SUSTAINING OPEN SOURCE SOFTWARE PRODUCTION: AN EMPIRICAL ANALYSIS THROUGH THE LENS OF MICROECONOMICS

by

Samuel Jospeh Boysel

A Dissertation Presented to the
FACULTY OF THE USC GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the
Requirements for the Degree
DOCTOR OF PHILOSOPHY
(ECONOMICS)

December 2022

Copyright 2022 Samuel Jospeh Boysel

Dedication

To everyone.

Acknowledgements

Thank you to everyone.

Table of Contents

List of Tables

List of Figures

Abstract

In this manuscript, we explore microeconomic behavior shaping the production of open source software (OSS). We fortify our analysis with economic structure to guide our narrative and assess our hypotheses empirically, filling important gaps in the literature on the supply side of markets for OSS goods. Our motivation is rooted in a desire to better understand how various microeconomic phenomena influence sustained development of widely used OSS infrastructure. Following an introduction in Chapter 1, our contribution is divided into two distinct chapters.

In Chapter 2, we examine the extent to which peer effects influence the private provision of public goods. In the case of public information goods, peer contribution may facilitate or otherwise incentivize further contribution from others, effectively subsidizing private provision. We first utilize a reduced form approach to derive causal estimates of net peer effects in public goods contribution by exploiting a peers-of-peers identification strategy. We next develop a structural model of peer-influenced public good provision that both (1) separates extensive and intensive margin contribution decisions and (2) decomposes contribution into marginal private benefits and costs. We apply these methodologies using a sample of peer contribution histories for 2,287 OSS projects hosted on the GitHub collaboration platform. Both reduced form and structural approaches suggest peer effects are much stronger along

the extensive margin than the intensive margin. Contemporaneous intensive margin effects, while heterogenous across time and projects, are small and centered around zero, suggesting that strategic complementarity and substitution in peer contribution likely offset one another. Our counterfactual analysis suggests (extensive margin) peer effects account for nearly 56% of cumulative aggregate contribution for our sample, which translates to a value-added of 1-1.5 million software developer labor hours. These results support the notion that OSS is largely developed by disproportionate efforts from smaller groups of dedicated core maintainers, who integrate incremental contributions from the wider community, and casts doubt on the potential for peer effects alone to deliver sustained maintenance labor to individual projects.

In Chapter 3, we turn our attention to the formation of software dependency networks. Developers of software projects can leverage the functionality of existing open source projects. This practice can potentially lower the cost of development albeit at the inherent risk of relying on external components. A "downstream" project maintainer can choose to "import" elements of an "upstream" project to outsource functionality, but is uncertain how future changes in this dependency project may expose her own project to software faults or vulnerabilities. Software dependency networks therefore represent a "digital supply chain", an ecosystem of interdependent public goods that confer an intricate set of both positive and negative externalities for project maintainers and end users. Focusing on microeconomic fundamentals of the dependency management problem faced by the risk averse project maintainer, we use both reduced form and structural approaches to study how dependency networks create value, what forces shape their formation, and how individual behavior can influence the robustness of equilibrium network structure. We use a sample of open source software

projects from the Node.js JavaScript packaging ecosystem for which contribution and dependency formation decisions are observed in real-time. Finally, we consider several policy interventions that can improve equilibrium welfare. In particular, we find that removing less that 1% of core projects can reduce aggregate project quality by more than 5% for the remaining peers.

Chapter 1

Introduction

ch:introduction

1.1 Motivation

The production of Open Source Software (OSS), a special class of public information goods (**varian2000buying**), is both intriguing and puzzling from an economic perspective.[1] Most typically, OSS is incrementally and collaboratively developed by software developers willing to forgo property rights over their contributions.[2] In doing so, these efforts by contributors provide a rich and dynamic space of open technology available to the wider public.[3] In theory, allowing open access to a project's[4] source code and encouraging contributions from the wider community has attractive benefits for the individual developer, by potentially re-distributing the maintenance workload, encouraging more diverse perspectives over design

[1]Throughout this manuscript, we will refer to open source software by the acronym OSS. By *software*, we mean a set of digital instructions for a machine to perform a task.

[2]Invoking a prevailing definition, we say that software is open source if (1) users can freely access, (2) copy or distribute, and (3) make modifications to the software's underlying codebase (**perens1999open**). See **laurent2004understanding** for an overview of the specifics of OSS licensing.

[3]As a subset of digital technology, software goods can be credited for a considerable share of innovation and economic development over the past half-century (**goldfarb2015economic**).

[4]Throughout this study, we will reference the atomistic unit for an OSS codebase as a *project*, a collection of code typically organized in a repository designed for a specific purpose. For the purposes of this work, we will occasionally use the terms *codebase*, *package*, *library*, or *repository* as synonyms for an OSS project.

input, and facilitating a public audit of the codebase.[5] A form of "digital capital" or infrastructure[6], the availability of OSS allows developers to integrate subsets of functionality into more complex applications and greatly mitigates the need to "reinvent the wheel" in subsequent software development.[7] Both ubiquitous and characterized by widespread adoption, OSS has become an integral component of software development and technological innovation. It is estimated that modern software projects derive 70–90% of their functionality from open source code (**nagle_census_2022**).[8] In this way, open software ecosystems operate as modular public goods production networks, characterized by intricate and dynamic sociotechnical interactions both within and across projects and individual contributors.

Despite the attractive properties of OSS goods, there remains concern over the sustained maintenance of widely used as public good infrastructure (**eghbal2016roads**; **eghbal2020working**). Case studies[9] in problems arising from the under-maintenance of such OSS projects motivate our interest in exploring how economic behavior shapes production patterns. In an empirical analysis grounded in microeconomic theory, we seek to understand how peer effects create value in OSS production networks and what these forces imply for sustainability.

[5] To paraphrase the oft used adage of **raymond1999cathedral**, "with enough eyes, all bugs are shallow."

[6] See (**eghbal2016roads**) for a description of how OSS has become inextricably linked with and depended upon by contemporary software applications.

[7] Creating software requires human capital and labor but has the distinct advantage of being replicated at a marginal cost of zero.

[8] We provide further details on the prevalence of OSS usage and dependence in Chapter 2, Sections 2.1 and 2.2 and Chapter 3, Section 3.1.

[9] See for example, **mutton_half_2014**, **schlueter_kik_2016**, **carey_heartbleeds_2017**, **cfpb_equifax_2022**, and **ftc_equifax_2022**.

1.1.1 Economic Principles embedded in OSS Production

Although the benefits and importance of OSS goods appear to be significant, open software suffers from problems common to most classes of public goods: the presence of positive externalities, a non-excludable nature, and the absence of narrowly defined property rights preclude the establishment of a private market. In terms of production, OSS can clearly be subject to free-ridership[10] and may therefore suffer from under-provision from private contributors in the classic sense (**samuelson1954pure**). However, the ubiquitous nature and widespread uptake of OSS in the present day seemingly assuage concerns of under-provision in aggregate, suggesting strong individual benefits for motivated contributors are at work (**bergstrom1986private**; **lerner2002some**). In this work, we therefore instead focus on more nuanced problems surrounding *under-maintenance*: what happens when an integral OSS component, widely used and depended upon by many, lacks an adequate level of maintenance bandwidth to ensure security and proper functionality?[11] By encouraging additional contribution from peers or improving their productivity, can maintainers structure self-sustaining projects?

To guide us toward understanding mechanisms at the heart of production and maintenance concerns, a deeper look into contributor behavior within OSS ecosystems reveals complex interactions that can be explained by economic theory. Beyond any intrinsic desire

[10]By "free-riding", we mean that users of a software project can enjoy the benefits of development efforts made by their peers while contributing relatively little, or not at all, themselves. In a broader sense, the wider population of technology consumers often use OSS at least indirectly and do not contribute whatsoever. Throughout this study, we will emphasize a narrower scope for potential free-riders by looking within the subpopulation of developers who make at least a single contribution to OSS projects in our empirical sample.

[11]**eghbal2020working** provides an excellent overview of modern OSS development and places particular emphasis on the problems faced by oversubscribed projects.

for pure altruism, a common motivation for releasing software code to the public under a permissive license is to encourage the use and adoption of the codebase.[12] Project maintainers benefit from putting the code into the hands of many by opening a pathway for community contributions, ranging from the addition of new features to identifying software faults, which serve to ultimately benefit the project's quality. A common contribution pattern can be described as follows. A maintainer of an OSS project first releases their code, making it publicly accessible and distributing it under a permissive OSS license. Interested parties may then inspect the source code, use and modify it, and potentially contribute changes. To contribute, the developer formally requests the original maintainer integrates their proposed changeset into the codebase of the original project.[13][14]

Individuals who directly engage in the OSS space are confronted with many choices: which projects to use, which to contribute to, and how much to contribute. By choosing which external projects to use as dependencies, maintainers balance expediting the development of new features at the risk of relying on codebases outside their direct control.[15] Even after a maintainer decides to utilize software dependencies, she must choose which in particular to source based on their observable features. Users of software projects will contribute to projects if they find a net benefit in doing so. They must then determine an optimal level of contribution effort to allocate, conditional on the choices made by their peers. A common theme we explore throughout this work is the complex substitution patterns that arise as

[12] This can be seen as a special case of "impure" altruism (**andreoni1990impure**).

[13] This is known as either a *merge* or, in the development terminology popularized by software forges such as GitHub, a *pull request*.

[14] Open source theoretically allows any given codebase to be *forked* into any number of alternative versions. It must therefore be the case that attractive benefits exist for users to center their development efforts around a canonical version of a project.

[15] "Outsourcing" functionality can also be seen as a special case of the firm's "make versus buy" dilemma addressed in the literature (**coase1937nature**; **williamson1975markets**; **williamson1985economic**; **grossman1986costs**).

consequences of these choices. It may be the case that some OSS participants opt to "free-ride" on the outsized efforts of dominant contributors. Alternatively, some projects used as dependencies may greatly reduce the amount of work required in a dependent project while others may actually facilitate further development.

Externalities proliferate within OSS ecosystems, both between projects and individual contributors. The core inquiry in our analysis in Chapter 2 assess the extent to which contributors influence both contribution decisions and the productivity levels of their peers. Perhaps it is the case that some contributions induce peers to make additional follow-on contributions. This may arise when certain contributions lower the cost of subsequent development efforts or add critical features that encourage additional work. On the other hand, outsized contributions by dominant insiders may dissuade or "crowd out" contributions from outsiders, leading to increased free-ridership. It is unclear *ex ante* whether peers find the contributions of peers to be net substitutes or complements, which determines a distribution of the contribution workload within projects. Our interest in peer effects continues in Chapter 3, where we explore influences between interrelated software projects. As upstream dependencies can serve a portion of functionality to any number of downstream dependents simultaneously, network externalities exist by way of these dependencies projects exerting some influence over the quality and contribution costs for dependents. This arrangement has the advantage of efficiency: organizing functionality into modular software packages reduces duplicated efforts and can lower upfront costs of project development. As public information goods, dependency projects expose their functionality in a non-rival and non-excludable fashion, giving rise to economies of scale. However, relying on external software under some level of development imposes additional costs, ranging from mild maintenance costs to exposure to exploits or software faults, and ultimately exposing dependent projects to external, network-mediated shocks.

1.1.2 Related Work

Despite OSS serving as a foundational element of modern software development, the microeconomic fundamentals of the supply side have thus far received surprisingly little attention from the empirical economic literature. Some noteworthy exceptions include theoretical contributions by **lerner2002some**; **lerner2005economics** and **athey2014dynamics** and the empirical exploration of innovation by **fershtman2011direct**. This absence motivates the focus of the body of work contained in this manuscript. Specifically, we seek to examine microeconomic forces that shape contributor behavior and ultimately drive the production of OSS in equilibrium. Our analysis of peer effects between contributors in Chapter 2 draws from a literature of private production of public goods by heterogeneous agents (**bergstrom1986private**; **jacobsen2017public**). Our study of software dependency formation in Chapter 3 is related to descriptive studies of OSS ecosystems (**decan2018evolution**; **decan2019package**; **decan2019empirical**), OSS supply chain valuation (**keller2018opportunities**; **robbins2018open**), and both network innovation (**acemoglu2016innovation**) and formation under risk (**blume2013network**; **kovavrik2014risk**;

elliott2022supply).[16] More specifics links between the work in this manuscript and the literature are made in Chapter 2 Section 2.3 and Chapter 3 Section 3.2.

1.2 Empirical Setting

A distinct advantage for the empirical researcher studying OSS production in the modern era is the wealth of available observable data on both contributor behavior and project characteristics. Software forges, collaborative online platforms that serve as hubs for OSS development, have gained immense popularity.[17][18] Maintainers can host project codebases, share documentation, communicate or coordinate design decisions, and easily integrate contribution requests from the wider community in a unified public forum.[19] Most pertinent to the present study, these platforms chronicle publicly available usage data at precise temporal granularity[20] and distribute them in digestible formats (**Gousi13**). Furthermore, software development tools facilitate empirical analysis by preserving a timestamped record of project

[16]Additionally, we refer the interested reader to broader discussions of OSS development and history covered by previous authors: the transition to stable OSS adoption (**fitzgerald2006transformation**), a case study of the development of the Apache Server (**mockus2000case**), general OSS history (**bretthauer2001open**; **lerner2005economics**), understanding OSS development (**feller2002understanding**; **von2003special**; **fogel2005producing**), developer motivations (**lerner2002some**; **lakhani2003hackers**), general discussions of open innovation (**chesbrough2003open**; **von2006democratizing**) and commons-based peer production (**benkler2002coase**; **benkler2006commons**; **benkler2006wealth**).

[17]Noteworthy examples include GitHub, SourceForge, Bitbucket, and GitLab. See **squire2012describing** for an overview of the software forge ecosystem.

[18]As of October 2022, GitHub plays host to over 98 million users contributing across 43 million public repositories (**ghsize2022**). This can be seen as a lower bound estimate of platform activity as some share of GitHub activity occurs in private, unlisted repositories.

[19]See Figure 2.B.1 for an example of the GitHub repository page for Twitter's `bootstrap` library.

[20]Since observable actions by software developers are recorded and timestamped on platforms like software forges, OSS characteristics can be observed down to seconds or even milliseconds.

development for each line of code authored by a contributor. Software version control[21] systems allow the researcher to "rewind" the state of a software project to a specific moment in time. By exploiting these advantageous sources of data, researchers can derive refined and disaggregated sociotechnical measures for the evolution of project development activity within the OSS ecosystem across time: contribution levels, the relative popularity of projects, dependency relationships, technical features of individual projects, and other characteristics.[22]

While the sheer volume and granularity of public OSS data facilitate much of the methodology used in this study, it is wise to temper expectations by noting some key unobservables from the perspective of the econometrician. First and foremost, measuring uptake, usage, or valuation of OSS projects in the broadest sense remains elusive. In the absence of a well-defined market, it is at the very least challenging to completely observe a comprehensive measure of the public's willingness to pay for OSS public goods. Some simple yet imperfect proxies for OSS uptake exist. The researcher can gauge popular interest in a project via download counts (**fershtman2011direct**), contribution activity (**kalliamvakou2014promises**), search term frequency[23], or by measuring the extent to which other projects rely on certain OSS packages (**robbins2018open**).[24] Another important set of unobservables are characteristics of individual contributors. By nature of version control systems, developers at

[21]For example, `git`, `svn`, `mercurial`, and `bazaar` are examples of popular version control systems. Among these, `git` is the most widely used by far (**so_survey_2022**).

[22]See Chapter 2 Section 2.4 and Chapter 3 Section 3.4 for more specifics on OSS data and observable metrics utilized in this study.

[23]For example, how prevalent individuals search for, discuss, or mention certain OSS technologies across various internet forums.

[24]Note that none of these approaches truly capture a willingness to pay for OSS goods by end users. By using the revealed preferences of OSS contributors, a key contribution of Chapter 2 is to characterize peer effects in terms of labor development costs. See Section 1.3.

a minimum contribute under a chosen email address and can potentially disclose some information about themselves on collaboration platforms.[25] Software forges can observe a contributor's location when they interact with the platform, but this data is typically not publicly available.[26] Despite these these disadvantages, the observable data gives the researcher an incredibly detailed level of insight into the OSS production process and allows us to place these dynamics into the context of economic theory.

1.3 Contribution

sec:intro-contrib

In this manuscript, we develop microeconomic structure for two distinct phenomena within OSS production and discuss the implications our empirical analysis suggests for OSS sustainability. We exploit both disaggregated data on OSS contributor behavior and the structure of sociotechnical networks[27] to develop novel empirical contributions, filling a gap in the microeconomic literature on OSS. To the best of our knowledge, the work in this manuscript represents the first empirical tests of key theoretical hypotheses on OSS production dynamics (**athey2014dynamics**).

Our contribution is organized into two distinct chapters.[28] In Chapter 2, we study the role of peer effects in shaping equilibrium contribution levels to OSS. We explore the extent to which the contributions of peers can influence both an individual's contribution level

[25]Such as their true name, geographic location, firm affiliation, and links digital identities.

[26]This is usually due to privacy concerns or compliance with public policy, such as California's Consumer Privacy Act (2018) or the European Union's General Data Protection Regulation (GDPR). For further details on privacy policy and compliance on the GitHub platform, see `https://docs.github.com/en/site-policy/privacy-policies`.

[27]For example, we employ a "peers-of-peers" identification strategy in Section 2.5.2 to overcome endogeneity concerns (**manski1993identification**) and recover unbiased estimates of peer effects in individual contribution levels.

[28]These two chapters, while both concerned with the microeconomic details of OSS production, are written to serve as independent research contributions.

or productivity. By observing the revealed preferences of contributors, we provide the first empirical estimates of these peer effects in terms of development labor costs, which we subsequently argue represent a lower bound for the willingness to pay for OSS goods. In Chapter 3, we turn our attention to the formation of software dependency networks. We discuss the maintainer's decision-making framework for optimally balancing a level of internal development with external dependency usage and the implications this aggregate behavior has for dependency network structure in equilibrium. Extending the literature on strategic network formation, our structural approach characterizes the formation of software development networks under uncertainty and exploits highly disaggregated data to simplify estimation. Both studies fortify a theoretical framework with empirical analysis, each pairing reduced form approaches with fully specified structural models. This methodology has several advantages. First, we can capture descriptive statistics and build intuition for complex OSS production processes via reduced form exploration. Second, a structural methodology allows us to rigorously specify and micro-found the mechanisms through which our hypothesized production effects operate. Finally, we can take our fully specified structural model to data and conduct policy counterfactuals. As our structural approach seeks to characterize the data-generating process for each production phenomenon, we can modify the underlying fundamentals of each process and assess welfare implications under the new equilibria.

The key insights from our analysis can be summarized at a high level. First, we empirically confirm an often cited dilemma in OSS: simple contribution does not immediately imply sustained maintenance (**eghbal2020working**). Under our counterfactual analysis of contributor peer effects, we find that extensive margin peer effects account for approximately

56% of aggregate contribution (Section 2.6). We also find little evidence that intensive margin peer effects have any influence on contemporaneous individual contribution levels, on average.[29] Moreover, when individuals make larger contributions, they tend to do so at greater marginal cost. Hence, while we can say the peer contribution activity may lead to individuals contributing in small share, we cannot say that peers make each other more productive. Second, and related to the previous point, externalities are quite prevalent in the OSS setting in general. When studying digital supply chains, we find that removing less than 1% of core dependency packages reduces quality for the remaining peers by more than 5% (Section 3.7). Third, the role of maintainer risk aversion in delivering robust software dependency networks seems to be second order (Section 3.7). More often than not, attractive features of particular projects seem to outweigh considerations for package security from the perspective of the project maintainer choosing which external packages to import as dependencies.[30] Finally, it is abundantly clear than complex substitution patterns exist in this space. We find considerable heterogeneity in estimates of externalities across both projects and time. For example, we find a higher level of complementarity in contribution levels between both (1) individuals and (2) dependent projects in the early days of GitHub (Sections 2.5 and 3.5). We also find that free-ridership is more prevalent in larger projects with dominant "insider" contributors (Section 2.5).[31]

This study motivates and demonstrates the research potential of OSS dynamics by empirical microeconomists. An abundance of rich data now exist to tackle fundamental questions

[29]We find stronger intensive margin peer effects when relaxing the assumption of contemporaneous influence. See Section 2.5.4 and Table 2.A.5.

[30]As we discuss in Section 3.8, this may be a consequence of our chosen package quality metric.

[31]See Section 2.5.

regarding the nature of OSS production. Our findings have implications both for the management of OSS projects and future research in this space. First, there is some rationale for public support of critical yet oversubscribed[32] OSS projects from a public goods perspective. Peer effects alone cannot generate sustainable maintenance effort for key digital infrastructure. Second, a small number of core dependencies provide functionality for large segments of the OSS ecosystem. It would be prudent to prioritize support towards these cornerstone projects.

[32]Here meaning that the available maintenance bandwidth is insufficient to reliably ensure software quality for a widely used OSS codebase.

Chapter 2

Quid Pro Code: Peer Effects and Productivity in Open Source Software

ch:pg

2.1 Introduction

sec:pg-intro

Open Source Software (OSS) projects are public information goods produced through incremental efforts of individual contributors.[1][2] Interested parties can freely download software code for their own use and can also propose contributions to the original maintainer of

[1]Our use of the term "open source" requires some definition. In a general sense, OSS is a computer technology for which the underlying source code is made publicly available under a license permitting use, modification, and subsequent redistribution of derived products (**osi2007**). While there are many variations on the specifics of this definition, the most important feature of software projects considered in this study is that (1) they are distributed under some permissive OSS license (**choosealicense2022**) and (2) they are *collaborative* projects that allow for modifications to be submitted from a contributor base wider than the original developer.

[2]Throughout this chapter, we will use the terms "contributor", "developer", "individual", and "agent" interchangeably in reference to the population of study.

the project[3]. The very existence of OSS rebukes conventional wisdom on privately produced public goods[4] and various explanations have been offered to rationalize their provision, from signalling (**lerner2002some**), (impure) altruism (**andreoni1990impure**), need satiation (**athey2014dynamics**), and institutional structures imposed by self-organizing local communities (**ostrom1990governing**; **benkler2002coase**). In this study we examine an alternative channel through which widespread contribution to public OSS projects may be achieved: peer effects. Peer behavior can potentially affect the net returns to public good contribution through various channels, improving returns and ameliorating contribution costs. Can peer influence drive heterogeneity in preferences and contribution costs, effectively subsidizing the private provision of public goods?

Consider the quandary faced by maintainers of OSS projects.[5] Sindre Sorhus is a superstar OSS contributor. As of December 2021, he works on OSS full-time and is the author and primary maintainer of over 1,000 OSS projects (**sorhusweb2021**). As a prolific maintainer, Sorhus interacts with the wider community of OSS contributors and has personally reviewed tens of thousands of proposed contributions to his projects. Sorhus once reflected that "$\sim$ 80% of contributors doesn't [*sic*] know how to resolve a merge conflict, almost no one writes a good pull request titles, $\sim$ 30% don't run tests locally before submitting a

[3]For example, a user may wish to propose a new feature or fix a software fault (i.e., a "bug").

[4]Since contribution is costly, agents choose their contribution levels both with respect to private benefits of contribution as well as the level of the OSS public good delivered by the efforts of their peers. If the net benefit of contribution is negative, an individual may simply opt to free-ride on the efforts of others, leading to misallocation of contribution away from an efficient equilibrium.

[5]In this chapter, we will at times classify agents in the OSS public goods setting according to their level of participation in what is known as the "contributor funnel" (**mcquaid2018**). Users of an OSS project may utilize a software product but do not contribute to it. A subset of users are contributors and allocate some contribution effort to developing the project. A subset of contributors are maintainers, typically agents responsible for a large share of project contribution and may also have decision-making power over what proposed contributions are integrated into the project. We will also sometimes refer to these agents as developers.

[pull request]", and "$\sim$ 40% don't include docs/tests" (**sorhus2019**). In essence, Sorhus's concern centers around the lack of quality project contributions from his peers. Software development in general is a complex, ever-changing process and many potential contributors simply may lack the skills to contribute effectively. As opposed to shouldering the entire burden of OSS project development[6], to what extent can the contributions efforts of skilled contributors like Sorhus actually improve the productivity of their peers?[7]

A key difference between OSS projects and other public goods is that production of OSS generates both a community of contributors and a set of auxiliary information goods around the project that can potentially reduce subsequent costs of contribution. For example, OSS project maintainers provide assistance and guidance to new contributors by responding to inquiries via mailing lists, message boards, or real-time chat channels.[8] Moreover, OSS communities typically archive the history of such project-related interactions between contributors, creating a publicly accessible knowledge base for project development.[9] OSS projects typically feature documentation[10] that gives a broad overview of the project.[11], provides detailed information on how the software operates at a technical level, and suggest

[6]While the use of OSS is itself non-rival, the contribution bandwidth of project maintainers is not (**brown2018**).

[7]In other words, how do maintainers induce project users down the "contributor funnel" into becoming productive, recurring contributors?

[8]Users who receive feedback on their contribution from project maintainers are far more likely to return to contribute in the future (**sholler2019ten**).

[9]In a similar fashion, OSS projects are overwhelmingly managed using a *version control system*, making the entire projects incremental development history public record.

[10]Note that documentation is generated by developer labor and a contribution to the project itself.

[11]Examples of high-level documentation include project `README` files bundled with the project source code, "wiki" pages, and long-form vingettes on project usage. For an example of best practices on how these are actually integrated into an OSS project, see Sections 8, 10, and 11 of **wickham2015r**.

how to properly propose new contributions.[12] Popular OSS projects can also generate a significant amount of buzz outside the contribution platform itself, from community-authored articles demonstrating usage to external forums[13] where users can request help for various programming and software tasks. The combination of these features form the basis for peer effects on contributor productivity. Contribution activity itself can generate a form of "digital capital"[14] for subsequent OSS production, working to both lower the initial fixed cost of contribution for potential contributors and to make current contributors more productive. Hence, in contrast with many conventional public goods settings, there is scope for individual and peer contribution to become strategic complements.

Salient examples of OSS begin to illustrate the scale at which developers have contributed labor towards the production of complex public information goods. As each OSS developers's "contribution bandwidth" is both scarce and costly, the significance of peer effects that drive contributor labor can be measured naturally in terms of the opportunity cost of a developer's time: what is the equivalent private market labor expenditure to finance the development of large OSS projects? Consider the case of the Linux Kernel. Regarded as the largest collaborative OSS project in history, the Linux Kernel was first released in 1991 by Linus Torvalds and has become the most widely used operating system basis for web servers, mobile devices, and high performance computing infrastructure. As of September 2021, the Linux Kernel has amassed over 31.3 million single lines of code from 23,927 distinct

[12]For example, a project maintainer may include a "contribution template" so that novice contributors avoid common pitfalls for new project contributions. Referring back to the example of Sindre Sorhus, this improves the quality of the proposed change and reduces the "back-and-forth" between maintainer and contributor.

[13]A relevant example is the programming-focused question and answer website Stack Overflow which has been described as a sister community to OSS collaboration platforms such as GitHub (**eghbal2020working**).

[14]Or more accurately, human capital that is recorded or codified as a public information good and then used as an input in the production of additional public goods.

contributors over the past three decades. Using standard methods from software engineering cost estimation, it would take nearly 70 million person-hours to rewrite the entire kernel from scratch, which would cost over $1.05 billion today.[15][16] While estimates for the use-valuation of OSS is an important ongoing area of research (**greenstein2014digital**; **nagle2019open**), in this study we seek to characterize the extent to which peer effects can mitigate production costs of OSS public goods.

We seek to empirically assess peer effects on public good production using the context of OSS. Our methodology is organized into two phases. In the first phase, we build intuition on the magnitude of net peer effects in OSS contribution using a reduced form approach. To address concerns over endogeneity, we develop an identification strategy to determine to what extent individual effort levels are influenced by the contribution levels of their peers. Specifically, we instrument the likely endogenous contribution effort of an agent's peers in a given project with the effort levels of the agent's "peers-of-peers" defined by common contribution in outside projects.[17] The instrument operates by changing the relative incentives for peers to contribute to a given project by varying the incentives in external projects. This approach allows us to determine whether individual and peer contribution are strategic complements or substitutes on net, conditional on the set of developers that contribute at all. In the second phase, we develop a structural model of OSS contribution to pin down the microeconomic foundations for contributor behavior. We seek to place emphasis on disentangling contribution decisions along the extensive versus intensive margin and integrate peer influence into both decisions. To this end, we embed a micro-founded

[15]Estimated (conservatively) using the COCOMO model of software development cost estimation developed by **boehm1981software** and the software utility `scc` (Source: https://github.com/boyter/scc).

[16]The median annual salary for U.S. software developers in 2020 was $110,140 ($52.95 per hour) (**bls2021**).

[17]Details for this identification strategy are given in Section **??** and Figure **??**.

model of private public good provision (**bergstrom1986private**) into the selection model of (**heckman1979sample**). The structural approach facilitates the recovery of individual productivity parameters, allowing us to characterize the welfare of particular contribution profiles and conduct counterfactual analysis. Our main counterfactuals of interest estimates the value of aggregate contribution added by peer effects.

We apply this methodological framework in an empirical analysis, focusing on the context of Open Source Software contribution. We use individual-level contribution data for a random sample of 2,287 highly collaborative OSS projects hosted on the GitHub collaboration platform. The remainder of this chapter is organized as follows. We first provide additional background on OSS development in Section 2.2. Next, we survey segments of related literature in Section 2.3. We introduce the empirical setting in Section 2.4, describing the OSS contribution on the GitHub platform and giving an overview of data included in the empirical sample. We then develop a reduced form strategy to estimate peer effects in Section 2.5. With high-level insight on net peers effects in hand, we next develop a structural model of public good contribution with extensive and intensive margin peer effects in Section 2.6. We outline an estimation strategy, present estimation results, and conduct counterfactual analysis to measure the value of contribution generated by distinct peer effects channels. Finally, we summarize and interpret our findings in Section 2.7 and discuss promising directions subsequent research.

2.2 Background

sec:pg-background

OSS projects are typically organized around *software code repositories*, publicly accessible websites that host the project's source code and provide collaboration functionality[18]. Users can view and download the source code of OSS projects for their own use. They can also contribute to the OSS project's codebase. A typical contribution pattern works as follows: (1) a user downloads a copy of the source code, (2) makes a series of incremental changes to the codebase, and (3) submits a request to the owner of the original OSS repository to integrate their changes. Due to the open nature of the code and the permissiveness of OSS licenses in general, there is little to prevent a user from simply copying the codebase of an existing project into a new OSS good[19]. However, users can distribute contribution costs and share knowledge by working collaboratively with peers. It is therefore reasonable to assume there exist strong motivations for distributed users to rally around and contribute to particular OSS projects instead of splintering off into isolated endeavors, creating "digital communities" around OSS projects characterized by social norms and stocks of project-specific information capital.

