specified as:

$$P(J_{ipy+1}) = \mathit{logit}(\beta_0 P_{py} + \beta_1 S_{ipy} + \beta_2 T_{ipy} + \beta_3 C_{ipy}), \tag{3.1}$$

where $P(J_{ipy+1})$ is the likelihood that developer i joins project p in year $y+1$, and independent variables S_{ipy}, T_{ipy}, and C_{ipy} represent the social connection between potential contributor i and the existing developers of project p, the technical background fit between developer i and project p, and the factors relating to the competitive advantage of project p among the set of projects developer i can potentially join, respectively, all computed in year y.

Modeling Considerations

For simplicity, since the estimated β coefficients are stable, we compute only one individual model for $y = 2014$ and reuse the coefficients throughout Part II.

We also restrict our sample only to the labor-pool developers who were active (*i.e.*, made at least one commit) in 2014, because developers who are inactive for more than one year tend to

have a low probability to make commits in future years [39]. Until the end of 2014, there are 104,899 Python developers in our sample who made at least ten valid commits with changes to import statements, and were active in 2014. For each developer, we identify the projects whose labor pools the developer was part of, and model their tendency to join those projects in 2015. Since some developers may be in the labor pools of a large number of projects, for each developer, we randomly sample 30% of the labor pools they are part of. Consequently, we also exclude developers who are part of the labor pools of less than four projects, to ensure that at least one project per person is sampled. In total, we have 47,788 developers and 5,778,144 (developer–project) observations in our sample.

Finally, given the inherently nested structure of our data (the same developer being in the labor pool for multiple projects), we make clustering adjustments in the standard errors at both the project and developer levels to account for the possible within-cluster correlation [2].

Measuring the Technical Fit Between Projects and Developers

The fit between project technical requirements and individual technical background is hypothesized to be an important factor influencing the developer joining behavior. We use the packages (or libraries) a project imports to measure the *technical requirement* of a project, and the packages imported in past code commits of a developer to measure their *technical skills*. While prior research used the programming language as a proxy for technical skills [43], this coarse-grained measure is not suitable for our study as all the projects in our sample are mostly written in Python.

The World of Code dataset contains dependency information extracted from each commit (*i.e.*, the packages that a commit imports).[7] Therefore, we can obtain the packages that a project depends on, and the packages that a developer has used in their past commits. Following [63], we then train a Doc2Vec model to obtain the technical skill embedding of developers and projects. Analogous to a Doc2Vec model in natural language processing [122], we consider the packages to be the tokens, and the developers (projects) to be the document of tokens. We can thus learn a vector embedding of each token (package), and each document (developer and project). The cosine distance between vector embeddings will be small if the two documents are similar in terms of the tokens they contain, or the packages they use. Therefore, the cosine distance between the developer's and the project's embedding is a good measurement of technical skill fit.

Measuring the Project Relative Advantage

To operationalize possible competition between projects over potential contributors, we rank projects, from the perspective of an individual potential contributor, in terms of strength of social connection and technical similarity — we expect that given multiple options, on average, developers will typically choose projects for which they are a better fit. For a potential developer, we obtain all projects whose labor pools the focal developer is part of. For each project, we compute the characteristics that we consider to be influential on the developer joining behavior as hypothesized in Section 3.3. The relative advantage of each project is defined as the percentile of their characteristics, among the set of projects where the focal developer is part of the labor pool.

[7] `https://github.com/woc-hack/tutorial`

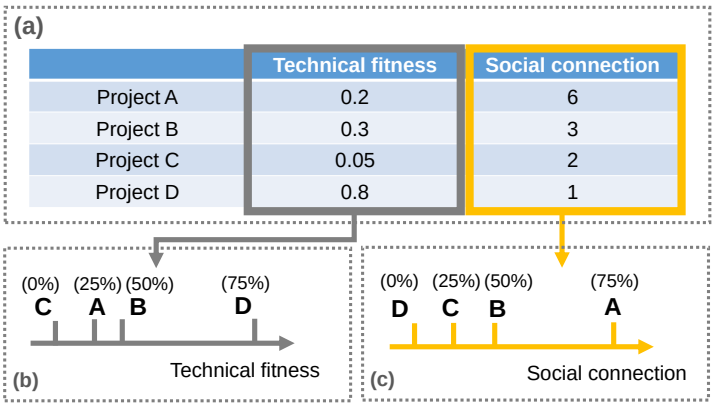


Figure 3.2: Illustration of the project relative advantage

In Figure 3.2 (a), for example, assume that the focal developer is in the labor pool of four projects (*i.e.*, A, B, C, and D). We first order projects based on their technical fit with the focal developer, and the resulting rank (or percentiles, as labeled on top of the project ID in Figure 3.2 (b)) is the relative advantage of projects in terms of their skill fit. Similarly, we compute the relative advantage of projects based on their strength of social connection with the potential developer, as shown in Figure 3.2 (c).

3.4.5 Part II: Project Models

Given the previous individual models estimating the probability of a labor-pool developer to join a focal project from Section 3.4.4, summing over all developers who are in the project's labor pool gives the mathematical expectation of the number of new contributors joining at the project level:

$$P_{pt} = \sum_{i} P(J_{ipt}), \tag{3.2}$$

where P_{py+1} is the tendency of all developers i in the project p's labor pool to join the project in year $y+1$, and $P(J_{ipy+1})$ is the tendency for any individual developer i to join project p, given that developer i is part of the p's labor pool.

Our modeling strategy is hierarchical regression, *i.e.*, we estimate separate individual models that incorporate different sets of factors, ranging from a baseline model with control variables only to a full model that includes all hypothesized factors. Since the aggregation to project level incorporates, therefore, the characteristics of developers in the labor pool, we refer to this outcome variable as the "effective (labor pool) size" in our model. For example, the effective labor pool of a project could be much smaller than the number of one-hop potential contributors would suggest, if the technical fit is relatively low or there is high competition from other projects.

Practically, we regress the number of new developers a project receives in year y. Similarly to the individual model, we restrict our sample to only active projects (*i.e.*, having at least

one commit) in year y, as projects that are inactive for more than one year should have lower probability of attracting new contributors in the next year compared to active projects [161] and we do not want our sample to be biased by the large number of inactive projects which attract no new contributors. New developers are those who made their first ever commit to the project in year y. As before, when measuring the number of new contributors (*i.e.*, the outcome variable), we only count the new contributors who made at least 10 commits involving import statements in the past,

For new project contributors without enough commit history to infer their technical skills, we cannot estimate their 'future' commit tendency, and thus we are unable to predict whether they will join a new project or not. As a robustness check, we have a separate model with the number of all new contributors as the outcome variable, and the influence of labor pool factors is still significant, though they explain less variance, as expected, because the contribution tendency of a large portion of new contributors is not computable; see Section 3.6.5.

The main result of this paper is reported for $y = 2021$, as this was the most recent complete snapshot in our sample and also the largest dataset; the 2021 snapshot contains 516,893 active Python projects. Since the results are consistent for other values of y, *i.e.*, the regression coefficients point in the same direction and have similar scale and statistical significance, we don't discuss the other models in detail but include the results in our replication package.

3.5 Results

3.5.1 Models of Individual Joining Tendency

We first present the result for modeling the developers' tendency to join other projects, conditioning on them being part of the project's labor pool, in Table 3.1.

In model I, we include social and technical variables that correspond to **H2** and **H3**. Both the social connection between potential new developers and the existing developers, and the technical similarity between potential developers' skill sets and the project's technical requirements are statistically significantly and positively associated to the likelihood of individuals joining new projects, which confirms **H2** and **H3**. In model II, we include the competition effects on the basis of model I, as discussed in **H4** and **H5**. **H4** is confirmed, as indicated by the negative effect of variable *Number of competing projects (log)*, which suggests that the more projects a developer was potentially exposed to (the more labor pools they are part of), the lower the likelihood that the developer will join any specific one of them. **H5** was also confirmed by the significant positive coefficient for variable *Social strength percentile* and *Technical similarity percentile*, suggesting that the project's relative advantage among the set of projects that a developer is potentially exposed to also influence the developer's tendency to join the project. Interestingly, that the effect of absolute technical similarity disappears after controlling for the relative similarity effect, suggesting that the relative technical fit among exposed projects are more important to influence developer join behavior compared to the absolute fitness. Compared with model I, model II has a lower AIC (Akaike information criterion) value overall which indicates a better match between the predicted value and the ground truth.

To better analyze the impact of social-technical factors on individual joining tendencies, we

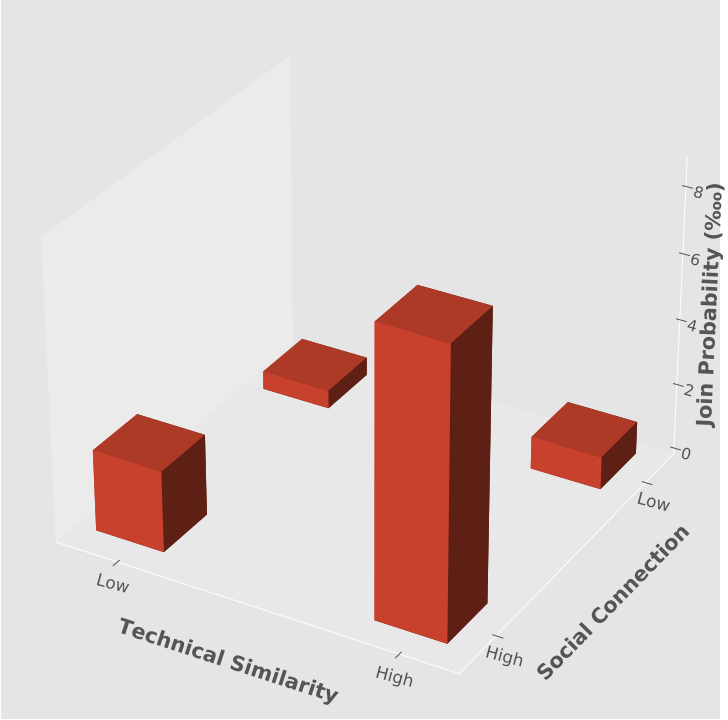


Figure 3.3: Illustration of the social-technical effects on developer joining tendency (y=2021).

examine project-developer pairs where the developer belongs to the project's labor pool and is a potential new contributor. We categorize these pairs into four groups based on whether the developer's technical similarity to the focal project and their social connection to existing developers are above or below the median values observed in our sample.

We then calculate the joining probability for each project-developer pair within each group. Figure 3.3 demonstrates the results. Developers who have strong social connections (top 50%) and high technical similarity (top 50%) with the project exhibit an average of 8.7 new developers joining the focal project per 10,000 project-developer pairs. In contrast, for developers with weak social connections and low technical similarity, this number drops to only 0.6 new developers per 10,000 project-developer pairs.

3.5.2 Models of the Number of New Contributors

Table 3.2 summarizes the models explaining the expected number of new contributors that a project receives in the next year; note the log-scaled outcome variable. Model I is our base model, where we explain the number of new developers a project receives only with the control variables (project-level characteristics). As the model suggests, the current and historical size of the project's developer team is positively associated with the number of future developers that a project receives, which suggests the existence of a preferential attachment effect that developers tend to attach to popular and well-known projects. This results is consistent with the one reported by prior work [173, 204]. In addition, the use of licenses and the project age both have a significant positive effect on the number of new developers. The effect of the current project size (or the

Table 3.1: Modeling the project-joining tendency of developers

	Model I	Model II
Non-competition effects		
Social strength (log)	$1.17(0.02)^{***}$	$0.89(0.03)^{***}$
Technical similarity	$2.03(0.21)^{***}$	$-0.14(0.25)$
Competition effects		
Num. competing projects (log)		$-0.68(0.02)^{***}$
Social strength percentile		$2.03(0.07)^{***}$
Technical similarity percentile		$0.93(0.13)^{***}$
Observations	$5,778,144$	$5,778,144$
Akaike Inf. Crit.	$64,668.72$	$60,465.68$
	$^{*}p < 0.05$; $^{**}p < 0.01$; $^{***}p < 0.001$	

number of commits) is also significant, though it is highly correlated with the historical project size and commits, as expected. Overall, this model explains 12.9% of the variance.

As hypothesized in **H1**, we include the labor pool size variable in model II to explain the number of new developers a project will receive. The size of the project labor pool is positively associated with the number of new developers, and the model explains 14.8% variance in total, or 1.9% more variance than model I.

The simple measurement of the project's labor pool size treats all developers in the labor pool equally, and fails to capture their different tendencies to join. In models III and IV, each developer is weighed differently based on their estimated tendency to join, as computed by the individual model. For the variable *Effective size, no-competition variables*, the estimated joining tendency for each developer is computed with non-competing variables only (or model I in Table 3.1), and the variable *Effective size, all variables* is computed based on both competition and no-competition variables (or model II in Table 3.1). Controlling for the simple measurement of the project's labor pool size, the effective labor pool size is still positively associated with the number of new contributors that a project receives, and it provides additional explanatory and predictive power on the outcome variable. Model III which includes the effective size computed with only non-competing variables explains 14.9% variance, or 0.1% more variance compared to model II, and by adding the effective size computed by both competition and no-competition effects, model IV explains 16.4% variance in total, and 1.5% more than model III. Therefore, we conclude that all hypotheses in Section 3.3 are confirmed at the project level.

3.6 Implications

In this paper, we explored the influence of the projects' labor pool on attracting new developers. We summarize the main results and discuss limitations, the scientific value, and the practical implications of our results below.

Table 3.2: Modeling the number of new developers a project will receive in the next year ($y = 2021$)

	Model I	Model II	Model III	Model IV
Control variables				
Project age (log)	$0.01(0.0004)^{***}$	$0.005(0.0004)^{***}$	$0.005(0.0004)^{***}$	$0.005(0.0004)^{***}$
Project total developer size (log)	$0.07(0.001)^{***}$	$0.06(0.001)^{***}$	$0.06(0.001)^{***}$	$0.06(0.001)^{***}$
Project recent developer size (log)	$0.05(0.001)^{***}$	$0.04(0.001)^{***}$	$0.04(0.001)^{***}$	$0.02(0.001)^{***}$
Project total commits (log)	$0.001(0.0004)^{*}$	$0.0003(0.0004)$	$0.001(0.0004)$	$0.001(0.0004)^{*}$
Project recent commits (log)	$0.01(0.0004)^{***}$	$0.01(0.0004)^{***}$	$0.01(0.0004)^{***}$	$0.01(0.0004)^{***}$
Has readme	$-0.01(0.001)^{***}$	$-0.01(0.001)^{***}$	$-0.01(0.001)^{***}$	$-0.01(0.001)^{***}$
Has license	$0.03(0.001)^{***}$	$0.02(0.001)^{***}$	$0.02(0.001)^{***}$	$0.03(0.001)^{***}$
Labor pool variables				
Labor pool size (log)		$0.02(0.0002)^{***}$	$0.01(0.0002)^{***}$	$0.01(0.0002)^{***}$
Effective size, no-competing variables (log)			$0.10(0.004)^{***}$	
Effective size, full variables (log)				$0.42(0.004)^{***}$
Observations	$516,893$	$516,893$	$516,893$	$516,893$
Adjusted R^2	0.129	0.148	0.149	0.164

Note: $^{*}p < 0.05$; $^{**}p < 0.01$; $^{***}p < 0.001$

3.6.1 Labor Pool as an Important Factor for Project Sustainability

The prevailing empirical studies on open-source sustainability, and attracting new contributors in particular, have focused on the influence of project-level characteristics. In our paper, we provide an alternative, ecosystem-level perspective by suggesting the project's labor pool as an important factor.

The labor pool of a project corresponds to the communities of developers that may be of help to the development and maintenance of the project. They consist of developers who learn about the project through many possible channels, like recommendations from their friends or collaborators, exposure on social media spaces, recommendation systems, web search engines, and many others. They are the contribution resources potentially 'accessible' by the project at a given time.

Considering labor pool factors following a methodology similar to ours can help open-source stakeholders to better understand and predict the amount of contributions projects may have in the future, which helps with the project management and resource allocation. Moreover, users who are seeking sustainable open-source projects to adopt can use labor pool factors to better evaluate the sustainability of candidate projects.

3.6.2 The Size of the Labor Pool Is Important, but It Is Not All That Matters

It is not surprising that expanding the size of the project's labor pool helps to attract new developers, which is why so much effort has been devoted to promoting open-source projects to a larger audience [31, 77, 182]. However, our results indicate that the characteristics of the developers in the project's labor pool, in addition to the raw number of developers, also have a significant effect to explain the number of new developers that a project will receive.

With many previous studies about attracting and retaining open-source contributors focusing on expanding the influence of projects and reaching out to a large community, our work adds to the conventional wisdom by suggesting that expanding project influence to a *proper* audience also matters. In addition, our work is the first, to the best of our knowledge, to provide empirical evidence of the competition effects between projects, and suggests the practice of recruiting new developers is more complex beyond simply reaching out to a large community.

3.6.3 Towards Targeted Project Promotion

For open-source project promoters seeking to attract more attention and effort to their projects, our results suggest that it is not only the number of developers that the promotion reaches that matters, but also the characteristics of those developers. Therefore, a targeted promotion that diffuses the project information to a developer community that possesses the related technical skills and is socially close to the project's existing developers may be more efficient and effective to attract new developers compared to promotions to a wide audience, as they reach a community that is more likely to join the project as contributors. In addition, as the promotion of projects becomes more targeted, developers may be less overwhelmed by the wide range of projects hosted online as the information and promotion they are exposed to will be more relevant to their skills, social connections, and needs. It can reduce the cognitive workload for individual developers and should help to better allocate attention and effort at the open-source ecosystem level [49].

Some project promotion channels or tools are potentially suitable for targeted promotions. For example, project promotion on social media such as Twitter may easily reach socially connected groups as the diffusion of information depends on the social connections; and the project recommendation tools can incorporate technical relatedness and social connections as factors for project recommendations. With most open-source promotion research so far concentrating on the size of developers that a promotion reaches [31, 77], we argue for the importance of studying the characteristics of developers that those promotions reach, and reconsider the value of promotion campaigns based on their effectiveness to reach a targeted audience instead of reaching a large developer community.

3.6.4 The Relationship Between Projects and the Success of Open-Source Ecosystems

The significant competition effects between projects revealed by our models suggest that the attraction of developers to projects is not a local question. Given two projects with overlapping labor pools, any effort to make one project more attractive is likely to have a negative effect on the sustainability of the other, when they are competing for the same effort pool.