Peer effects have long been discussed as a driving force behind the "success" of particular OSS projects. An early discussion on net effect of peer contributions on software goods in collaborative projects began with the conjecture by **brooks1995mythical**, who observed

[18]For example, Figure 2.B.1 depicts a snapshot of the web user interface for the `bootstrap` project's GitHub repository, a popular JavaScript framework for web development: `https://github.com/twbs/bootstrap`. The history of all user contributions to the pandas codebase can be viewed here: `https://github.com/twbs/bootstrap/commits/main`. The repository contains a README with the source code, a document that contains links to detailed documentation, installation and usage notes, and guidance for prospective contributors.

[19]This process is known as "forking" in the OSS community. Forks of original projects can also be come active contribution communities in their own right. This typically happens when there is a sufficient number of contributors interested in pursing a different direction of development.

that the addition of developers to a software project slows down the pace of development. In a response to the so-called “Brook’s Law”, **raymond1999cathedral** countered this postulate with the example of OSS collaboration and “Linus’s Law” that roughly states that the likelihood that faults in a software’s codebase will be identified and fixed rises with the number of users and contributors working with it. **raymond1999cathedral** argues that the proliferation of highly collaborative, decentralized OSS projects is itself a rebuke of Brook’s Law.

As the production of OSS can clearly be subject to peer influence, a core impetus for this study is to disentangle the various channels through which peer effects operate and estimate the empirical implications for these effects on equilibrium contribution. In theory, peer effects can have both negative and positive impacts the level of privately provided OSS. What anecdotal evidence do we have for either (1) free-riding or (2) productivity externalities in OSS development? With the rise of OSS use, a common concern amongst OSS project maintainers is over-subscription of their projects: users who flood communication channels with support requests without contributing the fix themselves (**eghbal2020working**). A related concern is that many OSS projects originating from small groups of contributors are widely used as part of the “digital infrastructure” (**eghbal2016roads**) that underpins modern information and communication technologies. Consider the case of OpenSSL, an encryption library that by some estimates is used by two-thirds of public facing web servers to secure private information (**openssl2021**). In 2011 a bug, now known as Heartbleed, was introduced into the OpenSSL codebase and was not discovered until 2014[20], exposing

[20]Consequently, some have pointed to the Heartbleed exploit as a repudiation of Linus’ Law (**meneely2014empirical**).

a vast swath of internet communications that were previous thought to be secure. The estimated cost to simply limit the extent of this vulnerability was estimated to be over $500 million USD (**eweek2014**) and does not consider the cost of any secure data lost through the exploit. The OpenSSL team at the time "never had more than three to four core developers" overseeing more than a half a million lines of code on an annual donation budget of $2,000 USD (**oberhaus2019**). Whether it was the sheer size and complexity the OpenSSL codebase or the preferences of the maintenance team deterred potential contributors, the fact that an OSS project serving as a critical component of internet infrastructure did not receive more attention from the wider community of users who rely on it ought to be cause for concern for OSS sustainability.

While free-riding on OSS contribution is likely prevalent and perhaps inescapable when considering a project's user-base in the broadest sense[21], it may also be the case that increased participation in OSS distributes the joint cost contribution and improves individual productivity. How can the development of OSS itself either make subsequent contribution less costly or induce the marginal free-rider to contribute? Recommended practices in software engineering encourage developers to include documentation, testing frameworks, and use automated processes whenever possible (**fogel2005producing**). Documentation explains the functionality and inner workings of software code in plain language, making it easier for both users and potential contributors to work with the software. Testing frameworks ensure the code functions as intended and are essential for a large collaborative OSS project. Continuous Integration (CI), a form of automation in the integration and testing

[21]Modern software projects, both proprietary and open source, typically borrow 70 to 90% of their functionality OSS components (**nagle_census_2022**).

of changes to software projects, facilitates a greater volume of contribution and has been shown to allow software projects to release[22] more frequently (**hilton2015ci**). Investments in these features lower the cost burden of maintenance and lower the barriers to entry for new contributors. Moreover, active contributors in OSS communities often provide "non-code" contribution services to the project, answering user inquiries, reviewing and integrating proposed changes, establishing design principles and community guidelines, and other functions peripheral to contributing code. It's natural to imagine that all else equal, a potential contributor would prefer allocating their contribution bandwidth to an OSS project with sociotechnical infrastructure that makes it easier to work with.

The collaborative and decentralized nature of OSS development suggests a setting rife with intricate peer effects. The wider population of OSS users may lack the skills or resources needed to contribute to OSS codebases and may simply free-ride on the contributions of more prolific developers. At the same time, OSS contribution itself generates an abundance of features that reduce the cost of and further incentivize wider OSS participation. We use this study as an opportunity to develop a microeconomic framework decomposing these forces and to empirically estimate their implications.

2.3 Literature

sec:pg-lit

We review a subset of academic literature that can be divided into several distinct strands: (1) motivations for OSS contribution, (2) the private provision of public goods, and (3) peer effects.

[22]In software development, a "release" is a particular version of the project distributed to users. In an appeal to Linus' Law, OSS proponents such as **raymond1999cathedral** and **fogel2005producing** encourage frequent releases.

2.3.1 Why Contribute to OSS?

Although initially puzzling, the existence and proliferation of OSS goods has been studied through an economic lens for over two decades (**lerner2002some**). A common interest in early research on the economics of OSS focuses on the incentives for participation in public good production by both individuals and profit-maximizing firms. Many different hypotheses have been offered to explain OSS provision and contribution behavior:

- Individual private benefits: intrinsic motivation (**lakhani2003hackers**), need satiation (**bessen2006open**; **athey2014dynamics**), signalling and status (**glazer1996signaling**; **lerner2002some**; **roberts2006understanding**), "warm glow" (**andreoni1990impure**), option value of modular codebases **baldwin2006architecture**, permissive licensing (**fershtman2004determinants**; **lerner2005scope**; **fershtman2007open**).
- Social effects: pure altruism (**bonaccorsi2003altruistic**), social norms and reciprocal altruism (**raymond1999cathedral**; **bergquist2001power**; **benkler2002coase**), project productivity (**fershtman2011direct**)
- Strategic motivations for firms: innovation, market power (**bonaccorsi2006entry**), labor search[23], cost reduction (**andersen2012commercial**)

Some closely related work examines contribution to OSS and open source content in general empirically. **fershtman2004determinants** find that permissive software licenses induce greater levels of contribution. **hahn2008emergence** find that OSS developers are more likely to join projects with past collaborators. **fershtman2011direct** demonstrate an empirical relationship between the success of an OSS project, measured in downloads, and

[23]See https://github.com/t9tio/open-source-jobs for a list of job listings for private firms with primary products centered around GitHub OSS repositories.

the extent to which its contributors work in other common projects, suggesting the existence of both direct and indirect project knowledge spillovers. In contrast, the present study uses microdata to measure peer effects on contribution at the individual level. Several authors have used the context of Wikipedia to study peer effects within collaborative production of open content. Exploiting blockages of Chinese language Wikipedia for mainland China, **zhang2011group** find that pro-social peer effects are increasing in the number of peers: individuals contribute more to Wikipedia when they have more peers. **slivko2014peer** use an indirect peers strategy to find modest evidence for positive, intensive margin peer effects amongst frequent contributors.

2.3.2 Private Public Good Provision

Seminal work seeks to rationalize private provision of public goods. While the canonical public goods model of **samuelson1954pure** suggests strong incentives to free-ride on the contributions of others, heterogeneity in both preferences and the marginal cost of provision can explain positive levels of private provision in many contexts (**tiebout1956pure**; **stiglitz1981public**; **stiglitz1982theory**; **bergstrom1986private**; **cornes1985simple**; **andreoni1990impure**; **fischbacher2006heterogeneous**; **kotchen2009voluntary**; **jacobsen2017pub**
In the case of OSS, online collaboration dramatically reduces transaction costs inherent to the production of other types of public goods (**coase1937nature**; **nitzan1990private**). Social norms develop around projects in order to efficiently manage the needs of the community and the time constraints faced by contributors (**hollander1990social**; **ostrom1990governing**).

Moreover, agents are subject to contribution externalities and can confer productivity benefits on peers, which in turn confer additional benefits to the original agent (**elliott2019network**). In this sense, agents "pass through" benefits of increased contribution and can be compensated for these investments.

Several authors have focused on public good provision specifically within the context of OSS. **johnson2002open** analyzes a model of OSS public good contribution. As expected, the assumption of the fixed costs of contribution preclude the efficiency of the decentralized equilibrium. **baldwin2006architecture** find that highly "modular" codebases provide contributors with option value and ultimately attract more contribution.

2.3.3 Peer Effects

Productivity Spillovers

Particularly of concern to our reduced form analysis, we link this work to an expansive body of literature concerning peer effects and their estimation. Experimental evidence suggest peer effects in public goods settings can be driven by punishment (**fehr2000cooperation**), cooperation (**falk2006clean**), and can ultimately increase voluntary contribution to public projects (**archambault2016can**). Several empirical studies find evidence of labor productivity "spillovers" when high ability peers are introduced (**mas2009peers**; **lindquist2015network**). There is mixed evidence for peer effect heterogeneity across individuals (**arcidiacono2005peer**; **cornelissen2017peer**), suggesting the context and estimation strategy matter. A related literature investigates the importance of group sizes on treatment and peer effects (**angrist1999using**; **krueger2003economic**).

Identification

Identification of peer effects in non-experimental settings is of great concern to this literature. **manski1993identification** posits a "reflection problem" which **bramoulle2009identification** suggest can be solved by using instruments generated by the network structure itself: the behavior of indirectly linked agents can generate quasi-random variation needed to address endogenity concerns with estimating peer effects in observational data. It should be noted that the identification strategy of **bramoulle2009identification** relies purely on characteristics of the network structure between agents and without qualification, can be devoid of microeconomic foundations or even lack appealing quasi-random variation for causal identification (**angrist2014perils**). Other authors have used alternative strategies, such as true random assignment of peers (**sacerdote2001peer**; **guryan2009peer**; **carrell2011natural**), exploiting quasi-experimental designs (**dahl2014peer**), overlapping peer groups (**de2010identification**), directly modelling endogenous peer networks (**goldsmith2013social**
the use of panel data (**patnam2011corporate**), and explicit structural approaches (**ciliberto2016playing**
Our study will draw several techniques from this literature to develop an identification strategy for peer effects, including social connections, changes in peer groups, and individual fixed effects to develop a unique, microfounded "peers-of-peers" identification strategy in Section **??**. To the best of our knowledge, the closest use of the peers-of-peers identification strategy in public good contribution is **slivko2014peer**, who uses the number and average contribution level of indirect peers to instrument for peer contribution.

2.4 Data

sec:pg-data

We use observational data to measure individual contribution levels over time for a sample of OSS projects. We draw this sample from projects hosted on GitHub, the world's largest collaborative software development platform.[24] For each project, we observe agent-level contribution efforts in continuous time[25], measured in "commits", or atomistic modifications to the codebases of OSS projects.[26] For the purposes of this study, we will define an agent's peer group in a given project as the set of other developers contributing code to that project. Individual and peer contribution levels across projects and time will form the core of the reduced form and structural analysis. Additional details on the data used in this paper can be found in Section 2.C of the appendix.

We begin by describing the dataset in broad strokes. Since the universe of OSS repositories on the GitHub platform is incredibly vast[27], we restrict our empirical sample to a randomly selected subset of popular and highly collaborative projects. Specifically, we take a 10% random sample of GitHub projects with 15 or more distinct contributors and 100 or more "stars"[28] as of June 2019. This results in an empirical sample containing 2,287 projects

[24]Launched on April 2008, GitHub has become the world's largest source code host and *de facto* collaboration platform for OSS projects

[25]Each commit to a project is recorded with a timestamp (e.g. 2009-10-31 01:48:52).

[26]Note that a commit can encompass changes to any number of lines across any number of files. A natural concern may be that variation in the size of individual commits makes it difficult to compare as equivalent units of contribution effort. For example, a single commit might be a simple typo correction requiring little effort or a complicated "bug" fix that took many hours to address. Some have argued for simpler measures to estimate labor commitment to software development, such as the number of days a developer makes at least one contribution to a project in a given time period (**sherwood2015estimating**).

[27]As of January 2020, GitHub has over 40 million users and hosts more than 190 million software repositories (**octoverse2020**).

[28]On the GitHub platform, users can mark interesting projects by "starring" them, which subscribes the user to a newsfeed covering project development. For the purposes of this study, we use project stars as a proxy for user interest or quality of the project. Stars also distinguish highly collaborative, "engineered" software projects from small, single-user projects (e.g. abandoned forks or repositories containing personal files like notes or school projects) (**munaiah2017curating**).

and 107,921 distinct contributors observed from the launch of GitHub in April 2008 through June 2019.[29] We aggregate individual contribution to a monthly frequency and therefore the unit of analysis is individual-project-time.[30] The most commonly represented programming languages for these projects are JavaScript (31%), Python (11%), and Java (9%). Of the contributors represented in the sample, 3.7% are members of the projects they contribute to and only 0.57% are project owners.[31] The average project in the empirical sample is 5 years old and is the product of 2,490 cumulative commits made by 56 distinct contributors. The average individual in the sample contributes 13 commits to a particular project a month and 53 commits across all projects over the sample period. It is critical to note the (right) skewness of contribution, both between and within projects: the median project has 829 cumulative commits made by 29 distinct contributors while the median agent makes only 3 commits to a single project each month. Furthermore, the share of individual contribution within projects is bimodal (see Figure 2.B.3). Roughly 45% of contributions in our sample are made by agents who represent 5% or less of total project contribution for that month. On the other end of the spectrum, about 8% of observations in the sample represent individual contributions that account for over 95% of total project commits for that month. In simpler

[29]Since GitHub is simply the web platform hosting the project, some projects in the sample have contributions made either prior to the existence of GitHub or it's arrival on the GitHub platform. Projects are managed using version control systems (VCS) that record a complete history of changes in the project since its inception. GitHub's namesake comes from the VCS tool used by the projects it hosts: `git`. Figure 2.B.2 overlays histograms for (1) the earliest recorded commit in each project and (2) the date the project was created or moved to the GitHub platform.

[30]In other words, each observation is the level of contribution by an individual to a particular project for the given month.

[31]For the projects in this sample, "ownership" does not imply property rights over the software code itself. Project "ownership" and membership on the GitHub platform simply means the user has certain administrative privileges within the repository, most important of which is the ability to merge proposed contributions of outsiders into the main project codebase. It should be noted that many projects may feature core contributors with a considerable amount of influence on project design decisions who are not officially project owners or members in the GitHub system.

terms, the most common contribution pattern within projects involves many individuals contributing a small share[32] relative to a dominant core contributor in each period. The sample provides evidence that even though both aggregate contribution and the number of distinct contributors have grown over time (see Figure 2.B.4 and Figure 2.B.5), average individual contribution levels have remained quite stable (see Figure 2.B.6). Consistent with anecdotal evidence from the OSS literature (**eghbal2020working**) and theory on contributor behavior (**athey2014dynamics**), these characteristics suggest the growth of an OSS projects is a combination of (1) small number of dominant core contributors and (2) the aggregate effect of small contributions from a wider population of software developers.

With a general understanding of the GitHub contribution sample used in this chapter, we now direct attention towards measures of peer and individual contribution germane to both reduced form and structural analysis. We present key descriptive statistics for this empirical sample of contributions measures in Table 2.A.1. The average agent contributes 13 commits to a project each month and has an average of 17 peers contributing 188 commits in aggregate. As noted before, individual contribution is highly right-skewed. The median agent contributes just 3 commits per month and has 7 peers who contribute 59 total commits. Approximately 6.8% of observations in the empirical sample involve a sole contributor with no peers in that time period. Since the mean and median individual contribution levels coincide with project-specific contribution, these data suggest that most agents contribute to a single project in a month.[33] The average agent's mean cumulative contribution to a

[32]This is known as "drive-by" or "casual" contribution (**fogel2005producing**; **eghbal2020working**).

[33]It should be noted that the apparent lack of contributors contributing to multiple projects may simply be an artifact of sample construction. We simply take a random sample of projects and observe contribution activity of agents within those particular projects. Therefore, individuals in the sample may be contributing to other OSS projects not recorded in the present sample. We at least partially account for this deficiency when constructing our instrument for peer contribution in Section 2.5.2, a measure that sums contribution

particular project is 256 commits (median 23), a pattern that naturally is similar in peers. Together, insights from the empirical sample suggest agents form affinities with a particular project and continue to contribute to it over time.[34]

Finally, we collect two additional measures most relevant for our structural approach described in Section 2.6. First, for each project and time period, we measure the number of "stars" associated with the project. This is a rough proxy for an OSS repository's popularity and is used to measure the level of public good quality. Similar to contribution levels, project quality is highly skewed: the mean (median) project has 910 (161) stars in a given month. Second, we observe individual time allocated on the platform. Specifically, for each individual and time period, we measure how many days they spend contributing to any project on the GitHub platform (**sherwood2015estimating**). We use this measure of time allocation to proxy for numeraire good consumption, which in turn facilitates the estimation of time and project-varying productivity shocks for each agent. In a given month, the average (median) agent makes commits over 3.88 (2) days to projects in the sample. Compared with the skewness in contribution levels, this descriptive suggests a stark difference between the extensive and intensive margin contribution decisions.

levels of "peers-of-peers" across all projects recorded in the GHTorrent sample of **Gousi13**, some of which are not contained in our empirical sample.

[34]This may be explained by many alternative mechanisms, including individual need (**bergstrom1986private**; **lerner2002some**; **lakhani2003hackers**), the discipline of social norms (**ostrom1990governing**), or an accumulated expertise within a project.

2.5 Reduced Form

sec:pg-reduced

Before developing a structural model, we build intuition for net peer effects in public goods contribution using a simple reduced form framework. We seek to understand how an individual agent's individual contribution level is influenced by the contribution level of her peers. This section is organized as follows. We first outline a baseline econometric specification to assess peer effects in public good contribution. Next, in an effort to address endogeneity concerns and give a causal interpretation to the peer effect estimates, we propose an instrumental variable for peer contribution, define its measurement, and discuss various possible threats to identification. The final subsection discusses the empirical results.

2.5.1 Peer Effects on Individual Contribution

sec:reduced-spec

Consider a setting in which individuals $i \in \mathcal{N}$ contribute to OSS projects $p \in \mathcal{P}$ in each period $t \in \mathcal{T}$. The outcome of interest, $a_{ipt} \geq 0$, is the contribution level for agent i to project p at time t. The aggregate contribution level of agent i's peers to project p at time t, denoted by $a_{\text{-}ipt} \equiv \sum_{j \neq i} a_{jpt}$, is the regressor of interest. We present a baseline specification[35] for contribution peer effects in Equation (2.1):

eq:pg-reduced

$$a_{ipt} = \delta a_{\text{-}ipt} + \boldsymbol{\beta}' \boldsymbol{X}_{ipt} + \epsilon_{ipt} \tag{2.1}$$

[35] Alternative specifications similar to Equation 2.1 are presented in Section 2.5.4, serving to both provide robustness checks and to consider different characterizations of peer influence.

. Here the vector $\boldsymbol{X}_{ipt}$ is a set of observable exogenous factors driving agent i's level of contribution to project p at time t. The term ϵ_{ipt} represents unobservable factors driving contribution.

The coefficient of interest in Equation (2.1) is δ, which captures the (average) effect of aggregate peer contribution on individual contribution.[36] We will refer to the coefficient δ as the reduced form peer effect in contribution. This term is sometimes referred to in the literature as the "endogenous effect" (**manski1993identification**) or "social multiplier" (**glaeser2003social**). Our empirical analysis seeks to test the null hypothesis of no peer effects ($\delta = 0$) against an alternative that there exists some relationship between contribution levels of peers ($\delta \neq 0$). If there is evidence of peer effects, we are also interested in the net effect of the opposing externalities. The core premise of this study is that peer influence is the net effect of two distinct externalities in contribution. In the canonical public goods model, individual and peer contribution to public goods are strategic (gross) substitutes and therefore voluntary provision is vulnerable to free-riding. If incentives to free-ride dominate, we should expect $\delta < 0$[37] in equilibrium. On the other hand, if an increased level of peer contributions also leads to an increase in agent i's contribution *ceteris paribus*, it is likely the case that some other peer effect (e.g., externalities in productivity and contribution costs, pro-social behavior) dominates incentives to free-ride. This would imply $\delta > 0$.

[36]In other words, the effect on individual contribution when peer contributions increase by 1 commit, on average and *ceteris paribus*.

[37]Note that this assumes that $\boldsymbol{\beta} \neq \mathbf{0}$ in the true model for individual contribution. If the population model in Equation (2.1) is such that $\boldsymbol{\beta} = \mathbf{0}$ (i.e., a model without covariates or an intercept), then $\delta \in [0, 1]$ by construction. As $\delta \to 0$, a single contributor dominates and all others free-ride. As $\delta \to 1$, contribution is uniform across peers.

Other observable factors that influence agent contribution are captured in a vector $\boldsymbol{X}_{ipt}$ and may potentially vary across agents, OSS projects, and time. Examples of these influences may include individual and peer contribution history[38], observable quality or popularity of the OSS project, the size of the contribution peer group, technical characteristics of the projects[39], and other agent characteristics.[40][41][42] In terms of the specification in Equation (2.1), we can also include a battery of individual, project, or time fixed effects[43] in $\boldsymbol{X}_{ipt}$.

2.5.2 Identification

sec:reduced-identification

The specification in Equation (2.1) describes a model in which peer groups are defined as the set of agents contributing to a particular OSS project at a given point in time: individual contribution is a function of contemporaneous, aggregate peer contribution. The specification is a simplified linear-in-sums[44] formulation similar to reduced form models studied widely in the peer effects literature (**manski1993identification**; **bramoulle2009identification**; **goldsmith2013social**).

[38]Such as an agent's cumulative contribution to a project at time t or their contribution in previous periods. Cumulative and temporal lags of contribution can capture an agent's accumulated experience or affinity with a particular project.

[39]Such as project age and programming language used

[40]In the context of GitHub data available, agent characteristics may include whether the agent is the owner or member of the project or if they identify with a particular employer.

[41]Agents can voluntarily include the name of their employer in their GitHub profile and can make contributions with a company email address.

[42]On the GitHub platform, an agent can be added as a member to a project, potentially giving them more discretion over what proposed changes by the wider community are integrated. It also is plausibly a signal of an agent's affinity with a particular project.

[43]Inclusion of individual level fixed effects $\boldsymbol{X}_{ipt}$ accounts for an agent's intrinsic proclivity to contribute to OSS goods, independent of other factors (**andreoni1990impure**).

[44]Technically, the specification is linear in "leave out" sums by definition of $a_{\text{-}ipt}$.

Point identification of the parameter δ in Equation (2.1) is demonstrated by **lee2007identification** by exploiting "leave out" sums and variation in peer group sizes, overcoming the well known non-identification result of **manski1993identification**. In our setting, we point to the descriptive statistics in Table 2.A.1 as evidence that contributors are likely to have different groups of contributing peers in each period for any particular project. Since peer groups in the current empirical setting are naturally quite dynamic, we argue that point identification is established. Therefore we wish to go one step further and establish causal identification for the parameter δ in Equation (2.1). Under what conditions can we interpret an estimate of δ as the local average treatment effect (LATE) of peer contribution on individual contribution? The immediate challenge is that since individual and peer contribution are both observed choice variables, a naive estimate of δ likely suffers from endogeneity bias (**angrist2014perils**; **lewbel2019identification**). An experimental ideal to causally identify the net peer effects parameter δ would involve first randomly assigning agents to projects and then allowing them to decide contribution levels, ensuring random peer groups in which choice of contribution levels ought to be uncorrelated unobservables, or $\text{Cov}(a_{-ipt}, \epsilon_{ipt}) = 0$. In reality, agents select into and choose contribution levels on the basis of potentially unobservable influences, such as personal need, technical ability, and their endowment of time to work on OSS projects. For example, high ability agents might select into and make above-average contributions to common projects, generating positive bias in the ordinary least squares estimate of δ in Equation (2.1) (i.e., $\text{Cov}(a_{-ipt}, \epsilon_{ipt}) > 0$). On the other hand, low ability agents may also select into projects with highly skilled developers and make minimal contributions. Taken to the extreme, agents who free-ride completely do not contribute at all and therefore do not appear in the sample whatsoever. Since we cannot completely account for all free-riders in OSS[45], we must caveat any interpretation of the estimated effects in this analysis to be conditional on the population individuals who contribute at all.

In the absence of purely random assignment of contributors to projects, we address the concern over endogeneity in peer contribution $a_{\text{-}ipt}$ by use of an instrumental variables strategy. We seek a valid instrument for peer contribution that, conditional on other observable and exogenous factors $\boldsymbol{X}_{ipt}$, (1) exerts some influence on the contribution levels of Agent i's peers $j \neq i$ and (2) only influences Agent i's contribution to project p through its effect on peer contribution $a_{\text{-}ipt}$.[46] In other words, in lieu of random assignment to projects, we must opt for an instrument that generates quasi-random variation in peer contribution levels, conditional on the set of agents that contribute at all to a given project. Furthermore, we combine this instrumental variables approach with a battery of both control variables that plausibly explain OSS contribution and fixed effects at the individual, project, and time period to account for common but unobservable shocks across each unit.[47]

Contribution by peers-of-peers

sec:pg-reduced-peers-of-peers

Consider an agent i who contributes to OSS project p. If for some reason i's peers suddenly find contribution to other projects relatively more (less) attractive, they may allocate efforts away from (towards) project p through a channel with no direct influence over Agent i's contribution to p. This strategy is facilitated by the project-mediated "social network" of individual developers in which connections are defined by the projects they

[45]Given its wide-reaching prevalence, it's difficult to imagine there exist consumers of information technology who have *not* used OSS at some point.

[46]In the language of instrumental variable estimation, an instrument that satisfies both the (1) relevance and (2) exclusion conditions.

[47]In the context of our notation, across each i, p, and t.

commonly contribute to.[48] Agent i has peers $j \neq i$ in project p, who in turn also have peers $k \neq i, j$ in other projects $q \neq p$ they also contribute to. Hence we argue we can use the contribution network structure itself in a "peers-of-peers" identification strategy in the spirit of **bramoulle2009identification** to recover the effect of peer effort levels on equilibrium contribution levels.[49] An important departure is that while the strategy of **bramoulle2009identification** is designed to exploit general characteristics of the peer social network, the identification used in this study is based on microeconomic principles of substitution. We sketch out the identification strategy graphically in Figure ??. To guide the graphical intuition, consider the following hypothetical scenario. Suppose there are three contributors $\mathcal{N} = \{i, j, k\}$ and two OSS projects $\mathcal{P} = \{p, q\}$. Assume that at the beginning of period t, contribution profiles are $a_{ipt}, a_{jpt}, a_{jqt}, a_{kqt} > 0$. Hence Agents i and j contribute positive amounts to Project p while Agents j and k contribute positive amounts to Project q. Agent i's direct peer is Agent j and indirect or "peer-of-peer" is Agent k. In this sense, Agents i and k are connected only indirectly through the contribution patterns of Agent j.[50] This initial setting is represented in Panel (a) of Figure ??. Next, suppose Agent k increases their contribution to Project q (e.g., Panel (b) of Figure ??). If changes in Agent k's contribution to project q influence the time-constrained contribution behavior of Agent j, then Agent j may have incentives to change her contribution levels to Project p. The case in which Agent j finds her contribution to Project p a strategic complement with Agent k's is depicted in Panel (c) of Figure ??. An example in the OSS setting may occur when Agent

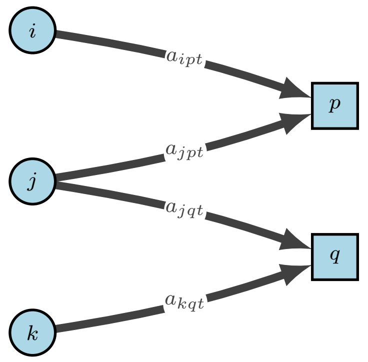
[48]In a similar fashion, **fershtman2011direct** use a bipartite graph to model connections between OSS projects and contributors.

[49]As noted when surveying related literature, **slivko2014peer** uses a similar "peers-of-peers" identification strategy by using a network of Wikipedia editors mediated by articles commonly contributed to.

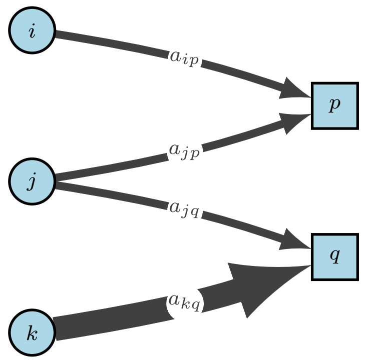
[50]That is to say, Agents i and k are not *directly* connected through the contribution networks. Any influences Agent k's contribution has on that of Agent i operate *only* through changes in Agent j's behavior.

fig:identification

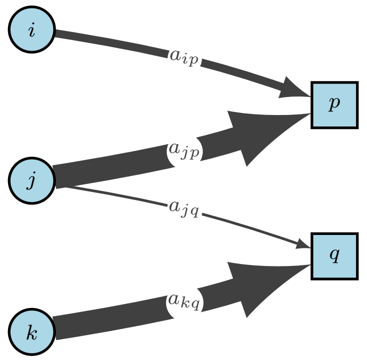
Figure 2.1: Identification Strategy (“Peers-of-peers” Contribution). Agents $\{i, j, k\}$ contribute to Projects $\{p, q\}$. Assume i and j contribute to p while j and k contribute to q.



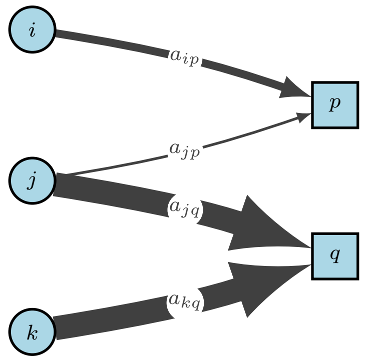
(a) Initial Setting



(b) Suppose Agent k increases contribution to Project q



(c) Case 1: Agent j substitutes *towards* Project p (contribution is a strategic substitute)



(d) Case 2: Agent j substitutes *away from* Project p (contribution is a strategic complement)

k contributes a fix for an issue in Project q that was consuming Agent j's contribution bandwidth. Conversely, the case in which Agent j finds her contribution to Project p a strategic complement with Agent k's is depicted in Panel (d) of Figure ??. This may arise if Agent k contributes an attractive fix or feature to Project q that encourages additional contribution from Agent j. In either case, the contribution pattern of Agent i's indirect peer Agent k influences Agent i only through changes in the behavior of Agent j.

In summary, we propose the use of aggregate contribution of peers-of-peers effort to instrument for peer effort. The instrument operates by inducing substitution of contribution effort across projects, generating quasi-random variation in aggregate peer effort from the perspective of individual developers. In the following two subsections, we define the measurement of the peers-of-peers instrument and provide a set of assumptions for its validity.

Instrument Measurement

sec:pg-reduced-instrument

Denote the "peers-of-peers" instrument for peer contributions $a_{\text{-}ipt}$ as z_{ipt}. Roughly speaking, we choose to define z_{ipt} as the aggregate contribution of peers of i's peers in project p at time $t-1$.[51] Formally we measure z_{ipt} as t:

eq:instrument

$$z_{ipt} = \sum_{j \neq i} \sum_{q \neq p} \sum_{k \neq i,j} 1\{a_{jq,t-1} > 0\} 1\{a_{jk,t-1}\} a_{kq,t-1} \tag{2.2}$$

. Hence z_{ipt} represents "aggregate contribution by i's peers-of-peers defined by project p in month $t-1$".[52] To avoid concerns of reverse causality, we use peers-of-peers contribution in the previous month $t-1$ to instrument for peer contribution in month t. Since Equation (2.1) postulates that agents respond to aggregate as opposed to average peer contribution, we construct the peers-of-peers instrument in a similar fashion.

Threats to Identification

sec:pg-reduced-id-threats

The validity and strength of the peers-of-peers instrument z_{ipt} instrument for peer contribution $a_{\text{-}ipt}$ rests on several assumptions.

Assumption 1. *No isolated contributors.*

[51] We consider the contribution of peers-of-peers in the previous period to mitigate concerns over reverse causality.

[52] Since the sample is constructed by measuring all contribution around a set of core collaborative projects, it is important to note that agents in the empirical sample also contribute to outside projects. Hence projects $q \neq p$ and contribution levels $\{a_{jqt}, a_{kqt}\}$ may not be present in the sample.

Most obviously, contributors need to have peers in order to assess the influence of peer effects. Moreover their peers must also have peers. While by construction of the sample each project has at minimum 15 distinct contributors over its lifespan, we acknowledge that the empirical sample includes a small share of observations in which only a single agent makes a contribution in that time period.[53] We should reasonably expect such observations to both weaken the relationship between the instrument z_{ipt} and peer effort a_{ipt} and introduce downward bias to estimates of the peer effect coefficient δ.

Assumption 2. *For each agent i, there exists a set of projects i will never contribute to, independent of the cost of contribution.*

Agents cannot be peers with everyone. For the exclusion restriction to hold, it is necessary that peers-of-peers contribution influences individual contribution only through peer contribution. Hence we need a sufficient level of contribution behavior where agents are connected indirectly through peers.[54] Consider a setting in which all agents contribute to all projects. In this setting, agent i's contribution level is directly influenced by other agents since the "peers of i's peers" are really just i's peers.

Assumption 3. *Conditional on observable influences, agents substitute contribution effort between projects.*

In other words, the peers-of-peers effort z_{ipt} is conditionally correlated with aggregate peer contribution $a_{\text{-}ipt}$ for the relevance condition to hold: $\text{Cov}(a_{\text{-ipt}}, z_{ipt} \mid \boldsymbol{X}) \neq 0$. This is our most critical assumption. For the instrument to be relevant, there needs to exist some

[53]As noted in Section 2.4, these observations comprise about 6.8% of the empirical sample.

[54]In other words, the social network of contribution cannot be a complete graph and a sufficient number of "intransitive triads" exist in the contribution network (**bramoulle2009identification**; **de2010identification**; **graham2015methods**).

degree of influence of peers-of-peers contribution on peer contribution in aggregate. The immediate concern with using "peers-of-peers" contribution to instrument for peer effort in Equation (2.1) is that the objective of reduced form analysis is to test the null hypothesis of $\delta = 0$: no influence of peer contribution on individual contribution within projects. However, it is important to note that the peers-of-peers instrument operates through substitution with peer contribution *between* projects while the null hypothesis for Equation (2.1) only accounts for substitution with peer effort *within* projects. There is no reason *ex ante* that one substitution pattern precludes the other. Additionally, we argue that the peers-of-peers contribution levels are relevant conditional on other exogenous or predetermined factors drive peer contribution, such as cumulative contribution in a project (i.e. $\text{Cov}(a_{\text{-}ipt}, z_{ipt} \mid \boldsymbol{X}_{ipt}) \neq 0$).[55] Combining these arguments, we asserts that if these assumptions hold then the peers-of-peers instrument z_{ipt} drives some degree of meaningful variation in peer contribution $a_{\text{-}ipt}$ that is quasi-random and therefore exogenous from the perspective of the individual i.

2.5.3 Results

sec:reduced-results

We present baseline estimates for the peer effects parameter δ of the reduced form model from Equation (2.1) in Table 2.A.2. Columns (1) through (3) of Table 2.A.2 present ordinary least squares (OLS) estimates while Columns (4) through (6) present instrumental variables estimates using two-stage least squares (IV 2SLS). We use peers-of-peers contribution z_{ipt} to instrument for peer effort $a_{\text{-}ipt}$. Columns (1) and (4) estimate a specification of Equation (2.1) with only an intercept and the endogenous regressor $a_{\text{-}ipt}$. Columns (2) and (5) add covariate

[55]Furthermore, we note there is some degree of mechanical correlation between $a_{\text{-}ipt}$ and z_{ipt}: a greater number of peers to individual i is likely to subsequently generate a greater number of peers-of-peers.

controls[56] and Columns (3) and (6) add covariate controls alongside project and year-month fixed effects.

The estimates in Table 2.A.2 suggest little evidence of peer effects in contribution on average for the full sample. The OLS estimates in Column (3) are not statistically different from zero after accounting for project fixed effects and observables. The same is true for the corresponding 2SLS estimates. These specifications explain roughly 18% of the variation in individual contribution for the full sample. We note the F statistic from the first stage of the 2SLS estimate in Column (6) of Table 2.A.2 is 64.37.[57] Hence given the model in Equation (2.1) and the sample at hand, we cannot reject the null hypothesis of no peer influence on individual contribution on average ($\delta = 0$) once we account for project fixed effects and covariate controls.

2.5.4 Detailed Analysis and Robustness

sec:pg-reduced-robust

Estimates for the population average δ in Equation (2.1) mask considerable heterogeneity in peer influence on individual level contribution. To explore heterogeneity and provide additional robustness for the reduced form peer effect estimates, we estimate a series of alternative specifications and present the results in Appendix 2.D. Most notably, we find evidence that although the number of contributors has grown over time, contemporaneous peer effects have diminished over time (see Figure 2.B.8). It's reasonable to suspect that peer effects are stronger in the earlier days of GitHub as most projects in the sample were

[56]Control variables include three temporal lags of individual and peer commits to the project, cumulative individual and peers project commits, project quality measured in GitHub stars, quadratic terms for project age, and dummy variables indicating if the individual is a project owner, project member, or if they are affiliated with a firm.

[57]Since the model is just-identified, we report the heteroskedasticity-robust F statistic proposed by **olea2013robust** and recommended by **andrews2019weak**.

still in their infancy. Peer groups were smaller and there were simply fewer developers active on the platform. We also find considerable heterogeneity in peer effects at the project level (see Figure 2.B.7). We interpret this result as heterogeneity in net complementarity of contribution effort that likely varies across projects.[58] Moving beyond contemporaneous peer effects, we find that peer effects are stronger when regressing individual contribution 3 months after on peer contribution 3 months preceding a given period t (see Table 2.A.5). It is likely the case that peer influence takes some time to operate and individuals are induced to contribute on the basis of relatively recent development activity, not necessarily occurring in the same month. This result is important as it suggests that intensive margin peer effects are likely stronger after relaxing our rather restrictive assumption of contemporaneous influence.[59] Finally, we consider the effect of contribution by project "insiders" on the level of contribution by project "outsiders" (see Figure 2.B.9), and find evidence that increased contribution from project insiders "crowds out" contribution from outsiders.

Our results together suggest that while contemporaneous intensive margin peer effects in contribution are limited on average for the entire sample, (1) there exists significant heterogeneity in peer effects across time and projects, (2) contemporaneous peer effects may too narrowly restrict the scope of peer influence, and (3) free-ridership is likely prevalent if dominant core contributors and project insiders carry out the bulk of OSS development.

[58]Moreover, our data and econometric specification treat each individual commit as equivalent contribution. In reality, some contributions might be more important than others. Consider the difference between a typo fix and the introduction of a new feature set. Since the specific lines of code changed by each commit can be observed, an worthwhile continuation of this work ought to examine the interaction between specific types of contribution and peer influence. This approach can give context to the heterogeneity observed in peer effect estimates and guide recommendations for OSS sustainability policy.

[59]We return to this assumption when interpreting our structural estimates of intensive margin peer effects in Section 2.6.5.

We discuss these results in more detail alongside the findings of the structural analysis in Section 2.7.

2.6 Structural Model

sec:pg-structural

While the reduced form analysis begins to reveal patterns of peer influence in OSS contribution, a more refined approach is needed to operationalize the various channels through which peer contribution can influence equilibrium behavior. Importantly, the reduced form specification in Equation (2.1) conflates peer influence along both the extensive and intensive margin into a single parameter. Therefore estimates of contribution peer effects δ are conditional on agents who contribute positive amounts and does not separately account for *why* agents decide to contribute to particular projects. A structural approach allows us to rigorously define micro-founded channels for both equilibrium contribution decisions and peer influence.

There are several key features of our structural model. First, we seek to separately identify marginal private benefits and costs of contribution for each agent. Second, we can characterize each agent's equilibrium contribution decision along both the extensive (i.e. whether to contribute) and intensive (i.e. how much to contribute) margins. Third, we can integrate peer influence into both of these features. Peer effects can potentially influence contribution benefits and productivity as well as intensive and extensive margin contribution in equilibrium. Finally, a fully specified structural approach permits counterfactual analysis. This will allow us to place a value-added estimate for both intensive and extensive margin peer effects in terms of changes to equilibrium contribution. Specifically, we can compare contribution from the observed equilibrium to a counterfactual under which peer effects are absent.[60]

The remainder of this section is organized as follows. First, we set up the model of OSS contribution and introduce its various elements in Section 2.6.1. Our approach combines a model of private provision of public goods (**bergstrom1986private**) into a selection model (**heckman1979sample**). Second, we define an equilibrium in Section 2.6.2. Third, in Section 2.6.3 we specify how peer effects enter into the structural framework. Fourth, we detail our estimation strategy in Section 2.6.4. Fifth, we describe the structural estimates in Section 2.6.5. Finally, we conduct a counterfactual analysis to estimate "value-added" by peer effects in Section 2.6.6.

2.6.1 Setup

sec:pg-structural-setup

Individual agents (i.e. OSS developers) are indexed $i \in \mathcal{N} = \{1, \ldots, N\}$. In each period $t \in \mathcal{T} = \{1, \ldots, T\}$, agents choose contribution levels $a_{ipt} \geq 0$ across a set of OSS projects indexed $p \in \mathcal{P} = \{1, \ldots, P\}$ to maximize incremental contribution utility in each period. To summarize what follows, Table 2.A.8 collects notation for the structural model.

[60]Or, as if software developers contributed to projects in isolation from one another. We operationalize this by setting intensive or extensive margin peer effects equal to zero.

Project Quality

Projects are indexed by their quality y_{pt} at time t. We assume project quality y_{pt} is a simple linear function y of cumulative contribution through t, $a_{pt} \equiv \{a_{ips}\}_{s \leq t}^{i \in \mathcal{N}}$, and parameters b_{pt}:

$$y_{pt} = y(a_{pt}, b_{pt}) = b_{pt} \sum_{i \in \mathcal{N}} \sum_{s \leq t} a_{ips} \tag{2.3}$$

eq:pg-quality

. Note that this specification implies that the parameter b_{pt} represents the marginal product of contribution labor in terms of the quality of project p at time t.[61]

Preferences

Agent preferences are styled after **bergstrom1986private**'s model of private public good provision. We extend this framework to include multiple public goods and time periods. In each period t and for each project p, agents derive utility over (1) direct contribution benefits, (2) project quality, and (3) a numeraire consumption good x_{it} (e.g. time). Specifically,

$$u_{it} = u(a_{it}, y_t, x_{it}) = \sum_{p \in \mathcal{P}} \left(v_{ipt} a_{ipt} - \frac{1}{2} (a_{ipt})^2 + y_{pt} \right) + x_{it} \tag{2.4}$$

eq:pg-utility

. where $a_{it} \equiv \{a_{ipt}\}_{p \in \mathcal{P}}$ and $y_t \equiv \{y_{pt}\}_{p \in \mathcal{P}}$ respectively collect agent i's contributions and project quality across for all $p \in \mathcal{P}$ at time t. Following **bergstrom1986private**, we assume linear preferences: contribution, public good quality, and private good consumption are perfect substitutes.[62] This simplifies the utility maximization problem into independent

[61] While the project quality specification in Equation 2.3 may give rise to concerns over "over-fitting" parameter estimates to the data, we choose this specification purposefully to capture the reality that the marginal product of contribution labor is arguably higher when the project is in early development stages.

[62] More specifically, preferences are quasilinear in x_{it} and therefore increasing an agent's endowment of the numeraire good does not influence demand for contribution.

choices of optimal contribution between projects, subject only to a budget constraint. Agent preferences are shaped by private contribution benefit shocks $v_{ipt} \in \mathbb{R}$, which partially determine the optimal level of contribution in equilibrium. It's critical to note that a realization of v_{ipt} may be such that the agent decides not to contribute to project p at all. Individual project-specific benefit shocks are similar to **athey2014dynamics**'s "arrival of needs" model of OSS dynamics at a macro-level.

Contribution Constraint

We assume that agent contribution a_{ipt} and consumption of the private good x_{it} are constrained by (1) productivity shocks[63] $c_{ipt} > 0$ and (2) endowments ω_{it}:

eq:bc

$$x_{it} + \sum_{p \in \mathcal{P}} c_{ipt} a_{ipt} \leq \omega_{it} \tag{2.5}$$

. In our empirical application, ω_{it} is the agent's endowment of time in period t (i.e. 1 month) and the private good x_{it} is the amount of time spent *not* contributing.[64] As in the reduced form analysis, we measure a_{ipt} as the number of commits agent i makes to project p at time t. This implies that the (inverse) productivity parameters c_{ipt} measure the time cost incurred by agent i making a_{ipt} commits to project p.[65] If $c_{ipt} > c_{jpt}$, agent j is *more* productive

[63]As specified in the contribution constraint in Equation (2.5), c_{ipt} technically represents agent i's "cost" of contribution to project p at time t. The inverse of c_{ipt} is therefore a measure of contribution productivity. Throughout the structural analysis, we will refer to c_{ipt} as both "productivity" and "cost" interchangeably.

[64]In other words, the number of days in month t in which agent i authored no commits.

[65]When an agent's endowment ω_{it} is measured as the number of days in period t and x_{it} is measured as the number of days in the period i was not active on the GitHub platform, the shock c_{ipt} can be interpreted as the number of commits i makes to project p per days i was active on GitHub.

contributing to project p at time t. Finally, we naturally normalize $\omega_{it} = 1$ for all i and t given its interpretation. Since $a_{ipt} \geq 0$, this will in turn imply $0 \leq x_{it} < 1$ and $0 < c_{ipt} < 1$.[66]

Selection Mechanism

Obviously, agents can elect to contribute nothing to certain projects. We therefore introduce a selection mechanism in the spirit of **heckman1979sample**. We assume that projects feature fixed costs of contribution, modelled as a latent productivity threshold z_p.[67][68] Agent i will contribute $a^{\star}_{ipt} > 0$ to project p at time t if their private project-specific ability z_{ipt} exceeds z_p. Furthermore, we assume that z_{ipt} is a linear function of observables $\boldsymbol{W}_{ipt}$, $z_{ipt} = \boldsymbol{\gamma}'\boldsymbol{W}_{ipt} + \epsilon^{z}_{ipt}$ where $\epsilon^{z}_{ipt} \sim \mathcal{N}(0,1)$. Therefore the probability that i contributes to project p in period t is

eq:selection

$$\Pr(a^{\star}_{ipt} > 0) = \Pr(z_{ipt} \geq z_p) = \Phi(\boldsymbol{\gamma}'\boldsymbol{W}_{ipt}) \tag{2.6}$$

. where $\Phi(z)$ is the standard normal cumulative distribution function. In applications, we normalize the contribution threshold $z_p = 0$ for all projects.[69] The vector $\boldsymbol{W}_{ipt}$ contains a set of characteristics that influences i's decision to contribute to project p at time t: the number of peers contributing to project p and both cumulative and lagged contribution for individual i as well as for all agents $j \neq i$ (e.g., historical peer contribution). These factors give important signals to prospective contributors deciding which projects to participate in and will serve as the basis for our extensive margin peer effects discussed in more detail in

[66]In general, we only bound productivity shocks such that $c_{ipt} > 0$. However, in the data the smallest value of positive contribution is $a^{\star}_{ipt} = 1$. It can therefore be shown that this normalization implies also $c_{ipt} < 1$ for all $a^{\star}_{ipt} > 0$.

[67]We acknowledge that this selection mechanism could also be interpreted as a latent *benefit* threshold. See the discussion of structural estimates of extensive margin peer effects in Section 2.6.5.

[68]See **hsieh2018superstar** and **hsieh2020collaboration** for examples of similar selection mechanisms used in models of public good contribution.

[69]To rationalize this normalization, we detail project-specific estimation of $\boldsymbol{\gamma}$ for each p in Section 2.6.3.

Section 2.6.3. For example, an established project featuring many active contributors can provide a useful signal to newcomers uncertain about its quality and maturity, who may be more inclined to contribute under a belief that their efforts will go towards a worthwhile endeavor.

2.6.2 Equilibrium

sec:pg-structural-eq

Timing and Information

At the beginning of each period t, each agent first learns their extensive margin shock ϵ^z_{ipt} for each project. Next, the set of agents who meet the productivity threshold $z_{ipt} \geq z_p$ and decide to contribute to project p learn their benefit and productivity shocks (v_{ipt}, c_{ipt}). We assume that all shocks are public information: agents know who will contribute to which project and how much they will contribute. In the following subsections, we characterize both extensive and intensive margin decisions and the resulting equilibrium.

Extensive Margin Decision

Following the selection mechanism described in Equation (2.6), agent i will contribute $a^\star_{ipt} > 0$ if and only if $z_{ipt} \geq z_p$ upon learning ϵ^z_{ipt}. Otherwise, if an agent does not cover the productivity threshold, they will decide not to contribute to project p at all: $z_{ipt} < z_p \iff a^\star_{ipt} = 0$.

Intensive Margin Decision

Agents with $z_{ipt} \geq z_p$ next determine an optimal, positive contribution level $a^{\star}_{ipt} > 0$. Taking marginal private benefit and productivity shocks (v_{ipt}, c_{ipt}) as given, each agent i chooses an allocation $(a_{ipt}, y_{pt}, x_{it})$ to maximize incremental utility u_{it}:

$$\begin{aligned} \max_{a_{ipt}>0, y_{pt}, x_{it} \in [0,1)} \quad & \sum_{p \in \mathcal{P}} \left(v_{ipt} a_{ipt} - \frac{1}{2} (a_{ipt})^2 + y_{pt} \right) + x_{it} \\ \text{s.t.} \quad & x_{it} + \sum_{p \in \mathcal{P}} c_{ipt} a_{ipt} \leq 1 \\ & y_{pt} = b_{pt} \sum_{j} \sum_{s \leq t} a_{jps} \end{aligned} \tag{2.7}$$

eq:pg-ump

. Under the intensive margin decision characterized by System 2.7, each agent i explicitly takes into account (1) shocks (v_{ipt}, c_{ipt}) and (2) cumulative contribution to project p. To account for affinities and experience formed in particular projects, we allow an agent's cumulative and lagged contribution history to influence their benefit and productivity shocks in Section 2.6.3.

To characterize each agent's intensive margin contribution behavior in equilibrium, we observe that the first order necessary conditions for optimal, non-zero contribution $a^{\star}_{ipt} > 0$ imply

$$a^{\star}_{ipt} = b_{pt} + v_{ipt} - c_{ipt} \tag{2.8}$$

eq:eq-contrib-level

. In other words, should agent i decide to contribute to project p at time t, her optimal level of contribution equals the sum of the marginal product of labor in terms of public good quality b_{pt}, the marginal private benefit of contribution v_{ipt}, and the marginal private cost of contribution c_{ipt}. All else equal, agents contribute more when either their marginal product of labor or marginal private benefits of contribution are higher and less when the marginal cost of contribution (i.e. inverse productivity) is higher.[70]

Combining the optimal intensive margin choice of contribution in Equation (2.8) and the extensive margin decision (i.e. selection mechanism) in Equation (2.6), a given agent i's equilibrium contribution strategy for project p at period t can be summarized as

$$a^{\star}_{ipt} = \begin{cases} b_{pt} + v_{ipt} - c_{ipt} & \text{if} \quad \boldsymbol{\gamma}'\boldsymbol{W}_{ipt} \geq \epsilon^{z}_{ipt} \\ 0 & \text{if} \quad \boldsymbol{\gamma}'\boldsymbol{W}_{ipt} < \epsilon^{z}_{ipt} \end{cases} \qquad \text{eq:eq-contrib} \quad (2.9)$$

.

2.6.3 Peer Effects

sec:pg-structural-peer-effects

We allow peers to influence equilibrium contribution decisions along both the extensive and intensive margins for equilibrium contribution behavior described in Equation (2.9). To disentangle these margins, we will assume separate channels of influence for each mechanism. Historical peer contribution will form the basis for peer effects along the extensive margin. Conditional on the set of agents who contribute a strictly positive level, correlation between the realized benefit and productivity shocks, (v_{ipt}, c_{ipt}), of an individual and her peers will form the basis for peer effects for the intensive margin choice. We formalize these peer effect channels in the following two subsections.

[70]Equation (2.8) is a linear form of the optimal public good contribution level of derived by **bergstrom1986private**, reflecting that private public good contribution is driven by heterogeneity in both benefit and cost heterogeneity.

Extensive Margin

To integrate peer influence into the extensive margin contribution decision, we disaggregate influences over agent i's latent ability threshold for project p at period t, $z_{ipt} = \boldsymbol{\gamma}'\boldsymbol{W}_{ipt} + \epsilon^z_{ipt}$, into characteristics specific to i or project p, $\boldsymbol{\beta}'_z\boldsymbol{X}_{ipt}$ (individual controls), and those related to peers $j \neq i$, $\boldsymbol{\gamma}'\boldsymbol{W}_{ipt}$ (peer influences).[71] Specifically, we include (1) the number of agents contributing to project p in period $t-1$ as well as (2) cumulative and lagged peer contribution to project p in the vector $\boldsymbol{W}_{ipt}$. This is designed to capture the fact that past contribution to OSS projects by peers is a public information good itself and may lay the foundation for subsequent contribution.[72] On the other hand, agents may also choose to free-ride should cumulative project contribution reach a particular level. The vector $\boldsymbol{X}_{ipt}$ contains measures such as individual i's cumulative and lagged contribution, and therefore accounts for i's own accumulated experience with project p. Furthermore, we allow parameter vectors $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}_z$ to vary by project and period. In addition to simplifying estimation[73], estimating separate parameters for each project implicitly accounts for project-varying characteristics that may influence selection beyond contribution history.[74] For each project, the selection mechanism in Equation (2.6) becomes

eq:extensive-peer

$$\Pr(a^\star_{ipt} > 0) = \Phi(\boldsymbol{\gamma}'\boldsymbol{W}_{ipt} + \boldsymbol{\beta}'_z\boldsymbol{X}_{ipt}) \tag{2.10}$$

[71]With a slight abuse of notation for simplicity.

[72]The influence of past actions by peers is also considered in a similar fashion by **bollinger2012peer**, who use cumulative solar panel installations in a neighborhood to predict current period adoptions.

[73]Estimating an analogous model $\Pr(a^\star_{ipt} > 0) = \Phi(\boldsymbol{\gamma}'\boldsymbol{W}_{ipt} + \boldsymbol{\beta}'_z\boldsymbol{X}_{ipt})$ with a single coefficient $\boldsymbol{\gamma}$ would entail a single regression with $N \cdot P \cdot T = 107,250 \cdot 2,287 \cdot 134 = 32,867,620,500$ observations at the individual level.

[74]This amounts to including a distinct constant term in each N-length vector $\boldsymbol{X}_{ipt}$ for each p.

. The parameter vector $\boldsymbol{\gamma}$ captures project-specific peer effects along the extensive margin as a function of historical peer contribution activity. If $\boldsymbol{\gamma} > 0$, the likelihood of contribution is increasing in past peer contribution $\boldsymbol{W}_{ipt}$.[75]

Intensive Margin

For each $a^{\star}_{ipt} > 0$, we can separately recover the shocks v_{ipt} and c_{ipt} by using the equilibrium contribution level in Equation (2.8), the budget constraint in Equation (2.5), and the project quality function in Equation (2.3).[76] Therefore we can develop a framework for assessing contemporaneous peer influence for both individual private benefits and productivity along the intensive margin, conditional on the set of agents with strictly positive contribution levels. In the context of our model, this can be measured by the degree to which shocks (v_{ipt}, c_{ipt}) are correlated between peers in a given project and period.

We separate peer effects in contribution productivity, c_{ipt}, from peer effects in private contribution benefits, v_{ipt}, by using distinct peer effect specifications similar in structure to the reduced form peer effects specification in Equation (2.1). First, we assume that agent productivity is at least partially determined by peer effects:

eq:productivity-peer

$$c_{ipt} = \delta_c \bar{c}_{\text{-}ipt} + \boldsymbol{\beta}_c' \boldsymbol{X}_{ipt} + \epsilon^c_{ipt} \tag{2.11}$$

[75]Notice that

$$\frac{\partial \Pr(a^{\star}_{ipt} > 0)}{\partial \boldsymbol{W}_{ipt}} = \boldsymbol{\gamma}\phi(\cdot) > 0$$

[76]Estimation is covered in detail in Section 2.6.4 as well as Section 2.E of the appendix.

. where $\overline{c}_{\text{-}ipt} \equiv \frac{1}{n_{pt}-1}\sum_{j\neq i} 1\{a^{\star}_{ipt} > 0\}c_{jpt}$ and $n_{pt} \equiv \sum_{i\in\mathcal{N}} 1\{a^{\star}_{ipt} > 0\}$ define the mean of productivity shocks for i's contemporaneous peers in project p.[77] Like the extensive margin specification in Equation (2.10), $\boldsymbol{X}_{ipt}$ are a vector of observables and fixed effects such as lagged and cumulative contribution. Conditional on covariates, δ_c captures the average correlation in productivity shocks amongst peers for a given project and time period. When $\delta_c < 0$, individual costs of contribution are negatively correlated with peer costs, suggesting positive peer effects in terms of productivity.

Similarly, private benefit shocks (e.g., private "needs" or returns to contribution) are modelled as follows:

eq:benefit-peer

$$v_{ipt} = \delta_v \overline{v}_{\text{-}ipt} + \boldsymbol{\beta}_v' \boldsymbol{X}_{ipt} + \epsilon^v_{ipt} \tag{2.12}$$

. When $\delta_v > 0$, individual private benefits are positively correlated with those of their peers, suggesting pro-social peer effects.

To summarize, extensive margin peer effects are parameterized by $\boldsymbol{\gamma}$. Conditional on the set of agents who do contribute, intensive margin (contemporaneous) peer effects are parameterized by (δ_c, δ_v).[78] The framework for extensive and intensive margin peer effects in this structural approach captures several desirable properties. First, we model each margin independently, allowing us to estimate them separately. The source of extensive margin effects is historical peer contribution and the source for intensive margin effects is contemporaneous correlation with contributing peers. Second, given that we observe each agent's

[77] When there is only a single agent contributing to a project, $\overline{c}_{\text{-}ipt} = 0$.

[78] In a more general sense, elements in the vectors $(\boldsymbol{\beta}_v, \boldsymbol{\beta}_c)$ may include terms related to historical (i.e., lagged or cumulative) peer contribution that, similar to the extensive margin parameterization, may also plausibly influence positive contribution levels. With respect to the counterfactual analysis in Section 2.6.6, we are more broadly interested in parameters related to both types of peer influence, contemporaneous and accumulated, on private benefits and productivity.

extensive margin decision for every project and period, we can estimate $\boldsymbol{\gamma}$ separately for each p. Motivated in part by the considerable project-level heterogeneity revealed in the reduced form analysis, this parameterization is more flexible than estimating a single parameter and can account for a range of project-varying extensive margin influences.[79] Finally, we use the OSS contributor's time-constrained utility maximization problem to separately recover benefit and productivity shocks. Unpacking net benefits allows us to further isolate the channels of peer influence in intensive margin contribution. In the next section, we turn our attention to estimating parameters of interest.

2.6.4 Estimation

sec:pg-structural-estimation

In this section, we provide a high-level overview of our structural estimation strategy and objectives. A more thorough and detailed treatment is provided in Section 2.E of the appendix. Given data $(a_{ipt}, y_{pt}, x_{it})$ for all $i \in \mathcal{N}, p \in \mathcal{P}$, and $t \in \mathcal{T}$, we develop an estimation strategy to recover the following:

1. Marginal product of labor parameters $\boldsymbol{b} = (b_{pt})$ from the project quality function in Equation (2.3).
2. Private benefit and productivity shocks $\boldsymbol{s} = (v_{ipt}, c_{ipt})$ for all $a^{\star}_{ipt} > 0$ from the equilibrium contribution level in Equation (2.8).
3. (Extensive margin peer effects) Parameters $(\boldsymbol{\gamma}, \boldsymbol{\beta}_z)$ from Equation (2.10).
4. (Intensive margin peer effects) Parameters $(\delta_c, \delta_v, \boldsymbol{\beta}_c, \boldsymbol{\beta}_v)$ from Equations (2.11) and (2.12).■

[79] Note that given data limitations, we cannot estimate intensive margin peer effects (δ_v, δ_c) separately for each project and period. In many cases, our empirical sample contains only a single contribution for the month to a given project.