This finding raises the important question about the allocation of effort among open-source projects, and the role of individual projects in the success of the open-source ecosystem. While open-source developers and researchers have devoted much effort to making individual projects more successful and sustainable, little was discussed about the influence of those efforts on other projects in the ecosystem. Our result is one of the first works to understand the relationship and competition between projects, and we call upon future research in this direction to further study the sustainability of individual projects jointly with that of the open-source ecosystem.

3.6.5 Validations and Robustness Checks

Validate the Robustness of Our Results Over Years We evaluate the effectiveness of labor pool factors over years and run the same project model for data in three other years (*i.e.*, selecting $y = 2016, 2017, 2018, 2019, 2020$ and 2021). The result suggests that the labor pool factors are important predictors of the new developer attraction for four consecutive years, see our replication package for full results.

Validate the Results With a Negative Binomial Regression Model Negative binomial regression is a powerful tool to model discrete outcome variables such as the number of new developers [103]. To validate the influence of labor pool factors, we use a negative binomial regression model to estimate the number of new developers that a project receives, with all the independent variables the same as the ordinary least squares (OLS) regression used to report the main result; see replication package.

All labor pool factors are still significantly associated with the outcome variable, with the direction of effects qualitatively similar except for the effect from *Effective size, no-competition variables* in model III, which changes from positive in the OLS regression model to the current negative effect. Further analysis shows that the effect is only negative when controlling for the *Labor pool size* variable, but significantly positive otherwise. Despite this inconsistency, the effective labor pool size with the individual model of full variables still shows a significant positive effect, and the Akaike information criterion (AIC) value decreases when adding labor pool variables, which indicates better model fit. Therefore, we conclude the results with negative binomial regression are generally consistent with the OLS regression.

3.7 Conclusions

In this work, we show that an open-source project's labor pool, defined as the set of developers who are possibly aware of the project and may serve as potential future project contributors, is an important factor that can impact project sustainability. We found that adding the labor pool factors, being both the amount and characteristics of developers in a project's labor pool, explains 27% more variance compared to baseline models that only include project characteristics. We also show that the technical fit, the social connection, and the project competition effects are three factors that can affect how developers move between projects. Our work contributes to the scientific understanding of what leads to the attraction of new open-source developers beyond

individual project-level factors, and provides important implications to open-source stakeholders for better project promotion and community management.

Chapter 4

Novelty of OSS Projects

4.1 Introduction

It has long been recognized that open-source software development is an avenue for innovation and creative expression — "how creative a person feels when working on the project is the strongest and most pervasive driver" of participation in open source [120]. Unsurprisingly, we have seen an explosion in production of open-source software, especially in the last decade, with the proliferation of the social coding philosophy [60]. Nowadays, open-source software seems more popular than ever before [64], and it is hard to find any sectors of the economy that do not rely heavily on open-source infrastructure [74].

At the same time, with growing use and ubiquity of open source, there are growing concerns about the maintainability and sustainability of this digital infrastructure [88]. It is not always without barriers for newcomers to join projects [137, 188], turnover rates for open-source contributors are high [112, 140], and even widely-used projects can end up being maintained by a single person or, sometimes, by no one at all [13, 50]. The insufficient maintenance of open-source projects can have disastrous consequences, as prominent security incidents like Heartbleed [213], the Equifax breach [42], and Log4Shell [102] have shown, just to name a few.

These days, significant attention is being paid by policy makers, practitioners, and researchers to understanding open source health and improving open source sustainability, with many open questions remaining around determinants of project success and failure, governance models, procurement and allocation of resources, and others. In this general context we focus on one important but poorly understood concept — innovation. While open source as a whole is a catalyst for innovation [72] (*e.g.*, technology startup companies would have much slower starts without access to open-source infrastructure) and understanding, and being able to identify, innovations in other fields has always been of great interest to investors, governments, etc, we know very little about how innovation emerges at the individual project level and what are its consequences, in either open-source or commercial software development.

Taking one step in this direction, in this paper we begin to study innovation in open source in the Schumpeterian tradition [176] of viewing innovation as emerging from the novel recombination of existing bits of knowledge, a typical perspective in the science of science field [84]. Operationalizing innovation at code level as a function of the libraries and packages a project

imports (see Section 4.3.3) — projects built on top of more *atypical* combinations of libraries are considered to be more innovative — we find that in the Python open-source ecosystem projects with higher levels of innovation tend to be more popular on average, in terms of GitHub star counts. Stated differently, novelty begets popularity, though no main effects are detected in terms of the PyPI package download counts which represents the actual use of projects. At the same time, we find that projects with higher levels of innovation tend to involve smaller teams and are more likely to become abandoned sooner, suggesting that the benefits of increased popularity also carry a cost of limiting the available pool of potential maintainers of a code base that, on average, fewer people may be familiar with.

Based on these results, we argue that innovation and open source sustainability are closely related and, to some extent, antagonistic. With creative expression being such a dominant driver of contributing to open source, as discussed above, one can expect that there will always be an incentive to *create* ever-new open-source software (innovation seems to be rewarded with increased popularity) over doing the work of *maintaining* existing systems through bug fixes and the like, which is often perceived as "grunt work" [104, 130]. This can contribute to exacerbating the resource allocation problem at the global, ecosystem level, by making it even harder to ensure that sufficient maintenance attention (developers, funding, etc) is being allocated to the projects that need it the most.

4.2 Theoretical Framework and Hypothesis

We start by reviewing prior work and developing our hypotheses. Innovation has long been an important but elusive construct in all domains, including the literature reporting on the software industry [71]. Various theoretical models of innovation and innovativeness of firms have been proposed [4, 194], touching on multiple dimensions such as process, product, and organization. And while innovation is often operationalized in terms of patents [134], as of 2020 "there is [still] little consensus on how innovation measurement should be carried out" [23].

Instead, we adopt the Schumpeterian [176] view of innovation as a novel recombination of existing bits of knowledge, and draw inspiration from studies of innovation in research publications and its relationship to scientific impact. In a widely cited paper, [203] analyzed the list of references in research articles indexed by the Web of Science database, measuring the extent to which papers cite atypical versus conventional combinations of prior work as a proxy for how innovative the papers are — more innovative papers are expected to combine existing bits of knowledge, *i.e.*, cite prior papers or publication venues, in novel ways. For example, aggregating at the level of publication venues, a paper that is early to cross disciplinary boundaries would tend to rank as innovative because within a discipline researchers tend to publish in relatively small and disjoint sets of venues. Similarly, a later paper in an established interdisciplinary area would be perceived as less innovative, if there is a history of prior work citing combinations of venues from both disciplines. A similar argument can be made at the individual publication level, instead of aggregating at the level of publication venues. A key finding from this work by [203] is that papers citing some amount of atypical combinations of prior work at publication time "are unusually likely to have high impact," as measured by citation counts to the focal papers over the following years post publication.

By analogy, software is also rarely created from scratch but, rather, by recombining existing bits of functionality in novel ways. These days, there is a wealth of open-source code that developers can reuse, and mature package managers and package registries like PyPI (Python) and npm (JavaScript) to facilitate such reuse. Naturally, not every piece of code using some libraries will be hugely innovative. Conversely, at some level almost all open-source projects are at least somewhat innovative, except in some relatively rare instances of hard forking [233] or code copying [142]. However, similar to the previous argument about scientific innovations, we expect that combining existing software libraries in novel ways indicates, on average, a higher degree of creativity and innovation, and tend to draw much attentions from the community. Consequently, we hypothesize that:

H$_6$. *Open-source projects reusing more atypical combinations of software libraries tend to be more popular.*

At the same time, higher levels of innovation may come at a cost and not every team may be equally positioned and equally able to pursue innovations. In the economics literature it is generally understood that large business organizations tend to be more risk averse when it comes to untested ideas, which may have potential for greater returns if successful, but also carry higher risk of failure [48]. Similar effects are present in scientific research. Indeed, Wu et al.[222] recently showed that it is the smaller teams that most innovate, "disrupt[ing] science and technology with new problems and opportunities."

One can expect a similar effect in open source through at least two complementary mechanisms. One mechanism is related to a similar willingness to experiment and take greater technological risks in smaller teams compared to larger ones, as expected in commercial firms and scientific research. Another mechanism in open source could be related to the availability of appropriately skilled maintainers and contributors — the more atypical a project is in its technology stack, the fewer people may have the relevant knowledge to participate in it. Either way, regardless of the mechanism, we hypothesize that:

H$_7$. *Open-source projects using more atypical combinations of software libraries tend to involve smaller groups of contributors.*

In the long term, having a more atypical technology stack and having smaller teams may both pose a sustainability risk for projects. Except in rare cases of "feature completeness" [204], open-source projects rely on a constant stream of contributors for survival. There is a rich literature on attracting and retaining contributors to open source, exploring a diversity of topics ranging from motivations to participate [90] and barriers to placing a first contribution [137, 188], to engagement [39, 161] and disengagement factors [112, 140]. Similarly, prior research has also tried to explain the factors associated with project abandonment and survival [13, 50, 204]. We add to this literature by exploring the link between project innovativeness and project long-term survival. On the one hand, there is prior evidence suggesting that higher project popularity is associated with higher attractiveness to new contributors [86]; in turn, this should increase a project's chances of long-term survival. On the other hand, while possibly more popular, we expect that more innovative projects will involve smaller teams per **H**$_7$ above and will have a harder time recruiting contributors, because there may be fewer people with the right expertise in their potential contributor pools. In the long term, we hypothesize that:

H$_8$. *Open-source projects reusing more atypical combinations of software libraries tend to be at*

greater risk of abandonment.

4.3 Methods

4.3.1 Data Collection

The main source of data in this study is the *World of Code* [132] dataset. This dataset records the development activities of millions of open-source projects hosted in public git repositories online, including all of GitHub, Gitlab, and BitBucket. World of Code also includes timestamped package dependency information for many programming languages, extracted by parsing import statements, which we use to compute the atypicality of projects' package combinations. This information is usually not available in other open-source software datasets. The closest are package manager dependency datasets (*e.g.*, libraries.io), which contain only within-package-manager dependencies; in World of Code we can observe also dependencies in projects that are not themselves libraries hosted by a package manager registry.

To keep the analysis tractable (our data collection involves aggregation of raw data across platforms and complex, computationally expensive measurement steps), we focus within World of Code only on Python projects hosted on GitHub as the subjects of study. Python is one of the top languages used for open-source projects [25] and supports a wide range of applications and projects of different purposes.[1] It is a large, diverse, and interesting ecosystem. GitHub is the largest platform for hosting open-source development. Therefore, the effects observed in our sample may generalize to other open-source projects and programming language ecosystems, though that remains to be tested.

We consider projects with over 50% of their source code files written in Python as Python projects in the World of Code dataset and we have the complete commit activities for all projects before the end of 2021. To address the problem of including individual projects that are not intended to be used or developed by others (*e.g.*, class projects), we require projects to have at least one release on GitHub to be included in further analysis.

We obtain the commit activities of projects directly from the World of Code dataset, which provides de-aliased developer information through the approach suggested in [87]. Commits from bots are removed following the approaches in [62]. We extract the timestamped GitHub star, release information and data to compute other control variables through the GitHub API.[2]

Lastly, for projects that implement a Python package, we use their number of downloads from the PyPI package manager as the outcome measurement of project usage. We obtain the amount of monthly download data from the public PyPI download statistics,[3] which was collected with Google BigQuery API and the download counts from common mirroring tools are excluded to reflect accurate download behaviors by users.

Overall, there are 75,388 projects left after this procedure, and those projects will be further filtered for each specific analysis.

[1] https://www.python.org/about/apps/

[2] https://docs.github.com/en/graphql

[3] http://www.lesfleursdunormal.fr/static/informatique/pymod_stat_en.html

4.3.2 Pre-processing of package dependencies

We extracted package dependency information from the World of Code dataset by parsing the content of source code files.[4] To ensure accuracy, we de-aliased packages that were imported with different names but provided very similar functions (*e.g.*, *urllib* and *urllib3*). We accomplished this by collecting the package homepage URL from libraries.io, [5] and we merged two packages if they pointed to the same homepage.

To focus on commonly used packages, we removed all packages that were imported by no more than ten projects, as well as Python standard libraries. The latter tend to provide little information about project functions and appear in many projects (e.g., os). We obtained a list of Python standard libraries by crawling the standard library page. [6]

After pre-processing, our dataset consisted of 7,728 packages.

4.3.3 Quantifying Project Atypicality

At a high level, we first compute the atypicality of pairwise package combinations, which reflects the frequency of importing two packages in the same project. We then aggregate the package-level atypicality at the project level by taking the average atypicality of all package combinations in a project. Since the atypicality of package combinations can change over time (e.g., two packages that were previously used separately may become commonly imported in the same project years later), we compute the atypicality of packages and projects for each year.

To measure the pairwise package combination atypicality, we use a variation of the Markov Chain Monte Carlo algorithm and construct simulations of importing events where projects import the same number of packages in the same year, but the choice of packages to import is decided randomly. We compare the empirically observed combination of two packages with the simulating combinations to understand the frequency of using two packages together compared to random chance, with low frequency indicating atypical combinations. Following [203], we compute the atypicality of pairwise package combinations using the following steps.

Obtain timestamped project dependency

The World of Code dataset is utilized in this study to extract information about the packages that projects import. Specifically, only the Python source code and packages are considered, and a list of all the packages that a project imports and the earliest importing time are obtained. Additionally, by analyzing all the commits and the packages they import, the earliest time that a package was imported by any projects in the sample is identified, which indicates how old a package is. Therefore, each project and the package it imports can be represented as a tuple $[P, p, t_P, t_p]$, where P is the project, p is the package, and t_P and t_p represent the time when project P first imports package p and when package p was first imported by any projects, respectively. To simplify the representation, t_P and t_p only record the year instead of the exact time.

[4] https://github.com/woc-hack/tutorial
[5] https://libraries.io/
[6] https://docs.python.org/3/library

Measure the frequency of pairwise package combinations

Based on the timestamped dependency information, we compute the number of times that two packages are imported in the same project. In Figure 4.1, the focal project imports three packages X, Y and Z, with package X and Y being imported in 2016 and package Z in 2017. There are three possible pairwise combination of packages (*i.e.*, X and Y, Y and Z, and X and Z), and we consider the later time that one package was imported into a project as the time when new package combinations appear. For the focal project in Figure 4.1, there are one package combination in 2016 (X-Y, shown in blue), and two package combinations in 2017 (X-Z and Y-Z, shown in red).

For each given year, we aggregate over all projects in our sample and measure the frequency that two packages appear in the same project until that year, which gives us a dynamic measurement of package combinations and its evolution over time.

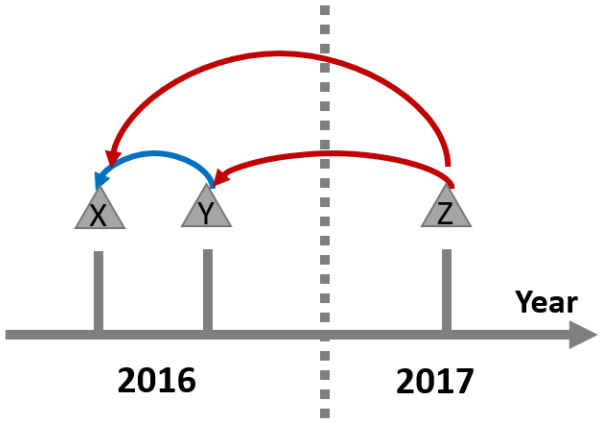


Figure 4.1: Package combinations within a project

Simulation of package importing events

In the description below, we use *importing events* to refer to the importing activities represented by the tuple in Section 4.3.3. In this step, we generate random simulation of importing events which change the packages that a project imports, but preserve the other property of project-package dependency as much as possible through a variation of the Markov Chain Monte Carlo algorithm. *Importing events* refers to the importing activities represented by the tuple in Section 4.3.3, and the process is illustrated in Figure 4.2.

In Figure 4.2, there are three project nodes A, B, and C (project C appears two times), and three package nodes X, Y, Z. A directed edge from one project to a package corresponds to an importing event, where the year that the project node is in represents the earliest time when the project imports the package (or t_P in the tuple), and the year that the package node is in corresponds to t_p and stands for the earliest time that the package was ever imported by any projects. For example, the edge from C (project) to Z (package) indicates that project C imported package Z (first ever imported in 2015) in 2016, and one year later project C imported package Y, which was first ever imported in 2016. For simplicity, we use P-p links to describe an importing event where project P imports package p (*e.g.*, A-X link in Figure 4.2)

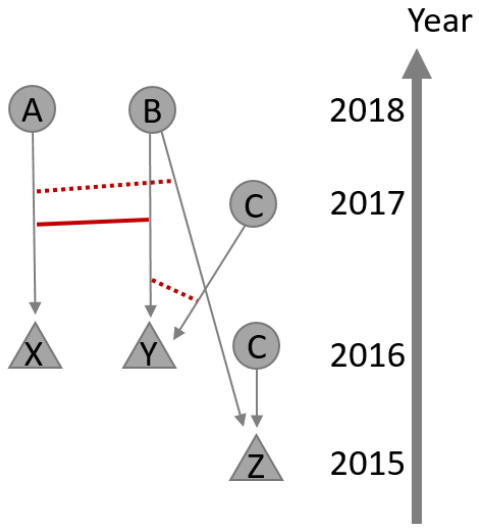


Figure 4.2: Switching package importing events

A random switch of importing events is defined as randomly taking two importing events, and exchange the package being imported between the two events. Notice that A-X and B-Y links exist in Figure 4.2, and a switch between those two links will lead to simulated importing events where project A imports package Y, and project B imports package X (or A-Y and B-X links). This random switch changes the packages that a project uses (and the package combination within a project), but keeps the number of packages that a project imports, and the number of projects that a package was imported in the same.

In addition, because the atypicality of package combinations are computed dynamically, we preserve the longitudinal pattern of imports by restricting the switch to be between importing events of the t_P and t_p. Therefore, the switch between A-X and B-Y links are allowed, because the time of the event are both in 2018, and the imported packages were both firstly imported in 2016. A switch between B-Y and C-Y links will not be allowed, because project B imported the package in 2018 while project C made the import in 2017. Similarly, switching between A-X and B-Z are not allowed because package X was first imported in 2016, which differs from 2015 for package Z.

For each given t_P and t_p, and the set of events selected by the two timestamps, we conduct the described switch within this set of events for enough times to simulate random dependency relationship. There is no guarantee when the Monte Carlo algorithm will converge, but [203] suggests to run the switch $100 * E$ times, where E corresponds to the number of links (or importing events).