The parameters of interest are $\boldsymbol{\delta} = (\delta_c, \delta_v)$, which drive intensive margin peer effects, and $\boldsymbol{\gamma}$, which drive extensive margin peer effects. For each project $p \in \mathcal{P}$, our estimation strategy is as follows:

1. Assume disturbances are jointly normally distributed $(\epsilon^z_{ipt}, \epsilon^v_{ipt}, \epsilon^c_{ipt}) \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$, independent and identically distributed between agents and time. Within the variance-covariance matrix $\boldsymbol{\Sigma}$, assume that $\sigma^2_z = 1$.
2. Given data (a_{ipt}, y_{pt}), recover b_{pt} using Equation (2.3).
3. Given data $(a_{ipt}, y_{pt}, x_{it})$ and b_{pt}, recover shocks (v_{ipt}, c_{ipt}) using Equation (2.9), Equation (2.5), Equation (2.3) by means of GMM[80].
4. Given data $(1\{a_{ipt} > 0\}, \boldsymbol{W}_{ipt}, \boldsymbol{X}_{ipt})$ and shocks (v_{ipt}, c_{ipt}) recover $(\boldsymbol{\gamma}, \boldsymbol{\delta}, \boldsymbol{\beta}, \boldsymbol{\Sigma})$, where $\boldsymbol{\delta} = (\delta_v, \delta_c)$ and $\boldsymbol{\beta} = (\boldsymbol{\beta}_z, \boldsymbol{\beta}_v, \boldsymbol{\beta}_c)$, via MLE (**zhao2020new**).

Parameters $\boldsymbol{\theta} = (\boldsymbol{b}, \boldsymbol{\gamma}, \boldsymbol{\delta}, \boldsymbol{\beta}, \boldsymbol{\Sigma})$ allow us to completely characterize the data generating process for the structural model, a necessary prerequisite simulating policy counterfactuals.

2.6.5 Structural Estimates

sec:pg-structural-estimates

Benefit and Productivity Shocks

We present the recovered values for marginal product of labor parameters b_{pt} and shocks (v_{ipt}, c_{ipt}) for all $a^\star_{ipt} > 0$ in Figure 2.B.10. The first panel of Figure 2.B.10 contains distributions of marginal private benefit shocks v_{ipt} grouped by year. Similarly, productivity (inverse marginal cost) shocks are presented in the second panel. Several patterns emerge

[80] For each i and t, there are $2P$ unknowns: v_{ipt} and c_{ipt} for each $a_{ipt} > 0$. There are P first order conditions from Equation (2.9), P equations for project quality form Equation (2.3), and one budget constraint. Overall, this implies $NT(2P + 1)$ moment conditions and $2NPT$ unknowns.

from the recovered shocks. First, these distributions are relatively stable over time. Second, when considering the entire sample, benefit and productivity shocks are relatively uncorrelated with one another at the individual level ($\text{Corr}(v_{ipt}, c_{ipt}) = -0.081$). There is, however, evidence of a temporal trend in shock correlation over the sample period: Figure 2.B.11 reveals that benefits v_{ipt} demonstrate a strong negative correlation with productivity c_{ipt} ($\text{Corr}(v_{ipt}, c_{ipt}) \approx -0.6$ to -0.5) in early periods of GitHub that trend towards 0 nearer the end of the sample period. Recall that $\text{Cov}(v_{ipt}, c_{ipt}) < 0$ implies that greater marginal private benefits are associated with lower private marginal costs of contribution. Together, these data seem to suggest that early stages of GitHub OSS collaboration featured highly productive individuals with greater net benefits of contribution relative to later entrants. In later periods, incentives to become more productive may be weaker given greater peer participation. Corroborating the findings of the reduced form analysis, this structural evidence further supports the notion that the prevalence of free-ridership has likely increased on average as the GitHub platform has grown in size.

The third panel of Figure 2.B.10 contains estimates of marginal product of labor parameters b_{pt} from Equation (2.3). By virtue of functional form assumption for project quality, b_{pt} tend to be largest in the early stages of project development: the initial commits tend to be the most important in determining project quality. Since b_{pt} tends to decline over a project's lifespan, productivity and benefit shocks explain sustained contribution.

Extensive Margin Peer Effects

Figure 2.B.12 contains estimates for extensive margin peer effects captured by the parameter $\boldsymbol{\gamma}$ of Equation (2.10). Two key patterns emerge. First, the likelihood of contribution is increasing in the number of peers who contributed in the previous period while decreasing in lagged and cumulative contribution levels. Second, the coefficients for lagged number of peers are much larger in magnitude compared with lagged and cumulative contribution. Taken together, these estimates underscore an intuitive if not trivial fact: agents are more likely to join projects growing in the number of contributors. To a lesser extent, the likelihood of contribution declines as projects grow larger in terms of the size of the codebase. We can interpret this finding in several ways. On one hand, actively developed projects provide positive peer effects that incentivize contribution from outsiders. On the other, it may simply be the case that increased development activity in the early stages of a project may signal a project's promise or quality to prospective contributors. To rule out this signalling mechanism, we estimate extensive margin peer effects at the project level and control for observable project quality. Moreover, it appears that contribution incentives lessen as a project matures into a stable state[81], as it is likely that less contribution is required.

In Equation (2.6) of Section 2.6.1, we model extensive margin selection into projects as a latent productivity threshold. We acknowledge that the largest driver of project participation, the number of peers contributing, can influence both benefits and costs of contribution. Given that z_{ipt} is unobserved and a function of both individual and peer historical contribution, we could just as easily have modelled z_p as a latent benefit threshold for project p. At best, we can only say our structural approach finds evidence that projects with many actively contributing members increase an individual's *net* benefit of contribution and therefore positively impacts extensive margin participation.

[81]This phase of project development is sometimes referred to as "maintenance mode" as opposed to "active development".

Intensive Margin Peer Effects

Project-level estimates of the intensive margin peer effects δ_v and δ_c are summarized in Figure 2.B.13. Much like the project-level reduced form estimates displayed in Figure 2.B.7, both benefit and productivity peer effects are distributed relatively symmetrically around 0. A relatively strong positive correlation between δ_v and δ_c, $\text{Corr}(\delta_v, \delta_c) = 0.843$, implies greater benefit correlation between peers within projects is also associated with greater marginal cost correlation between peers. Ultimately, this suggests an *inverse* relation between benefit and productivity shocks correlation: the net effect of peer influence along the intensive margin leads developers to contribute more at greater marginal cost. The lack of correlation between v_{ipt} and c_{ipt} at the individual level further supports this finding. We interpret this positive correlation between δ_v and δ_c as evidence that pro-social peer effects dominate productivity peer effects. Consistent with the reduced form analysis, there is no strong evidence that contemporaneous peer effects improve intensive margin productivity across projects on average. In other words, we cannot say that OSS contributors make each other more productive along the intensive margin when considering contemporaneous influence.

Summary

sec:pg-structural-estimates-summary

To summarize, structural estimation of benefit and cost shocks along with extensive and intensive margin peer effects seem to corroborate evidence from our reduced form approach and descriptive statistics from the empirical sample. First, extensive margin peer effects are a much more important driver of project growth relative to intensive margin effects. Consistent with the "casual contributor" phenomenon described anecdotally by OSS maintainers, projects with many contributors are more likely to attract incremental contributions from outsiders than they are to attract dedicated maintainers. Second, in terms of the ratio of private contribution benefits to costs, early OSS contributors on the GitHub platform enjoyed greater net benefits of contribution relative to later entrants. Finally, pro-social forces seem to trump peer effects with respect to intensive margin productivity. There is little evidence to suggest peers reduce marginal costs of contribution along the intensive margin.

It is important to note that these results and their subsequent interpretation rest on some assumptions made in our modelling approach. First, as in the reduced form analysis, we place a restrictive assumption that intensive margin peer influence operates contemporaneously. As shown in Section 2.5.4, relaxing this assumption will likely lead to larger estimates of peer effects along this margin. Second, the functional form assumptions made in our structural approach may simplify estimation at the expense of some flexibility. Specifically, Equations (2.3) (project quality) and (2.4) (agent preferences) omit certain terms such that benefit and productivity shocks can be point identified. These assumptions may bias our parameter estimates away from their true values. Subsequent work would do well to relax these assumptions by either additional structure, data, or a more flexible estimation strategy.

2.6.6 Counterfactual Analysis

sec:pg-structural-counterfactual

Value of Peer Effects

sec:counterfactual-peer

While the presence of positive[82] peer effects precludes a socially optimal level of contribution under private provision, they may increase equilibrium contribution beyond what would be provided in a world without peer influence. In this sense, peer effects have the potential to effectively subsidize the cost of private provision. Indeed, the preliminary analysis of the structural estimates in the preceding subsection gives reason to believe that peer behavior can drive preference and cost heterogeneity along both the extensive and intensive margins, albeit to differing degrees. To gauge the "value–added" by highly nuanced peer effects in terms of aggregate contribution labor, we use the estimates of the structural model to derive a counterfactual equilibrium in which peer effects are absent.

We consider the following policy counterfactual: suppose peer effects do not exist. In other words, past peer contribution does not influence an individual's likelihood of contribution and private benefit and productivity shocks are uncorrelated for individuals who decide to contribute. This scenario roughly corresponds to "siloed" development: agents independently contribute to a public good but do so without interaction with peers or the contribution levels of peers. What is the resulting level of contribution? Specifically, we begin by setting extensive and intensive margin peer effect parameters to zero: $\boldsymbol{\gamma} = \boldsymbol{\delta} = \mathbf{0}$. We then use the remaining parameter estimates $(\boldsymbol{\beta}, \boldsymbol{\Sigma}, b_{pt})$ to re-simulate the data generating process described by the structural model for the entire sample period.[83] To compare the

[82]Note that in the canonical public goods model of **samuelson1954pure**, negative peer effects (e.g., a congestion externality) could potentially offset the classic positive externality that drives free-riding and under-contribution relative to the social optimum.

[83]We use data $(a_{ipt}, y_{pt}, x_{it})$ and recovered shocks (v_{ipt}, c_{ipt}) to "seed" initial conditions for period $t =$ 2008–04–01.

relative impact of extensive and intensive margin effects, we also simulate a counterfactual under which extensive margin peer effects exist while only intensive margin peer effects are absent.

The results of these counterfactuals, in terms of aggregate contribution across all projects, are summarized alongside the observed data in Figure 2.B.14. Two key patterns emerge. First, the counterfactual without peer effects results in aggregate contribution approximately 55.6% lower compared with the observed equilibrium. By June 2019, aggregate contribution across all projects in the observed sample totals in excess of 5.519 million commits. Under the counterfactual scenario with no peer effects, aggregate contribution is reduced to approximately 2.452 million commits. If contributors make 2–3 commits per labor hour on average, a back-of-the-envelope calculation for this shortfall of 3.067 million commits implies a loss of 1–1.5 million OSS labor hours relative to the observed equilibrium. The median hourly wage of \$52.95 for software developers in the U.S. suggests the value–added by OSS peer effects in our sample at \$54.132 to \$81.199 million USD. Second, extensive margin peer effects constitute the overwhelming of the value added. Figure 2.B.14 shows that the counterfactual scenario in which only extensive margin peer effects exist (i.e., $\boldsymbol{\delta} = 0$) closely matches the observed equilibrium under which both extensive and intensive margin effects are active. As discussed in Sections 2.5.4 and 2.6.5, the diminished role of intensive margin effects may be a result of our narrowly tailored definition for peer influence.

2.7 Discussion

sec:pg-discussion

Using the context of OSS, we have studied the influence of peer effects on the private provision of public goods in detail. We use both reduced form and structural approaches to (1) address non-random selection, (2) distinctly model intensive and extensive margin peer effects, and (3) disentangle marginal private benefits and costs of contribution as distinct channels of influence. Our findings in both approaches are consistent with anecdotal evidence: OSS project growth is largely driven by some combination of dedicated large-share core contributors and the arrival of many small-share contributors. We find little evidence that peers make each other more productive on average: contemporaneous intensive margin peer effects are heterogeneous across projects but do appear to have been larger in the early days of GitHub. Moreover, structural estimates suggest the effect of peer influence on average is that agents contribute greater levels when their peers do, but at greater marginal cost. Our counterfactual analysis seeks to estimate the value-added of peer effects in terms of private public good provision. Driven almost exclusively by extensive margin peer effects, we find that cumulative contribution is approximately 56% lower under the scenario where peer effects are not present.

We can interpret the findings of this analysis to highlight some limitations for the potential for peer effects to foster the production of public information goods. We find that while extensive margin effects can drive a significant share of contribution, these effects are decreasing in the size of the peer group. This may arise either (1) if small share contributors free-ride on the efforts of dominant core contributors or (2) become less likely to contribute once a project matures in size. Moreover, the lack of strong, positive peer influence in contribution productivity along the intensive margin suggests that any strategic complementarity or substitution in contribution may simply offset on net. Compared with previous studies which document strong pro-social effects to collaboratively produced public goods (**zhang2011group**; **slivko2014peer**), peer effects in the production of more complex information goods like OSS may be significantly more nuanced.

As noted above, a key takeaway of this study is that agents differ in their willingness to contribute their labor towards the sustained maintenance of OSS used by a larger community. The extent to which peer effects matter for sustaining the quality of OSS public goods likely depends on both (1) the project's use valuation from the wider community and (2) the project's position as a component of OSS infrastructure (**eghbal2016roads**).[84] Whereas the study of peer effects in this chapter focuses purely on the production side of public goods, a promising direction for future research is to explore the welfare implications for behavioral patterns uncovered thus far.[85] Better characterizations of optimal contribution patterns that consider the wider set of beneficiaries to OSS quality allow the researcher to better discern the extent to which peer influences on collaboration truly matter. Such efforts can continue to place the economic significance of peer effects, externalities, and public good production into context for OSS.

[84]For example, if OSS production fits a combinatoric production model in which developers make small, specialized contributions and move on to other projects, peer effects may be of less importance to delivering an efficient equilibrium. On the other hand, network externalities might amplify the importance of maintenance labor and positive peer effects. Consider the example of OpenSSL and the Heartbleed Bug. OSS infrastructure that is widely depended upon but maintained by a small group could likely benefit from additional contribution labor that can at least partially be generated through peer effects.

[85]In other words, subsequent studies would do well to distinguish between the welfare implications of production versus sustained maintenance for complex public information goods like OSS.

Appendices

2.A Tables

app:pg-tables

Table 2.A.1: Descriptive Statistics - Primary Measures in Empirical Sample

tab:desc-stats-sample

Measure	Notation	Count	Mean	SD	Min	Median	Max
Project commits (total)	a_p	2,287	2,490	7,720	23	825	188,292
Individual commits (total)	a_i	107,921	53	669	1	2	186,464
Project commits (monthly)	a_{pt}	96,453	59	294	1	16	73,161
Individual commits (monthly)	a_{it}	421,879	13	129	1	3	73,145
Cumulative individual commits (monthly)	$\tilde{a}_{it}$	421,879	278	1,109	1	28	186,464
Cumulative project commits (monthly)	$\tilde{a}_{pt}$	96,453	1,989	6,295	1	532	188,292
Individual commits (project-month)	a_{ipt}	440,111	13	126	1	3	73,145
Peer commits (project-month)	$a_{\text{-}ipt}$	440,111	188	398	0	59	73,160
Cumulative individual commits (project-month)	$\tilde{a}_{ipt}$	440,111	256	1,076	1	23	186,447
Cumulative peer commits (project-month)	$\tilde{a}_{\text{-}ipt}$	440,111	2,096	6,630	0	262	124,932
Number of peers (project-month)	n_{ipt}	440,111	17	29	0	7	310
Cumulative GitHub Stars (project-month)	y_{pt}	96,294	910	2,924	0	161	81,817
GitHub active days (monthly)	gad_{it}	411,427	3.88	4.67	1	2	31

Table 2.A.2: Reduced Form - Individual Level Peer Effects Estimates (Baseline Estimates for peer effect δ from Equation (**??**))

tab:reduced-full-sample

	OLS			IV 2SLS		
	Individual Commits			Individual Commits		
	(1)	(2)	(3)	(4)	(5)	(6)
Peer Commits	0.0078	0.0065	0.0035	-0.0035	0.0089	0.0102
	(0.0011)	(0.0018)	(0.0034)	(0.0015)	(0.0037)	(0.0251)
Individual Commits (cumulative)	-	0.0332	0.0529	-	0.0333	0.0529
	-	(0.0134)	(0.0434)	-	(0.0134)	(0.0435)
Individual Commits (previous month)	-	0.2604	0.1853	-	0.2604	0.1853
	-	(0.1763)	(0.0902)	-	(0.1763)	(0.0902)
Peer Commits (cumulative)	-	-0.0020	-0.0024	-	-0.0020	-0.0024
	-	(0.0009)	(0.0019)	-	(0.0009)	(0.0019)
Peer Commits (previous month)	-	0.0051	0.0036	-	0.0044	0.0039
	-	(0.0022)	(0.0032)	-	(0.0024)	(0.0046)
Peer Group Size	-	0.0017	0.0898	-	-0.0093	0.0158
	-	(0.0662)	(0.1457)	-	(0.0553)	(0.3760)
Controls	No	Yes	Yes	No	Yes	Yes
Fixed Effects	No	No	Yes	No	No	Yes
N	440,111	433,867	433,867	436,287	433,867	433,867
R^2	0.0006	0.1802	0.2268	-0.0007	0.1802	0.2267
First stage F statistic				6,520	1,151	64.37

Note: Columns (1)-(6) present the coefficient estimate $\hat{\delta}$ from Equation (**??**) in which aggregate peer commits are regressed on individual commits. Standard errors appear in parentheses below the coeffficient estimate. Columns (1), (2), (4), and (5) use heteroskedasticity-robust standard errors while Columns (3) and (6) cluster standard errors by project. Column (4) through (6) additionally report the cluster-robust F-statistic from the first stage of the two-stage least squares procedure. Control variables include three lags of individual and peer commits, cumulative individual and peers commits, project quality measured in GitHub stars, quadratic terms for project age and peer group size, and dummy variables indicating if the individual is a project owner, project member, or if they are affiliated with a firm. Fixed effects included are individual, project, and year-month.

Table 2.A.3: Reduced Form - Individual Level Peer Effects (Estimates of peer effect δ from Equation (**??**) with covariate interaction terms)

tab:reduced-interactions

	OLS			IV 2SLS		
	(1)	(2)	(3)	(4)	(5)	(6)
Peer Commits	0.0078	0.0065	0.0167	-0.0035	0.0497	-0.0721
	(0.0011)	(0.0020)	(0.0105)	(0.0015)	(0.0354)	(0.1562)
Peer Commits ×		-1.47 $\times 10^{-5}$	-0.0002		-0.0002	0.0003
Peer Group Size		(2.59 $\times 10^{-5}$)	(0.0001)		(0.0001)	(0.0009)
Peer Commits ×		0.0004	0.0004		0.0004	0.0004
Lagged Individual Commits		(0.0001)	(0.0002)		(0.0001)	(0.0002)
Peer Commits ×		-3.09 $\times 10^{-6}$	-2.32 $\times 10^{-6}$		-2.54 $\times 10^{-6}$	-2.6 $\times 10^{-6}$
Lagged Peer Commits		(1.54 $\times 10^{-6}$)	(1.13 $\times 10^{-6}$)		(1.16 $\times 10^{-6}$)	(1.5 $\times 10^{-6}$)
Peer Commits ×		-3.82 $\times 10^{-5}$	-4.9 $\times 10^{-5}$		-3.74 $\times 10^{-5}$	-4.89 $\times 10^{-5}$
Cumulative Individual Commits		(1.86 $\times 10^{-5}$)	(3.39 $\times 10^{-5}$)		(1.81 $\times 10^{-5}$)	(3.36 $\times 10^{-5}$)
Peer Commits ×		1.81 $\times 10^{-6}$	1.93 $\times 10^{-6}$		1.83 $\times 10^{-6}$	2.03 $\times 10^{-6}$
Cumulative Peer Commits		(1.07 $\times 10^{-6}$)	(6.3 $\times 10^{-7}$)		(1.08 $\times 10^{-6}$)	(7.57 $\times 10^{-7}$)
Peer Commits ×		6.18 $\times 10^{-8}$	2.82 $\times 10^{-7}$		5.24 $\times 10^{-8}$	-3.03 $\times 10^{-7}$
Project Quality		(1.87 $\times 10^{-8}$)	(3.5 $\times 10^{-7}$)		(2.42 $\times 10^{-8}$)	(9.98 $\times 10^{-7}$)
Peer Commits ×		0.0451	0.3513		0.0329	0.3732
Project Owner		(0.0753)	(0.2380)		(0.0848)	(0.2455)
Peer Commits ×		0.0093	0.0070		0.0001	0.0311
Project Member		(0.0065)	(0.0066)		(0.0039)	(0.0412)
Peer Commits ×		-3.55 $\times 10^{-6}$	7.89 $\times 10^{-6}$		-1.23 $\times 10^{-5}$	3.17 $\times 10^{-5}$
Project Age		(1.54 $\times 10^{-6}$)	(7.89 $\times 10^{-6}$)		(8.59 $\times 10^{-6}$)	(4.73 $\times 10^{-5}$)
Controls	No	Yes	Yes	No	Yes	Yes
Fixed Effects	No	No	Yes	No	No	Yes
N	440,111	433,867	433,867	436,287	433,867	433,867
R^2	0.0006	0.1910	0.2372	-0.0007	0.1891	0.2351
First stage F statistic				6,520	389.5	57.177

Note: Columns (1)-(6) estimate specifications corresponding to those presented in Table 2.A.2 with the inclusion of terms interacted with aggregate peer effort (RHS endogenous term).

Table 2.A.4: Reduced Form - Temporal Heterogeneity in Individual Level Peer Effects (Estimates of peer effect δ from Equation (**??**) for subsamples disaggregated by time period)

tab:reduced-by-period

	OLS			IV 2SLS		
	(1)	(2)	(3)	(4)	(5)	(6)
2008 - 2012	0.0119	0.0271	0.0267	-0.0104	-0.0270	-0.0359
	(0.0011)	(0.0017)	(0.0077)	(0.0020)	(0.0235)	(0.0487)
N	20,399	20,209	20,209	20,262	20,262	20,209
R^2	0.0077	0.5158	0.6288	-0.0196	0.4807	0.6010
First stage F statistic				3,677	57.00	85.29
2012 - 2016	0.0178	0.0155	0.0228	-0.0237	0.0037	-0.1262
	(0.0005)	(0.0006)	(0.0028)	(0.0008)	(0.0016)	(0.2851)
N	146,778	145,952	145,952	146,256	145,952	145,952
R^2	0.0279	0.4975	0.5837	-0.1244	0.4937	0.3716
First stage F statistic				6,637.1	1,543	8.405
2016 - 2019	0.0056	0.0052	0.0032	-0.0013	0.0165	-0.0009
	(0.0010)	(0.0017)	(0.0026)	(0.0020)	(0.0076)	(0.0385)
N	272,934	267,706	267,706	269,769	267,706	269,706
R^2	0.0003	0.1835	0.2696	-0.0001	0.1829	0.2696
First stage F statistic				3,491.0	607.9	35.63

Note: Columns (1)-(6) estimate specifications corresponding to those presented in Table 2.A.2 distinctly for sub-samples disaggregated by time period.

Table 2.A.5: Reduced Form - Beyond Contemporaneous Individual Level Peer Effect Estimates (Estimates for peer effect δ from Equation (2.13))

tab:reduced-pooled

	OLS Individual Commits			IV 2SLS Individual Commits		
	(1)	(2)	(3)	(4)	(5)	(6)
Peer Commits	0.0139	0.0212	0.0068	-0.0145	0.0575	0.0881
	(0.0009)	(0.0042)	(0.0070)	(0.0018)	(0.0110)	(0.0914)
Individual Commits (cumulative)	-	-0.0051	-0.0063	-	-0.0074	-0.077
	-	(0.0020)	(0.0047)	-	(0.0021)	(0.0056)
Individual Commits (previous month)	-	0.3158	0.1036	-	0.3014	0.0949
	-	(0.2402)	(0.2433)	-	(0.2355)	(0.2410)
Peer Group Size	-	-0.0175	-0.0071	-	-0.7290	-1.835
	-	(0.1270)	(0.4223)	-	(0.2148)	(2.068)
Controls	No	Yes	Yes	No	Yes	Yes
Fixed Effects	No	No	Yes	No	No	Yes
N	440,111	433,867	433,867	436,287	433,867	433,867
R^2	0.0021	0.1802	0.2274	-0.0066	0.2200	0.3100
First stage F statistic				4,376	124.9	32.16

Note: Columns (1)-(6) present the coefficient estimate $\hat{\delta}$ from Equation (2.13) in which aggregate peer commits from the preceding 3 months are regressed on individual commits for the subsequent 3 months. Covariates controls and fixed effects correspond to the estimates in Table 2.A.2.

Table 2.A.6: Reduced Form - Project-Level Estimates (Historical Project Contribution regressed on Contemporaneous Project Contribution)

tab:reduced-project-level-contrib

	OLS Project Commits						
	(1)	(2)	(3)	(4)	(5)	(6)	(7)
Project Commits (1 month prior)	0.4502	0.3188	0.3188	0.3186	0.2804	0.3186	0.2803
	(0.2216)	(0.1998)	(0.2000)	(0.0387)	(0.0449)	(0.0386)	(0.0450)
Project Commits (2 months prior)	-	-0.0383	-0.0383	-0.0384	-0.0632	-0.0384	-0.0633
	-	(0.0666)	(0.0659)	(0.0289)	(0.0349)	(0.0288)	(0.0350)
Project Commits (3 months prior)	-	0.1224	0.1223	0.1222	0.0882	0.1221	0.0880
	-	(0.0391)	(0.0384)	(0.0311)	(0.0394)	(0.0309)	(0.0396)
Project Commits (cumulative)	-	0.0156	0.0156	0.0157	0.0159	0.0157	0.0160
	-	(0.0039)	(0.0033)	(0.0067)	(0.0148)	(0.0067)	(0.0149)
Controls	No	Yes	Yes	Yes	Yes	Yes	Yes
Fixed Effects: Time	No	No	Yes	No	No	Yes	Yes
Fixed Effects: Project	No	No	No	No	Yes	No	Yes
Fixed Effects: Language	No	No	No	Yes	No	Yes	No
N	96,453	96,294	96,294	96,294	96,294	96,294	96,294
R^2	0.19555	0.27487	0.27567	0.27494	0.30079	0.27575	0.30159

Note: Columns (1)-(7) contain coefficient estimates for current month total project contribution regressed on project contribution in previous months and the cumulative project contribution. Other controls include lagged and cumulative numbers of project contributors, project quality, and quadratic terms for project age. Columns (1) and (2) report heteroskedasticity-robust standard errors, Column (3) clusters standard errors by month, Columns (4) and (6) by project language, and Columns (5) and (7) by project.

Table 2.A.7: Reduced Form - Project-Level Estimates (Historical Number of Contributors regressed on Contemporaneous Number of Contributors)

tab:reduced-project-level-ncontrib

	OLS						
	Number of Project Contributors						
	(1)	(2)	(3)	(4)	(5)	(6)	(7)
Number of contributors (1 month prior)	0.8822	0.6082	0.6085	0.6080	0.5187	0.6082	0.5188
	(0.0166)	(0.0430)	(0.0459)	(0.0422)	(0.0380)	(0.0423)	(0.0382)
Number of contributors (2 months prior)	-	0.1175	0.1167	0.1173	0.0778	0.1166	0.0774
	-	(0.0408)	(0.0477)	(0.0248)	(0.0346)	(0.0251)	(0.0348)
Number of contributors (3 months prior)	-	0.1772	0.1769	0.1769	0.1307	0.1767	0.1306
	-	(0.0311)	(0.0336)	(0.0266)	(0.0196)	(0.0263)	(0.0194)
Number of contributors (cumulative)	-	0.0007	0.0007	0.0007	-0.0008	0.0007	-0.0008
	-	(0.0002)	(0.0002)	(0.0002)	(0.0004)	(0.0003)	(0.0008)
Project Commits (cumulative)	-	1.01e-5	1.04e-5	1.01e-5	0.0001	1.05e-5	0.0001
	-	(8.09e-6)	(8.07e-6)	(8.1e-6)	(1.69e-5)	(1.13e-5)	(4.33e-5)
Controls	No	Yes	Yes	Yes	Yes	Yes	Yes
Fixed Effects: Time	No	No	Yes	No	No	Yes	Yes
Fixed Effects: Project	No	No	No	No	Yes	No	Yes
Fixed Effects: Language	No	No	No	Yes	No	Yes	No
N	96,453	96,294	96,294	96,294	96,294	96,294	96,294
R^2	0.77107	0.79349	0.79494	0.79354	0.81159	0.79499	0.81299

Note: Columns (1)-(7) contain coefficient estimates for the number of contributors for a project in the current month regressed on the number of contributors in previous months and the cumulative number of project contributors. Other controls include lagged and cumulative project contribution, project quality, and quadratic terms for project age. Columns (1) and (2) report heteroskedasticity-robust standard errors, Column (3) clusters standard errors by month, Columns (4) and (6) by project language, and Columns (5) and (7) by project.