Compare the empirically observed events with the simulated ones

We run the same simulation twenty times which results in twenty different randomly generated importing event sets. For each simulated set, we measure the frequency of package combination until a given year similarly to Section 4.3.3. Given twenty simulated event sets and the empirically observed pairwise package combination frequency, we compute a z-score for each package

combination with equation 4.1,

$$z_{ijt} = (obs_{ijt} - exp_{ijt})/(\sigma_{ijt}) \tag{4.1}$$

where obs_{ijt} corresponds to the empirically observed number of times that package i and j appear in the same project until year t, exp_{ijt} is the average number of times (or the expected times) that package i and j appear in the same project until year t over twenty simulated event sets, and σ_{ijt} is the standard deviation of the co-appearance frequency of package i and j in those sets as well.

Intuitively, the z-score represents how many standard deviations more (or less) that the observed combination frequency between two packages are, compared to that expected by random chance. The numeric value of z-score can be quite dispersed, and we transform it with equation 4.2 to make it smooth.

$$Z_{ijt} = \begin{cases} \log(z_{ijt} + 1) & z_{ijt} \geq 0 \\ -\log(-z_{ijt} + 1) & z_{ijt} < 0 \end{cases} \tag{4.2}$$

A high Z-score corresponds to low atypicality.

Aggregate the package combination atypicality at the project level

Given the package combination atypicality (measured by Z-score) until each year, we aggregate it at the project level by taking the average atypicality of all its package combinations, with package atypicality measured in the year when the combination appears in the focal project. Using Figure 4.1 again as an example, the atypicality of the focal project is measured by averaging over atypicality of X-Y combination (measured until 2016), and X-Z, Y-Z combination (both measured until 2017).

4.3.4 Studying the Association between Project Atypicality and Interest from Developers and Users

Regression analysis was conducted to understand the association between project atypicality and the outcome of interest, as described in Section 4.2. Only projects with at least X years (where X is a hyper-parameter) of historical record before 2022-01-01 as selected for analysis. Both the outcome and independent variables were computed based on activities in the first X-year after the projects' first commit. To validate the robustness of our results when modeling the effect of project atypicality on the number of project developers and project stars, we selected X =1, 2, 3, 4 and 5 and report the results for all values of X.

4.3.5 Survival analysis on the sustainability of project and its relationship to atypicality

In addition to measuring the amount of attention a project receives from the community, we also employ a Cox proportional hazards model to conduct survival analysis and understand how project

conventionality influences its long-term sustainability. To do so, we sample projects that received at least one commit in year Y (where Y is a hyper-parameter) and analyze how long they are sustainably maintained before being abandoned and no longer receiving contributions. Following the practice outlined in [161], we consider projects with no commit activity for a period of 12 months to be dormant or abandoned, as the duration between a large majority of two consecutive commits is less than 12 months. As we have commit activity data until the end of 2021, we can detect project abandonment events that occurred before the end of 2020. Projects that are still receiving contributions by the end of this period are labeled as censored and considered to be actively maintained. We use the same control variables and atypicality measurement as in our regression analysis (See section 4.3.4), with the exception that they are computed based on activities until the end of year Y instead of within the first X years of the project's first commit. Additionally, we include the number of commits a project receives in year Y and the number of developers who made at least one commit to the project in year Y as controls to account for recent project activity. We select $Y = 2016, 2018$ and find that the results are qualitatively similar. We report the survival analysis effects with $Y = 2016$ in Section 4.4 with the other results available in the replication package.

4.4 Results

4.4.1 Understanding the Atypicality Measurement

The atypicality measurement, or the Z-score, is the key variable of interest in the study. To validate the effectiveness of this measurement and provide more context, we describe what this measurement captures for both individual package combinations and communities of package connections.

Project Atypicality and Popularity

Table 4.1 shows the results of Model I, which examines the relationship between project atypicality and the number of stars that the project receives. After controlling for the number of commits, the number of developers, project owner identity, and the number of imported packages, we find that the conventionality of a project's package combination is negatively associated with the number of project stars on GitHub.

The conventionality variable is standardized to have a mean of zero and a standard deviation of one. The coefficient indicates that, after controlling for other variables, a one-standard-deviation increase in project conventionality (or a decrease in atypicality) corresponds to a 4% decrease in the number of project stars.

For example, the 25% most atypical projects in our sample have a conventionality that is 0.5 standard deviations below the mean, while the 25% most typical projects have a conventionality that is 0.4 standard deviations above the mean. Given that the mean number of stars for projects in our sample is 118.3 (median 7.0), shifting from the 25% most conventional projects to the 25% most atypical ones leads to an average increase of 4.3 stars.

The orange line in figure 4.4 presents the estimated atypicality effects on project star counts

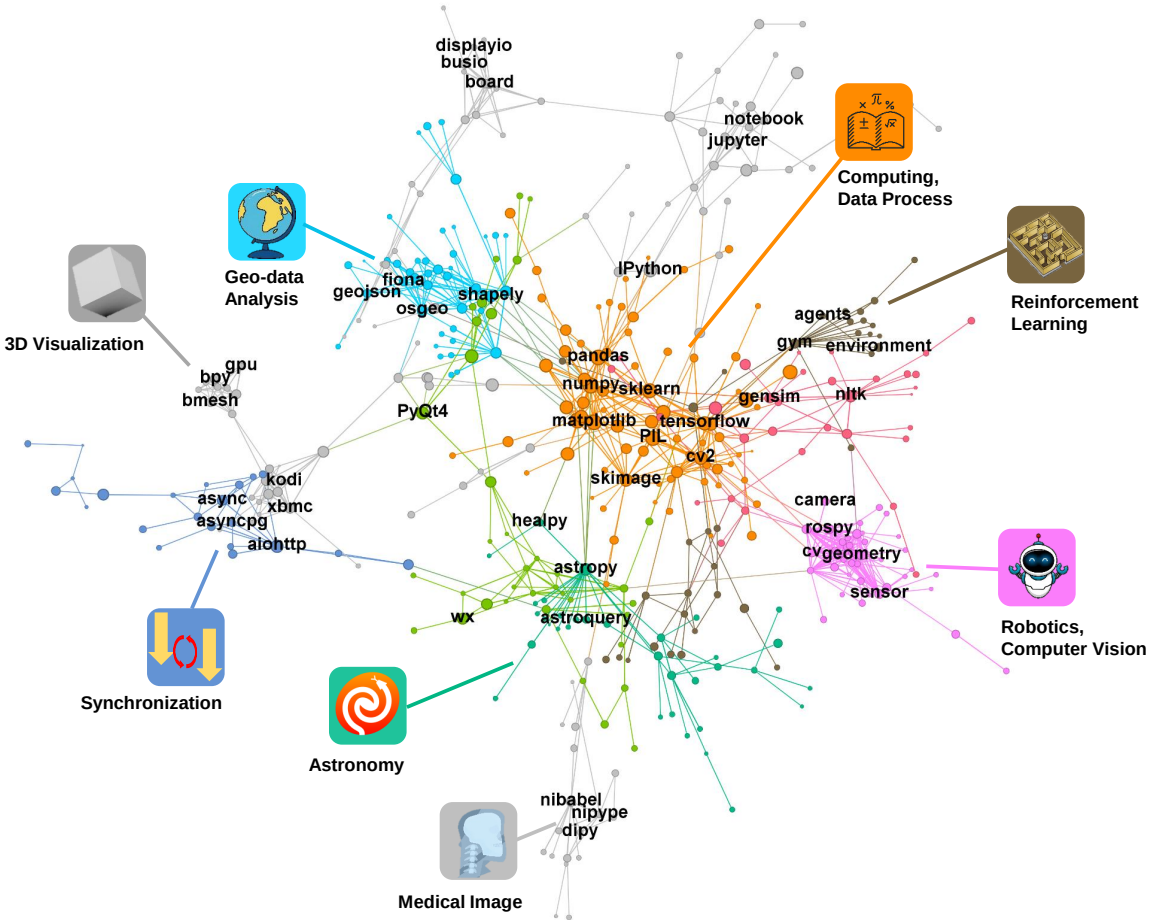


Figure 4.3: Typical combination of Python packages (until 2019)

measured for different length of period, and the estimated coefficient is significantly below zero in most cases, suggesting atypical projects tend to have more stars. The exception appears in the early stage when we only measure the project star counts within one-year after project creation, and atypical projects tend to receive less star during the period. One possible explanation is that atypical projects take longer time to be recognized and appreciated, which is known as *sleeping beauty* effects in the science of science literature [205].

Alternatively, we measure the relationship between project atypicality and the number of downloads that the project receives. In contrast to the effect on star counts and shown in Model II, the estimated coefficient for conventionality indicates a slight position correlation. Given that the p-value is only marginally below 0.05 and the estimated coefficient on download are not significant in all other models measuring different period lengths (See the red line in Figure 4.4), we report no conclusive association between project atypicality and the number of downloads.

Given our results, we partially confirm $\mathbf{H}_6$ by suggesting that the project atypicality associates with higher GitHub star in the long term, but may leads to a lower number of stars in the short run. In addition, we find no consistent association between project atypicality and its download counts.

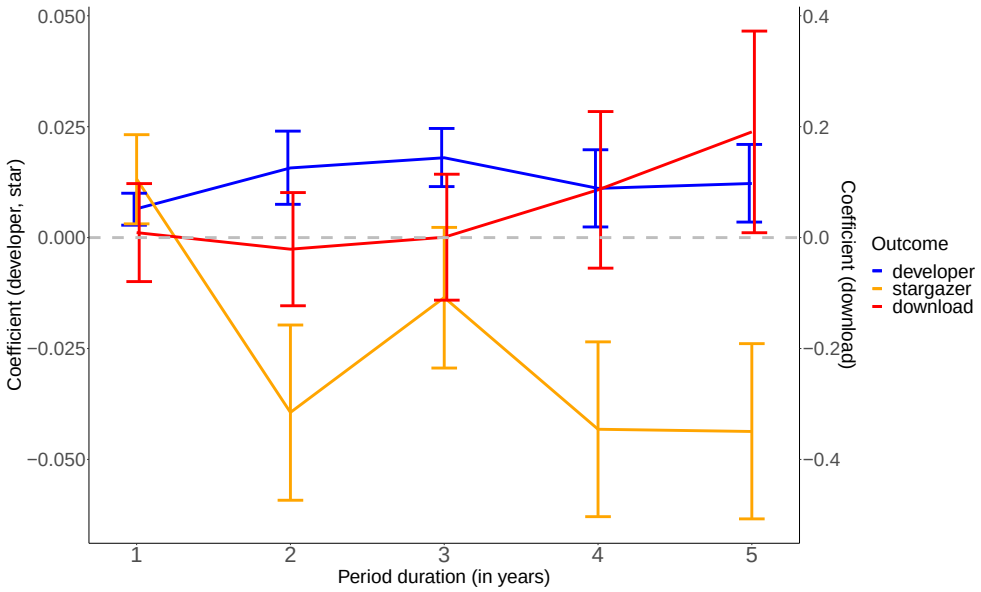


Figure 4.4: Estimated coefficient of Z-score with different outcome variables across different period duration (Error bar represents 95% confidence interval)

4.4.2 Project Atypicality and the Size of Developer Team

Model III, presented in Table 4.1, investigates the relationship between project atypicality and the size of the development team. After controlling for the number of commits, project owner, project popularity, and the number of imported packages, we find that the conventionality of a project's package combination is positively associated with the number of developers working on the project.

The estimated coefficient indicates that, after controlling for other variables, a one-standard-deviation increase in project conventionality (or a decrease in atypicality) corresponds to a 2% increase in the number of developers working on the project. A similar change from the quarter of most conventional projects to the 25% most atypical projects corresponds to a change in conventionality of -0.9 standard deviations, which would lead to a decrease of 0.13 developers per project, or approximately one fewer developer per every eight projects, after controlling for other variables. Therefore, $\mathbf{H}_7$ is partially confirmed by our findings.

4.4.3 Project Atypicality and its Sustained Development

Table 4.2 presents the results of our survival analysis when setting $Y = 2016$. Our findings show that, when controlling for the number of project developers, the number of commits to the project, and the project-level popularity, conventional projects tend to have a lower hazard rate or a lower probability of being abandoned. This is indicated by the significant negative coefficient of the conventionality variable.

Specifically, we found that a one standard deviation increase in the conventionality score changes the probability of project abandonment to 93% of its original value. In this sample, shifting from the 25% most atypical projects to the quarter of most conventional projects would

Table 4.1: Results for regression analysis (X=5)

	Dependent variables		
	Model I (star)	Model II (download)	Model III (developer)
Control variables			
Commit count (log)	0.10^{***} (0.01)	0.04 (0.12)	0.30^{***} (0.005)
Package imported (log)	-0.10^{***} (0.02)	-0.79^{***} (0.16)	-0.03^{***} (0.01)
Developer count (log)	1.36^{***} (0.01)	1.23^{***} (0.11)	
Star count (log)			0.26^{***} (0.003)
Is owned by organization	-0.74^{***} (0.02)	-0.26 (0.20)	0.46^{***} (0.01)
Time in GitHub	1.52^{***} (0.04)		
Time in PyPI		-0.12 (0.08)	
Year of first commit (fixed effect)	X	X	X
Atypicality			
Conventionality (scaled)	-0.04^{***} (0.01)	0.19^{*} (0.09)	0.01^{**} (0.004)
Observations	18,194	1,165	18,194
Adjusted R^2	0.55	0.50	0.69
Note:			*p<0.05; **p<0.01; ***p<0.001

result in a likelihood of project abandonment that is a 93.7% of its original value ($0.93^{0.9}$). Therefore, we confirm $\mathbf{H}_8$.

4.5 Implications

As one of the first research onto the innovation in the open-source software context, our work reveals the impact that atypical combination of packages have to the outcomes related to the success of the project. We summarize the main findings below and discuss the future research directions and potentially impactful implications ahead.

4.5.1 Atypical Projects Draw User Attentions

Innovation is often praised and encouraged as it leads to success in business [152], technology [185], and scientific publications [203]. With atypical combination of packages as one possible way to measure innovation in open-source software context, we are able to show that the atypical projects tend to draw much user attention and receive many stars. We suggest that the pursuit of atypical, or innovative project does come with rewarding outcome in the project popularity.

Table 4.2: The effect of project atypicality on the sustained development (Y=2016)

	Coefficient	Exp(coefficient)
Control variables		
Commit count (log)	-0.08^{***} (0.01)	0.93
Package imported (log)	-0.02 (0.02)	0.98
Developer count(log)	-0.30^{***} (0.03)	0.74
Star count (log)	-0.11^{***} (0.01)	0.90
Is owned by organization	-0.38^{***} (0.03)	0.68
Year of first commit (fixed effect)	X	
Atypicality		
Conventionality (scaled)	-0.07^{***} (0.01)	0.93
Observations	11,803	11,803
Note:		*p<0.05; **p<0.01; ***p<0.001

4.5.2 Atypical Projects Face Difficulty in Development

While developing atypical projects brings unusual level of user attention, it also come with challenges in project development. We observe that atypical projects tend to be developed by a smaller set of developers, which makes the workload on individual developers larger compared to conventional projects as the latter have more contributors to share the workload with.

While projects developed by small teams may enjoy advantages such as reduced communication overhead and easier coordination [114], it also poses great risk to the long-term sustainability of the project as its development and maintenance are only dependent on a few set of developers and the overall status of the project may be strongly influenced by the status change of one or two of those developers [13]. This is indeed what we observe empirically for projects in our analysis, as the conventionality of a project is positively associated with an decreased possibility of project abandonment, or increase in the project sustainability.

4.5.3 The Tension Between Popularity and Participation

There is always the puzzle why many open-source projects, though being popular and widely adopted by the open-source users on the one hand, but were only maintained by essentially a very small set of developers. This puzzle is not only of scientific interest, but also contain important practical implications towards a more sustainable open-source development and the construction of more reliable open-source projects [72].

In our work, we provide a new perspective to explain this puzzling tension between project popularity and developer participation through the measurement of project atypicality and innovation. Empirical evidence suggests that atypical works tend to draw more attention from the users, likely due to its difference from other packages available to the open-source community, but on the other hand are usually developed by smaller set of developers which lead to problems in its sustainability, with higher barrier and higher uncertainty of contribution as

possible mechanisms. Following this observation, new questions can be asked on how to reduce the seemingly negative consequences of atypical combinations (fewer contributors with relevant expertise) while maintaining the positive consequences (higher popularity)? And how to reduce the long-term abandonment risk of open-source projects through the identification of atypical projects and understanding the mechanism for why the tension between project popularity and developer participation exists in the first place.

4.5.4 Future Work: The Context Surrounding Innovativeness

Our paper suggests the possibility of measuring innovation in open-source context, and associates it with the commonly-used metrics for open source software success. This opens up the discussion about how innovation happens in open-source context? What developers or teams are most likely to produce innovative projects and what are their characteristics? And how can we support innovative teams and address the challenges faced by emerging innovations? Those questions provide important practical guidance on open-source sponsors about how to allocate their resources to better support open-source ecosystems, to open-source users and adopters on the choice of sustainable and reliable projects, and to open-source developers who work to create and maintain open-source projects and help them better decide how to distribute their time and resources into different projects.

4.5.5 Future Work: Atypicality and Measurements of Innovation

Measuring innovation has been a challenge in almost every field that it was touched. Borrowing the established measurement from other fields, particularly the study on scientific publication and new knowledge discovery [203], we use the atypicality of package combinations as a measurement of project innovation.

Still, we do not yet know to what extent it captures the concept of innovation in open-source ecosystems, and how do we validate the effectiveness of this, or any form of innovation measurement in open-source context. We consider this question being the building blocks for many future research into the innovation in open-source and we call upon more efforts into the discussion of measurement and the development of validation approaches and datasets.

4.6 Conclusions

In this project, we build a theoretical framework around innovation in open-source software and how would it relate to project popularity and sustainable development. We suggest a measurement of innovation based on atypicality of package combinations, and explain the scientific intuition behind and provide empirical explanation about what does it captures. We found that innovative projects tend to draw much attention among open source users and developers, especially in the long term, but at the same time also be developed by smaller teams and face greater challenges on project sustainability. This observation connects the project popularity and developer participation together and provides a plausible explanation on the tension between the two. Our work opens up the discussion about the role of innovation in open source and it comes with great practical

implications about how to boost the commercial success of projects while at the same time make it sustainable. The measurement, theoretical framework, and the empirical association observed from our works contribute to both the scientific understanding of open-source software ecosystem but also provides implications to open-source practitioners.

Chapter 5

The Strength of Weak Ties in Open-Source Software Innovation

5.1 Introduction

Innovation holds a prominent position in the literature of science and technology development. Innovative products and ideas distinguish themselves from incremental improvements by introducing profound and radical changes. These innovations possess the potential not only to generate significant impact but also to disrupt established competencies, leading to discontinuous advancements in the field [210].

Innovation plays a crucial role in open-source software (OSS) development. Within the OSS community, developers select tasks and projects based on personal interest, and the level of creativity they experience while working on these projects stands as one of the most influential and pervasive drivers of participation in open-source initiatives [120]. Furthermore, numerous organizations adopt open-source software as a manifestation of open innovation. This approach allows them to explore innovative software development opportunities through interactions with users and developers beyond their organizational boundaries [211, 215].

The prevailing discourse within the scholarly literature on scientific and technological innovation asserts that novel advancements often arise from preexisting components of knowledge [203], with the reconfiguration of knowledge serving as a predominant avenue towards innovation [176]. Within the realm of software development, the amalgamation of programming knowledge forms the cornerstone of any software project. While certain combinations of knowledge may be conventional and frequently utilized, resulting in projects that bear little divergence from existing ones, others entail the fusion of rarely associated knowledge domains, thereby yielding more innovative outcomes. Recent empirical studies adopted the integration of novel combinations of software packages as a metric for assessing the degree of innovation within projects, and they reveal a significant variance in novelty across Python open-source projects [79].

The question arises: what elements contribute to the differing degrees of innovation observed in open-source projects? Considering innovation in open-source software development as the implementation of fresh combinations of knowledge, we propose and empirically assess the theory that developers' access to a broad spectrum of knowledge, information, and skills likely plays a

crucial role in fostering innovation.

In the open-source context, developers acquire new skills and knowledge through interactions with peers and engagement with project artifacts [226]. These interactions manifest in various forms, spanning from direct contributions to projects—which often entail coordination with other developers and a profound understanding of the project's technical intricacies [56]—to discussions on project issue reports [10], and passive reviews of projects without explicit actions. Through those interactions, developers form social relationships between each other, and they learn about new programming skills from their peers involved in the interactions [29, 107].

Scholars in the domain of network science have delved into the transmission of knowledge via social interactions and the resultant social ties, examining their significance in catalyzing the emergence of innovative ideas and products. Specifically, they have identified that the nature of social connections facilitates knowledge exchange across different scales and types [94, 116]. A significant focus has been on the "strength" of social relationships and the knowledge exchanged through these interactions. Researchers have defined individuals with frequent and intimate interactions as having strong ties, whereas those with less frequent or intimate interactions are considered to have weak ties [94]. Theoretical frameworks suggest that weak ties often bridge individuals from different communities, thereby facilitating the diffusion of novel knowledge to the recipients [38, 93, 94].

Building on this theoretical framework, we establish three distinct networks that portray developer interactions, spanning from 'strong' to 'weak' interactions: direct code commits, issue posts, and project stars. These networks function as surrogates for the knowledge accessible to developers via social interactions of varying intensities. We then conduct an empirical analysis to investigate the correlation between the access to knowledge through these interaction networks and the innovativeness of the projects created by contributors within the open-source community.

Our findings indicate that the diversity of projects accessed by developers through the three networks positively predicts the novelty of the projects they subsequently create, whereas the volume of accessed projects does not show a significant correlation with project novelty. Moreover, we observe that the diversity of projects and knowledge accessed through weak interactions, such as simply previewing code as indicated by stars, has a greater impact on project novelty compared to interactions involving stronger engagements like comments and commits.

This study contributes to the theoretical comprehension of novelty and diversity, and highlighting the role of passive participation within open-source software ecosystems. Furthermore, our research offers practical insights that can help enhance open-source innovation for developers, open-source platforms, and tool designers alike.

5.2 Theoretical Frameworks and Hypothesizes

5.2.1 Social-Technical Interactions, Knowledge Diffusion and Innovation

Knowledge is a fundamental component of innovation, as it enables the integration of existing knowledge and the creation of new insights [67]. This knowledge is acquired and diffused through interactions with other individuals [101] or through engagement with artifacts, often created by humans [46].

In the context of open-source software, Knowledge about programming and project design is disseminated through developers' conversations and coordination efforts. For instance, technical discussions among developers involve sharing knowledge and absorbing insights from others, contributing to a collective pool of expertise [212]. Furthermore, knowledge acquisition can occur passively through interactions with technical artifacts like reviewing code segments and online discussions, which are readily accessible in open-source environments [18, 60].

Generally speaking, an increase in interactions among developers corresponds to a higher amount of knowledge transfer, thus better preparing developers for future innovation. Therefore, we hypothesize that:

H$_9$. *The more interactions developers have with other developers and projects, the more innovative they will be when developing their own projects.*

In our study, we define innovation as the novel combination of software packages, following the approach taken by Fang et al. [79]. From this perspective, the diversity of knowledge accessible to developers appears crucial for the level of innovation. The specific knowledge developers gain from interacting with individuals of similar technical and social backgrounds can often be redundant, offering limited contributions to novel combinations of existing knowledge. We anticipate that this effect would also extend to interactions with technical artifacts. Therefore, we hypothesize that:

H$_{10}$. *The greater the diversity of interactions among developers and with technical artifacts, the higher the level of innovation observed in the development of their own projects.*

5.2.2 The Differed Tie Strength and their Effects on Innovation

Sociologists have extensively investigated the social networks among individuals and their consequential influence on social dynamics. A fundamental concept that distinguishes various social connections is the notion of tie strength, which encompasses factors such as the duration of shared interactions, emotional depth, level of intimacy, and reciprocal exchanges [94]. Weak ties typically involve infrequent and less intimate interactions, while strong ties are characterized by higher frequency and intimacy.

As posited by Granovetter, strong ties tend to develop among individuals within the same community, while weak ties are more common between individuals from different communities. Here, a community is defined as a group of people who are predominantly interconnected with each other. This pattern arises because individuals within the same community have numerous opportunities for interaction and often share many mutual friends. This shared social context leads to increased similarity in behaviors, interests, and other characteristics, thereby enhancing intimacy among individuals [94].

Furthermore, Heider's Cognitive Balance Theory proposes that when two individuals are linked only by weak ties yet share many common friends, it creates a "psychological strain." Such situations, characterized by cognitive imbalance, are typically avoided by individuals [147]. Therefore, weak ties between individuals sharing many mutual friends or belonging to the same community are unlikely to persist over time [94].

The bridging role of weak ties between people of different communities offering great advantages for the diffusion of novel information. Within the confines of a single community, individuals

tend to become more similar through frequent social interactions, leading to the consumption and circulation of similar types of information. Consequently, the information disseminated with the strong ties connecting individuals of a single community often contains less novel content, and at times, may even be redundant, rendering it less valuable overall. In contrast, information acquired through weak ties, or cross-community connections, is generated by individuals with diverse backgrounds and knowledge bases. This information is often unavailable within one's own community, making it more likely to be novel and of greater value [94].

The advantage of weak ties in accessing novel and valuable information has been empirically evaluated across various contexts, including job seeking [93, 163], generation of innovative ideas [38], and the diffusion of news feed content [16].

Therefore, the knowledge transferred through weak ties should encompass more information that is valuable to the creation of novel projects, and thus the amount, and diversity of knowledge transferred through weak ties should have a stronger effect compared to the strong ties.

$\mathbf{H}_{11}$. *More interactions with technical artifacts through weak ties is linked to higher levels of innovation observed in the development of their own projects.*

$\mathbf{H}_{12}$. *Greater diversity in interactions with technical artifacts through weak ties is linked to higher levels of innovation observed in the development of their own projects.*

5.3 Related Works

5.3.1 Innovation in open-source software

Innovation constitutes a fundamental concept within the realm of open-source software. Early investigations into the motivational factors driving open-source software developers indicate that the perceived level of creativity experienced during engagement with open-source projects stands out as a primary catalyst for participation in such initiatives [117, 120]. Furthermore, the pursuit of innovative software solutions incentivizes organizations and enterprises to allocate resources towards open-source software development. This is attributed to the opportunities it affords for novel software development through interactions with the diverse user and developer base within the open-source community. Such interactions provide access to resources external to the organizations, which would otherwise remain inaccessible [126, 211, 215].

For an extended duration, quantifying innovation in open-source software projects has presented a complex challenge. Among the pioneering contributions in this domain is the work of Couger and Dengate, who advocated for assessing the creativity of software products through their novelty, denoting the degree of uniqueness relative to existing solutions, and utility, reflecting their quality and impact [54]. Subsequently, Beghelli and Jones advanced this discourse by introducing a more comprehensive framework for gauging innovation, considering distinctions in functionality, aesthetics, and other pertinent features [21]. Additionally, Kuzmickajaet al.conducted notable research wherein they employed a 5-point Likert scale for direct evaluation of the novelty of ideas underlying mobile services, utilizing human evaluators for the assessment [119].

The prevailing approaches to measuring innovation predominantly rely on human evaluations or criteria that are challenging to operationalize, thereby impeding the assessment of software innovation on a large scale. Consequently, quantitative methods for evaluating software innovation

remain relatively scarce and often exhibit coarse granularity. For instance, Donget al.conducted an empirical investigation into the pace of innovation within open-source software, utilizing the number of software releases as a proxy for innovation speed without differentiating between individual releases [66]. More recently, Fang and colleagues introduced a novel approach, utilizing the unique combinations of software packages as an indicator of software novelty [79]. This method offers a scalable computational measure of novelty, supported by a robust theoretical foundation within the innovation literature. In this paper, we adopt this approach to measuring software innovation.

5.3.2 Social Ties, Knowledge Flow and Innovation

Participation in open-source software (OSS) development constitutes a social learning process [55], wherein social ties among OSS developers serve as channels for information dissemination and knowledge exchange. Developers acquire knowledge about emerging tools and projects by monitoring trends in the OSS community [182]. Furthermore, they gain insights into software design principles and programming skills through the review of code authored by peers [151]. Additionally, interactions with project maintainers facilitate learning about software management practices, as well as specific bug fixes or feature implementations [45].

The knowledge accessible to developers influences their decisions and practices in software development. For example, Singhet al.and Penget al.found that developers are more likely to choose an open-source license that is adopted by other projects to which they are socially connected [158, 183]. Similarly, Hemanket al.revealed that developers are more likely to adopt an automated tool in project development if they are connected to other developers who adopt the same tool. This effect is observed both for collaboration ties and project watching connections among developers [121].

Accessing diverse knowledge acts as a catalyst for innovation, allowing individuals to integrate disparate concepts and develop unconventional and novel products [175]. Tortorielloet al.observed a positive association between employees' access to diverse knowledge within the research departments of high-tech companies and their innovation levels [200]. Similarly, Basit and Medase found that the integration of knowledge from both internal research teams and customers in the public sector enhances innovation [3].

5.3.3 Social Network Analysis in Open Source Software

Social network analysis (SNA) has been a prominent methodology employed by researchers in the field of open-source software (OSS) to study social relationships. Early contributions by Crowston and Howison investigated the diverse structures of developer communication networks within bug tracking systems, revealing significant variations in communication networks in terms of activity centrality and interaction modularity [55]. Subsequent studies have further examined the evolution of communication network structures in OSS projects [127] and investigated how different network structures correlate with the effectiveness of bug fixes and overall project success [128, 221].

Another line of work examines the collaboration networks among developers, considering them as conduits for knowledge exchange and resource sharing. Research in this area investigates

the association between a project's or developer's network position and their success. Grewal et al.found that the number of commits received by a project is associated with both the network position of the project and its managers [95]. Similarly, Fershtman and Gandal identified a relationship between a project's closeness centrality in the project-by-project network and its likelihood of being selected as the "project of the month" [83].

Tan, Singh, and their colleagues adopted social capital theory to empirically evaluate multiple network centrality metrics and their associations with a project's popularity and productivity [184, 198]. Furthermore, networks can also elucidate developer activities. For example, Qiu et al. found that social networks influence the sustained engagement of OSS developers, with effects varying among individuals of different genders [161].

The study of developer social networks extends beyond those formed on code hosting platforms and is not limited to direct interactions. For instance, Yang et al.examined the developer follower network on the Ohloh platform and investigated its association with project success [225]. Additionally, Hu et al.analyzed developers' star, fork, and follow activities to quantify their influence [105].

5.4 Methods

5.4.1 Dataset

We utilized the World of Code dataset and the GitHub API as the primary data sources for our study. The World of Code dataset is arguably the most comprehensive record of open-source projects and their commit activities [132]. Importantly, it processes the source code of each project and extracts software dependency information, which we use to compute an OSS project's novel combination of packages under development.

Additionally, we collected data on three different types of relationships among OSS developers using the GitHub API. These relationships encompass code commits, issue discussions, and starring activities, many of which are not available in the World of Code dataset.

5.4.2 Data Pre-process

Our study focused on projects primarily written in the Python programming language, defined as projects where more than 50% of the source code files are Python scripts, and examined the activities of developers involved in these projects. We selected Python as the subject of our study due to its widespread popularity in open-source software (OSS) [186] and its usage across various domains by developers from diverse backgrounds [171]. Consequently, the Python project ecosystem exhibits considerable variability in terms of development practices and social relationships. The findings from this ecosystem are likely to be generalizable to other ecosystems.

We initially collected all Python projects from the World of Code dataset and filtered out projects that were forks of others, had fewer than ten commits, or were created after the year 2022.

Next, we utilized the GitHub API to collect developer activities in all the selected projects. Specifically, we gathered all commit contributions, issues, and stars associated with these projects,

including author and timestamp information. This data was used to compute the networks among OSS developers and projects.

To identify bot accounts, we used keywords in user logins. For instance, logins containing terms such as *-bot* or *-robot* were categorized as bots. Additionally, we manually inspected the profile descriptions of the top 100 user accounts in our sample, ordered by the number of commits made, to ensure the removal of highly active bot accounts that could significantly bias our results. All identified bot accounts and their associated activities were subsequently excluded from our analysis.

5.4.3 Network Construction

GitHub functions as a code hosting platform that facilitates collaboration among developers during project development processes. Our study focused on collecting data related to three key categories of developer activities within the realm of open-source projects. These categories include project commits, issue postings, and project stars.

Following data collection, we proceeded to construct networks of projects based on the aforementioned types of activities. These networks were built using activity data from core developers who contribute significantly to the project and wield substantial influence over its development, thereby capturing the flow of knowledge across different projects.

Identification of Project Core Contributors

Numerous methods have been proposed for identifying a project's core developers. Empirical research by Joblin et al.demonstrated that the resulting core contributors generally exhibit agreement when identified using various count-based or network-based metrics [109]. In our study, we classified core developers based on their significant commit activity [109, 143].

For each project in our sample, we collected all commit activities from the main branch. We defined core developers as individuals who authored at least five percent of all commits and had contributed a minimum of ten commits to the project.

Network Definition

We developed directed topological networks to represent the flow of knowledge across projects. In these networks, nodes were used to symbolize projects, while directed edges were established from one project, referred to as A, to another project, designated as B, if and only if a core developer of project A interacted with project B within one year prior to their initial commit to project A. The weight of each edge denoted the number of core developers from project A who engaged with project B in a specific type of interaction.

Our investigation focused on three types of interactions: project commits, issue postings, project forks, and project stars. As a result, we created three distinct directed networks, each corresponding to one of these interactions.

The Collaboration (Commit) Network. A commit to a project signifies a developer's contribution to the codebase. While commits can vary in size [7], a successful commit often requires a

significant understanding of the project's technical details [212], indicating a knowledge flow between the project and the developer.

The Issue Network. GitHub issues provide platforms for discussions on various project-related topics, encompassing feature suggestions, bug reports, and user problems [138]. Engaging in issue discussions contributes significantly to a developer's learning process, fostering a deeper understanding through feedback and comments. Therefore, posting an issue reflects a knowledge flow from the project to the author of the issue.

The Star Network. Starring a project is often undertaken as a gesture of appreciation, followed by reasons such as bookmarking and usage-related intentions [29]. Starring a project suggests that a developer possesses some level of understanding of the project. However, it's important to note that starring a project is considered a low-commitment action and may only indicate a superficial knowledge of the project itself.

5.4.4 Variable Measurements

Our study aimed to investigate the associations between the network positions of a software project and the atypical combinations of software packages adopted. The unit of analysis was individual projects, and all variables were computed at the project level. Below, we describe the computation methods for the variables used in our study.

Using Out-Degree Centrality for Measuring the Amount of Interaction and Knowledge Received

For every project, we calculated its out-degree centrality within each of the three networks: commit, issue, and star networks. Out-degree centrality for a project was defined as the total sum of the weights of all its out-edges. These out-edges represent the number of actions initiated by the core developers of the project towards other projects within the open-source community, or the amount of knowledge likely to flow from other projects to the core developers of the focal project through interactions, thereby influencing its development.

Using Node2Vec Model for Measuring the Diversity of Knowledge Received

Next, we compute the diversity of projects to which a focal project is connected within the interaction networks. This diversity metric reflects the extent to which the focal project receives knowledge from a variety of sources.

While conventional metrics such as network constraints [36] and betweenness centrality [33] are commonly used to measure diversity, they often do not scale well and can be computationally intensive, especially when dealing with large networks such as those in our study.

Moreover, these metrics typically consider both the in-degree and out-degree edges of the focal node, whereas our study focuses solely on the out-degree as it represents the flow of knowledge from other projects to the focal one.

To address this challenge, we adopt the Node2Vec model [96], a classical approach from the machine learning domain, to generate vector representations (embeddings) for each node in the network based on its topological position.

Initially, we generate sequences of random nodes from the graph using a random walk process: Starting from a random node in the network, we sample the next node by randomly selecting another node in the network that is adjacent to the current one. The likelihood of selecting each node is proportional to the edge weight between the candidate next node and the current node. This process continues until we have selected enough nodes to complete the walk (in our study, each walk comprises 20 nodes). We repeat this process to generate multiple walks (or sequences of nodes).

Subsequently, we employ the skip-gram model [139], commonly utilized for generating embeddings for words in natural languages, on the generated walks to obtain embeddings for each node. Nodes that frequently appear together in the same walk tend to have vector representations that are closely situated in the embedding space.

For each project in the network, we examine all other projects that receive a directed edge from the focal project (denoted as set P) and calculate the pairwise distance between any two projects in set P according to Equation 5.1:

$$D = \frac{\sum_{i,j \in P, i \neq j} sim(v_i, v_j)}{|P| * (|P| - 1)} \tag{5.1}$$

Here, i and j represent projects in set P, and v_i and v_j correspond to their vector representations generated by the Node2Vec model. The distance between v_i and v_j is determined by computing the cosine similarity between the two vectors.