Table 2.A.8: Structural Model Notation

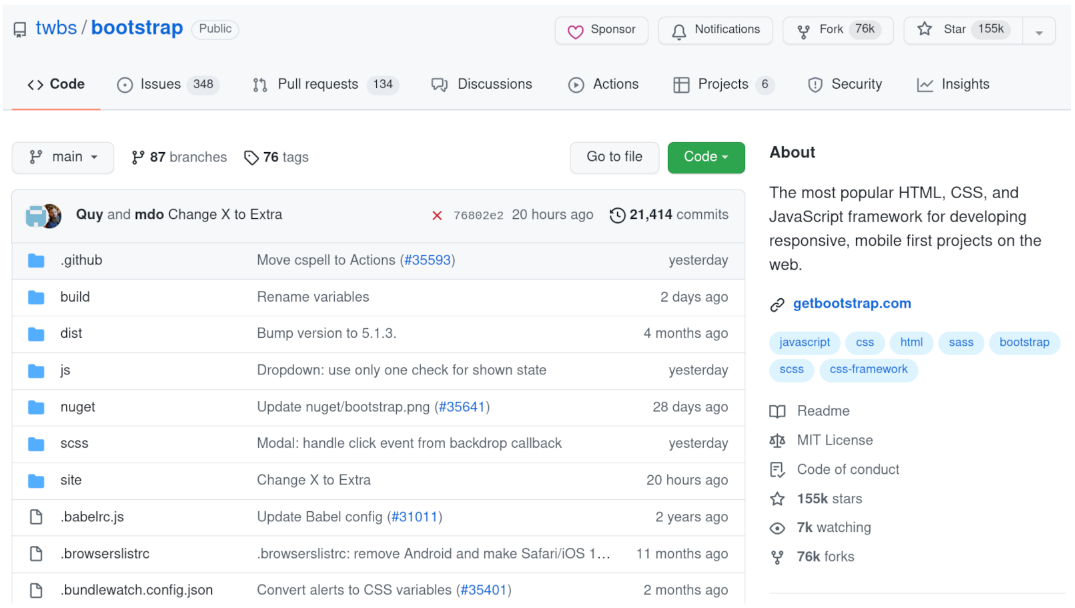
$i, j \in \mathcal{N}$	agents where $\lvert\mathcal{N}\rvert = N$
$p \in \mathcal{P}$	OSS projects where $\lvert\mathcal{P}\rvert = P$
$t \in \mathcal{T}$	time periods where $\lvert\mathcal{T}\rvert = T$
$a_{ipt} \in \mathbb{R}_+$	agent i's contribution to project p in period t
$y_{pt} \in \mathbb{R}$	quality of project p in t
$b_{pt} = \frac{\partial y_{pt}}{\partial a_{ipt}}$	marginal product of labor in terms of public good quality
$x_{it} \in \mathbb{R}_+$	agent i's consumption of numeraire good (e.g. time)
$\omega_{it} \in \mathbb{R}_+$	agent i's numeraire endowment
z_{ipt}	agent i's latent "ability" in project p at time t (see Equation (2.6))
z_p	project p's latent "ability threshold" (see Equation (2.6))
v_{ipt}	private contribution benefit for agent i in project p at time t
c_{ipt}	contribution cost (inverse productivity) for agent i in project p at time t
$\boldsymbol{\gamma}$	extensive margin peer effects (see Equation (2.10))
δ_v	intensive margin peer effects for marginal private benefits of contribution
δ_c	intensive margin peer effects for marginal private costs of contribution (see Equation (2.11))
$\boldsymbol{\beta}_z$	control variable parameters for latent agent productivity z_{ipt} in extensive margin decision (see Equation (2.10))
$\boldsymbol{\beta}_v$	control variable parameters for marginal private benefits of contribution (see Equation (2.12))
$\boldsymbol{\beta}_c$	control variable parameters for marginal private cost of contribution (see Equation (2.12))
ϵ^z_{ipt}	Unobserved factors influencing extensive margin decision (see Equation (2.10))
ϵ^v_{ipt}	Unobserved factors influencing marginal private benefit shock v_{ipt} (see Equation (2.12))
ϵ^c_{ipt}	Unobserved factors influencing marginal private cost shock c_{ipt} (see Equation (2.11))

tab:struct-notation

2.B Figures

app:pg-figures

Figure 2.B.1: Example GitHub Repository Page: twbs/bootstrap



twbs / bootstrap Public

Sponsor · Notifications · Fork 76k · Star 155k

Code · Issues 348 · Pull requests 134 · Discussions · Actions · Projects 6 · Security · Insights

main · 87 branches · 76 tags · Go to file · Code

Quy and mdo Change X to Extra ✗ 76802e2 20 hours ago 21,414 commits

.github	Move cspell to Actions (#35593)	yesterday
build	Rename variables	2 days ago
dist	Bump version to 5.1.3.	4 months ago
js	Dropdown: use only one check for shown state	yesterday
nuget	Update nuget/bootstrap.png (#35641)	28 days ago
scss	Modal: handle click event from backdrop callback	yesterday
site	Change X to Extra	20 hours ago
.babelrc.js	Update Babel config (#31011)	2 years ago
.browserslistrc	.browserslistrc: remove Android and make Safari/iOS 1…	11 months ago
.bundlewatch.config.json	Convert alerts to CSS variables (#35401)	2 months ago

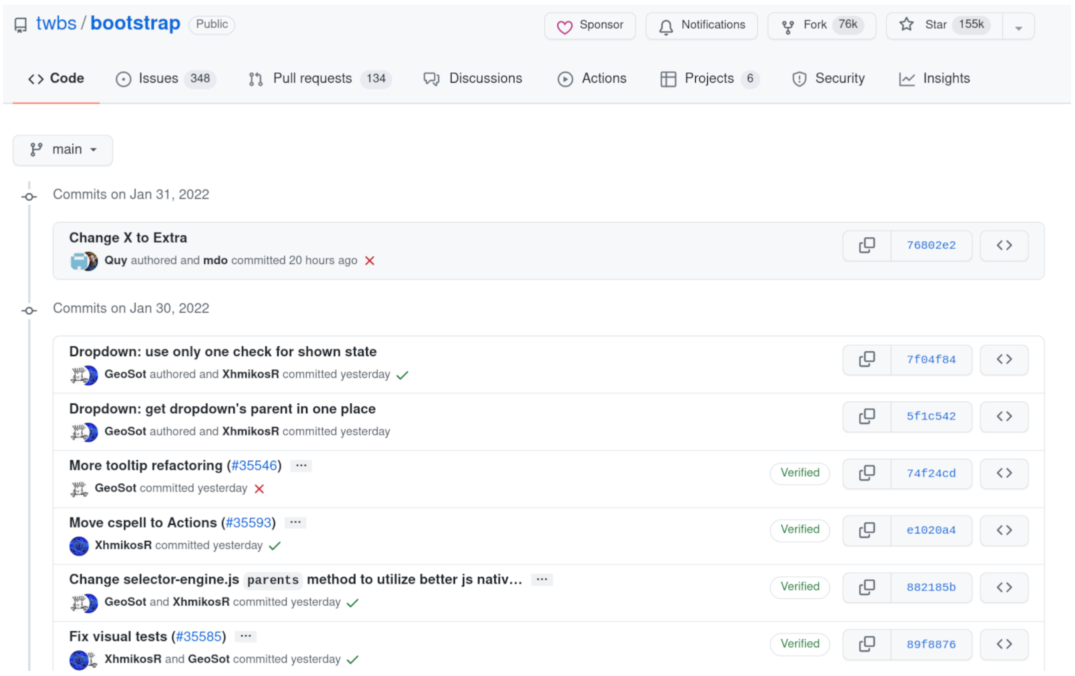
About

The most popular HTML, CSS, and JavaScript framework for developing responsive, mobile first projects on the web.

getbootstrap.com

javascript · css · html · sass · bootstrap · scss · css-framework

Readme
MIT License
Code of conduct
155k stars
7k watching
76k forks



twbs / bootstrap Public

Sponsor · Notifications · Fork 76k · Star 155k

Code · Issues 348 · Pull requests 134 · Discussions · Actions · Projects 6 · Security · Insights

main

Commits on Jan 31, 2022

Change X to Extra — 76802e2
Quy authored and mdo committed 20 hours ago ✗

Commits on Jan 30, 2022

Dropdown: use only one check for shown state — 7f04f84
GeoSot authored and XhmikosR committed yesterday ✓

Dropdown: get dropdown's parent in one place — 5f1c542
GeoSot authored and XhmikosR committed yesterday

More tooltip refactoring (#35546) … — Verified — 74f24cd
GeoSot committed yesterday ✗

Move cspell to Actions (#35593) … — Verified — e1020a4
XhmikosR committed yesterday ✓

Change selector-engine.js `parents` method to utilize better js nativ… … — Verified — 882185b
GeoSot and XhmikosR committed yesterday ✓

Fix visual tests (#35585) … — Verified — 89f8876
XhmikosR and GeoSot committed yesterday ✓

fig:example-repo

Figure 2.B.2: Descriptive Statistics - Project Creation Dates and Earliest Commits

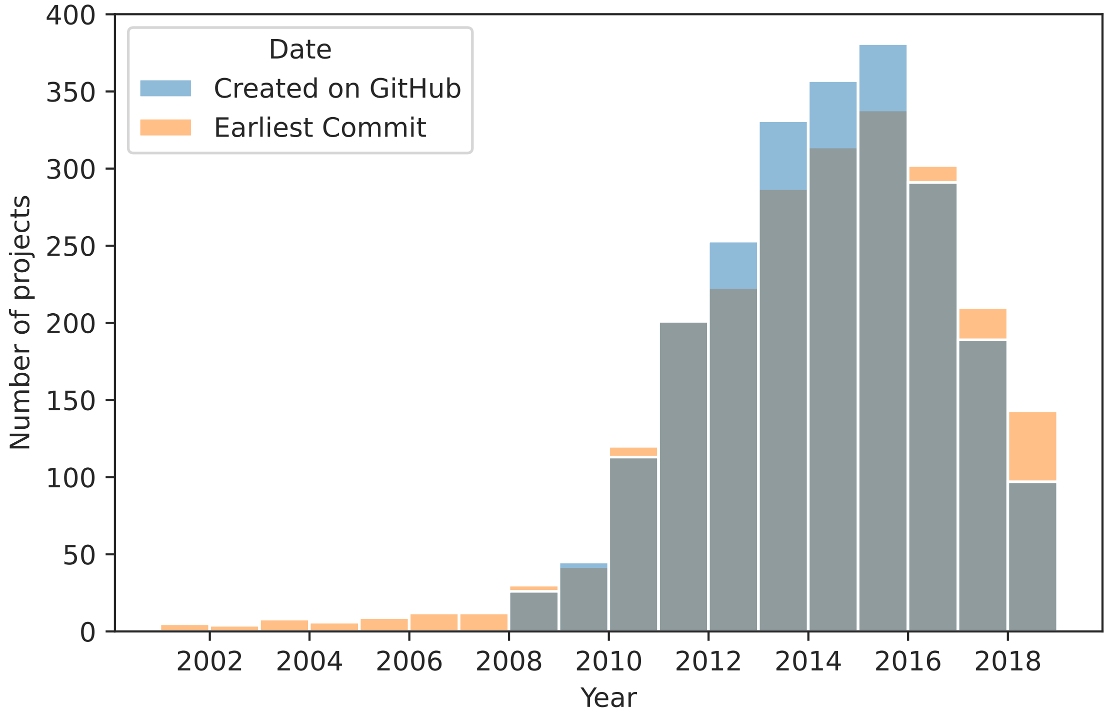


fig:hist-proj-births

Figure 2.B.3: Descriptive Statistics - Distribution of Project-level Contribution Shares

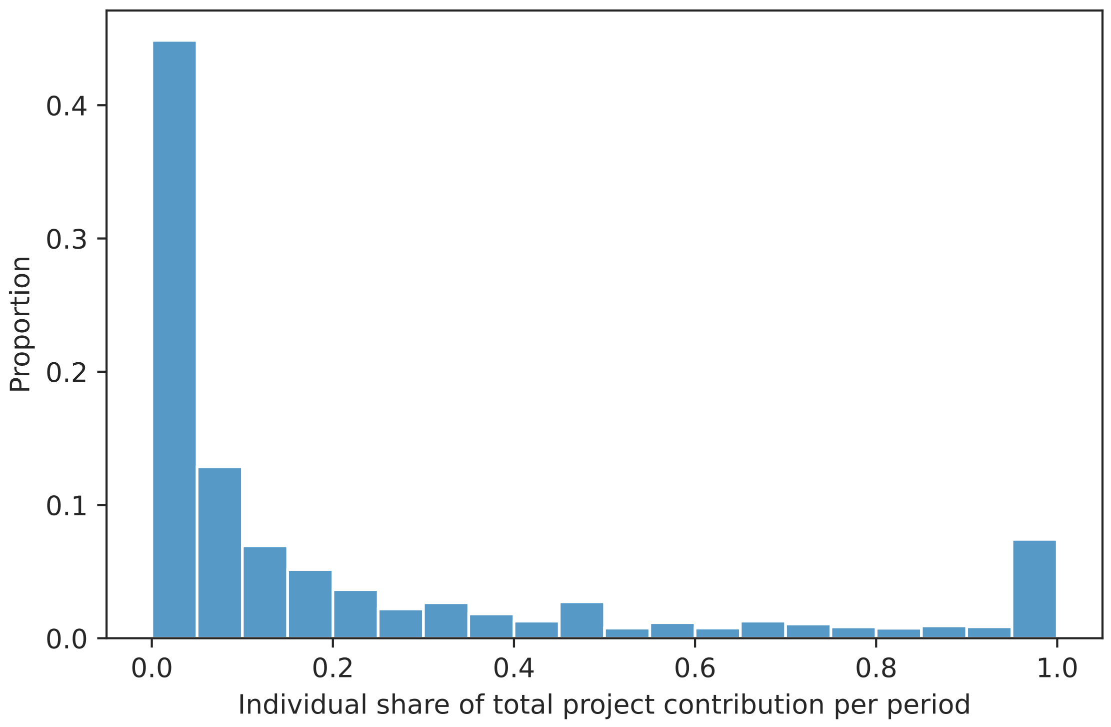


fig:distplot-contri

Figure 2.B.4: Descriptive Statistics - Aggregate contribution in sample

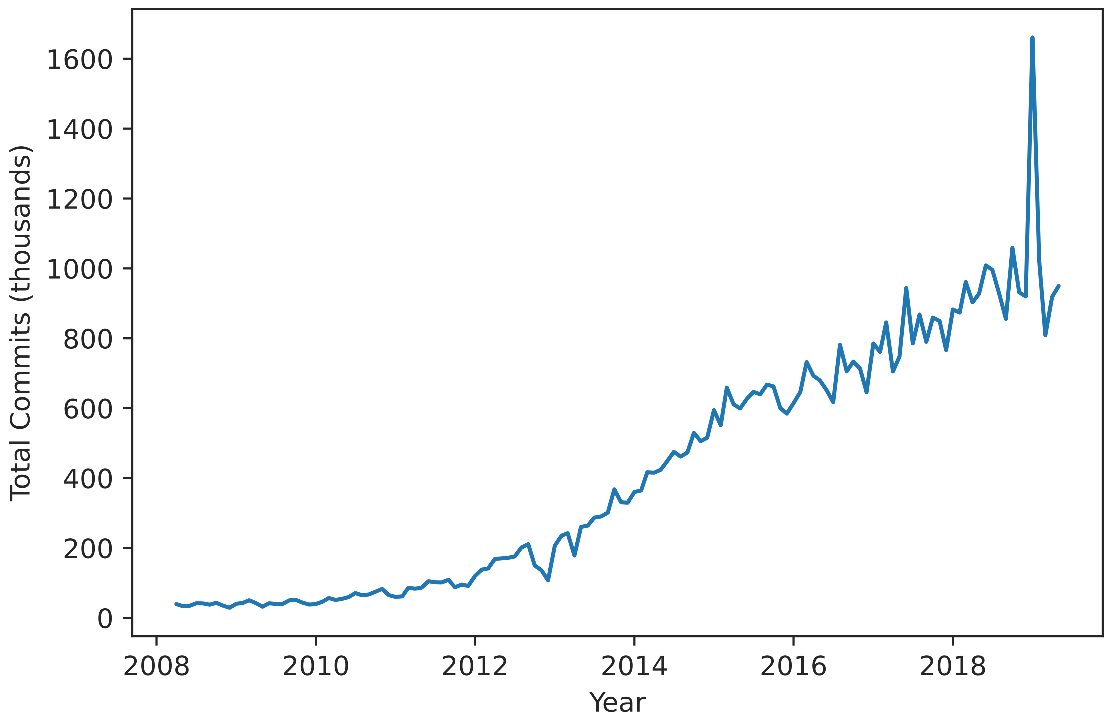


fig:lineplot-agg-co

Figure 2.B.5: Descriptive Statistics - Distinct contributors in sample

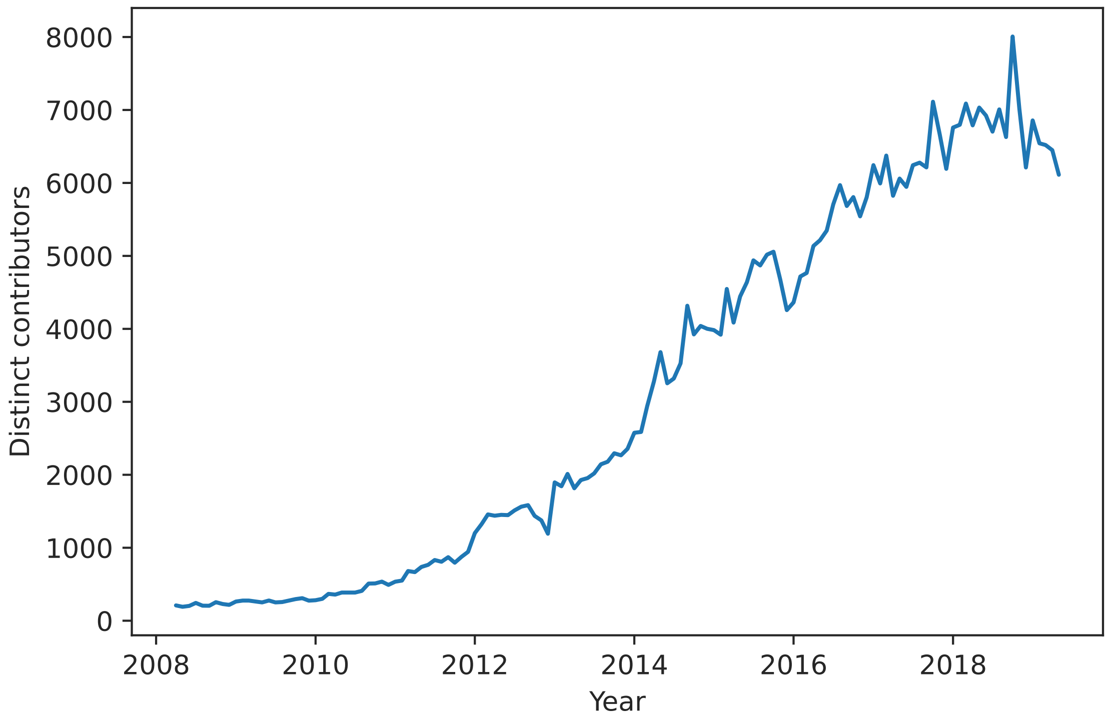


fig:lineplot-distinc

Figure 2.B.6: Descriptive Statistics - Mean individual and peer contribution per project

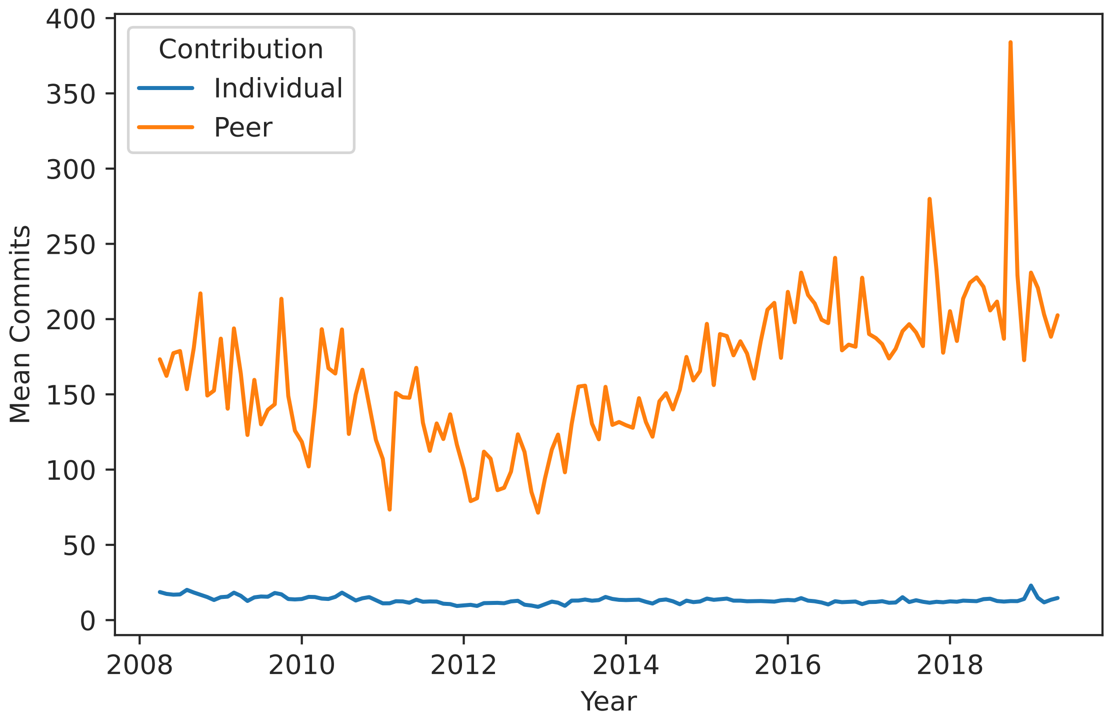


fig:lineplot-ind-pe

Figure 2.B.7: Reduced Form - Project Heterogeneity (Distribution of project-level estimates of $\hat{\delta}$ for Equation (**??**))

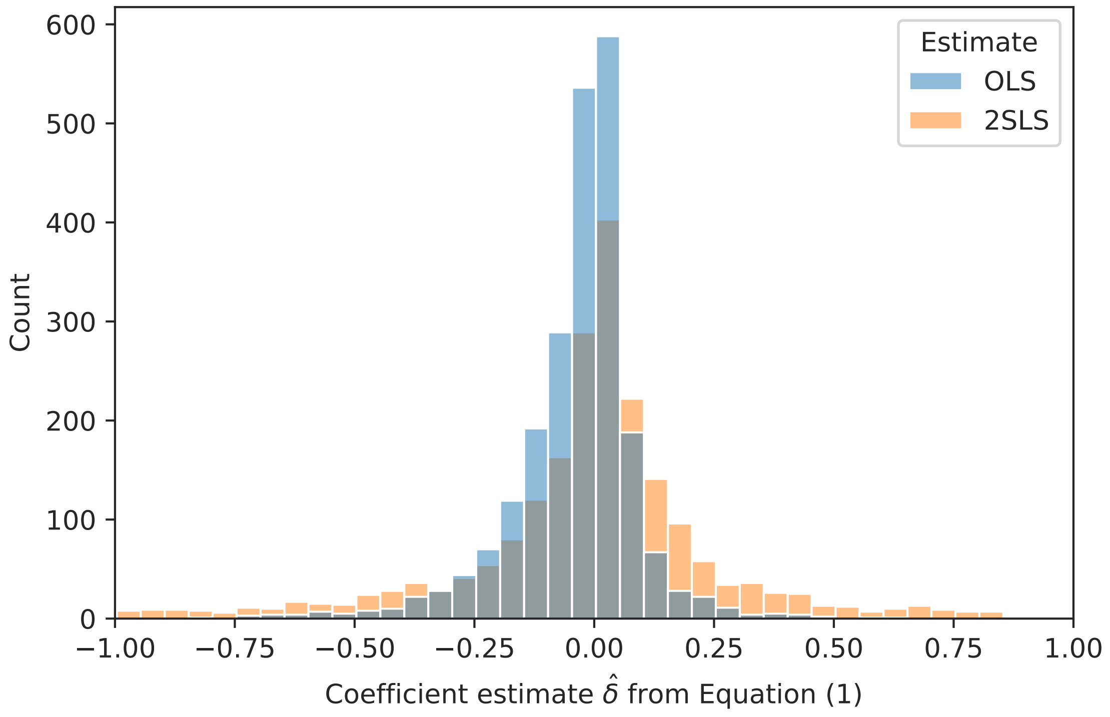


fig:distplot_reduc

Figure 2.B.8: Reduced Form - Temporal Heterogeneity (Estimates for Equation (**??**) over sample period. The top subplot includes estimates for the peer effect coefficient $\hat{\delta}$ of Equation (**??**) within annual cross-sectional sub-samples. The bottom subplot includes estimates for the peer effect coefficient within cumulative sub-samples (i.e. all observations $\leq t$).)

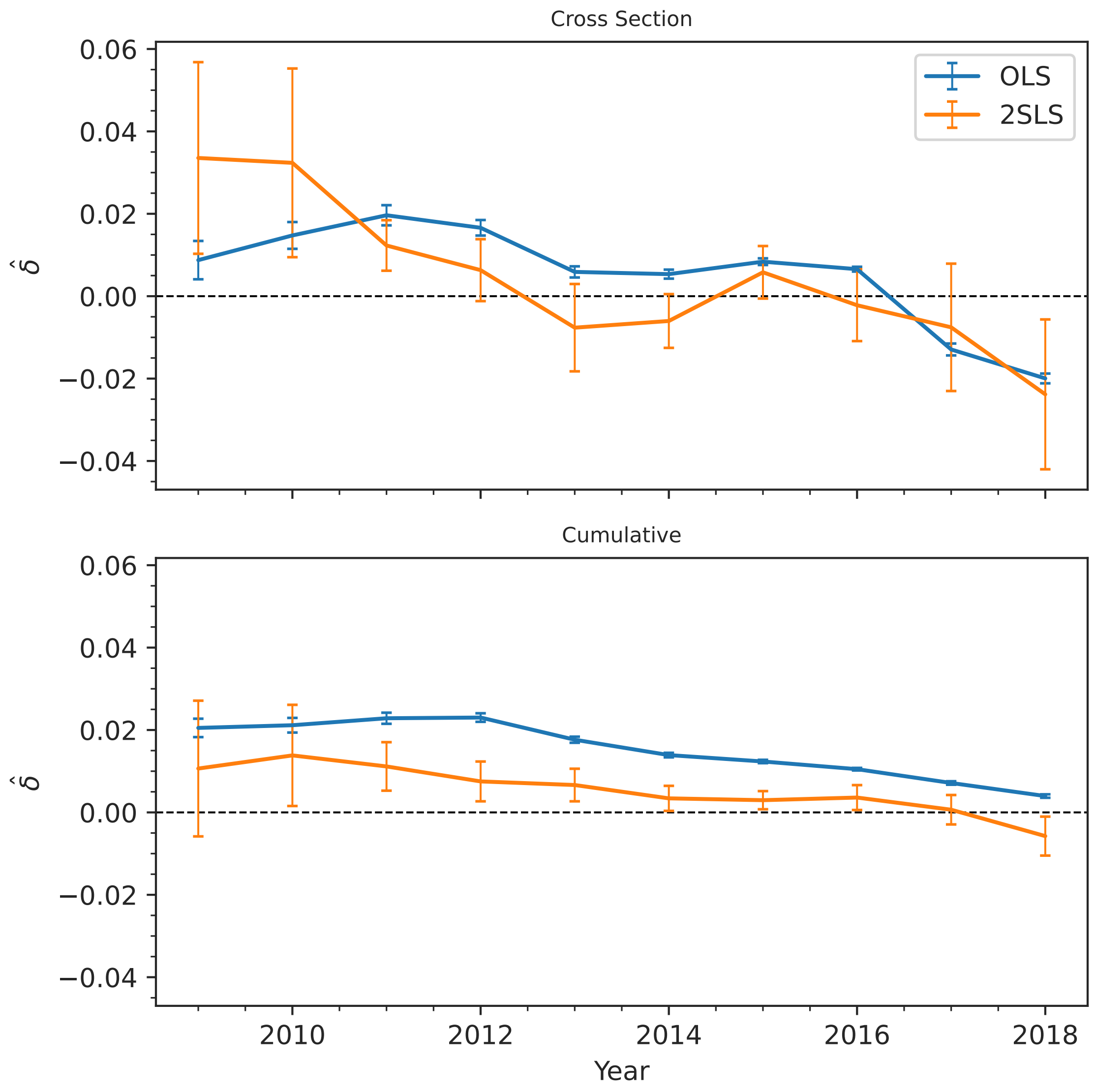


fig:lineplot_time_

Figure 2.B.9: Reduced Form - Insider Contribution and Crowding Out (Estimates for δ in Equation (2.14))

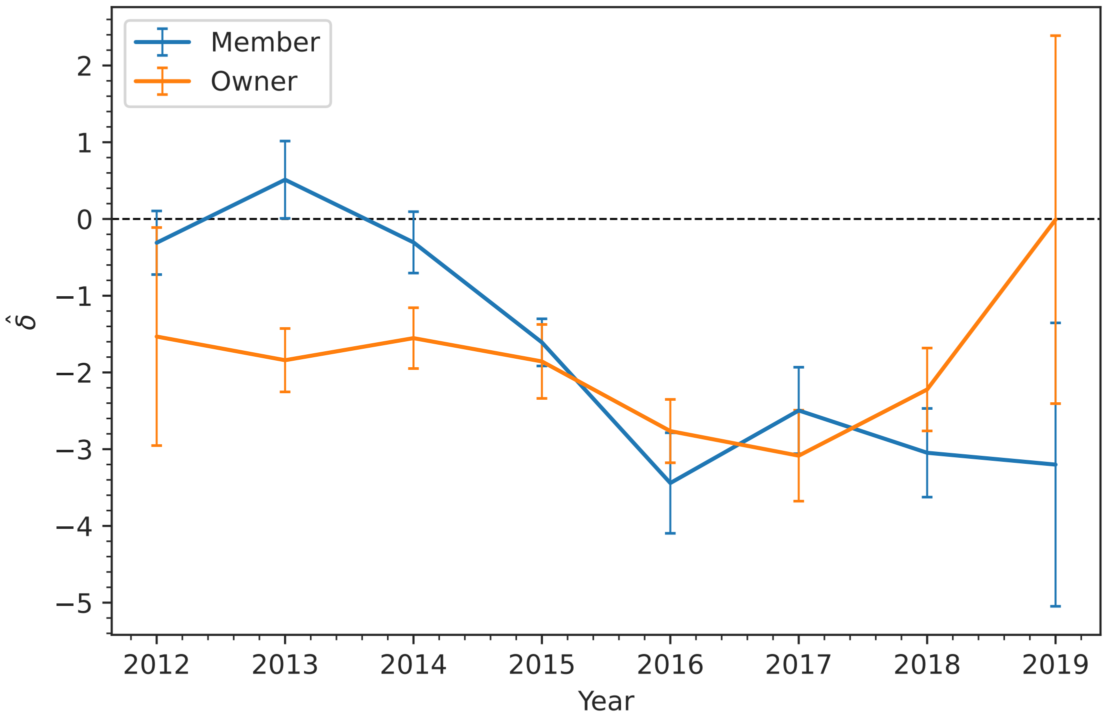


fig:insider

Figure 2.B.10: Structural Model - Recovered Benefit and Productivity Shocks v_{ipt}, c_{ipt} and marginal product of labor parameters b_{pt} for all observed contribution $a^{\star}_{ipt} > 0$ (Equation (2.8))