We calculate the diversity index for each node in each network. However, the index is undefined for projects with an out-degree centrality less than two. Therefore, such projects will be excluded from our analysis when considering diversity metrics as independent variables.

Using Atypical Combination of Software Packages as Measurement for Project Novelty

The modern software ecosystem encompasses a vast array of software packages, numbering in the hundreds of thousands. The incorporation of these packages, often containing reused code snippets, serves to streamline software development processes and enhance accessibility [118].

Fang et al.introduced a metric for assessing project novelty based on the unconventional combinations of software packages imported within a project [79]. The underlying rationale behind this metric is that the software packages imported by a project reflect its functionality. While certain packages, such as *numpy* and *tensorflow*, are frequently imported together, others, like *numpy* and *requests*, are less commonly combined compared to what would be expected by random chance. Projects that import "atypically paired packages" are deemed "novel" as they employ packages in nontraditional ways.

In their study, computing the atypical combination of software packages requires random reshuffling of the entire network, a computationally challenging process given the scale of networks in our current study. To overcome this challenge, we adopt the skip-gram model once again to generate embeddings of software packages, representing their technical usage and function, drawing inspiration from prior works by Dey and others [63].

For each project, we extract the sequence of packages imported (excluding internal Python packages as per [79]) and generate embeddings for each package based on the dependency relationship. Unlike the random walks of nodes used in the Node2Vec model to generate embeddings,

there is no inherent order in the sequence of packages imported within the same project. Therefore, we set the window size in the skip-gram model to be sufficiently large, enabling it to consider all packages in the same sequence as its context when generating embeddings.

Next, we determine the atypicality of the combination of two packages by computing the negation of the cosine similarity of their embeddings. We then calculate the average pairwise atypicality of all pairs of packages imported by a project to serve as the measure of novelty for that project, following Fang's methodology [79].

It's important to note that we can only compute the atypicality score for projects that import at least two packages. Any other packages are subsequently removed from the analysis.

5.4.5 Using Principal Component Analysis (PCA) to Decompose the Correlated Variables

We employed a linear regression model to investigate the correlational relationship between projects' network positions in different networks and project atypicality. However, we encountered the issue of multicollinearity, where network metrics across networks were highly correlated. This multicollinearity problem can complicate the interpretation of coefficients and reduce the statistical power of our model [61].

To mitigate the multicollinearity issue, we conducted principal component analysis (PCA) as outlined by Wold et al. [219]. This analysis was carried out separately for the out-degree centrality across the three networks (referred to as the *degree variables*) and the diversity measurements across the three networks (referred to as the *diversity variables*).

The PCA resulted in three principal components for the degree variables and another three for the diversity variables. These new variables were created as linear combinations of the original metrics and were orthogonal to each other.

Upon manual examination of these components, we decided to retain two principal components each for the degree and diversity variables. One component represents the average degree (or diversity of ties) of the project across the three networks, while the other represents the strength of network ties in the degree (or diversity of ties) metric. In this latter component, higher weights were assigned to measurements from networks formed of weak ties (*e.g.*, the star network).

Other Control Variables

The exhaustive list of variables employed in our study, encompassing both outcome and independent variables, is delineated in Table 5.1.

5.5 Results

We outline the empirical results below.

5.5.1 Empirical Evidence on the Strength of Different Ties

To start, we present empirical evidence to substantiate the disparities in interaction "strength" across the three networks and to verify whether these strengths align with the theoretical discussions outlined in Section 5.4.3.

We assess the transitivity of networks to empirically evaluate the "strength" of ties in each network. Consider the focal project illustrated in Figure 5.1: if project A is connected to developers B and C, a triad is formed. If A's connections to B and C are both strong ties, the triad is unstable and will evolve into a triangle where A, B, and C are mutually connected. This evolution occurs because B and C have numerous opportunities to interact and foster familiarity due to their strong ties with developer A. Thus, networks characterized by strong ties typically exhibit a greater number of triangles, as opposed to triads, compared to those with weak ties [94]. Formally, this can be computed using the measurement of transitivity as outlined in equation 5.2.

$$T = \frac{3 * N_{\text{triangles}}}{N_{\text{triads}}} \tag{5.2}$$

Here, $N_{\text{triangles}}$ and N_{triads} represent the number of triangles and triads in the network, respectively. A higher transitivity value indicates a network of stronger ties.

As network transitivity is usually defined and computed in undirected networks, for simplicity to provide transitivity descriptive statistics, we convert all three networks (which are all directed) to their undirected form. This is achieved by considering there is an undirected edge between nodes A and B if there is either an edge from A to B or from B to A.

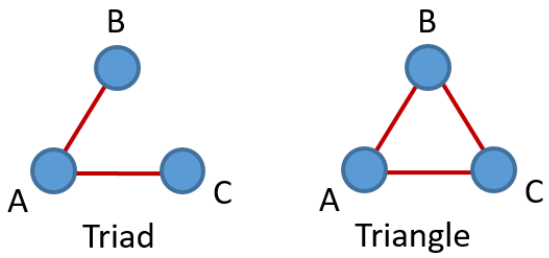


Figure 5.1: The triads and triangles in networks

Table 5.2 presents descriptive statistics for the project networks, delineated by commit, issue, and star interactions.

The reported number of nodes excludes isolates (*i.e.*, nodes without edges) from each network, and the number of edges is calculated in the undirected version of each network.

The descriptive statistics underscore significant differences in network transitivity among various network types. Specifically, the commit network displays the highest levels of transitivity, followed by the issue network, while the star network exhibits the lowest level. These findings are consistent with our theoretical understanding of tie strength.

5.5.2 PCA Analysis of Degree and Diversity Variables

We performed Principal Component Analysis (PCA) on both the degree variables and diversity variables representing a project's position in the commit, issue, and star networks. For the degree variables, PCA was conducted using the complete dataset of study samples. However, for the diversity variables, PCA was restricted to projects with an out-degree of at least two in all three networks to ensure the feasibility of their diversity computations.

Table 5.3 provides the proportion of variance explained by each principal component, while Table 5.4 presents the loadings of the principal components on the original variables.

Prior to analysis, all input variables underwent logarithmic scaling and normalization to achieve a mean of zero and a standard deviation of one.

In both the degree and diversity variables, we retain the first two principal components as they altogether capture over 80% of variance from the original variables, as detailed in Table 5.3.

Principal components represent linear combinations of the original variables, with the loadings indicating the contributing coefficients. As detailed in Table 5.4, the first and second principal component (*i.e.*, PC1 and PC2) for both degree and diversity variables are computed according to equations 5.3 and 5.4.

$$\begin{aligned} PC1_{\text{degree}} &= 0.60 * Deg_{\text{commit}} + 0.61 * Deg_{\text{issue}} + 0.52 * Deg_{\text{star}} \\ PC2_{\text{degree}} &= -0.45 * Deg_{\text{commit}} - 0.28 * Deg_{\text{issue}} + 0.85 * Deg_{\text{star}} \end{aligned} \tag{5.3}$$

$$\begin{aligned} PC1_{\text{diversity}} &= 0.63 * Div_{\text{commit}} + 0.64 * Div_{\text{issue}} + 0.43 * Div_{\text{star}} \\ PC2_{\text{diversity}} &= -0.36 * Div_{\text{commit}} - 0.24 * Div_{\text{issue}} + 0.90 * Div_{\text{star}} \end{aligned} \tag{5.4}$$

The loadings onto the first principal component (PC1) are relatively consistent across networks for both degree and diversity variables. We interpret PC1 as an average of the network metric (either degree or diversity) from each network.

Conversely, the loadings on the second principal component (PC2) exhibit a descending trend. The highest loading on the degree (or diversity) comes from the star network, followed by the issue network, and the lowest from the commit network. Drawing upon our theoretical framework elucidated in Section 5.4.3, we consider the loadings on PC2 as indicative of network metrics influenced by interactions of varying strengths, with a heavier emphasis on weak ties.

When incorporating both PC1 and PC2 into the same linear regression model, we interpret their meanings as follows:

- PC1 reflects the average degree (or diversity) of out-edges from a project across the three networks.
- PC2 reflects the extent to which the degree (or diversity) is derived from weak ties within the network.

Maintaining the value of PC1 constant, higher values of PC2 indicate a greater degree (or diversity) of the project's edges originating from weak ties within the network, and conversely, lower PC2 values signify a lesser influence of weak ties on the network metrics.

5.5.3 The Influence of Degree Centrality and Diversity of Ties on Project Novelty

Table 5.5 presents the results of the regression analysis conducted on project atypicality. Model I encompassed all samples within the study, whereas Models II, III, and IV were exclusively applied to projects with a minimum out-degree centrality of two across all three networks, thereby enabling computation of their tie diversity.

Model I and Model II shed light on the impact of degree variables. Model I reveals a significant positive effect from average degree centrality, whereas Model II shows a negative effect. However, the effect sizes for both models are relatively small. For instance, in Model I, transitioning a project's Deg_{ave} from the 25th percentile (-1.08) to the 75th percentile (0.54) results in a mere 0.002 increase in project atypicality. Considering that the 25th percentile of a project's atypicality score is -0.33, and the 75th percentile is -0.12 in our sample, the change in project novelty due to variations in average degree centrality is practically minimal. Furthermore, no statistically significant effect is observed from the $Deg_{weakness}$ variable. Therefore, we conclude that **H9** and **H11** are not confirmed, as they do not demonstrate a practically significant effect from degree variables to project novelty.

In Model III, we observe a positive association between the average diversity (Div_{ave}) of ties originating from the focal project and project novelty. This effect size is notably stronger compared to that of average degree. Interestingly, despite the mean and standard deviation of Deg_{ave} and Div_{ave} being very similar across Models I, II, and III, the estimated coefficient of Div_{ave} in Model III is nearly seven times greater than that of Deg_{ave} in Models I and II.

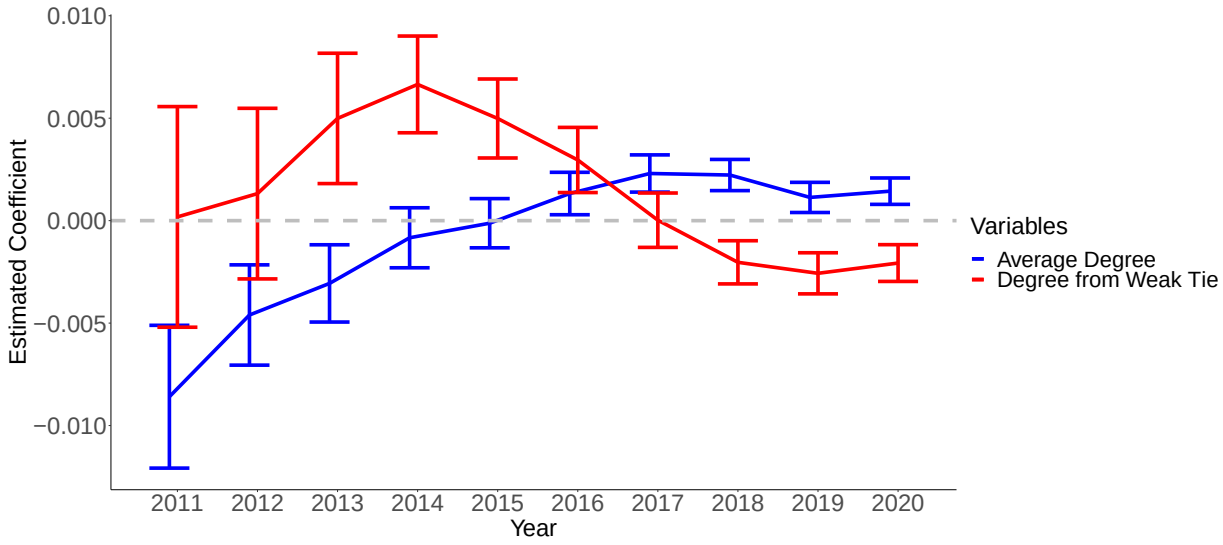
Transitioning a project from the 25th percentile (-0.89) to the 75th percentile (0.86) in terms of Div_{ave} results in an increase of approximately 0.012 in project novelty, representing roughly a 4% change in the distribution (e.g., a shift from the median project in terms of novelty to the 54th percentile).

Additionally, we observe a significant effect for the $Div_{weakness}$ variable. This suggests that, when controlling for the average diversity across the three networks, higher diversity in the 'weak network' (e.g., star network) corresponds to increased project novelty. However, the practical effect size for this variable is much smaller. For instance, transitioning from the 25th percentile of $Div_{weakness}$ to the 75th percentile only increases project novelty by 0.003, representing approximately a 1% increase in the distribution.

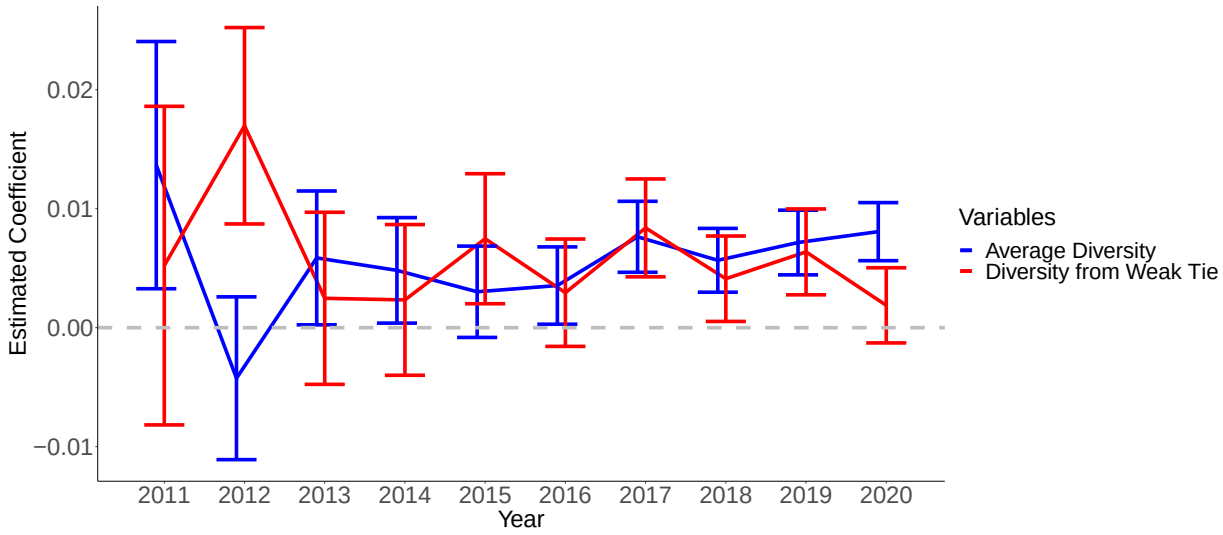
Based on these findings, we conclude that **H10** and **H12** are confirmed.

When incorporating both degree and diversity variables in the same regression model, as in Model IV, we discovered that the positive effects from Div_{ave} and $Div_{weakness}$ remained significant. This suggests a consistent effect stemming from tie diversity and diversity in weak ties. However, we also noted a significant negative effect from Deg_{ave} and $Deg_{weakness}$ after adjusting for project diversity.

The negative effect indicates that, while holding tie diversity constant, higher out-degree centrality and more ties in the 'weak network' tend to decrease project novelty rather than enhance it. This observation could be attributed to the increased constraints within a local community associated with higher out-degree, as discussed by Burt [37]. Such constraints can lead to a reduced inclination for combining packages of diverse functions within the project, consequently contributing to the diminished novelty of the project.



(a) The effects of degree variables on project novelty across years



(b) The effects of diversity variables on project novelty across years

Figure 5.2: Knowledge access and its relationship to project novelty across years (error bar represents the 95% confidence interval)

5.5.4 The Influence of Degree and Diversity Variables Across Years

We divided our dataset based on the year of project creation and conducted regression analyses similar to Models I and III to assess the effects of degree and diversity variables on projects created in different years. The estimated coefficients of the degree and diversity variables are illustrated in Figure 5.2 (The other control variables are included in the model, but their estimated effects are not shown in the figure). Figure 5.2a presents the estimated coefficients for degree variables across project creation years.

An intriguing trend is observed: before the year 2015, the average degree of projects is negatively associated with project atypicality, while more ties in the 'weak networks' are positively associated with project atypicality. However, this pattern is reversed in recent years, where higher average degree corresponds to higher atypicality, and more degree from the 'weak networks' corresponds to lower atypicality.

This longitudinal trend persists even after several adjustments to the regression model. For instance, computing the yearly estimated coefficients for the degree variables after controlling for the diversity variables (similar to Model IV in Table 5.5), the estimated coefficients and their trends remain consistent.

This changing pattern could also be influenced by varying development times for projects created in different years. For example, our observation period concluded on the last day of 2021, and we computed the atypicality score for projects created in 2019 using all packages imported before 2022, representing roughly one to two years of development time. In contrast, projects created in earlier years have much longer development times.

To mitigate the influence of varied development lengths, we adjusted the outcome variable in the regression model (*i.e.*, atypicality) to be computed based only on commits made within X years of project creation for projects created in different years. We set X as 1, 3, and 5, and observed similar qualitative trends for the variables Deg_{ave} (Average Degree) and $Deg_{weakness}$ (Degree from Weak Tie).

The persistence of the trend in Figure 5.2a suggests a potential shift in how innovative projects are created within open-source software communities. While in the past, a higher out-degree in weak ties appeared to be positively associated with project novelty, recent years have seen a shift where a greater emphasis on the amount of interactions through strong ties may better facilitate the creation of novel projects.

The reasons behind this changing trend and its implications remain open questions worth exploring for future research. It could reflect evolving collaboration patterns, changes in the nature of software development, or shifts in community dynamics within open-source ecosystems. Understanding these dynamics can offer valuable insights into the factors driving innovation and project success in open-source software environments.

The effects of diversity variables appear to be more consistent over time. As illustrated in Figure 5.2b, both the average diversity of ties from the focal project (Div_{ave} in Table 5.5) and the diversity from weak ties ($Div_{weakness}$) consistently exhibit a positive effect on project novelty. It is worth noting that the effects may not always be significant for certain years, likely due to the small number of projects available for estimating the diversity variables in those years.

5.6 Discussions

5.6.1 Accessing Diverse Knowledge Drives the Development of Innovative Projects

Developers learn about new programming skills and knowledge as they participate in the development of open-source software projects. Particularly as reported by our study, that the diversity of knowledge that developers learn, not the amount, explains and predicts the novelty of projects those developers will create in the future, where novelty is defined as adopting software packages that are usually not combined together in the same project.

It contributes to the theory about diversity in the context of open-source software. While the existing literature often looks at the diversity in terms of superficial demographical traits such as gender [148, 161, 208], tenure [208], and geographic location [11], our study looks at the diversity of the knowledge and skills developers possess, and conclude the empirical correlation between a diverse knowledge background and the creation of novelty software projects.

This founding also have important practical implications on the creation of novel projects in open-source software. Research have shown that developers tend to join, or pay attention to projects that are technical familiar to them [63, 78], and many automated project recommendation tools tend to recommend projects to developers with related technical background [229]. However, our result suggests that we decrease the level of project innovation by only recommending the projects to developers who have the related technical skills, and it may further decrease the level of project novelty if developers are exposed to, and interact with many projects of similar technical details without a high diversity of projects being interacted.

To help boost innovation in open-source software community, we suggest to add the diversity dimension when recommending projects to developers, and make sure developers are exposed to projects that contain diverse knowledge. In addition, software knowledge can be diffused through multiple channels online such as StackOverflow [206] and Twitter [182], and there are empirical evidence to suggest that developers are able to access different information through different channels [80]. Therefore, we recommend developers to access information through multiple channels and tools, so that they will access diversity information and likely increase the novelty of their project.

5.6.2 Learning through Weak Ties Boosts the Creation of Novel Projects

Accessing diverse knowledge through social ties is instrumental in fostering the development of innovative projects. Specifically, gaining diverse project knowledge via ”weak ties” (e.g., star networks) proves more beneficial compared to strong ties.

Existing literature describes passive forms of engagement, such as starring, as merely expressing appreciation for a project or bookmarking it for future reference [29]. Developers who solely star a project without engaging in discussions or contributing to its development are often labeled as passive participants or peripheral users [57, 150]. While some studies highlight the positive impact of passive participants on project diffusion and quality improvement [179], our understanding of the broader implications of passive participation within the developer community and the open-source ecosystem remains limited.

Our empirical findings contribute to elucidating the role of passive and weak interactions in the open-source software community. We propose the existence of a learning mechanism through passive interactions with projects, particularly when developers star a diverse array of projects (indicating reviewing and gaining a fundamental understanding of their code). This learning process is closely tied to the novelty of both the developer and the projects they create in the future. Moreover, encouraging developers to engage in reviewing diverse projects online, even without active participation in development, could potentially boost innovation in the entire open-source ecosystem.

5.7 Conclusions

In this study, we conducted empirical investigations into the correlation between developers' access to knowledge via social networks and the novelty of the projects they develop. Our results indicate that developers who acquire knowledge from a wide range of projects tend to produce more innovative outcomes. Specifically, the acquisition of knowledge from a diverse array of projects through weak ties, such as within a star network, exhibits greater significance than contributing to a varied project portfolio.

These findings contribute to the theoretical comprehension of diversity's role within open-source environments, the determinants of novel project creation, and the influence of passive engagement in open-source endeavors on project efficacy. Furthermore, we offer practical insights for developers on effective strategies for enhancing their capacity to innovate in project development. Additionally, we propose recommendations for open-source platforms and tool developers to better support developers in creating innovative projects.

Table 5.1: The definitions of the variables in our models.

Outcome Variables	
Atypicality	The extent to which a project imported software packages that were typically not imported together in one project, computed based on all the packages imported until the end of year 2021.
Degree Variables (original)	
$\text{Deg}_{\text{Commit}}$	The sum of edge weights from the focal project to the others in the commit network.
$\text{Deg}_{\text{Issue}}$	The sum of edge weights from the focal project to the others in the issue network.
Deg_{Star}	The sum of edge weights from the focal project to the others in the star network.
Degree Variables (post-PCA)	
Deg_{Ave}	The first principal component resulted from the PCA analysis over three original degree variables, representing average network degree across the three networks.
$\text{Deg}_{\text{Weakness}}$	The second principal component resulted from the PCA analysis over three original degree variables, representing the degree from weak ties.
Diversity Variables (original)	
$\text{Div}_{\text{Commit}}$	The diversity of projects to which the focal project has a directed edge in the commit network.
$\text{Div}_{\text{Issue}}$	The diversity of projects to which the focal project has a directed edge in the issue network.
Div_{Star}	The diversity of projects to which the focal project has a directed edge in the star network.
Diversity Variables (post-PCA)	
Div_{Ave}	The first principal component resulted from the PCA analysis over three original diversity variables, representing average diversity of connection across the three networks.
$\text{Div}_{\text{Weakness}}$	The second principal component resulted from the PCA analysis over three original diversity variables, representing the diversity from to weak ties.
Controls Variables	
$\text{Year}_{\text{creation}}$	A fixed effect variable indicating the year in which the project received its first commit.
Owned by Org?	A binary variable indicating whether the project was owned by an organizational account or not.
$N_{\text{Owner\_Star}}$	The number of total stars that the project owner received from all other projects it owns before the creation of the focal project.
$N_{\text{Core\_Dev}}$	The total number of core developers in the project, as computed before the end of year 2021.
N_{Packages}	The number of software packages that the project import before the end of year 2021.

Table 5.2: Empirical evaluation of the "strength" of social ties

	#Nodes	**#Edges**	**Transitivity** (*10^{-2})
Commit Network	$763,062$	$1,926,978$	30.04
Issue Network	$278,945$	$727,255$	3.42
Star Network	$480,394$	$3,658,543$	0.23

Table 5.3: Proportional of variance explained by each principal component for degree and diversity variables

	PC1	**PC2**	**PC3**
Variance Explained (Degree/Diversity)	$0.64/0.51$	$0.22/0.29$	$0.14/0.19$
Cumulative Variance (Degree/Diversity)	$0.64/0.51$	$0.86/0.81$	$1.00/1.00$

Table 5.4: Loadings of principal components on degree/diversity variables

	PC1	**PC2**	**PC3**
D_{commit}(Degree/Diversity)	$0.60/0.63$	$-0.45/-0.36$	$0.67/0.69$
D_{issue}(Degree/Diversity)	$0.61/0.64$	$-0.28/-0.24$	$-0.74/-0.72$
D_{star}(Degree/Diversity)	$0.52/0.43$	$0.85/0.90$	$0.11/0.08$

Table 5.5: Regression analysis on factors influencing the project atypicality

	Model I	Model II	Model III	Model IV
Variables of interest				
Deg_{ave}	0.001^{***}	-0.001^{**}		-0.002^{***}
	(0.0002)	(0.001)		(0.001)
$Deg_{weakness}$	-0.00002	-0.0001		-0.005^{***}
	(0.0002)	(0.001)		(0.001)
Div_{ave}			0.007^{***}	0.008^{***}
			(0.001)	(0.001)
$Div_{weakness}$			0.005^{***}	0.007^{***}
			(0.001)	(0.001)
Controls				
Owned by Org?	0.027^{***}	0.017^{***}	0.018^{***}	0.018^{***}
	(0.001)	(0.002)	(0.002)	(0.002)
$N_{\text{Owner\_Star}}$*	0.003^{***}	0.001	0.0004	0.001^{**}
	(0.0002)	(0.0004)	(0.0004)	(0.0004)
$N_{\text{Core\_Dev}}$*	0.014^{***}	0.019^{***}	0.018^{***}	0.019^{***}
	(0.001)	(0.002)	(0.002)	(0.002)
N_{Packages}*	0.040^{***}	0.023^{***}	0.023^{***}	0.024^{***}
	(0.0003)	(0.001)	(0.001)	(0.001)
Fixed effect				
$\text{Year}_{\text{creation}}$	X	X	X	X
Observations	602,375	38,164	38,164	38,164
Adjusted R^2	0.052	0.063	0.068	0.069

*Note: Variables donated by * are log-scaled* *p<0.1; **p<0.05; ***p<0.01

Bibliography

[1] Alberto Abadie. Semiparametric difference-in-differences estimators. *The Review of Economic Studies*, 72(1):1–19, 2005.

[2] Alberto Abadie, Susan Athey, Guido W Imbens, and Jeffrey Wooldridge. When should you adjust standard errors for clustering? Technical report, National Bureau of Economic Research, 2017.

[3] Shoaib Abdul Basit and Kehinde Medase. The diversity of knowledge sources and its impact on firm-level innovation: Evidence from germany. *European Journal of Innovation Management*, 22(4):681–714, 2019.

[4] Richard Adams, John Bessant, and Robert Phelps. Innovation management measurement: A review. *International journal of management reviews*, 8(1):21–47, 2006.

[5] Amritanshu Agrawal, Akond Rahman, Rahul Krishna, Alexander Sobran, and Tim Menzies. We don't need another hero? the impact of "heroes" on software development. In *International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 245–253, 2018.

[6] Divyakant Agrawal, Ceren Budak, and Amr El Abbadi. Information diffusion in social networks: Observing and influencing societal interests. *Proceedings of the VLDB Endowment*, 4(12):1512–1513, 2011.

[7] Abdulkareem Alali, Huzefa Kagdi, and Jonathan I Maletic. What's a typical commit? a characterization of open source software repositories. In *2008 16th IEEE international conference on program comprehension*, pages 182–191. IEEE, 2008.

[8] Sadika Amreen, Audris Mockus, Russell Zaretzki, Christopher Bogart, and Yuxia Zhang. Alfaa: Active learning fingerprint based anti-aliasing for correcting developer identity errors in version control systems. *Empirical Software Engineering*, 25:1136–1167, 2020.

[9] Maurício Aniche, Christoph Treude, Igor Steinmacher, Igor Wiese, Gustavo Pinto, Margaret-Anne Storey, and Marco Aurélio Gerosa. How modern news aggregators help development communities shape and share knowledge. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pages 499–510. IEEE, 2018.

[10] Deeksha Arya, Wenting Wang, Jin LC Guo, and Jinghui Cheng. Analysis and detection of information types of open source software issue discussions. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 454–464. IEEE, 2019.

[11] Joop Aué, Michiel Haisma, Kristín Fjóla Tómasdóttir, and Alberto Bacchelli. Social

diversity and growth levels of open source software projects on github. In *Proceedings of the 10th ACM/IEEE International symposium on empirical software engineering and measurement*, pages 1–6, 2016.

[12] Guilherme Avelino, Leonardo Passos, Andre Hora, and Marco Tulio Valente. A novel approach for estimating truck factors. In *International Conference on Program Comprehension (ICPC)*, pages 1–10. IEEE, 2016.

[13] Guilherme Avelino, Eleni Constantinou, Marco Tulio Valente, and Alexander Serebrenik. On the abandonment and survival of open source projects: An empirical investigation. In *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–12. IEEE, 2019.

[14] Aida Azadegan, K Nadia Papamichail, and Pedro Sampaio. Applying collaborative process design to user requirements elicitation: A case study. *Computers in Industry*, 64(7): 798–812, 2013.

[15] Ali Sajedi Badashian and Eleni Stroulia. Measuring user influence in GitHub: the million follower fallacy. In *International Workshop on CrowdSourcing in Software Engineering (CSI-SE)*, pages 15–21. IEEE, 2016.

[16] Eytan Bakshy, Itamar Rosenn, Cameron Marlow, and Lada Adamic. The role of social networks in information diffusion. In *Proceedings of the 21st international conference on World Wide Web*, pages 519–528, 2012.

[17] Sogol Balali, Umayal Annamalai, Hema Susmita Padala, Bianca Trinkenreich, Marco A Gerosa, Igor Steinmacher, and Anita Sarma. Recommending tasks to newcomers in oss projects: How do mentors handle it? In *Proceedings of the 16th International Symposium on Open Collaboration*, pages 1–14, 2020.

[18] Margherita Balconi. Tacitness, codification of technological knowledge and the organisation of industry. *Research policy*, 31(3):357–379, 2002.

[19] Ann Barcomb, Andreas Kaufmann, Dirk Riehle, Klaas-Jan Stol, and Brian Fitzgerald. Uncovering the periphery: A qualitative survey of episodic volunteering in free/libre and open source software communities. *IEEE Transactions on Software Engineering*, 46(9): 962–980, 2018.

[20] Andrew Begel, Robert DeLine, and Thomas Zimmermann. Social media for software engineering. In *Proceedings of the FSE/SDP workshop on Future of software engineering research*, pages 33–38, 2010.

[21] Alejandra Beghelli and Sara Jones. On the novelty of software products. In *Proceedings of the Sixth International Conference on Design Creativity (ICDC 2020)*, pages 11–18. The Design Society, 2020.

[22] Pankaj Bhatt, Gautam Shroff, and Arun K Misra. Dynamics of software maintenance. *ACM SIGSOFT Software Engineering Notes*, 29(5):1–5, 2004.

[23] Nauman bin Ali, Henry Edison, and Richard Torkar. The impact of a proposal for innovation measurement in the software industry. In *International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–6, 2020.

[24] Christian Bird, Alex Gourley, Prem Devanbu, Michael Gertz, and Anand Swaminathan. Mining email social networks. In *International Conference on Mining Software Repositories (MSR)*, pages 137–143, 2006.

[25] Tegawendé F Bissyandé, Ferdian Thung, David Lo, Lingxiao Jiang, and Laurent Réveillere. Popularity, interoperability, and impact of programming languages in 100,000 open source projects. In *2013 IEEE 37th annual computer software and applications conference*, pages 303–312. IEEE, 2013.

[26] Sue Black, Rachel Harrison, and Mark Baldwin. A survey of social media use in software systems development. In *International Workshop on Web 2.0 for Software Engineering*, pages 1–5. ACM, 2010.

[27] Kelly Blincoe, Jyoti Sheoran, Sean Goggins, Eva Petakovic, and Daniela Damian. Understanding the popular users: Following, affiliation influence and leadership on github. *Information and Software Technology*, 70:30–39, 2016.

[28] Christopher Bogart, Christian Kästner, James Herbsleb, and Ferdian Thung. How to break an api: cost negotiation and community values in three software ecosystems. In *Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE)*, pages 109–120, 2016.

[29] Hudson Borges and Marco Tulio Valente. What's in a github star? understanding repository starring practices in a social coding platform. *Journal of Systems and Software*, 146: 112–129, 2018.

[30] Hudson Borges, Andre Hora, and Marco Tulio Valente. Understanding the factors that impact the popularity of github repositories. In *International Conference on Software Maintenance and Evolution (ICSME)*, pages 334–344, 2016.

[31] Hudson Silva Borges and Marco Tulio Valente. How do developers promote open source projects? *Computer*, 52(8):27–33, 2019.

[32] Gargi Bougie, Jamie Starke, Margaret-Anne Storey, and Daniel M German. Towards understanding twitter use in software engineering: Preliminary findings, ongoing challenges and future questions. In *International Workshop on Web 2.0 for Software Engineering*, pages 31–36. ACM, 2011.

[33] Ulrik Brandes. A faster algorithm for betweenness centrality. *Journal of mathematical sociology*, 25(2):163–177, 2001.

[34] Frederick P Brooks Jr. *The mythical man-month: essays on software engineering*. Pearson Education, 1995.

[35] Chris Brown and Chris Parnin. Understanding the impact of GitHub suggested changes on recommendations between developers. In *Joint Meeting on the Foundations of Software Engineering (ESEC/FSE)*, pages 1065–1076. ACM, 2020.

[36] Ronald S Burt. A note on social capital and network content. *Social networks*, 19(4): 355–373, 1997.

[37] Ronald S Burt. The network structure of social capital. *Research in organizational behavior*, 22:345–423, 2000.

[38] Ronald S Burt. Structural holes and good ideas. *American journal of sociology*, 110(2): 349–399, 2004.

[39] Fabio Calefato, Marco Aurelio Gerosa, Giuseppe Iaffaldano, Filippo Lanubile, and Igor Steinmacher. Will you come back to contribute? investigating the inactivity of oss core developers in github. *Empirical Software Engineering*, 27(3):76, 2022.

[40] Marco Caliendo and Sabine Kopeinig. Some practical guidance for the implementation of propensity score matching. *Journal of Economic Surveys*, 22(1):31–72, 2008.

[41] Gerardo Canfora, Massimiliano Di Penta, Rocco Oliveto, and Sebastiano Panichella. Who is going to mentor newcomers in open source projects? In *Proceedings of the 20th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE)*, pages 1–11, 2012.

[42] Brandon Carlson, Kevin Leach, Darko Marinov, Meiyappan Nagappan, and Atul Prakash. Open source vulnerability notification. In *IFIP International Conference on Open Source Systems (OSS)*, pages 12–23. Springer, 2019.

[43] Casey Casalnuovo, Bogdan Vasilescu, Premkumar Devanbu, and Vladimir Filkov. Developer onboarding in github: the role of prior social links and language experience. In *Proceedings of the 2015 10th joint meeting on foundations of software engineering*, pages 817–828, 2015.

[44] Carlos Castillo, Mohammed El-Haddad, Jürgen Pfeffer, and Matt Stempeck. Characterizing the life cycle of online news stories using social media reactions. In *ACM Conference on Computer Supported Cooperative Work & Social Computing (CSCW)*, pages 211–223. ACM, 2014.

[45] Antonio Cerone. Learning and activity patterns in oss communities and their impact on software quality. *Electronic Communications of the EASST*, 48, 2013.

[46] Gina N Cervetti, Carolyn A Jaynes, and Elfrieda H Hiebert. Increasing opportunities to acquire knowledge through reading. *Reading more, reading better*, pages 79–100, 2009.

[47] InduShobha Chengalur-Smith, Anna Sidorova, and Sherae L Daniel. Sustainability of free/libre open source projects: A longitudinal study. *Journal of the Association for Information Systems*, 11(11):5, 2010.

[48] Clayton M Christensen and Clayton M Christensen. *The innovator's dilemma: The revolutionary book that will change the way you do business*. HarperBusiness Essentials New York, NY, 2003.

[49] Johan SG Chu and James A Evans. Slowed canonical progress in large fields of science. *Proceedings of the National Academy of Sciences*, 118(41):e2021636118, 2021.

[50] Jailton Coelho and Marco Tulio Valente. Why modern open source projects fail. In *Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*, pages 186–196. ACM, 2017.

[51] Jailton Coelho, Marco Tulio Valente, Luciana L Silva, and Emad Shihab. Identifying unmaintained projects in GitHub. In *International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–10, 2018.

[52] Jailton Coelho, Marco Tulio Valente, Luciano Milen, and Luciana L Silva. Is this github project maintained? measuring the level of maintenance activity of open-source projects. *Information and Software Technology*, 122:106274, 2020.

[53] Eleni Constantinou and Tom Mens. Socio-technical evolution of the ruby ecosystem in github. In *2017 IEEE 24th international conference on software analysis, evolution and reengineering (SANER)*, pages 34–44, 2017.