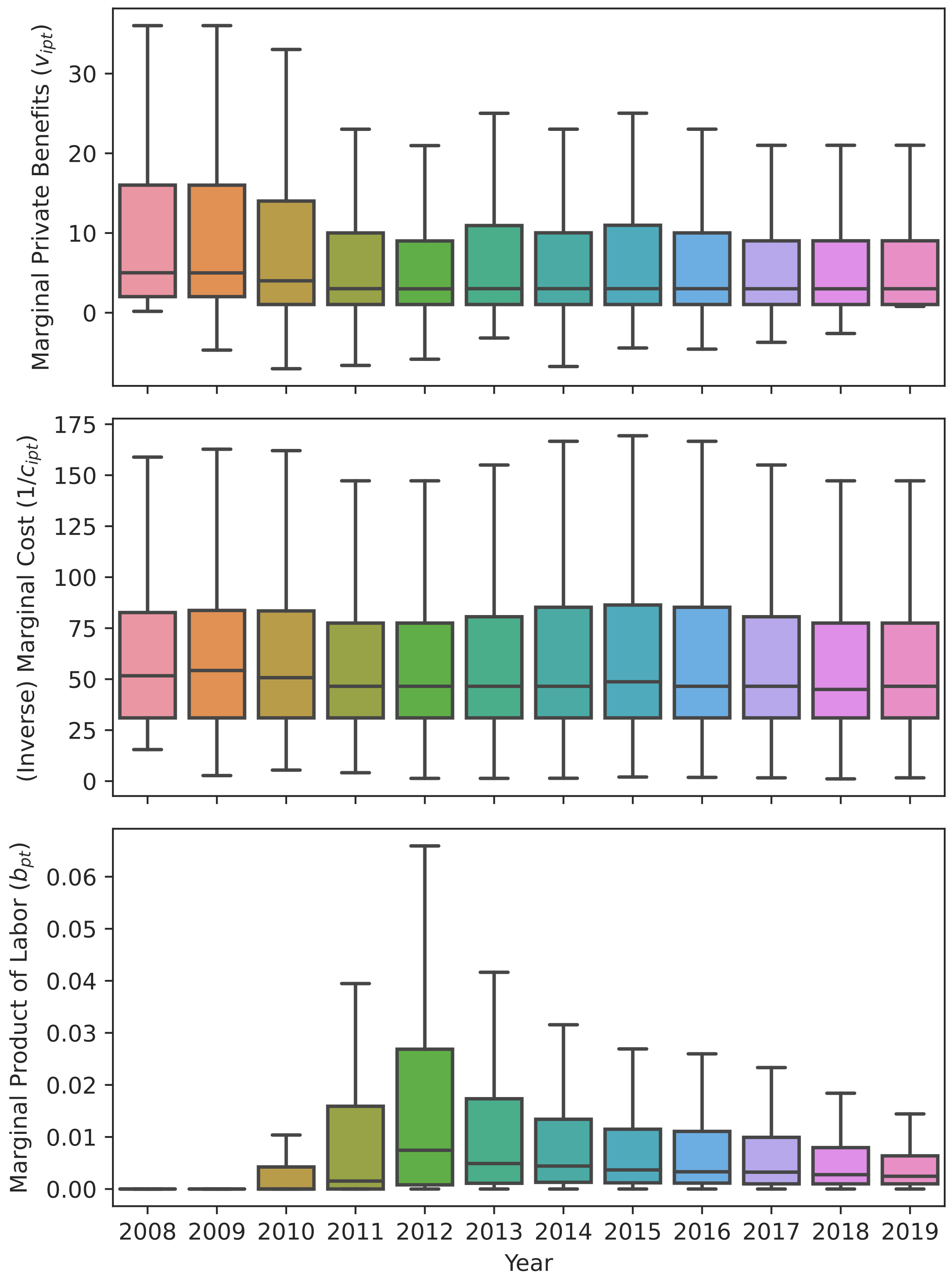


fig:shocks

Figure 2.B.11: Structural Model - Correlation between Benefit and Productivity Shocks v_{ipt}, c_{ipt} over sample period

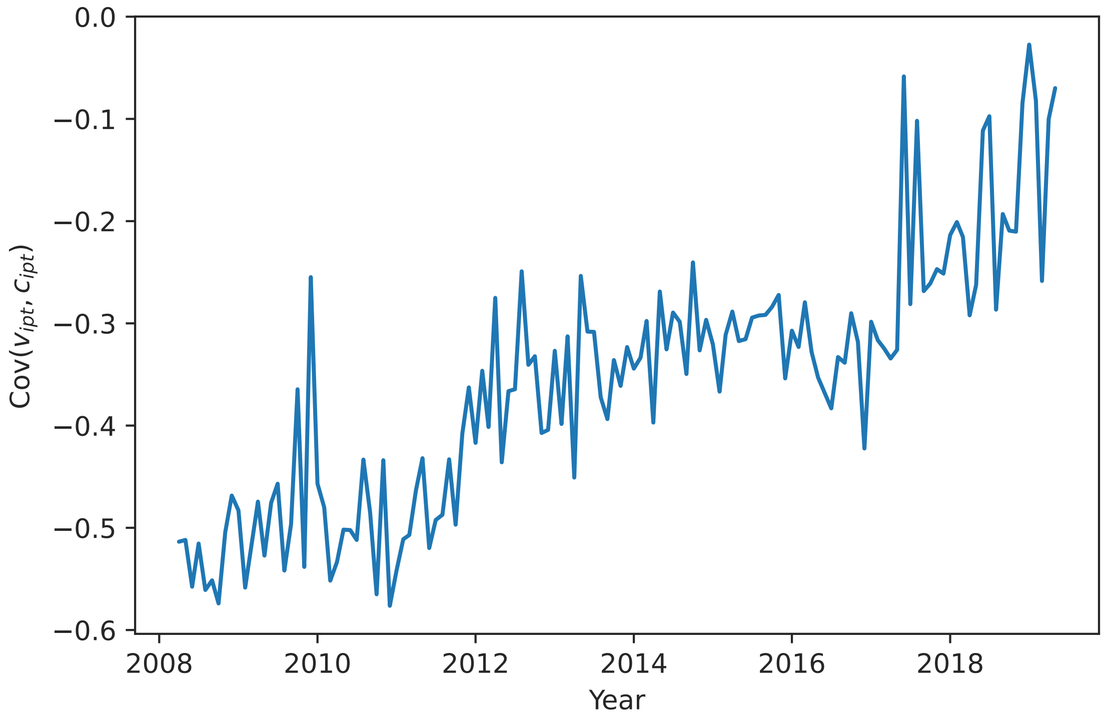


fig:lineplot_shock

Figure 2.B.12: Structural Model - Extensive Margin Peer Effects (Project-level estimates for $\boldsymbol{\gamma}$ from Equation (2.10))

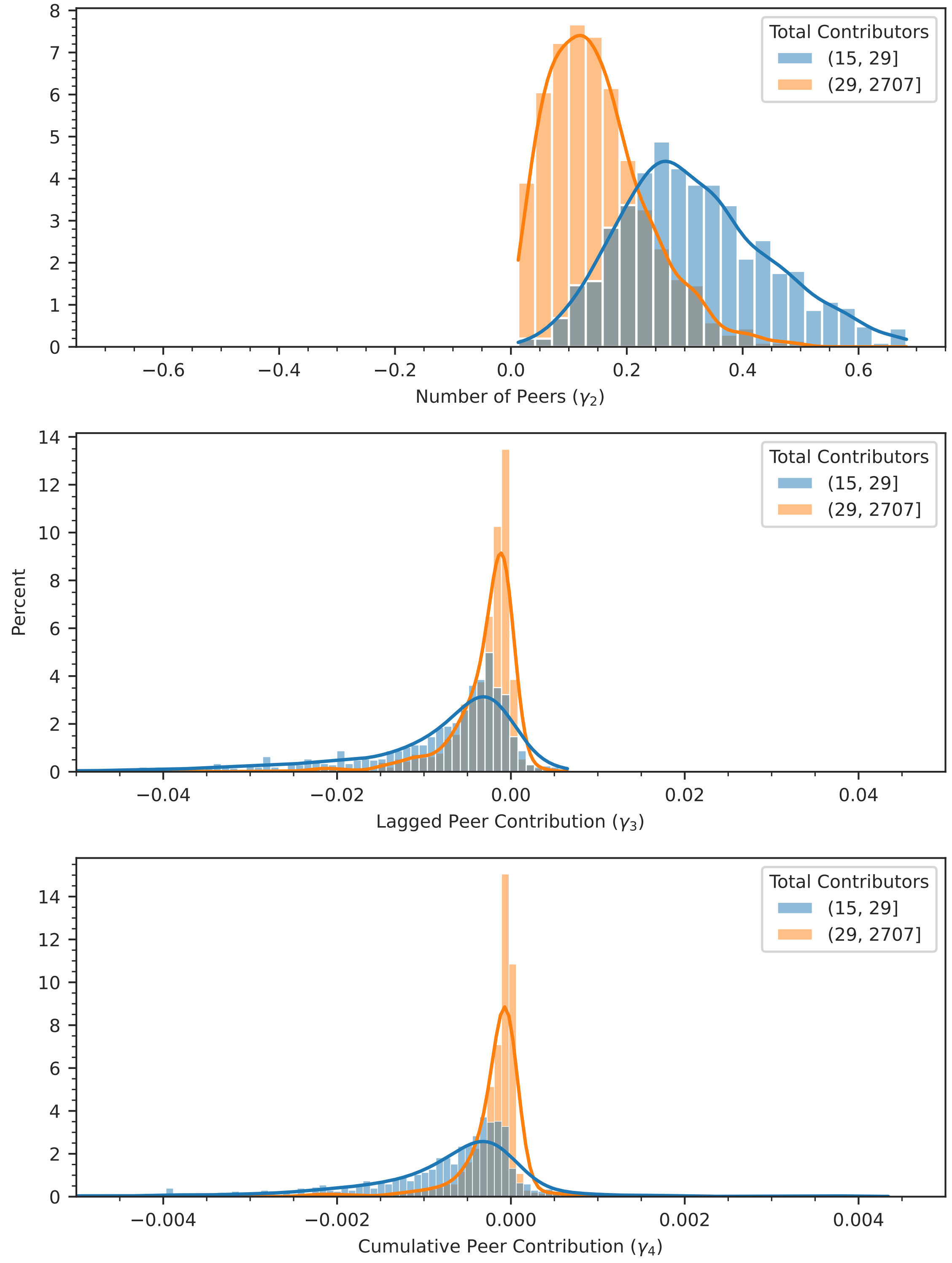


fig:extensive-marg

Figure 2.B.13: Structural Model - Intensive Margin Peer Effects (Project-level estimates for δ_v from Equation (2.12) and δ_c from Equation (2.11))

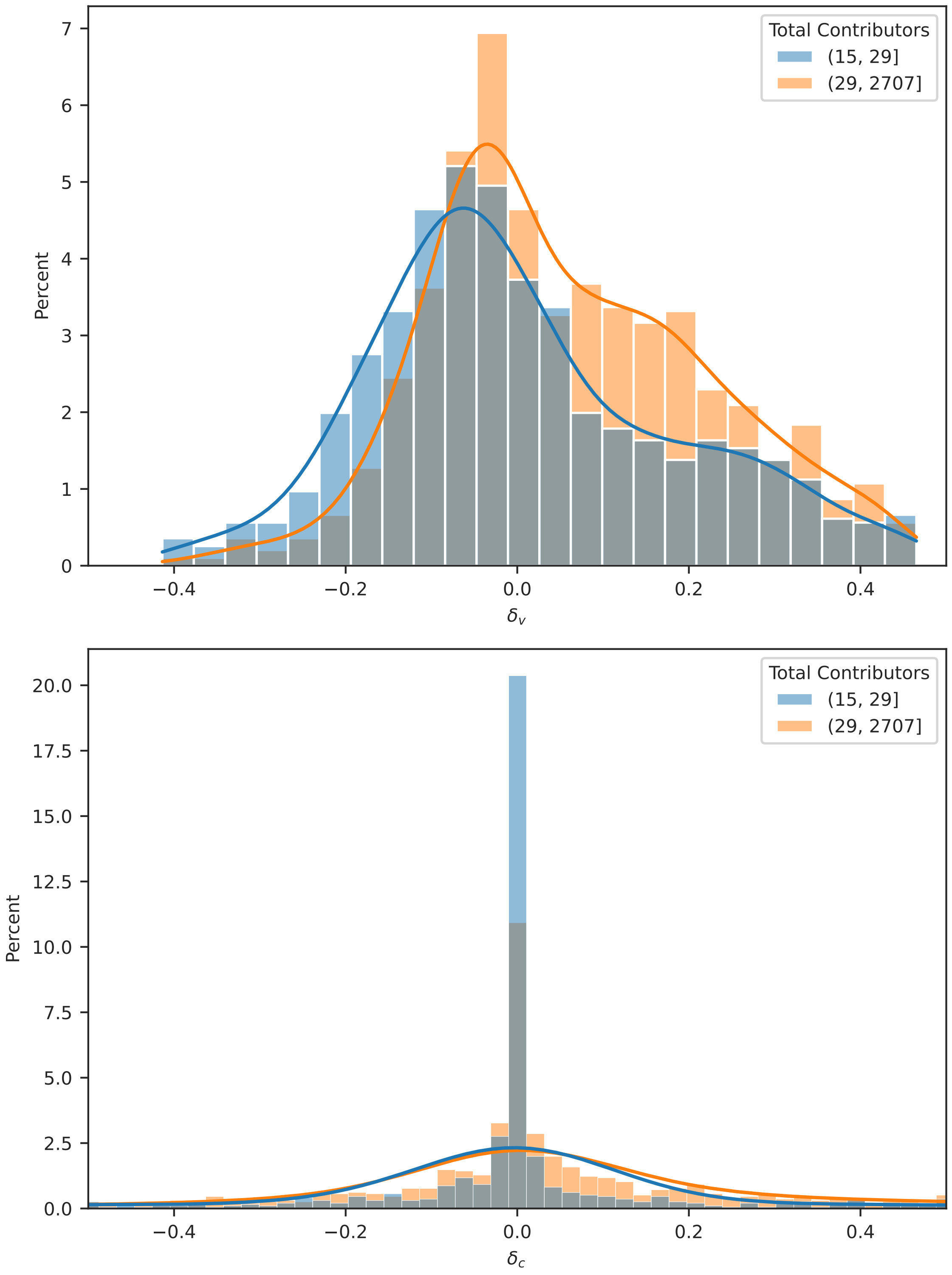


fig:intensive-marg

Figure 2.B.14: Structural Model - Counterfactual Growth in Aggregate Contribution without Peer Influence

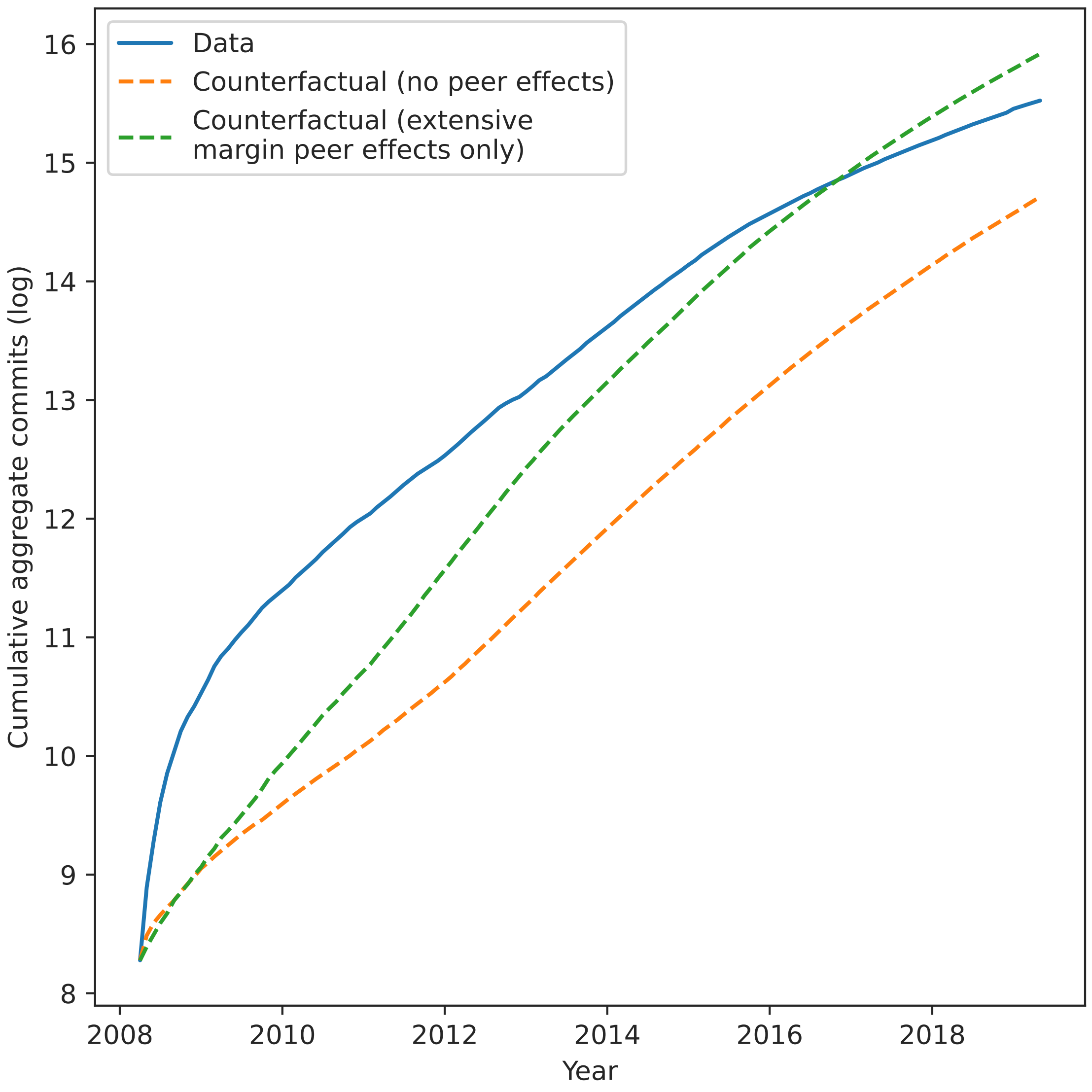


fig:counterfactual

2.C Data Details

app:pg-data-app

Sources

We use several data sources for our empirical sample

- GHTorrent[86], an archive that seeks to provide an offline, historical record of all public activity on the GitHub platform. The data is very large (the June 2019 archive is 104 GB compressed) but provides a set of tools so that researchers can analyze the data in a relational database management system. As an alternative, the data is also hosted on Google Big Query.
- Project source code hosted on the GitHub platform. Projects are usually managed by a version control system (VCS) that, among many other technical features, records a chronological history of changes to the project's codebase. This allows us to create measures for project characteristics over each point in time in the project's history. On GitHub, the VCS tool used is `git`.

Sample Selection

As the number of projects recorded in the GHTorrent dataset is rather unwieldly for analysis by conventional means, we resort to sampling. We use the following procedure to develop a sample of popular, collaborative OSS projects hosted on GitHub:

[86]Source: https://ghtorrent.org/

1. From the set of public GitHub projects created before 2019–06–01, select the subset with (1) 15 or more distinct contributors and (2) 100 cumulative “stars”. Denote this set of top projects $\mathfrak{P}$.
2. Take a 10% random sample $\mathcal{P} \subset \mathfrak{P}$ from the set of top projects. This set of core project will form the basis of projects considered in both the reduced form and structural analysis.
3. For all projects $p \in \mathcal{P}$, determine the set of agents $\mathcal{N}_p \equiv \{i \mid a_{ipt} > 0 \, \forall t \in \mathcal{T}\}$ that contribute to p. For core projects $\mathcal{P}$, contribution is observed over time periods $t \in \mathcal{T}$ where $\inf(\mathcal{T}) = 2009\text{–}11\text{–}01$ and $\sup(\mathcal{T}) = 2019\text{–}05\text{–}01$. Collect all core agents into the set $\mathcal{N} \equiv \cup_{p \in \mathcal{P}} \mathcal{N}_p$.
4. With $\mathcal{N}, \mathcal{P}$ and $\mathcal{T}$ defined, we can proceed in collecting measures of contribution levels, project characteristics, and agent characteristics.

2.D Additional Reduced Form Results

app:pg-reduced-details

We provide some deeper analysis into the reduced form peer effects estimates in an effort to (1) provide robust support for the baseline peer effects estimates in Table 2.A.2 and (2) disentangle the various forces embedded in the full sample estimates.

Interactions

Various observable factors may be associated with different levels of peer effect on contribution. For example, agents larger or higher quality projects may respond differently to the contribution levels of their peers. Peer effects might also vary by the size of the project or peer group itself. We investigates these effects by estimating a version of Equation (2.1) that includes interactions between peer effort and various observables. We present interaction term coefficient estimates in Table 2.A.3.

At first glance, it is apparent that the interactions between peer effort and these various factors are second order compared to the primary peer effect. Moreover, most are statistically in consequential at conventional levels. It is interesting to note, however, that peer effects are strongest when they agent themself has a cumulative history of contribution with the project. This effect is stronger than the influence of peer group size and project quality. We interpret this effect as evidence to the notion that agents form strong affinities to OSS projects and contribute to them with limited concern over other exogenous factors. There is weak evidence that peer effects are stronger for agents invested in the project, such as owners and members, but these effects are not statistically different from zero across specifications.

Temporal Heterogeneity

Given the dramatic growth of OSS participation on GitHub, it is likely that peer effects in the early days of the platform are different from later years. Table 2.A.4 collects estimates of the specifications in Table 2.A.2 for different time periods. Two patterns emerge. First, positive peer effects are stronger in earlier periods. The Column (3) OLS estimates for years 2008 through 2012 are 0.0208 and statistically different from zero at conventional significance levels (compared with -0.0014 for the full sample). This estimate falls to -0.0088 for years 2016 through 2019. It should be noted that the number of observations in this subsamples are 20,209 and 267,706 respectively. Second, comparing Columns (3) and (6) across time periods in Table 2.A.4 suggests some evidence that the OLS estimates are positively biased. Finally, we further disaggregate the sample by time to estimate Equation (2.1) for (1) annual cross sectional sub-samples and (2) cumulative sub-samples (i.e., for all observations $\leq t$ for $t \in \{2008, 2009, \ldots, 2019\}$). We plot these coefficient estimates in Figure 2.B.8.

Both Table 2.A.2 and Figure 2.B.8 suggest that peer effects were more likely to be positive in the early days of OSS on GitHub. This is consistent with **eghbal2020working**'s observation that "platforms broke the commons": early OSS collaboration likely featured smaller, more cohesive project communities in which work was distributed evenly. As GitHub grew in size, the arrival of many, small-share contributors helped grow projects in aggregate but coincide with diminished estimates for the peer effect.

Project Heterogeneity

There is also considerable project-level heterogeneity in peer effects. Figure 2.B.7 plots the distribution of peer effects obtained by estimating Equation (2.1) for each project individually. A key takeaway from Figure 2.B.7 is that after accounting for covariates, peer effect estimates are surprisingly rather symmetric around the null hypothesis of $\delta = 0$. The share of projects in which peer and individual effort are substitutes and those in which they are complements is relatively well balanced within the sample.

Beyond Contemporaneous Peer Effects

The contemporaneous specification in Equation (2.1) is likely too narrowly defined to capture peer effects that develop over a span longer than a single month. Since OSS contribution is public record, peer effects in a general sense need not be strictly contemporaneous. We estimate a version of this specification that seeks to estimate the effect of recent peer contribution (e.g., previous three months) on subsequent individual contribution (e.g. preceding three months):

eq:reduced-pooled

$$\sum_{\tau=0}^{2} a_{ipt+\tau} = \delta \sum_{\tau=0}^{2} a_{\text{-}ipt-\tau} + \boldsymbol{\beta}' \boldsymbol{X}_{ipt} + \epsilon_{ipt} \tag{2.13}$$

. We present estimates of the specification above in Table 2.A.5. The estimates in Table 2.A.5 are larger in magnitude compared to the baseline results in Table 2.A.2, suggesting peer effects are stronger when considered under a wider temporal bandwidth.

Project Level Effects

To begin to see how contribution patterns manifest along the extensive margin, we aggregate contribution to the project-month level and regress (1) aggregate project contribution on cumulative and lagged contribution (Table 2.A.6) and (2) the number of contributors on cumulative and lagged contributor groups (Table 2.A.7). The estimates in columns (6) and (7) of Table 2.A.6 imply that aggregate project contribution is greater, on average, when lagged project contribution is greater. Similarly, columns (6) and (7) in Table 2.A.7 demonstrate that contributor peer groups is autocorrelated on average. Both of these results suggest that past contribution behavior predicts future participation along the extensive margin. We explore the potential for extensive margin peer influence more thoroughly in Section 2.6.3.

Insider Contribution and Crowding Out

An alternative way to look at peer groups within OSS projects is to distinguish between project "insiders" and contributors from the wider community. A natural question is whether contribution from project insiders crowds out contributions from project outsiders. The wider community may have strong incentives to free-ride on disproportionate contributions from dominant core contributors. We define a project insider as an individual who is either the nominal project owner or a member of the project. We aggregate individual contribution to the project level and split it into insider contribution a_{pt}^{in} and outsider contribution a_{pt}^{out}. We then regress outsider contribution on insider contribution and project level controls:

eq:insider

$$a_{pt}^{\text{out}} = \delta a_{pt}^{\text{in}} + \boldsymbol{\beta}' \boldsymbol{X}_{pt} + \epsilon_{pt} \tag{2.11}$$

. We estimate Equation (2.14) for each period and plot the coefficient estimates for $\hat{\delta}$ in Figure 2.B.9. Estimates for $\hat{\delta}$ are consistently negative and statistically significant, giving strong evidence for crowding out by project insiders. It is also worthy to note that crowding out appears to increase in later periods, coinciding with diminishing peer effects over time in Figure 2.B.8.

2.E Structural Estimation Details

app:pg-estim-details

Given data $(a_{ipt}, y_{pt}, x_{it})$ for all $i \in \mathcal{N}, p \in \mathcal{P}$, and $t \in \mathcal{T}$, we develop an estimation strategy to recover

1. Marginal product of labor parameters $\boldsymbol{b} = (b_{pt})$ from the project quality function in Equation (3.9).
2. Private benefit and productivity shocks $\boldsymbol{s} = (v_{ipt}, c_{ipt})$ for all $a^{\star}_{ipt} > 0$ from the equilibrium contribution level in Equation (2.8).
3. (Extensive margin peer effects) Parameters $(\boldsymbol{\gamma}, \boldsymbol{\beta}_z)$ from Equation (2.10).
4. (Intensive margin peer effects) Parameters $(\delta_c, \delta_v, \boldsymbol{\beta}_c, \boldsymbol{\beta}_v)$ from Equations (2.11) and (2.12). ■

The parameters of interest are $\boldsymbol{\delta} = (\delta_c, \delta_v)$, which drive intensive margin peer effects, and $\boldsymbol{\gamma}$, which drive extensive margin peer effects. For each project $p \in \mathcal{P}$, our estimation strategy is as follows:

1. Assume disturbances are jointly normally distributed $(\epsilon^z_{ipt}, \epsilon^v_{ipt}, \epsilon^c_{ipt}) \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$, independent and identically distributed between agents and time. Within the variance-covariance matrix $\boldsymbol{\Sigma}$, assume that $\sigma^2_z = 1$. This implies

$$\begin{bmatrix} \epsilon^z_{ipt} \\ \epsilon^v_{ipt} \\ \epsilon^c_{ipt} \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & \sigma_{zv} & \sigma_{zc} \\ \sigma_{zv} & \sigma^2_v & \sigma_{vc} \\ \sigma_{zc} & \sigma_{vc} & \sigma^2_c \end{bmatrix} \right)$$

   Notice also that $\sigma_{zv} = \rho_{zv}\sigma_v$ and $\sigma_{zc} = \rho_{zc}\sigma_c$.

2. Given data (a_{ipt}, y_{pt}), recover $\boldsymbol{b}$ using Equation (3.9).

3. Given data $(a_{ipt}, y_{pt}, x_{it})$ and $\boldsymbol{b}$, recover shocks $\boldsymbol{s}$ using Equation (2.9), Equation (2.5), Equation (3.9) by means of GMM. Let $\mathcal{P}_{it} \equiv \{p \in \mathcal{P} \mid a^{\star}_{ipt} > 0\}$ be the subset of projects i contributes to in time t and $|\mathcal{P}_{it}| = P_{it}$. For each i and t, there are $2P_{it}$ unknowns: v_{ipt} and c_{ipt} for each $a^{\star}_{ipt} > 0$. There are P_{it} first order conditions from Equation (2.9), P_{it} equations for project quality form Equation (3.9), and one budget constraint (Equation (2.5)):

$$\begin{aligned} a_{ipt} &= b_{pt} + v_{ipt} - c_{ipt} > 0 \quad && \forall p \in \mathcal{P}_{it} \\ y_{pt} &\leq b_{pt} \sum_{j} \sum_{s \leq t} a_{jps} && \forall p \in \mathcal{P}_{it} \\ x_{it} &+ \sum_{p} c_{ipt} a_{ipt} \leq 1 \end{aligned} \tag{2.15}$$

eq:moment-conds

Combining the moment conditions in (2.15), the GMM formulation to recover (v_{ipt}, c_{ipt}) for each agent i and period t given data $(a_{ipt}, y_{pt}, x_{it})$ and parameters b_{pt}, becomes

$$\begin{aligned} (v_{ipt}, c_{ipt}) = \underset{v_{ipt}, c_{ipt}}{\arg\min} \quad & \frac{1}{P_{it}} \sum_{p \in \mathcal{P}_{it}} (a_{ipt} - b_{pt} - v_{ipt} + c_{ipt})^2 \\ \text{s.t.} \quad & 0 < b_{pt} + v_{ipt} - c_{ipt} && \forall p \in \mathcal{P}_{it} \\ & 0 < c_{ipt} \leq 1 && \forall p \in \mathcal{P}_{it} \\ & y_{pt} \leq b_{pt} \sum_{j} \sum_{t} (b_{pt} + v_{jpt} - c_{jpt}) + b_{pt} \sum_{j} \sum_{s<t} a_{jps} && \forall p \in \mathcal{P}_{it} \\ & x_{it} + \sum_{p} c_{ipt}(b_{pt} + v_{ipt} - c_{ipt}) \leq 1 \end{aligned} \tag{2.16}$$

eq:gmm

Across all agents and time periods, this implies $\sum_i \sum_t (2P_{it} + 1)$ total moment conditions and $\sum_i \sum_t 2P_{it}$ unknowns.