[54] J Daniel Couger and Geoff Dengate. Measurement of creativity of is products. In *Proceedings of the Twenty-Fifth Hawaii International Conference on System Sciences*, volume 4, pages 288–298. IEEE, 1992.

[55] Kevin Crowston and James Howison. The social structure of free and open source software development. *First Monday*, 2005.

[56] Kevin Crowston, Kangning Wei, Qing Li, U Yeliz Eseryel, and James Howison. Coordination of free/libre open source software development. 2005.

[57] Kevin Crowston, Kangning Wei, Qing Li, and James Howison. Core and periphery in free/libre and open source software team communications. In *Proceedings of the 39th Annual Hawaii International Conference on System Sciences (HICSS'06)*, volume 6, pages 118a–118a. IEEE, 2006.

[58] Kevin Crowston, Qing Li, Kangning Wei, U Yeliz Eseryel, and James Howison. Self-organization of teams for free/libre open source software development. *Information and software technology*, 49(6):564–575, 2007.

[59] Kevin Crowston, Kangning Wei, James Howison, and Andrea Wiggins. Free/libre open-source software development: What we know and what we do not know. *ACM Computing Surveys (CSUR)*, 44(2):1–35, 2008.

[60] Laura Dabbish, Colleen Stuart, Jason Tsay, and Jim Herbsleb. Social coding in github: transparency and collaboration in an open software repository. In *Proceedings of the ACM 2012 conference on computer supported cooperative work*, pages 1277–1286, 2012.

[61] Jamal I Daoud. Multicollinearity and regression analysis. In *Journal of Physics: Conference Series*, volume 949, page 012009. IOP Publishing, 2017.

[62] Tapajit Dey, Sara Mousavi, Eduardo Ponce, Tanner Fry, Bogdan Vasilescu, Anna Filippova, and Audris Mockus. Detecting and characterizing bots that commit code. In *Proceedings of the 17th International Conference on Mining Software Repositories (MSR)*, pages 209–219, 2020.

[63] Tapajit Dey, Andrey Karnauch, and Audris Mockus. Representation of developer expertise in open source software. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 995–1007. IEEE, 2021.

[64] Thomas Dohmke. Blog post: 100 million developers and counting. `https://github.blog/2023-01-25-100-million-developers-and-counting/`, 2023. Retrieved: March 2023.

[65] Stephen G Donald and Kevin Lang. Inference with difference-in-differences and other panel data. *The review of Economics and Statistics*, 89(2):221–233, 2007.

[66] John Qi Dong, Weifang Wu, and Yixin Sarah Zhang. The faster the better? innovation speed and user interest in open source software. *Information & Management*, 56(5):669–680, 2019.

[67] Marina Du Plessis. The role of knowledge management in innovation. *Journal of knowledge management*, 11(4):20–29, 2007.

[68] Esther Duflo. Schooling and labor market consequences of school construction in Indonesia: Evidence from an unusual policy experiment. *American Economic Review*, 91(4):795–813, 2001.

[69] Zakir Durumeric, Frank Li, James Kasten, Johanna Amann, Jethro Beekman, Mathias Payer, Nicolas Weaver, David Adrian, Vern Paxson, Michael Bailey, et al. The matter of heartbleed. In *Proceedings of the 2014 conference on internet measurement conference*, pages 475–488, 2014.

[70] Subhabrata Dutta, Sarah Masud, Soumen Chakrabarti, and Tanmoy Chakraborty. Deep exogenous and endogenous influence combination for social chatter intensity prediction. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1999–2008, 2020.

[71] Henry Edison, Nauman Bin Ali, and Richard Torkar. Towards innovation measurement in the software industry. *Journal of systems and software*, 86(5):1390–1407, 2013.

[72] Nadia Eghbal. *Roads and bridges: The unseen labor behind our digital infrastructure*. Ford Foundation, 2016.

[73] Nadia Eghbal. *Working in Public: The Making and Maintenance of Open Source Software*. Stripe Press, 2020.

[74] Nadia Eghbal. *Working in public: The making and maintenance of open source software*. Stripe Press San Francisco, 2020.

[75] Alberto Espinosa, Robert Kraut, Javier Lerch, Sandra Slaughter, James Herbsleb, and Audris Mockus. Shared mental models and coordination in large-scale, distributed software development. 2001.

[76] Hongbo Fang, Daniel Klug, Hemank Lamba, James Herbsleb, and Bogdan Vasilescu. Need for tweet: How open source developers talk about their github work on twitter. In *Proceedings of the 17th International Conference on Mining Software Repositories*, pages 322–326, 2020.

[77] Hongbo Fang, Hemank Lamba, James Herbsleb, and Bogdan Vasilescu. "This is damn slick!" estimating the impact of tweets on open source project popularity and new contributors. In *Proceedings of the 44th International Conference on Software Engineering*, pages 2116–2129, 2022.

[78] Hongbo Fang, James Herbsleb, and Bogdan Vasilescu. Matching skills, past collaboration, and limited competition: Modeling when open-source projects attract contributors. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 42–54, 2023.

[79] Hongbo Fang, Bogdan Vasilescu, and James Herbsleb. Novelty begets popularity, but

curbs participation-a macroscopic view of the python open-source ecosystem. In *2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE)*, pages 643–653. IEEE Computer Society, 2023.

[80] Hongbo Fang, Bogdan Vasilescu, and James Herbsleb. Understanding information diffusion about open-source projects on twitter, hackernews, and reddit. In *Proceedings of the 16th International Conference on Cooperative and Human Aspects of Software Engineering (CHASE)*, pages 56–67, 2023.

[81] Yulin Fang and Derrick Neufeld. Understanding sustained participation in open source software projects. *Journal of Management Information Systems*, 25(4):9–50, 2009.

[82] Fabio Ferreira, Luciana Lourdes Silva, and Marco Tulio Valente. Turnover in open-source projects: The case of core developers. In *Brazilian Symposium on Software Engineering (SBES)*, pages 447–456, 2020.

[83] Chaim Fershtman and Neil Gandal. Direct and indirect knowledge spillovers: the "social network" of open-source projects. *The RAND Journal of Economics*, 42(1):70–91, 2011.

[84] Santo Fortunato, Carl T Bergstrom, Katy Börner, James A Evans, Dirk Helbing, Staša Milojević, Alexander M Petersen, Filippo Radicchi, Roberta Sinatra, Brian Uzzi, et al. Science of science. *Science*, 359(6379):eaao0185, 2018.

[85] Matthieu Foucault, Marc Palyart, Xavier Blanc, Gail C Murphy, and Jean-Rémy Falleri. Impact of developer turnover on quality in open-source software. In *Joint Meeting on the Foundations of Software Engineering (ESEC/FSE)*, pages 829–841, 2015.

[86] Felipe Fronchetti, Igor Wiese, Gustavo Pinto, and Igor Steinmacher. What attracts newcomers to onboard on oss projects? tl; dr: Popularity. In *IFIP International Conference on Open Source Systems*, pages 91–103. Springer, 2019.

[87] Tanner Fry, Tapajit Dey, Andrey Karnauch, and Audris Mockus. A dataset and an approach for identity resolution of 38 million author ids extracted from 2b git commits. In *Proceedings of the 17th International Conference on Mining Software Repositories (MSR)*, pages 518–522, 2020.

[88] R Stuart Geiger, Dorothy Howard, and Lilly Irani. The labor of maintaining and scaling free and open-source software projects. *Proceedings of the ACM on Human-Computer Interaction*, 5(CSCW1):1–28, 2021.

[89] Andrew Gelman and Jennifer Hill. *Data analysis using regression and multilevel/hierarchical models*. Cambridge University Press, 2006.

[90] Marco Gerosa, Igor Wiese, Bianca Trinkenreich, Georg Link, Gregorio Robles, Christoph Treude, Igor Steinmacher, and Anita Sarma. The shifting sands of motivation: Revisiting what drives contributors in open source. In *Proceedings of the 43rd International Conference on Software Engineering (ICSE)*, pages 1046–1058, 2021.

[91] Todd A Gormley and David A Matsa. Growing out of trouble? corporate responses to liability risk. *The Review of Financial Studies*, 24(8):2781–2821, 2011.

[92] Georgios Gousios and Diomidis Spinellis. Ghtorrent: Github's data from a firehose. In *2012 9th IEEE Working Conference on Mining Software Repositories (MSR)*, pages 12–21.

IEEE, 2012.

[93] Mark Granovetter. The strength of weak ties: A network theory revisited. *Sociological theory*, pages 201–233, 1983.

[94] Mark S Granovetter. The strength of weak ties. *American journal of sociology*, 78(6): 1360–1380, 1973.

[95] Rajdeep Grewal, Gary L Lilien, and Girish Mallapragada. Location, location, location: How network embeddedness affects project success in open source systems. *Management science*, 52(7):1043–1056, 2006.

[96] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864, 2016.

[97] Mariam Guizani, Thomas Zimmermann, Anita Sarma, and Denae Ford. Attracting and retaining oss contributors with a maintainer dashboard. In *Proceedings of the 44th International Conference on Software Engineering (ICSE): Software Engineering in Society*, pages 36–40, 2022.

[98] Tomas Gustavsson. Managing the open source dependency. *Computer*, 53(2):83–87, 2020.

[99] Jungpil Hahn, Jae Yoon Moon, and Chen Zhang. Impact of social ties on open source project team formation. In *IFIP International Conference on Open Source Systems*, pages 307–317. Springer, 2006.

[100] Jungpil Hahn, Jae Yun Moon, and Chen Zhang. Emergence of new project teams from open source software developer networks: Impact of prior collaboration ties. *Information Systems Research*, 19(3):369–391, 2008.

[101] Tua Haldin-Herrgard. Difficulties in diffusion of tacit knowledge in organizations. *Journal of Intellectual capital*, 1(4):357–365, 2000.

[102] Joseph Hejderup. On the use of tests for software supply chain threats. In *ACM Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses*, pages 47–49, 2022.

[103] Joseph M Hilbe. *Negative binomial regression*. Cambridge University Press, 2011.

[104] James Howison, Keisuke Inoue, and Kevin Crowston. Social dynamics of free and open source team communications. In *IFIP Conference on Open Source Systems (OSS)*, pages 319–330. Springer, 2006.

[105] Yan Hu, Shanshan Wang, Yizhi Ren, and Kim-Kwang Raymond Choo. User influence analysis for github developer social networks. *Expert Systems with Applications*, 108: 108–118, 2018.

[106] Corey Jergensen, Anita Sarma, and Patrick Wagstrom. The onion patch: migration in open source ecosystems. In *Proceedings of the 19th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE)*, pages 70–80, 2011.

[107] Jing Jiang, David Lo, Jiahuan He, Xin Xia, Pavneet Singh Kochhar, and Li Zhang. Why and how developers fork what from whom in github. *Empirical Software Engineering*, 22: 547–578, 2017.

[108] Mitchell Joblin, Wolfgang Mauerer, Sven Apel, Janet Siegmund, and Dirk Riehle. From developer networks to verified communities: A fine-grained approach. In *Proceedings of the 37th International Conference on Software Engineering (ICSE)*, pages 563–573, 2015.

[109] Mitchell Joblin, Sven Apel, Claus Hunsen, and Wolfgang Mauerer. Classifying developers into core and peripheral: An empirical study on count and network metrics. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, pages 164–174. IEEE, 2017.

[110] Paul CD Johnson. Extension of nakagawa & schielzeth's r2glmm to random slopes models. *Methods in Ecology and Evolution*, 5(9):944–946, 2014.

[111] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M German, and Daniela Damian. The promises and perils of mining github. In *Proceedings of the 11st International Conference on Mining Software Repositories (MSR)*, pages 92–101, 2014.

[112] Rajdeep Kaur and Kuljit Kaur Chahal. Exploring factors affecting developer abandonment of open source software projects. *Journal of Software: Evolution and Process*, 34(9):e2484, 2022.

[113] Jymit Khondhu, Andrea Capiluppi, and Klaas-Jan Stol. Is it all lost? a study of inactive open source projects. In *IFIP International Conference on Open Source Systems (OSS)*, pages 61–79. Springer, 2013.

[114] Aniket Kittur and Robert E Kraut. Harnessing the wisdom of crowds in wikipedia: quality through coordination. In *Proceedings of the 2008 ACM conference on Computer supported cooperative work*, pages 37–46, 2008.

[115] Samara Klar, Yanna Krupnikov, John Barry Ryan, Kathleen Searles, and Yotam Shmargad. Using social media to promote academic research: Identifying the benefits of Twitter for sharing academic work. *PloS One*, 15(4):e0229446, 2020.

[116] David Krackhardt, N Nohria, and B Eccles. The strength of strong ties. *Networks in the knowledge economy*, 82, 2003.

[117] Sandeep Krishnamurthy. On the intrinsic and extrinsic motivation of free/libre/open source (floss) developers. *Knowledge, Technology & Policy*, 18(4):17–39, 2006.

[118] Charles W Krueger. Software reuse. *ACM Computing Surveys (CSUR)*, 24(2):131–183, 1992.

[119] Ilona Kuzmickaja, Xiaofeng Wang, Daniel Graziotin, Gabriella Dodero, and Pekka Abrahamsson. In need of creative mobile service ideas? forget adults and ask young children. *Sage Open*, 5(3):2158244015601719, 2015.

[120] Karim Lakhani and Robert Wolf. *Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects.* MIT Press, Cambridge, 2005.

[121] Hemank Lamba, Asher Trockman, Daniel Armanios, Christian Kästner, Heather Miller, and Bogdan Vasilescu. Heard it through the gitvine: an empirical study of tool diffusion across the npm ecosystem. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE)*, pages 505–517, 2020.

[122] Jey Han Lau and Timothy Baldwin. An empirical evaluation of doc2vec with practical insights into document embedding generation. *arXiv preprint arXiv:1607.05368*, 2016.

[123] Michael J Lee, Bruce Ferwerda, Junghong Choi, Jungpil Hahn, Jae Yun Moon, and Jinwoo Kim. Github developers use rockstars to overcome overflow of news. In *CHI'13 Extended Abstracts on Human Factors in Computing Systems*, pages 133–138. Nelson Education, 2013.

[124] Daniel Z Levin and Rob Cross. The strength of weak ties you can trust: The mediating role of trust in effective knowledge transfer. *Management science*, 50(11):1477–1490, 2004.

[125] Antonio Lima, Luca Rossi, and Mirco Musolesi. Coding together at scale: Github as a collaborative social network. In *Proceedings of the international AAAI conference on web and social media*, volume 8, pages 295–304, 2014.

[126] Johan Linåker, Hussan Munir, Krzysztof Wnuk, and Carl-Eric Mols. Motivating the contributions: An open innovation perspective on what to share as open source software. *Journal of Systems and Software*, 135:17–36, 2018.

[127] Yuan Long and Keng Siau. Social network structures in open source software development teams. *Journal of Database Management (JDM)*, 18(2):25–40, 2007.

[128] Yuan Long and Keng Siau. Impacts of social network structure on knowledge sharing in open source software development teams. *AMCIS 2008 Proceedings*, page 43, 2008.

[129] Jessica GY Luc, Michael A Archer, Rakesh C Arora, Edward M Bender, Arie Blitz, David T Cooke, Tamara Ni Hlci, Biniam Kidane, Maral Ouzounian, Thomas K Varghese Jr, et al. Does tweeting improve citations? one-year results from the tssmn prospective randomized trial. *The Annals of Thoracic Surgery*, 2020.

[130] Jeff Luszcz. How maverick developers can create risk in the software and iot supply chain. *Network Security*, 2017(8):5–7, 2017.

[131] Mark Lutter and Linus Weidner. Newcomers, betweenness centrality, and creative success: A study of teams in the board game industry from 1951 to 2017. *Poetics*, 87:101535, 2021.

[132] Yuxing Ma, Chris Bogart, Sadika Amreen, Russell Zaretzki, and Audris Mockus. World of code: an infrastructure for mining the universe of open source vcs data. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, pages 143–154. IEEE, 2019.

[133] Danaja Maldeniya, Ceren Budak, Lionel P Robert Jr, and Daniel M Romero. Herding a deluge of good samaritans: How github projects respond to increased attention. In *Proceedings of The Web Conference 2020*, pages 2055–2065, 2020.

[134] Edwin Mansfield. Patents and innovation: an empirical study. *Management Science*, 32(2): 173–181, 1986.

[135] Jennifer Marlow, Laura Dabbish, and Jim Herbsleb. Impression formation in online peer production: activity traces and personal profiles in GitHub. In *ACM Conference on Computer Supported Cooperative Work & Social Computing (CSCW)*, pages 117–128, 2013.

[136] Tadej Matek and Svit Timej Zebec. Github open source project recommendation system.

arXiv preprint arXiv:1602.02594, 2016.

[137] Christopher Mendez, Hema Susmita Padala, Zoe Steine-Hanson, Claudia Hilderbrand, Amber Horvath, Charles Hill, Logan Simpson, Nupoor Patil, Anita Sarma, and Margaret Burnett. Open source barriers to entry, revisited: A sociotechnical perspective. In *International Conference on Software Engineering (ICSE)*, pages 1004–1015, 2018.

[138] Thorsten Merten, Bastian Mager, Paul Hübner, Thomas Quirchmayr, Barbara Paech, and Simone Bürsner. Requirements communication in issue tracking systems in four open-source projects. In *REFSQ workshops*, pages 114–125, 2015.

[139] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[140] Courtney Miller, David Gray Widder, Christian Kästner, and Bogdan Vasilescu. Why do people give up flossing? a study of contributor disengagement in open source. In *IFIP International Conference on Open Source Systems*, pages 116–129, 2019.

[141] Samim Mirhosseini and Chris Parnin. Can automated pull requests encourage software developers to upgrade out-of-date dependencies? In *International Conference on Automated Software Engineering (ASE)*, pages 84–94. IEEE, 2017.

[142] Audris Mockus. Large-scale code reuse in open source software. In *First International Workshop on Emerging Trends in FLOSS Research and Development (FLOSS'07: ICSE Workshops 2007)*, pages 7–7. IEEE, 2007.

[143] Audris Mockus, Roy T Fielding, and James Herbsleb. A case study of open source software development: the apache server. In *Proceedings of the 22nd international conference on Software engineering*, pages 263–272, 2000.

[144] Audris Mockus, David M Weiss, and Ping Zhang. Understanding and predicting effort in software projects. In *Proceedings of the 25th International Conference on Software Engineering (ICSE)*, pages 274–284, 2003.

[145] Lukas Moldon, Markus Strohmaier, and Johannes Wachs. How gamification affects software developers: Cautionary evidence from a natural experiment on GitHub. In *International Conference on Software Engineering (ICSE)*, pages 549–561. IEEE, 2021.

[146] Suhaib Mujahid, Diego Elias Costa, Rabe Abdalkareem, Emad Shihab, Mohamed Ayme Saied, and Bram Adams. Towards using package centrality trend to identify packages in decline. *arXiv preprint arXiv:2107.10168*, 2021.

[147] Paul T Munroe. Cognitive balance theory (heider). *The Blackwell encyclopedia of sociology*, 2007.

[148] Dawn Nafus. 'patches don't have gender': What is not open in open source software. *New Media & Society*, 14(4):669–683, 2012.

[149] Shinichi Nakagawa and Holger Schielzeth. A general and simple method for obtaining r2 from generalized linear mixed-effects models. *Methods in Ecology and Evolution*, 4(2): 133–142, 2013.

[150] Kumiyo Nakakoji, Yasuhiro Yamamoto, Yoshiyuki Nishinaka, Kouichi Kishida, and Yunwen Ye. Evolution patterns of open-source software systems and communities. In *Pro-*

ceedings of the international workshop on Principles of software evolution, pages 76–85, 2002.

[151] Jagadeesh Nandigam, Venkat N Gudivada, and Abdelwahab Hamou-Lhadj. Learning software engineering principles using open source software. In *2008 38th Annual Frontiers in Education Conference*, pages S3H–18. IEEE, 2008.

[152] Andy Neely and Jasper Hii. Innovation and business performance: a literature review. *The Judge Institute of Management Studies, University of Cambridge*, pages 0–65, 1998.

[153] Linus Nyman and Juho Lindman. Code forking, governance, and sustainability in open source software. *Technology Innovation Management Review*, 3(1), 2013.

[154] Andrei Oghina, Mathias Breuss, Manos Tsagkias, and Maarten De Rijke. Predicting imdb movie ratings using social media. In *European Conference on Information Retrieval*, pages 503–507. Springer, 2012.

[155] Cassandra Overney, Jens Meinicke, Christian Kästner, and Bogdan Vasilescu. How to not get rich: An empirical study of donations in open source. In *International Conference on Software Engineering (ICSE)*, pages 1209–1221, 2020.

[156] Jagdish K Patel, CH Kapadia, and Donald Bruce Owen. *Handbook of statistical distributions*. M. Dekker, 1976.

[157] Gang Peng. Co-membership, networks ties, and knowledge flow: An empirical investigation controlling for alternative mechanisms. *Decision Support Systems*, 118:83–90, 2019.

[158] Gang Peng, Jifeng Mu, and C Anthony Di Benedetto. Learning and open source software license choice. *Decision Sciences*, 44(4):619–643, 2013.

[159] José C Pinheiro and Douglas M Bates. Linear mixed-effects models: basic concepts and examples. *Mixed-effects models in S and S-Plus*, pages 3–56, 2000.

[160] Huilian Sophie Qiu, Yucen Lily Li, Susmita Padala, Anita Sarma, and Bogdan Vasilescu. The signals that potential contributors look for when choosing open-source projects. *Proceedings of the ACM on Human-Computer Interaction*, 3(CSCW):1–29, 2019.

[161] Huilian Sophie Qiu, Alexander Nolte, Anita Brown, Alexander Serebrenik, and Bogdan Vasilescu. Going farther together: The impact of social capital on sustained participation in open source. In *International Conference on Software Engineering (ICSE)*, pages 688–699. IEEE, 2019.

[162] Israr Qureshi and Yulin Fang. Socialization in open source software projects: A growth mixture modeling approach. *Organizational Research Methods*, 14(1):208–238, 2011.

[163] Karthik Rajkumar, Guillaume Saint-Jacques, Iavor Bojinov, Erik Brynjolfsson, and Sinan Aral. A causal test of the strength of weak ties. *Science*, 377(6612):1304–1310, 2022.

[164] Naveen Raman, Minxuan Cao, Yulia Tsvetkov, Christian Kästner, and Bogdan Vasilescu. Stress and burnout in open source: Toward finding, understanding, and mitigating unhealthy interactions. In *International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*, pages 57–60, 2020.

[165] Sam Ransbotham and Gerald C Kane. Membership turnover and collaboration success in

online communities: Explaining rises and falls from grace in wikipedia. *Mis Quarterly*, pages 613–627, 2011.

[166] Dirk Riehle, Philipp Riemer, Carsten Kolassa, and Michael Schmidt. Paid vs. volunteer work in open source. In *2014 47th Hawaii International Conference on System Sciences*, pages 3286–3295, 2014.

[167] Peter C Rigby, Yue Cai Zhu, Samuel M Donadelli, and Audris Mockus. Quantifying and mitigating turnover-induced knowledge loss: case studies of Chrome and a project at Avaya. In *International Conference on Software Engineering (ICSE)*, pages 1006–1016. IEEE, 2016.

[168] Marian-Andrei Rizoiu, Lexing Xie, Scott Sanner, Manuel Cebrian, Honglin Yu, and Pascal Van Hentenryck. Expecting to be hip: Hawkes intensity processes for social media popularity. In *International Conference on World Wide Web*, WWW, 2017.

[169] Elizabeth Rosenthal. Social networks and team performance. *Team Performance Management: An International Journal*, 3(4):288–294, 1997.

[170] Suman Deb Roy, Tao Mei, Wenjun Zeng, and Shipeng Li. Towards cross-domain learning for social video popularity prediction. *IEEE Transactions on Multimedia*, 15(6):1255–1267, 2013.

[171] AS Saabith, MMM Fareez, and T Vinothraj. Python current trend applications-an overview. *International Journal of Advance Engineering and Research Development*, 6(10), 2019.

[172] Kazumi Saito, Ryohei Nakano, and Masahiro Kimura. Prediction of information diffusion probabilities for independent cascade model. In *International conference on knowledge-based and intelligent information and engineering systems*, pages 67–75. Springer, 2008.

[173] Carlos Santos, George Kuk, Fabio Kon, and John Pearson. The attraction of contributors in free and open source software projects. *The Journal of Strategic Information Systems*, 22 (1):26–45, 2013.

[174] Fabio Santos, Bianca Trinkenreich, João Felipe Pimentel, Igor Wiese, Igor Steinmacher, Anita Sarma, and Marco A Gerosa. How to choose a task? mismatches in perspectives of newcomers and existing contributors. In *2022 International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 114–124, 2022.

[175] Joseph A Schumpeter and Richard Swedberg. *The theory of economic development*. Routledge, 2021.

[176] Joseph A Schumpeter et al. *Business Cycles: A Theoretical, Historical and Statistical Analysis of the Capitalist Process*, volume 1. Mcgraw-hill New York, 1939.

[177] André Luis Schwerz, Rafael Liberato, Igor Scaliante Wiese, Igor Steinmacher, Marco Aurélio Gerosa, and João Eduardo Ferreira. Prediction of developer participation in issues of open source projects. In *2012 Brazilian Symposium on Collaborative Systems*, pages 109–114, 2012.

[178] Luiz Philipe Serrano Alves, Igor Scaliante Wiese, Ana Paula Chaves, and Igor Steinmacher. How to find my task? chatbot to assist newcomers in choosing tasks in oss projects. In *International Workshop on Chatbot Research and Design*, pages 90–107. Springer, 2021.

[179] Pankaj Setia, Balaji Rajagopalan, Vallabh Sambamurthy, and Roger Calantone. How peripheral developers contribute to open-source software development. *Information Systems Research*, 23(1):144–163, 2012.

[180] Sonali K Shah. Motivation, governance, and the viability of hybrid forms in open source software development. *Management science*, 52(7):1000–1014, 2006.

[181] Jyoti Sheoran, Kelly Blincoe, Eirini Kalliamvakou, Daniela Damian, and Jordan Ell. Understanding "watchers" on github. In *International Conference on Mining Software Repositories (MSR)*, pages 336–339, 2014.

[182] Leif Singer, Fernando Figueira Filho, and Margaret-Anne Storey. Software engineering at the speed of light: how developers stay current using twitter. In *Proceedings of the 36th International Conference on Software Engineering*, pages 211–221, 2014.

[183] Param Vir Singh and Corey Phelps. Networks, social influence, and the choice among competing innovations: Insights from open source software licenses. *Information Systems Research*, 24(3):539–560, 2013.

[184] Param Vir Singh, Yong Tan, and Vijay Mookerjee. Network effects: The influence of structural capital on open source project success. *Mis Quarterly*, pages 813–829, 2011.

[185] Stanley F Slater and Jakki J Mohr. Successful development and commercialization of technological innovation: Insights based on strategy type. *Journal of product innovation management*, 23(1):26–33, 2006.

[186] KR Srinath. Python–the fastest growing programming language. *International Research Journal of Engineering and Technology*, 4(12):354–357, 2017.

[187] Igor Steinmacher, Marco Aurélio Gerosa, and David Redmiles. Attracting, onboarding, and retaining newcomer developers in open source software projects. In *Workshop on Global Software Development in a CSCW Perspective*, 2014.

[188] Igor Steinmacher, Tayana Conte, Marco Aurélio Gerosa, and David Redmiles. Social barriers faced by newcomers placing their first contribution in open source software projects. In *Proceedings of the 18th ACM conference on Computer supported cooperative work & social computing*, pages 1379–1392, 2015.

[189] Igor Steinmacher, Marco Aurelio Graciotto Silva, Marco Aurelio Gerosa, and David F Redmiles. A systematic literature review on the barriers faced by newcomers to open source software projects. *Information and Software Technology*, 59:67–85, 2015.

[190] Margaret-Anne Storey. The evolution of the social programmer. In *Proceedings of the 9th IEEE Working Conference on Mining Software Repositories*, pages 140–140, 2012.

[191] Margaret-Anne Storey, Christoph Treude, Arie van Deursen, and Li-Te Cheng. The impact of social media on software engineering practices and tools. In *Proceedings of the FSE/SDP workshop on Future of software engineering research*, pages 359–364, 2010.

[192] Margaret-Anne Storey, Leif Singer, Brendan Cleary, Fernando Figueira Filho, and Alexey Zagalsky. The (r) evolution of social media in software engineering. *Future of software engineering proceedings*, pages 100–116, 2014.

[193] Margaret-Anne Storey, Alexey Zagalsky, Fernando Figueira Filho, Leif Singer, and

Daniel M German. How social and communication channels shape and challenge a participatory culture in software development. *IEEE Transactions on Software Engineering*, 43 (2):185–204, 2016.

[194] Ashok Subramanian. Innovativeness: redefining the concept. *Journal of engineering and technology management*, 13(3-4):223–243, 1996.

[195] Xiaobing Sun, Wenyuan Xu, Xin Xia, Xiang Chen, and Bin Li. Personalized project recommendation on github. *Science China Information Sciences*, 61:1–14, 2018.

[196] Lin Tan, Chen Liu, Zhenmin Li, Xuanhui Wang, Yuanyuan Zhou, and Chengxiang Zhai. Bug characteristics in open source software. *Empirical software engineering*, 19(6):1665–1705, 2014.

[197] Xin Tan, Minghui Zhou, and Zeyu Sun. A first look at good first issues on github. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE)*, pages 398–409, 2020.

[198] Yong Tan, Vijay Mookerjee, and Param Singh. Social capital, structural holes and team composition: Collaborative networks of the open source software community. *ICIS 2007 Proceedings*, page 155, 2007.

[199] Christopher Glen Thompson, Rae Seon Kim, Ariel M Aloe, and Betsy Jane Becker. Extracting the variance inflation factor and other multicollinearity diagnostics from typical regression results. *Basic and Applied Social Psychology*, 39(2):81–90, 2017.

[200] Marco Tortoriello, Bill McEvily, and David Krackhardt. Being a catalyst of innovation: The role of knowledge diversity and network closure. *Organization Science*, 26(2):423–438, 2015.

[201] Asher Trockman, Shurui Zhou, Christian Kästner, and Bogdan Vasilescu. Adding sparkle to social coding: an empirical study of repository badges in the npm ecosystem. In *International Conference on Software Engineering (ICSE)*, pages 511–522, 2018.

[202] Jason Tsay, Laura Dabbish, and James Herbsleb. Influence of social and technical factors for evaluating contribution in github. In *International Conference on Software Engineering (ICSE)*, pages 356–366, 2014.

[203] Brian Uzzi, Satyam Mukherjee, Michael Stringer, and Ben Jones. Atypical combinations and scientific impact. *Science*, 342(6157):468–472, 2013.

[204] Marat Valiev, Bogdan Vasilescu, and James Herbsleb. Ecosystem-level determinants of sustained activity in open-source projects: A case study of the pypi ecosystem. In *Joint Meeting on the Foundations of Software Engineering (ESEC/FSE)*, pages 644–655, 2018.

[205] Anthony FJ Van Raan. Sleeping beauties in science. *Scientometrics*, 59(3):467–472, 2004.

[206] Bogdan Vasilescu, Vladimir Filkov, and Alexander Serebrenik. Stackoverflow and github: Associations between software development and crowdsourced knowledge. In *2013 International Conference on Social Computing*, pages 188–195. IEEE, 2013.

[207] Bogdan Vasilescu, Alexander Serebrenik, Mathieu Goeminne, and Tom Mens. On the variation and specialisation of workload—a case study of the gnome ecosystem community. *Empirical Software Engineering*, 19(4):955–1008, 2014.

[208] Bogdan Vasilescu, Daryl Posnett, Baishakhi Ray, Mark GJ van den Brand, Alexander Serebrenik, Premkumar Devanbu, and Vladimir Filkov. Gender and tenure diversity in github teams. In *Proceedings of the 33rd annual ACM conference on human factors in computing systems*, pages 3789–3798, 2015.

[209] Bogdan Vasilescu, Kelly Blincoe, Qi Xuan, Casey Casalnuovo, Daniela Damian, Premkumar Devanbu, and Vladimir Filkov. The sky is not the limit: Multitasking across github projects. In *Proceedings of the 38th International Conference on Software Engineering (ICSE)*, pages 994–1005. ACM, 2016.

[210] Dennis Verhoeven, Jurriën Bakker, and Reinhilde Veugelers. Measuring technological novelty with patent-based indicators. *Research policy*, 45(3):707–723, 2016.

[211] Eric Von Hippel. Learning from open-source software. *MIT Sloan management review*, 42 (4):82–86, 2001.

[212] Georg Von Krogh, Sebastian Spaeth, and Karim R Lakhani. Community, joining, and specialization in open source software innovation: a case study. *Research policy*, 32(7): 1217–1241, 2003.

[213] James Walden. The impact of a major security event on an open source project: The case of openssl. In *International Conference on Mining Software Repositories (MSR)*, pages 409–419, 2020.

[214] Mairieli Wessel, Bruno Mendes De Souza, Igor Steinmacher, Igor S Wiese, Ivanilton Polato, Ana Paula Chaves, and Marco A Gerosa. The power of bots: Characterizing and understanding bots in oss projects. *Proceedings of the ACM on Human-Computer Interaction*, 2(CSCW):1–19, 2018.

[215] Joel West and Scott Gallagher. Challenges of open innovation: the paradox of firm investment in open-source software. *R&d Management*, 36(3):319–331, 2006.

[216] Joel West and Scott Gallagher. Patterns of open innovation in open source software. *Open Innovation: researching a new paradigm*, 235(11):82–106, 2006.

[217] Joel West and Karim R Lakhani. Getting clear about communities in open innovation. In *Online Communities and Open Innovation*, pages 109–117. Routledge, 2014.

[218] David C Wheeler and Morton E O'Kelly. Network topology and city accessibility of the commercial internet. *The Professional Geographer*, 51(3):327–339, 1999.

[219] Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52, 1987.

[220] Jeffrey M Wooldridge. *Introductory econometrics: A modern approach*. Nelson Education, 2016.

[221] Jing Wu, Khim-Yong Goh, and Qian Tang. Investigating success of open source software projects: A social network perspective. *ICIS 2007 Proceedings*, page 105, 2007.

[222] Lingfei Wu, Dashun Wang, and James A Evans. Large teams develop and small teams disrupt science and technology. *Nature*, 566(7744):378–382, 2019.

[223] Kazuhiro Yamashita, Shane McIntosh, Yasutaka Kamei, Ahmed E Hassan, and Naoyasu

Ubayashi. Revisiting the applicability of the pareto principle to core development teams in open source software projects. In *International Workshop on Principles of Software Evolution (IWPSE)*, pages 46–55, 2015.

[224] Heng-Li Yang and Jih-Hsin Tang. Team structure and team performance in is development: a social network perspective. *Information & management*, 41(3):335–349, 2004.

[225] Xuan Yang, Daning Hu, and Davison M Robert. How microblogging networks affect project success of open source software development. In *2013 46th Hawaii International Conference on System Sciences*, pages 3178–3186. IEEE, 2013.

[226] Yunwen Ye and Kouichi Kishida. Toward an understanding of the motivation of open source software developers. In *Proceedings of the 25th International Conference on Software Engineering (ICSE)*, pages 419–429, 2003.

[227] Chengxi Zang, Peng Cui, Chaoming Song, Christos Faloutsos, and Wenwu Zhu. Quantifying structural patterns of information cascades. In *Proceedings of the 26th International Conference on World Wide Web Companion*, pages 867–868, 2017.

[228] Wenbin Zhang and Steven Skiena. Improving movie gross prediction through news analysis. In *2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology*, volume 1, pages 301–304. IEEE, 2009.

[229] Xunhui Zhang, Tao Wang, Gang Yin, Cheng Yang, Yue Yu, and Huaimin Wang. Devrec: a developer recommendation system for open source repositories. In *Mastering Scale and Complexity in Software Reuse: 16th International Conference on Software Reuse, ICSR 2017, Salvador, Brazil, May 29-31, 2017, Proceedings 16*, pages 3–11. Springer, 2017.

[230] Minghui Zhou and Audris Mockus. Does the initial environment impact the future of developers? In *Proceedings of the 33rd International Conference on Software Engineering (ICSE)*, pages 271–280, 2011.

[231] Minghui Zhou and Audris Mockus. What make long term contributors: Willingness and opportunity in oss community. In *Proceedings of the 34th International Conference on Software Engineering (ICSE)*, pages 518–528, 2012.

[232] Minghui Zhou and Audris Mockus. Who will stay in the floss community? modeling participant's initial behavior. *IEEE Transactions on Software Engineering*, 41(1):82–99, 2014.

[233] Shurui Zhou, Bogdan Vasilescu, and Christian Kästner. How has forking changed in the last 20 years? a study of hard forks on github. In *International Conference on Software Engineering (ICSE)*, pages 445–456, 2020.