4. Given data $(d_{ipt} = 1\{a_{ipt} > 0\}, \boldsymbol{W}_{ipt}, \boldsymbol{X}_{ipt})$ and shocks $\boldsymbol{s}$ recover $(\boldsymbol{\gamma}, \boldsymbol{\delta}, \boldsymbol{\beta}, \boldsymbol{\Sigma})$, where $\boldsymbol{\delta} = (\delta_v, \delta_c)$ and $\boldsymbol{\beta} = (\boldsymbol{\beta}_z, \boldsymbol{\beta}_v, \boldsymbol{\beta}_c)$, using the maximum likelihood estimation (MLE) framework for the Heckman Selection model described by **zhao2020new**. Collect quantities either observed as data or recovered in the previous stages of estimation into a vector $\boldsymbol{D} = (b_{pt}, d_{ipt}, v_{ipt}, c_{ipt}, \boldsymbol{W}_{ipt}, \boldsymbol{X}_{ipt})$. Collect remaining unknown parameters in a vector $\boldsymbol{\theta} = (\boldsymbol{\gamma}, \boldsymbol{\delta}, \boldsymbol{\beta}, \boldsymbol{\Sigma})$. For each project $p \in \mathcal{P}$, the MLE optimization problem becomes

$$
\begin{aligned}
\max_{\boldsymbol{\theta} \in \boldsymbol{\Theta}} \quad & L(\boldsymbol{\theta} \mid \boldsymbol{D}) = \prod_i \prod_t \{f(v_{ipt} - c_{ipt} \mid d_{ipt} = 1)\Pr(d_{ipt} = 1)\}^{d_{ipt}} \Pr(d_{ipt} = 1)^{1-d_{ipt}} \\
\text{s.t} \quad & f(v_{ipt} - c_{ipt} \mid d_{ipt} = 1) = \frac{1}{\sigma}\phi\left(\frac{\epsilon^v_{ipt} - \epsilon^c_{ipt}}{\sigma}\right) \frac{\Phi\left(\frac{\rho}{\sqrt{1-\rho^2}}\left(\frac{\epsilon^v_{ipt} - \epsilon^c_{ipt}}{\sigma}\right) + \frac{\boldsymbol{\gamma}'\boldsymbol{W}_{ipt} + \boldsymbol{\beta}_z'\boldsymbol{X}_{ipt}}{\sqrt{1-\rho^2}}\right)}{\Phi(\boldsymbol{\gamma}'\boldsymbol{W}_{ipt} + \boldsymbol{\beta}_z'\boldsymbol{X}_{ipt})} \\
& \Pr(d_{ipt} = d) = \Phi(\boldsymbol{\gamma}'\boldsymbol{W}_{ipt} + \boldsymbol{\beta}_z'\boldsymbol{X}_{ipt})^{d_{ipt}} \Phi(-\boldsymbol{\gamma}'\boldsymbol{W}_{ipt} - \boldsymbol{\beta}_z'\boldsymbol{X}_{ipt})^{1-d_{ipt}} \\
& v_{ipt} = \delta_v \overline{v}_{\text{-}ipt} + \boldsymbol{\beta}_v'\boldsymbol{X}_{ipt} + \epsilon^v_{ipt} \\
& c_{ipt} = \delta_c \overline{c}_{\text{-}ipt} + \boldsymbol{\beta}_c'\boldsymbol{X}_{ipt} + \epsilon^c_{ipt} \\
& \sigma^2 = \sigma_v^2 + \sigma_c^2 - 2\rho_{vc}\sigma_v\sigma_c \\
& \rho\sigma = \sigma_{zv} - \sigma_{zc}
\end{aligned}
\tag{2.17}
$$

eq:mle

where ϕ and Φ are the standard normal density and distribution functions, respectively. For computation convenience, we solve the MLE problem by instead minimizing the negative log-transformation of L.

Chapter 3

No Free Lunch For Programmers: Digital Supply Chains and the Economics of Software Dependency Management

ch:dsc

3.1 Introduction

sec:dsc-intro

Modern software typically borrows 70 to 90% of functionality from free and open source software (FOSS) projects (**nagle_census_2022**). The use of external software components can significantly lower development costs, reduces the need to "reinvent the wheel", and allows specialized code to be organized into modular packages.[1] Relationships between projects within this ecosystem are known as *software dependency networks*, structures akin to "digital supply chains" in which any number of downstream dependents can costlessly share functionality served from an upstream dependency (**eghbal2016roads**).[2] Given the nature in which modern software services are produced and deployed, it is important to note

[1]To paraphrase the Unix philosophy espoused by Ken Thompson, "Make each program do one thing well." (**mcilroy1978unix**). See **lerner2002some** and **baldwin2006architecture** for discussions of the economics of software modularity.

[2]For example, a visual representation of the dependency network for a sample of Node.js JavaScript projects can be seen at `https://graphcommons.com/graphs/a7ec343d-2a0c-47bb-9658-bb8315e8a096`.

that dependent software components can be affected contemporaneously by the dependencies they borrow from.[3] For example, the maintainer of an upstream dependency project may introduce a change that is backwards incompatible for downstream dependents, forcing the downstream maintainer to expend development effort to maintain functionality. In more serious cases, upstream changes may even introduce faults or exploits that can affect the operation and security of downstream dependents (**ohm2020backstabber**). Therefore a central economic question in this setting is how the maintainer of a given software project must balance the benefits of development expedience offered by using existing codebases against the risk introduced by relying on a network of potentially problematic external dependencies.[4]

While attractive for efficiently building complex projects, the hidden costs of using software dependencies can range from mild maintenance costs[5] to catastrophic risk for downstream applications and end users. In practice, software developers are said to spend roughly as much time managing their code and dependencies as they do writing new features (**grams2019**). Digital supply chain risk is not just a problem for the maintainers of software projects. Some famous cases demonstrate the intricate and pervasive nature of open software components and how faults or changes can have widespread and costly impacts across dependent userbases.[6] In March 2016, the maintainer of Node.js package `left-pad` abruptly

[3]An overview of how services in modern software ecosystems are deployed and maintained can be found in **boldi2020fine**. Importantly, increased uptake of OSS components by users makes them attractive targets for malicious actors (**ladisa2022**).

[4]To paraphrase **devault2021**, there is "no free lunch" for the maintainers of software.

[5]A quote from an anonymous developer of the Eclipse integrated development environment (IDE) for Java: "I only depend on things that are really worthwhile. Because basically everything that you depend on is going to give you pain every so often. And that's inevitable." (**decan2019empirical**)

[6]By "userbase", we mean the incredibly broad set of stakeholders than have come to rely on the functionality and security of a given software component: developers who use software as intermediate inputs, individual end users, private firms, public institutions, etc.

removed the package from the `npm` package registry[7], making the software unavailable to thousands of downstream dependents (**schlueter_kik_2016**). This led to over 2% of all `npm` packages failing to operate properly until maintainers could replace the missing functionality. While the package itself was only 17 single lines of code and was replaced immediately, the aftermath of this abrupt removal begins to highlight the extent to which developers have come to rely on the availability of open code. In April 2014, a Google engineer reported an exploit that became known as the Heartbleed[8] bug in the source code of the OpenSSL library used for encryption, potentially exposing sensitive user information across an estimated 17% of public web servers (**mutton_half_2014**). Similarly, a fix was issued the same day the exploit was reported but hundreds of thousands of unpatched servers remained vulnerable as late as 2017, five years after the vulnerability had been introduced into the codebase (**carey_heartbleeds_2017**).[9][10] In September 2017, Equifax publicly announced a vulnerability stemming from their use of the Apache Struts website framework beginning in May 2017, exposing private records of over 147 million users (**cfpb_equifax_2022**). The company agreed to a settlement with the Federal Trade Commission and the Consumer Financial Protection Bureau that entitled compromised users of the service up to $425 million USD in restitution (**ftc_equifax_2022**). In general, the average cost of a data breach in 2021 was estimated to be $4.24 million USD (**ibm_cost_2021**). Together, these case

[7]`npm` is the Node Package Manager is the *de facto* standard for developing and distributing Node.js packages. See `https://www.npmjs.com/`.

[8]See `https://heartbleed.com/`.

[9]It is thought that the severity of an exploit is amplified if malicious actors are aware that valuable targets remain exposed to the vulnerability even after its disclosure.

[10]Yet another recent example of the wide-ranging impact of software faults is the Log4Shell vulnerability, introduced in 2013 and disclosed in December 2021, which allows an attacker to leak sensitive information passing between network connected devices (**wired_log4j_2021**). It is estimated that the exploit exposed hundreds of millions vulnerable devices or 93% of enterprise cloud environment (**wiz_log4shell_2021**).

studies illustrate the scope of vulnerability under which technological services reliant upon OSS ecosystems operate.

Software dependency networks share features with other networked settings commonly studied in the economic literature: joint research and development efforts between firms (**goyal2001r**), innovation and patents (**jaffe1993geographic**; **hall2005market**; **acemoglu2016innovat** academic publications (**hsieh2018superstar**), linkages between financial institutions (**elliott2014financi** **acemoglu2015systemic**), risk sharing (**fafchamps2007formation**), and inter-firm trade (**elliott2022supply**). OSS projects are collaboratively developed, interdependent network public goods that generate value as both intermediate and final goods.[11] The setting embeds both positive and negative network externalities in complex ways. Prudent or risk averse maintainers create value for downstream dependents by freely sharing software functionality with minimal fluctuations in dependency project quality. Linkages can also directly or indirectly transmit contagion between projects in the form of lapses in functionality, technical debt[12], and even software faults and vulnerabilities.

With these features in mind, we seek to study the evolution of these networks and the implications of equilibrium structure by focusing on the microeconomic behavior of software project maintainers. Specifically, we develop a framework in which each maintainer will make decisions over (1) a level of development resources to invest in their own project and (2) which external projects to use as dependencies in an effort to minimize development costs

[11]For example, a software engineer seeking to develop a project for consumers may opt to use an external dependency as a production input.

[12]In engineering and software development, technical or design debt occurs when a short term solution incurs larger costs over the long run (**techopedia_what_2017**). Proponents of efficient software design patterns argue that excessive dependency reliance contributes to technical debt (**jackson_reduce_2019**).

and maintain a preferred level of expected project quality.[13] In doing so, we can learn which factors influence both (1) the level of overall welfare induced by the dependency ecosystem and (2) the robustness of equilibrium dependency structure to cascading failures. After developing some intuition for these mechanics, we can then consider a set of potential policy interventions that can improve equilibrium welfare, allowing maintainers to make project development decisions more efficiently.[14]

The chapter is organized as follows. In Section 3.2 we survey the literature. In Section 3.3, we introduce a framework to ground our study of the sociotechnical software dependency ecosystem, which gives focus to the behavior of cost-minimizing yet risk averse maintainers making development decisions for interrelated projects under uncertainty. We illustrate the key features of this setting with simple examples. In Section 3.4, we introduce the data used in both the reduced form and structural empirical analyses while illustrating several descriptive patterns that guide our methodologies. With the empirical setting in place, we build intuition over equilibrium outcomes with a reduced form methodology in Section 3.5. Finally, in Section 3.6, we develop a complete structural model of software dependency network formation, assess its equilibrium properties, and discuss the specifics of estimation. In Section 3.7 use the estimates of structural parameters to conduct counterfactual analysis of potential policy interventions. We conclude with final remarks in Section 3.8.

[13]In this sense, transaction costs driven by information asymmetry confront maintainers with a "make-versus-buy" decision when developing the functionality of their project (**coase1937nature**; **williamson1975markets**; **williamson1985economic**). Under these conditions, some maintainers may prefer to invest more development effort in order to avoid external dependencies and provide a greater degree of "vertical integration" within their project (**grossman1986costs**).

[14]In other words, either at lower cost or lower uncertainty over project quality, or both.

3.2 Literature

sec:dsc-lit

We begin our study by reviewing relevant strands of literature to illustrate relevant empirical patterns in software dependency management, place the current study into context, and identify existing methodological approaches that can inform our analysis.

The empirical software engineering literature has established several salient stylized facts to characterize the state of software dependency networks in the wild. **kikas2017structure** and **decan2019empirical** find substantial indirect dependency between projects in software networks also characterized by limited direct dependency. Hence, empirical evidence suggests the observed behavior of project maintainers results in fragile dependency networks, vulnerable to contagion.[15] Common types of vulnerabilities can "break" functionality or expose sensitive user information for a package and its dependents (**prana2021out**). **decan2018impact** find that it takes on average 24 months to find 50% of all vulnerabilities[16], vulnerabilities are prevalent across releases, and downstream dependents often remain unpatched even after vulnerability is fixed upstream. Vulnerabilities can be further exacerbated by the reuse of software code, both in the form of reused code within packages as well as reusing outdated dependencies (**pham2010detection**). In some cases, up to 40% of errors in packages can be traced to changes in upstream projects (**decan2016github**). Up to 80% of maintainers do not keep their dependencies up to date while almost 70% are simply unaware of upstream version changes (**kula2017**). Vigilant maintainers must not only manage development decisions within their own codebase, but also track changes upstream

[15]As put by **zimmermann2019small**, such fragile networks can be described as "small world, high risk".

[16]The delay between the identification of a vulnerability and the distribution of a patch fixing it is known as "technical lag" (**decan2018evolution**; **zerouali2018empirical**).

A significant body of research has sought to better understand the mechanics driving these observed forces in software networks as well as their implications. One strand of literature has explored the costs of software vulnerabilities, including general inquiries into dependency risk (**schueller2022modeling**), the link between vulnerability disclosure and firm valuation (**acquisti2006there**; **telang2007empirical**; **anwar2018understanding**), the efficacy of vulnerability rewards programs (**finifter2013empirical**; **zhao2015empirical**; **roumani2016examining**), end economic theory behind optimal patch management (**cavusoglu2006eco**
finifter2013empirical). Several ambitious empirical studies have even endeavored to estimate the value of the entire OSS digital supply chain itself (**keller2018opportunities**; **robbins2018open**). Despite these efforts, the present study fills a gap in the literature by studying empirically the decision of the individual maintainer to outsource functionality for their project and how this behavior influences the resulting equilibrium.

Our preferred modelling approach attempts to explain the formation of software dependency networks based on the micro-founded behavior of individual maintainers and therefore draws from several distinct efforts within the economic literature. A starting point for the complementarities in production and the formation of fragile supply chains under risk was outlined by **kremer1993ring**.[17] More general theoretical treatments have developed theory for the micro-foundations of network formation under risk aversion (**kovavrik2009risk**; **blume2013network**; **kovavrik2014risk**).[18] The percolation of supply chain disruptions downstream has also been studied empirically through the use of natural experiments (**bernard2019produ**

[17]For overviews on production networks and input-output shock propogation, see **carvalho2014micro** and **carvalho2021supply**.

[18]Yet another adjacent strand of research has focused on incentives for agents to form networks insure against risk (**de2002risk**; **fafchamps2003risk**; **de2006risk**; **fafchamps2007formation**; **bramoulle2007risk**; **ambrus2014consumption**)

carvalho2021supply). Motivated by linkages between financial institutions, a considerable number of studies pay particular attention to conditions under which network structure is susceptible to contagion (**elliott2014financial**; **acemoglu2015systemic**; **erol2018network**; **marbukh2018network**).

The current study is most closely related to recent work by **elliott2022supply**, who consider the formation and robustness of supply chains in the presence of risk. The authors model complex production networks as the multi-sourcing strategies of individual firms, each subject to idiosyncratic disruptions which therefore exposes the entire network to contagious risk.[19] They find that even when firms can hedge against supply chain risk through multi-sourcing strategies, aggregate production remains quite sensitive to shocks in equilibrium. As anecdotal evidence suggests similar patterns may also be present within software dependency networks, the present study represents an empirical application continuing this strand of research in the domain of open source software and public good production.

For the purposes of structural estimation, we also look to a more general literature on strategic network formation (**bloch2006definitions**; **galeotti2010law**; **choi2019network**; **christakis2020empirical**). In particular we develop a structural approach to model the co-evolution of agent choice of actions and link formation by following the work of **hsieh2022structural**. In the spirit of **ballester2006s**, the authors also integrate a counterfactual analysis that considers the welfare impact of removing critical agents or "key players" from the network. In Section 3.7, we adapt this methodology by simulating dependency formation under the absence of "key dependency projects". We also draw from work defining graph theoretic measures of

[19]**elliott2022supply**'s consideration of endogenously chosen "relationship strengths" in inter-firm trade is analogous to our focus on the maintainer's choice between different dependency projects.

fig:basics

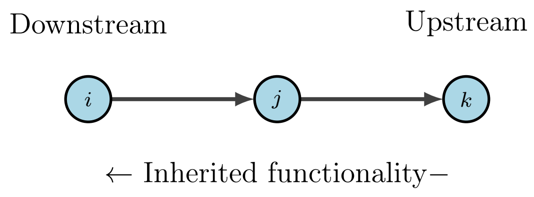


Figure 3.3.1: Project i depends directly on project j and project j depends directly on project k. Hence $G_{ij} = G_{jk} = 1$ are the only non-zero elements of the adjacency matrix G. We say project k is an *indirect dependency* of project i. Additional terminology: Project k is *upstream* of both projects i and j. Project i is *downstream* of both projects j and k. It is important to reiterate that $G_{ij} = 1$ implies that i inherits some functionality from j. In other words, the dependence relationship runs in the opposite direction of the flow of inherited functionality.

to characterize properties of social networks, such as node centrality (**bloch2019centrality**; **everett2022extended**) and network fragility (**doyle2005robust**; **wan2021survey**).

3.3 Framework

sec:dsc-framework

To build intuition for our setting and methodology, we next sketch out a framework for the evolution of software dependency networks, centering our attention on the problem of an individual project maintainer who seeks to efficiently develop a level of software quality under uncertainty. We illustrate, in turn, the general setting of software dependency networks, how indirect risk accumulates across interdependent projects, the maintainer's choice over dependencies, and factors that influence overall network robustness to perturbations. The discussion in this section is merely meant to fix ideas and serves as a primer to Section 3.6, in which we develop a complete micro-founded structural model to more formally characterize this behavior.

3.3.1 Setting

Consider a set of software projects $i \in \mathcal{N} = \{1, \ldots, N\}$. Each project can be indexed by a measure of its *quality* $y = (y_i)_{i \in \mathcal{N}}$. Software projects can *depend* on one another, in which case the dependent (downstream) project borrows a subset of functionality from the dependency (upstream) project at a nominal price of zero[20], assuming the upstream project is publicly available and released under a permissive license. These unilateral dependency relationships between projects can be collected into a directed graph $G = [G_{ij}]_{i,j \in \mathcal{N}}$, which itself can be represented by the $N \times N$ adjacency matrix $G = [G_{ij}]_{i,j \in \mathcal{N}}$ with elements $G_{ij} \in \{0, 1\}$ for all $i, j \in \mathcal{N}$[21] If $G_{ij} = 1$, then project i imports some functionality from project j and therefore depends the quality of project j to some extent:

$$G_{ij} = 1\{\text{project } i \text{ depends on project } j\}.$$

Otherwise, $G_{ij} = 0$. We say package j is a *direct* dependency of package i if $G_{ij} = 1$.[22] Package k is an *indirect*[23] dependency of package i if there exists a *directed path* from package i to package k.[24] In the parlance of software dependencies, we can also say that packages i is *downstream* of package j and package k is *upstream* of packages i and j. We illustrate the basics of this networked setting in Figure **??**.

[20]Note the "nominal" aspect of free and open source software. The very spirit of this chapter is to highlight sources of hidden costs associated with relying on public software infrastructure.

[21]Following convention, $G_{ij} = 0$ when $i = j$.

[22]Alternatively, the graph G can be represented as a tuple $G = (\mathcal{N}, \mathcal{E})$ where $\mathcal{E} = \{(i, j) \mid G_{ij} = 1\}$.

[23]In the software development community, indirect dependencies are sometimes known as transitive dependencies. As "transitive relationship" has a different meaning in social networks literatures, we opt to use the term "indirect dependency".

[24]That is, there exists a directed sequence of distinct dependency relationships $\{G_{ij}, \ldots, G_{lk}\}$ such that $G_{ij} \times \ldots \times G_{jk} = 1$.

Each project $i \in \mathcal{N}$ has a corresponding decision making agent whom we will refer to as the project maintainer.[25][26] The objective of each maintainer is to efficiently develop their project while maintaining its expected quality above a given threshold. The action space for each maintainer i consists of a choices over (1) the level of costly development effort to their project, $x_i > 0$, and (2) the subset of projects to import as dependencies, $\{G_{ij}\}_{j \neq i}$. We assume that the cost of a maintainers effort is given by $c_i(x_i, G)$ but that importing software dependencies has an upfront cost of zero.[27]

eq:cost-framework

$$c_i(x, G) = \frac{1}{2}x_i^2 - \left(a_i + \alpha \sum_{j \neq i} G_{ij} x_j\right) x_i \tag{3.1}$$

Suppose project quality y_i is a function of the dependency network G, and therefore the quality of external projects $y_{-i} \equiv (y_j)_{j \neq i}$ and aggregate effort $x \equiv (x_j)_{j \in \mathcal{N}}$, takes the following linear form:

eq:quality-framework

$$y_i(x, y_{-i}, G) = b_i x_i + \beta \sum_{j \neq i} G_{ij} y_j + \xi_i \tag{3.2}$$

To introduce risk and uncertainty in this framework, we assume that some share of project quality ξ_i is stochastic, unobservable, and known only in distribution by maintainers. Finally, we assume that maintainers are heterogenous in their relative level of risk aversion and define

[25] We will use the terms *project manager* and *maintainer* interchangeably.

[26] In reality, contribution and design decisions in large OSS project are often shaped by the consensus of many distinct developers. We simplify our modelling framework by assuming that any potentially collective decisions made in equilibrium are ultimately made by a single "project maintainer".

[27] The reality is that using external software likely entails some fixed cost, as the downstream developer needs to understand and integrate the upstream package into their project. When a developer imports a dependency, it is clear that the benefit of using the external package outweighs both fixed and marginal costs of working with the dependencies as well as the perceived risk of the dependency. In our modelling approach, dependency fixed costs are subsumed into the maintainer's choice across risky alternatives.

maintainer i's preferences over the quality of their project as $u_i(x, y, G) = \mathbb{E}\left[v_i(y_i)\right]$ where $v_i(\cdot)$ is a Bernoulli function. For example,

eq:utility-framework

$$u_i(x, y, G) = \mathbb{E}\left[-e^{-r_i y_i}\right] \tag{3.3}$$

Here $r_i > 0$ captures maintainer i's relative risk tolerance: as r_i increases, i becomes more risk averse and enjoys less utility under network G where the quality of her project is subject to greater fluctuations in quality. Putting all these elements together, we assume that at each maintainer chooses $(y_i^\star, x_i^\star, \{G_{ij}^\star\}_{j \neq i})$ to (1) minimize development costs while (2) keeping expected utility over project quality above a threshold $\underline{u}_i$:

eq:ump-framework

$$(y_i^\star, x_i^\star, \{G_{ij}^\star\}_{j \neq i}) = \underset{y_i, x_i > 0, \{G_{ij}\}_{j \neq i}}{\arg\min} \quad c_i(x, G) \quad \text{s.t.} \quad u_i(x, y, G) \geq \underline{u}_i \tag{3.4}$$

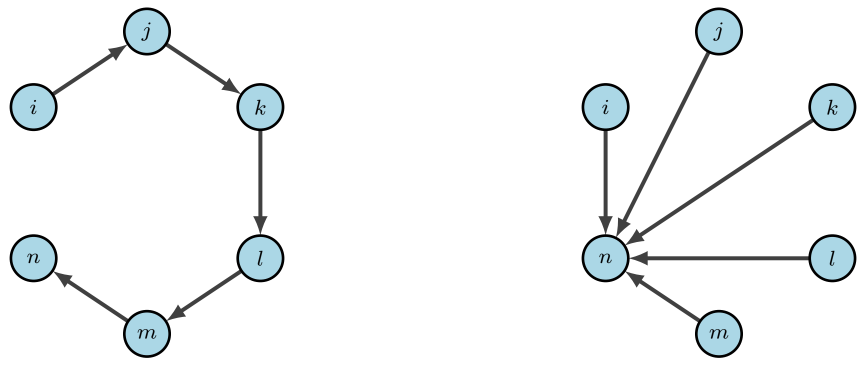
In the remaining subsections we discuss how dependency network structure embeds risk for individual projects, the decision of a project maintainer to depend on external projects, and how network structure can be prone to fragility.

3.3.2 Risk Embedded in Dependency Network Structure

We use Example (3.3.1) and Figure 3.3.2 to show how network structure exposes projects to risk in the form of quality shocks to direct and indirect dependencies.

ex:structure

Example 3.3.1 (Risk Embedded in Dependency Network Structure). *Consider the example networks in Figure 3.3.2. In this case, $\mathcal{N} = \{i, j, k, l, m, n\}$. In Panel 3.3.2a, $G_{ij} = G_{jk} = G_{kl} = G_{lm} = G_{nm} = 1$. In Panel 3.3.2b, $G_{in} = G_{jn} = G_{kn} = G_{ln} = G_{mn} = 1$. Assume*



(a) Indirect Network fig:indirect (b) Hub Network fig:hub

Figure 3.3.2: The network in Panel 3.3.2a is subject to more indirect risk than the network in Panel 3.3.2b. When considering the extent of both direct and indirect dependence, project n is the most critical project in both networks. fig:example-structure

that project quality for each project is the form given by Equation 3.2. Notice that in both networks, the removal of project n has the greatest effect for downstream projects. When considering the extent of both direct and indirect dependence, project n is the most critical project in both networks.

In Panel 3.3.2a, the profile of project quality can be represented by the following system:

$$y_i = b_i x_i + \beta y_j + \xi_i$$

$$y_j = b_j x_j + \beta y_k + \xi_j$$

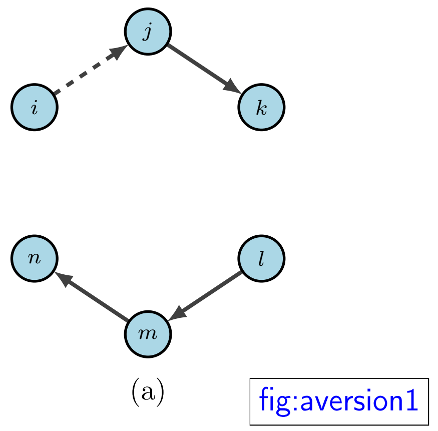
$$y_k = b_k x_k + \beta y_l + \xi_k$$

$$y_l = b_l x_l + \beta y_m + \xi_l$$

$$y_m = b_m x_m + \beta y_n + \xi_m$$

$$y_n = b_n x_n + \xi_n$$

Recursive substitution shows that while project n is subject only to fluctuations in ξ_n, the remaining projects inherit some risk from indirect upstream dependents. The unobservable



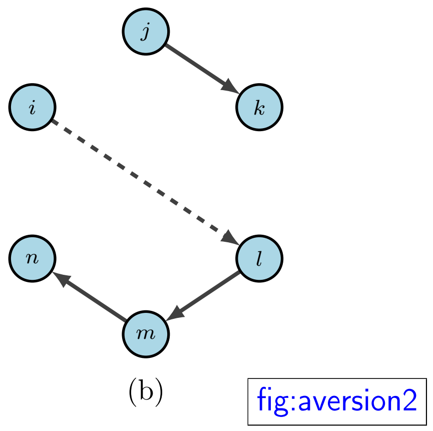


Figure 3.3.3: In Panel 3.3.3a, maintainer i prefers depending on project k over project l and avoids a greater level of indirect risk embedded in project l. In Panel 3.3.3b, maintainer i prefers l to j despite a greater level of indirect risk embedded by projec fig:example-aversion

portions of quality in projects m, l, k, j, and i are $\beta\xi_n + \xi_m$, $\beta^2\xi_n + \beta\xi_m + \xi_l$, $\beta^3\xi_n + \beta^2\xi_m + \beta\xi_l + \xi_k$, ... and so on.

In Panel 3.3.2b, consider the same set of projects under a different (hub) dependency network. Projects i, j, k, l, and m each depend on project n, which itself has no dependencies. Therefore relative to the network in Panel 3.3.2a, projects i and j are subject to less indirect risk, despite the fact that projects i, j, k all depend on the single project l in both networks.

As we have seen in real world examples in Section **??**, faults and vulnerabilities in upstream applications can have consequences in downstream dependents. Maintainer's understand this risk and make development decisions conditional on the current state of the dependency network.

3.3.3 A Maintainer's Choice Between Risky Alternatives

Using external software in a project can lower development costs and improve quality but also entails risk for maintainers. We seek to model dependency formation as a choice conditional on a maintainer's private level of risk aversion: a maintainer ought to only use an upstream project as a dependency if they find it beneficial to their project's quality net of any risk the dependency introduces. Therefore, conditional on project quality and effort choices (y, x), the dependency selection elements of the maintainer's decision in Equation (3.4) can roughly be summarized as follows:[28]

$$\text{Maintainer } i \text{ uses project } j \text{ as a dependency } \iff y_i(x, y_{-i}, G + ij) \succsim_i y_i(x, y_{-i}, G - ij)$$

We formalize this choice by specifying a utility function u_i for the preference relation $\succsim_i$ that reflects maintainer i's individual level of risk aversion in Section **??**. Since some portion of project quality is stochastic and unobservable (ξ_i), preferences can be represented in terms of expected utility à la von Neumann-Morgenstern.[29] Maintainer preference heterogeneity is a result of in variation $v_i(\cdot; r_i)$, a Bernoulli value function parameterized by a measure of risk aversion $r_i > 0$.[30] We illustrate the maintainers choice amongst dependencies with a simple example.

ex:aversion

Example 3.3.2 (Maintainer Risk Aversion and Dependency Choice)**.** *Consider maintainer i's choice between two candidate dependency projects, j and l, represented in Figure 3.3.3. In both panels, project j depends on project k while project l depends on project m and project n. Conditional on maintainer i's preferences for risk, she will chose to depend on a particular project if the it improves the expected quality of her own project. In Panel 3.3.3a, maintainer i imports project j as a dependency over packages l, indicating that she prefers*

[28]Some notation for modifying a single relationship in a given dependency graph G: Let $G + ij$ denote the dependency graph that differs from G only in that $G_{ij} = 1$ and therefore project manager i imports functionality from project j. Similarly, let $G - ij$ denote a dependency network where the only difference from G is that $G_{ij} = 0$.

[29]In other words, $\succsim_i$ and v_i are such that $y_i(x, y_{-i}, G + ij) \succsim_i y_i(x, y_{-i}, G - ij) \iff u_i(x, y, G + ij) \geq u_i(x, y, G - ij)$

[30]Therefore preferences are represented by $u_i(x, y, G) = \mathbb{E}\left[v_i(x, y, G; r_i)\right]$.

the quality improvement and lower level of indirect risk introduced by relying on project j over that offered by project l. In Panel 3.3.3b, maintainer i instead prefers to use project l as a dependency. This indicates that although project l embeds more indirect risk than project j, maintainer i finds the benefits of using l outweigh the costs.

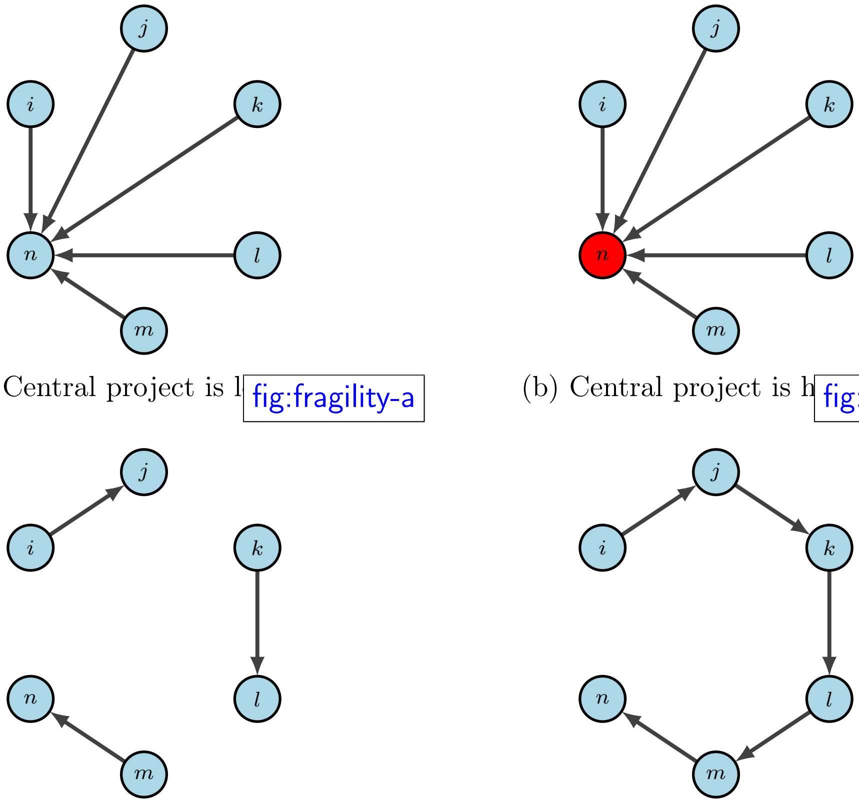
The decisions of maintainer's to prefer certain dependencies over others is driving force that determines equilibrium structure of the dependency network. As we discuss in the following section, this behavior has implications on the relative robustness or fragility of the entire ecosystem.

3.3.4 Fragile Dependency Networks

Both individual characteristics and the structure of the dependency network combine to expose individual projects to varying levels of risk, with implications for the overall value or health of the software ecosystem. We present two examples to illustrate these different channels.

ex:fragility

Example 3.3.3 (Fragile Dependency Networks)**.** *Consider to alternative dependency networks in Panel 3.3.4a and Panel 3.3.4b of Figure 3.3.4. Assume that the only difference between these networks is that the variability in quality for project n is greater in Panel 3.3.4b than it is in Panel 3.3.4a. Notice that given the structure of the dependency networks in both settings, all projects are exposed to disturbances stemming from project n. In this sense, the network in Panel 3.3.4b is relatively more fragile than the network in Panel 3.3.4a since the central or hub project n is riskier.*



(a) Central project is l fig:fragility-a (b) Central project is h fig:fragility-b

(c) Structure isolates fig:fragility-c (d) Structure amplifie fig:fragility-d

Figure 3.3.4: In Panels 3.3.4a and 3.3.4b, different project characteristics can influence system-wide fragility for networks with identical structure. In Panels 3.3.4c and 3.3.4d, different network structures can influence fragility when projects characteristics are held constant. fig:example-fragility

Next, consider the networks in Panel 3.3.4c and Panel 3.3.4d. In this case, difference in network structure can lead to increased fragility. The removal of project n in Panel 3.3.4c impacts only project m since the network is the union of three disconnected components. In Panel 3.3.4d, project n is a dependency, either direct or indirect, for all of it's peers. Hence we can say that the network structure in Panel 3.3.4d is relatively more fragile than in Panel 3.3.4c, since the removal of the same project is more disruptive to overall project quality.

As it is now clear that both project characteristics and network structure jointly determine the relative robustness of the software dependency ecosystem, it is useful to consider measures with the potential to characterize a given network G in terms of fragility. One approach is to consider measures of network centrality. Specifically, consider (1) Katz-Bonacich centrality and (2) betweenness centrality for the nodes of the graph G. Roughly speaking, a node has greater Katz-Bonacich centrality when it is a hub for many other high in-degree nodes. In the world of software networks, central hub dependencies lie at the core of the dependency network and serve, both directly and indirectly, as the foundation for many other projects. On the other hand, a node in graph G has a greater betweenness centrality when it lies along critical pathways between large components of the network. A software package with a large degree of betweeness centrality is an important conduit for a many downstream dependents with core upstream dependencies. Using the logic outlined in the beginning of this subsection, software networks characterized by highly centralized projects may efficiently serve functionality to many dependents, but do so at the cost of increased network fragility.

3.4 Data

sec:dsc-data

The data used in both our reduced form and structural approaches seeks to characterize (1) the features and outcomes within software projects and (1) the dependency relationships between them. How do upstream dependencies influence project contribution and quality? What social or technical characteristics of projects are associated with many upstream or downstream dependency relationships? Can the equilibrium structure of software dependency networks result from or contribute to these dynamics? What is the economic significance for the resulting equilibria?

To address these questions empirically, we develop a dataset of sociotechnical measures for a sample of interrelated OSS projects. We choose to focus on projects from the Node.js JavaScript ecosystem.[31] The dependency relationships between widely used open source Node.js projects are tracked over time by the `npm` (Node.js package manager) registry. Most critically, the `npm` registry records (1) timestamps for when specific versions of packages are published and (2) the set of external dependencies, along with their respective versions, declared by the parent package. Hence by knowing what external components a package relies on at a given point in time, we can observe the evolution of a software ecosystem and its dependency graph as a network panel. A notable advantage of this data is that captures more information about the exact timing of package publication dates and dependency formation compared to single-network observations prevalent in the literature.[32] In addition to simplifying structural estimation, this data enables our structural approach to consider the interrelated decisions of a project manager over internal project development and dependency formation with external projects.[33]

Another attractive property of this empirical setting is that it is possible to observe how sociotechnical features for each individual project evolve over time. The `npm` registry documents the repository URL for the package's source code. Furthermore, a project's source

[31]The rationale for this choice is discussed in sections below. Simply put, the `npm` is the largest open source package ecosystem in terms of number of packages (2.61 million packages as of January 2020 (**libio2020**)). Moreover, we focus on a single programming language ecosystem to make more appropriate comparisons between packages.

[32]Previous authors have developed estimation strategies to exploit repeated observations of networks (**snijders2010maximum**).

[33]Critically, we can capture the initial conditions of the sample network to overcome any bias that might arise characterizing the data generating process in our structural approach.

code is typically managed using a version control system (VCS), such as `git`, which has the benefit of chronicling development of the project at high granularity: one can use the version control log to know which developer contributed which lines of code to the project at specific moments in time. We use both the dependency relationships between packages[34] and the technical features of the project recorded in the VCS log.[35]

In the following sections, we discuss the procedure we use to develop our empirical sample, the measurement of software quality, and illustrate the dataset with a selection of descriptive statistics.

3.4.1 Sampling Procedure

Software dependency networks can grow incredibly large and can be observed at a high temporal granularity. We must therefore resort to sampling a set of representative projects with the potential to capture the essence of dependency management dynamics. We focus on a single packaging ecosystem[36] to minimize irregularities that may arise from cross-language comparisons. Motivated by the case studies of major disruptions caused by widely used software projects mentioned in Section **??**, we choose to focus on the largest packaging ecosystem tracked by the Libraries.io service: `npm` JavaScript packages.

We will begin by describing how we obtain a set of OSS projects and record their dependency relationships over time. Our sampling procedure can be summarized by the following steps. In Step 1, we sample the top 10 most widely depended upon Node.js packages in the `npm` registry as of September 2022. In Step 2, for each of these packages, we record a

[34]Tracked by the `npm` registry

[35]Observed in the source code of the repository. Some technical details: the granularity of the VCS log allows us to download the source code of a project and "rewind" it to its state at a specific point in time.

[36]In other words, a set of OSS projects written in a common language.

sequence of timestamps associated with minor version releases.[37]. In Step 3, for each package at a specific timestamped version, we record a set of upstream (versioned) dependencies the package depends upon at that point in time. We add this set of dependencies to the running list of sampled projects and return to Step 2. To reduce the dimensionality of the resulting sample, we restrict the set of timestamps sampled to minor package versions and limit the depth of upstream dependency projects sampled to 5th degree neighbors of the initial set of 10 seed projects. We refer to the set of versioned timestamps at which a sampled package and it's dependencies are observed as our set of *sample moments*, points in time at which the dependency network potentially changes.[38] Overall, we term this recursive network sampling procedure as an *upstream sample*.[39] The resulting dependency subnetwork contains 1,263 Node.js projects observed at 40,440 distinct sample moments from October 2010 through September 2022. A snapshot of the network sample as it was observed in September 2022 is depicted graphically in Figure 3.A.1. Growth in the size of the sample dependency graph over time can be seen in Figure 3.A.2.

Once we have obtained a panel of dependency relationships, we next use the source code of each project to derive measures of sociotechnical outcomes. For each package, we observe these outcomes for the set of project-specific moments, defined as when either (1) a minor version of the package is published or (2) a minor version of a package is declared as a

[37]Best practices in software development encourage the use of semantic versioning, a labelling system for published releases of software to indicate the degree to which the project has changed. Among other reasons, this is done to improve downstream compatibility, as managers of dependent projects can use the semantic version to determine if their dependency is likely to have any breaking or backwards-incompatible changes. See `https://semver.org/` for more details.

[38]It's important to note the use of the term "potentially". A new version release of a software package may likely contain the exact same set of dependencies as the previous version.

[39]We make a point of focusing on the most central set of Node.js packages and their dependencies. Any dynamics affecting this set of core packages will have widespread influence on downstream packages outside the sample. Hence any welfare effects estimated wthinin this core sample can be viewed as a lower-bound estimate for the `npm` ecosystem at large.

dependency of another project.[40] We use the repository URL of the project to download a copy of it's source code and version control history.[41] For each project moment, we use the VCS log to observe social features such as the cumulative level of commits to the project (i.e. contribution), the cumulative number of contributors, and the number of core contributors to the project and it's associated "bus factor"[42], the project's age, and an estimate of the number of hours spent[43] on project development. We also use the source code of the project itself to measure [44] technical features of the codebase such as the number of single lines of code (SLOC), the cumulative size of the codebase in megabytes, the number of files, the number of distinct languages used, and the number of lines in the codebase that are considered documentation, and other derived measures such as "modularity", defined as the number of lines per file in the codebase, and "churn", the ratio of cumulative commits to SLOC in the codebase. Projects in which the same sections of code are constantly under revision will,

[40]Note that to reduce the dimensionality of the empirical dataset, we do not observe every single package for each moment, only the packages specified in a version's changeset. We can get a sense of this technicality from the reduced for estimates in Tables 3.B.2 and 3.B.3, where the number of observations ranges from 206,598 to 196,894, depending on the availability of each covariate measure.

[41]A forensic analysis of software source code revision history for sociotechnical measures falls under a branch of research in the computer science literature known as mining software repositories (MSR). Notable tools in this space include `reaper` (**muniaiah2017reaper**), `pydriller` (**Spadini2018**), `augur` (**augur2022**), and `grimoirelab` (**duenas2021grimoirelab**).

[42]We defined the number of core contributors as the smallest number of contributors who together have contributed at least 80% of aggregate commits to the project. This measure is related to the so-called "bus factor" commonly discussed in the literature, which is used as an estimate of how susceptible a project is to the loss of key contributors. In our study, we define the bus factor for a project as the ratio of the cumulative number of core contributors to total cumulative contributors: the greater the bus factor (i.e. closer to 1), the more the project relies on a smaller set of core contributors. See `https://chaoss.community/metric-bus-factor/` for more details.

[43]We use a algorithm developed by **brunfeldt2022**. See `https://github.com/kimmobrunfeldt/git-hours`. The algorithm takes the revision history of the project (i.e. the `git` log) and identifies distinct "coding sessions" in which commits made less than 2 hours apart form a single contiguous coding session. For each coding session, time allocation is estimated by the duration as measured by the time between the first commit timestamp of the session and the last. Finally, the sum of all session durations, from the initial commit at t_0 and the final commit before observation at time t, is the estimated for time allocation for a codebase observed at some point in time t.

[44]To generate these technical metrics, we use the static code analysis tool Succinct Code Counter, `scc` (**boyter2022scc**). See `https://github.com/boyter/scc` for more information.

all else equal, have larger values for the churn measure. Finally, a most importantly for our measure of software project quality, we can derive measure of sophistication for the codebase known as cyclomatic complexity.[45]

3.4.2 Measuring Software Quality

The notion of a software's quality is a nebulous concept.[46] In the simplest sense, software code are instructions for a machine to perform a specific task. Developers and users of a particular project may derive value from it in different ways. For example, a user may consider a software of high quality if if can perform its stated purpose successfully, perform efficiently, and do so with minimal errors. Developers on the other hand, may understandably place more emphasis on the "maintainability" of the software's codebase.[47]

Even after settling on a particular definition of software quality, how can it be measured? The use quality of a project may be proxied by the extent of popular uptake. How many "followers" have indicated interest in the project on software development platforms like GitHub? How frequently is the software discussed by users in external communities?[48] Perhaps most pertinent to the present study, how many external projects depend a particular software package? The technical quality of the project can be measured in yet other

[45]Cyclomatic complexity measures the number linearly independent paths through the control flow of a software's functionality. Simply put, smaller and simpler software projects will likely have lower measures of cyclomatic complexity. See `https://www.ibm.com/docs/en/raa/6.1?topic=metrics-cyclomatic-complexity`

[46]See **spinellis2009evaluating** for an overview in methods for evaluating the quality of OSS.

[47]In economic terms, maintenance costs.

[48]For example, we can measure this using the relative frequency of the project's name in search engine trends or in software-specific Q&A forums such as StackOverflow.

ways. For example, static code analysis tools and "linters" can scan the project's codebases for potential vulnerabilities, poorly written or documented code, or other bad software development practices.

For the purposes of the reduced form and structural analyses, we opt for a rudimentary measure of software quality designed to reflect two distinct notions of codebase's overall value. We say a project is of high quality if it is (1) complex and (2) attracts a large number of contributors.[49] This measure captures both developer interest in contributing to the project along with a rough proxy for the level of engineering sophistication it entails.[50] In some reduced form specifications in Section **??**, we will also argue that the number of downstream dependents a project serves can also be used a measure of the value or quality of the software project.

3.4.3 Descriptive Statistics

The sample gives insight over the (1) actions, (2) outcomes, and (3) structure that characterize a software dependency network. We briefly provide some descriptive statistics for this empirical sample.

Figure 3.A.1 presents a snapshot of the sample dependency network as it is observed in September 2022. The key takeawy from this graphical depiction is the level of complexity and intricacy in the relationships between packages in the sample. Figure 3.A.2 shows the growth in the network over time, revealing that as new packages enter the ecosystem, the dependency network becomes less dense. Lower density networks may involve additional

[49]Specifically, we will define quality as the sum of log cyclomatic complexity and the log of the number of cumulative contributors to the project. We then scale the resulting sum to reside within the interval $[0, 1]$.

[50]Similar measures of OSS codebase quality are used in Libraries.io's SourceRank metric (**libio2020**). See https://docs.libraries.io/overview.html#sourcerankfor details.

software development expenditures but at the same time can satisfy a wider arrange of computing application needs and can also serve to isolate dependency risk. Another way to characterize the extent to which certain packages are relied upon in the dependency network is through measures of the package's centrality. For the purposes of illustration, we observe the network in several annual snapshots and calculate each node's (1) Katz-Bonacich centrality and (2) betweenness centrality.[51] We present a bivariate scatter plot of each node's centrality measures in Figure 3.A.3. Naturally, we can see that smaller networks featured nodes with greater centrality. However, we can also see that as the network grows larger in later periods, a small number of outlying nodes have exceptionally measures of centrality. In sense, the larger dependency networks diversify some risk away with the introduction of new packages but a small number of dependency "hubs" serve a larger number of dependents. Despite these insights, the overall effect of such network structures on maintainer welfare remains unclear.

Table 3.B.1 in Appendix **??** contains summary statistics, notation, and brief descriptions of the key sociotechnical project-level measures used in both the reduced form and structural analysis. We highlight the key insight from these features and leave any additional, detailed scrutiny to the reader. Most importantly, the vast majority of these project-level measures convey a common pattern of (right) skewness across projects that ought to have bearing on the interpretation of any sample-wide estimates. For example, the median project in the sample is a terminal dependency with no dependencies of its own, and hence dependency

[51] Roughly speaking, a node's Katz-Bonacich centrality is greater when many other central nodes depend on it. A nodes betweenness centrality is greater when it forms an important pathway. Greater levels of either centrality metric opens up the network to additional risk, all else being held equal. See **bloch2019centrality** and **everett2022extended** for deeper discussions of network centrality metrics and their implications for social networks.

quality and contribution are both absent (i.e. zero). Guided by the framework discussed in Section **??**, we give deeper consideration the relationships between these various features and network structure throughout the reduced form analysis in Section **??**.

3.5 Reduced Form

sec:dsc-reduced

Before developing a fully structural model of software dependency management, we begin our analysis with a reduced form approach to build intuition over empirical patterns. Our objectives in this section are two-fold. First, we begin to explore the extent to which upstream dependencies influence downstream dependent projects, by lowering contribution costs or improving project quality. Second, we estimate a set of models in which (1) the number of upstream projects a maintainer depends upon and (2) the number of downstream dependents a project supports are regressed in turn on set of observables such as features of the project itself and the current state of the dependency network as a whole.

We assume that panel data $\mathcal{D}_t \equiv (y_t, x_t, G_t, W_t)$, is observable by both maintainers and the econometrician where $t \in \mathcal{T}$ represent a sequence of observations.[52] To formalize this set of observables in the notation of our framework outlined in Section **??**, here the vector $x_t \equiv (x_{it})_{t\in\mathcal{T}}^{i\in\mathcal{N}}$ contains project contribution levels measure in number of commits, $y_t \equiv (y_{it})_{t\in\mathcal{T}}^{i\in\mathcal{N}}$ contains measures of project quality, G_t captures the dependency network structure, and $W_t \equiv (W_{it}, W_{ijt})_{t\in\mathcal{T}}^{i,j\in\mathcal{N}}$ captures node (project) and edge (dependency relations) characteristics for the graph (network) G_t at a point in time $t \in \mathcal{T}$ for project $i, j \in \mathcal{N}$. We assume that for equilibrium captured in these observables, behavior for maintainer i in each period

[52]We will assume that time is discrete and therefore without loss of generality, let $\mathcal{T} \subseteq \mathbb{N} = \{1, 2, \ldots\}$.

t is a function peer actions $j \neq i$, the state of the world $\mathcal{D}_{t-1}$, and stochastic shocks. Our discussion outlines a set of econometric specification, provides economic intuition for estimated parameters, and addresses issues pertaining to identification.[53]

3.5.1 Contribution Levels

reduced-contrib

Suppose we are first interested in the relationship between upstream and downstream contribution. Our preferred econometric specification is given in Equation (3.5):

eq:reduced-contrib

$$x_{it} = a_i + \alpha \sum_{j \neq i} G_{ijt} x_{jt} + \delta' W_{it} + \epsilon_{it} \tag{3.5}$$

In specification, fixed effect a_i captures a time invariant propensity for contribution to project i. The term $\sum_{j \neq i} G_{ijt} x_{jt}$ in Equation (3.5) is the sum of contribution activity in project i's dependencies. Therefore the parameter of interest in this specification, $\alpha \in \mathbb{R}$, measures the relative influence of upstream contribution on (downstream) contribution to project i. If $\alpha < 0$, then increased upstream contribution is associated with less downstream contribution on average. Conversely, $\alpha > 0$ implies that downstream contribution increases with the level of upstream dependency development. The direction of this net effect therefore maps into substitution: if $\alpha > 0$, upstream and downstream contribution are gross complements. We can interpret this in two ways. First, the level of upstream development lowers the marginal cost of downstream contribution, generating a positive productivity effect. Second, large dependency trees require the maintainer of the downstream dependent to exert considerable

[53]**chandrasekhar2016econometrics**, **bramoulle2020peer**, **de2020econometric**, **graham2020network**, and **graham2020econometric** provide excellent surveys of empirical methods in social network analysis.

effort to integrate and maintain. If $\alpha < 0$, upstream and downstream contribution are gross substitutes. This implies that larger dependencies allow downstream dependents to exert less development effort. Any one of these mechanisms seems plausible and none can be ruled out *ex ante*. Moreover while the specification assumes a common net pattern of substitution across projects and time, heterogeneous effects are likely more realistic.

The vector of controls W_{it} includes other observable characteristics for project i that might conceivably influence contribution levels. Specifically, we include controls such as a measure project quality y_{it}, a quadratic term in project age, the total number of contributors as well as the number of core contributors, technical characteristics of the project such as single lines of code and cyclomatic complexity, and temporal lags of both contribution to project i and upstream contribution[54]. The term ϵ_{it} represents project contribution influences that are unobserved by the econometrician, independent and identically distributed[55], and mean zero in expectation. In a setting in which (1) dependencies are formed unilaterally, (2) project managers are distinct across projects, and (3) the dependency network is acyclical, the terms on the right-hand side of Equation (3.5) are plausibly exogenous.[56][57] Therefore, in lieu of more rigorous argumentation, we are reasonable comfortable interpreting α as the causal effect of of upstream contribution on downstream contribution in our fixed effects specifications.

[54]That is to say, we include multiple lags of both the left-hand side endogenous and key right-hand side exogenous variables.

[55]$\mathbb{E}[\epsilon_{it}\epsilon_{js}]$ for each $j \neq i \in \mathcal{N}$ and $t \neq s \in \mathcal{T}$.

[56]That is, $\mathbb{E}[\epsilon_{it} \sum_{j \neq i} G_{ijt} x_{jt}] = \mathbb{E}[\epsilon_{it} W_{it}^k] = 0$ for $i \in \mathcal{N}$, $t \in \mathcal{T}$, all covariates W_{it}^k in the vector W_{it}.

[57]Consider the case in which the same set of developers contribute to a project i and its dependency j. In this case, the potential for simultaneity or reverse causality threatens naive estimates of α with endogeneity bias. A similar form of endogeneity may arise whenever a set of package dependencies form a cycle. For example, if for the set of projects i, j, and k, $G_{ij} = G_{jk} = G_{ki} = 1$. We assume away any pervasive threats from the former case and argue that software engineering best practices mitigate the latter.

We summarize coefficient estimates for the specification in Equation (3.5) in Table 3.B.2. To reiterate, the interpretation for the coefficient estimates for α is the effect of increased dependency contribution on the level of contribution in downstream projects.[58] The main takeaway from these results is that while in some specifications it would appear that there is a small positive productivity effect from upstream dependencies ($\hat{\alpha} = 0.034$ in Model 1 of Table 3.B.2), this effect seems to diminish or vanish completely on average after controlling for project-specific fixed effects ($\hat{\alpha} = 0.003$ in Model 3) and/or covariate controls ($\hat{\alpha} = 0.000$ in Model 2). Therefore we cannot say that on average a downstream project with many large dependencies is necessarily larger in terms of commits once individual project characteristics are accounted for.

There are several ways to interpret this pooled estimate. First, we must acknowledge that this particular reduced form model captures only intensive margin productivity effects. One may argue the sheer fact that the downstream dependent exists at all is simply because the upstream dependency sufficiently lowers some fixed cost of development. Second, the observed equilibrium may describe a situation where large, general-purpose dependencies enable the efficient development of smaller, more specialized dependent projects.[59]

[58]On average, *ceteris paribus*.

[59]An apt analogy in this case may be a parallel between basic (i.e. generic dependencies) versus applied (i.e. dependents) research studied in the innovation literature.

3.5.2 Project Quality

sec:reduced-quality

In a similar fashion, we can next turn our attention to the relationship between the quality of a project and the quality of its dependencies. We use the specification in Equation (3.6):

eq:reduced-quality

$$y_{it} = b_{0i} + b_{1i}x_i + \beta \sum_{j \neq i} G_{ijt} y_{jt} + \delta' W_{it} + \epsilon_{it} \tag{3.6}$$

The project quality fixed effect b_{0i} captures an intrinsic level of quality independent of upstream dependencies, controls, or temporal fluctuations. The term b_{1i} captures the marginal product of contribution in terms of improving the quality of project i. The aggregate quality of upstream dependencies at time t is $\sum_{j \neq i} G_{ijt} y_{ijt}$ and therefore the parameter of interest $\beta \in \mathbb{R}$ represents an attenuation factor with respect to quality influences transmitted through the dependency network. Similar to Equation (3.5), a vector of controls W_{it} includes other observables that can potentially influence quality: cumulative contribution, project age, the size of the contributor base, maintainer characteristics, and lags of project quality, contribution, and upstream quality. As in Equation (3.5), we make similar assumptions for the unobserved component ϵ_{it} in Equation (3.6).[60]

We summarize coefficient estimates for the specification in Equation (3.6) in Table 3.B.3. Similar to our analysis of contribution productivity in the previous section, the effect that upstream dependency projects have on downstream quality is small and largely determined by individual project characteristics. Moreover, it is interesting to note that the number of commits in a project has little effect on its quality (i.e. $\hat{b_{1i}} \approx 0$ in all specifications). We

[60] Note that the residuals of the regression from estimating the specification in Equation (3.6) can be used to proxy for volatility or uncertainty in project quality. More details can be found in our discussion of structural estimation in Section 3.6.3.

cannot say, given our chosen project quality metric and conditional on individual project features, that upstream dependencies significantly improve the quality of downstream dependents.

3.5.3 Dependency Formation

Up until now, our reduced form analysis on the influence of upstream dependencies has focused exclusively on intensive margin effects from upstream dependencies. We have not addressed factors that influence the likelihood of dependency formation and therefore know very little about the extent to which project characteristics and maintainer preferences can drive equilibrium dependency structure. In this section, we investigate features of OSS projects that either (1) form many upstream dependencies or (2) serve many downstream dependents. We operationalize this by regressing both the number of upstream dependencies or the number of downstream dependents a package has on covariate controls.[61]

Let $d^{\text{out}}_{ijt} \equiv \sum_{j \neq i} G_{ijt}$ denote the number of external projects that package i has declared as (upstream) dependencies at time t.[62] We study factors that drive a package to form many upstream dependency relationship using the specification described in Equation (3.7):

eq:reduced-upstream

$$d^{\text{out}}_{ijt} = \delta' W_{it} + \epsilon_{ijt} \tag{3.7}$$

[61]We acknowledge that there are alternatives to count regression to study characteristics of dependency formation. For example, we could assess the effect of observables on dependency formation using dyadic regression (**helmers2017board**; **bramoulle2020peer**):

$$G_{ijt} = 1\{r_i + \gamma_j + \delta' W_{ijt} + \epsilon_{ijt} \geq 0\}$$

Without sub-sampling, to estimate such a specification on the entire sample entails an onerous computation burden and hence we opt for the simpler and arguably more interpretable approach of count regression.

[62]In graph terminology, the out-degree of node i at time t for the graph G_t.

where W_{it} is a vector of observables for project i at time t drawn from Table 3.B.1. Similarly, let $d^{\text{in}}_{ijt} \equiv \sum_{j \neq i} G_{jit}$ denote the number of external (downstream) projects that declare package i as a dependency at t.[63] Factors that are associated with the attractiveness of package i as a dependency can be studied using the specification in Equation (3.8):

eq:reduced-downstream

$$d^{\text{in}}_{ijt} = \delta' W_{it} + \epsilon_{ijt} \tag{3.8}$$

As the number of downstream dependent is a way to measure a project's importance or quality, the specification in Equation (3.8) is an alternative to the specification in Equation (3.6) to reveal which observables features contribute to package quality.

We present coefficient estimates for the specifications in Equations (3.7) and (3.8) in Table 3.B.4. Several patterns emerge. First, models (1) through (4) of Equation (3.7) suggest that higher quality packages declare a larger number of upstream dependencies. This pattern is notably stronger than the intensive margin quality effects collected in Table 3.B.3 and underscores the notion that popular, complex projects likely outsource much of their functionality to external packages. Second, as expected from somewhat of a mechanical correlation, packages with more dependencies have higher dependency quality. However on the other hand, packages with many downstream dependents feature fewer upstream dependencies and therefore enjoy less quality effects from their own dependencies. Third, hub dependencies tend to be well documented while packages with many dependencies are not. It is likely that well documented software is easier to work with and are therefore more attractive to use. Fourth, specifications (1) and (5) suggest that software projects which rely

[63] In other words, package i's in-degree.

on a small set of contributors for project development (i.e. large bus factor) have both fewer upstream dependencies and downstream dependents. The direction of the influence of the bus factor switches somewhat when controlling for project fixed effects (specifications (2), (4), (6), and (8)).[64] Finally, a project's lines of code, the number of contributors, and age are all not strong predictors of either upstream or downstream dependency.

3.5.4 Robustness

Pooled estimates of the effect of dependencies on downstream contribution and quality may mask effects present in various sub-samples. To this end, we also estimate the dependency effects α of Equation (3.5) and β of Equation (3.6) at both (1) the project-level and (2) over time, and (3) for the sub-sample of projects with at least one dependency.

In Figure 3.A.4, we can see the project-level estimates for the impact of dependencies on downstream contribution α is somewhat symmetrically centered around zero. The same is true for upstream quality effects β at the project level. In Figure 3.A.5, we estimate α by annual sub-sample. Interestingly enough, we can see that the effect of dependencies on downstream productivity is much greater in earlier sample years when both projects and the dependency network itself were much smaller. This would suggest earlier periods of the sample dependency network featured a stronger degree of complementarity between upstream and downstream contribution for core Node.js package. On the other hand, upstream quality effects are not markedly different in earlier sample periods compared to later years or the fully pooled sample.

[64]In lieu speculating on the interpretation of this particular pattern, it would be wise to study the influence of project's bus factor with better data on maintainer characteristics.

Finally, we estimate these specifications for the sub-sample of projects with at least one dependency declared. These estimates ought to reflect dependency influences on the projects that actually rely on external software for some functionality. However, we find that these estimates are actually quite similar to those for the pooled sample, especially after controlling for project-specific fixed effects.

3.5.5 Summary

Overall, our reduced form methodology finds a limited impact of upstream contribution and quality on downstream project contribution or quality, respectively. The notable exception is that the impact of dependency contribution seems to have had a stronger impact on downstream contribution productivity in earlier periods of the sample (Figure 3.A.5). These insights guide our structural approach. First, reduced form analysis emphasizes the complexity of dynamics within the empirical setting of software dependency management. Without a well-specified structural model, it's unclear to what extent any of these estimated effects impact equilibrium welfare. Second, project-level fixed effects (i.e. a_i, b_{0i}, b_{1i}) seem to matter much more than average, intensive margin effects (e.g. α, β) when considering the influence of upstream dependencies on downstream outcomes. This result further motivates a structural approach that permits counterfactual analysis in which key central projects are removed. Third, the reduced form approach does not attempt to address factors such as project development costs, uncertainty over dependency quality, or maintainer risk aversion. We place these considerations at the forefront of our structural model. Finally, sample evidence suggests that (1) higher quality packages have more dependencies and (2) well documented are more likely to serve as dependencies.

3.6 Structural Approach

sec:dsc-structural

Reduced form analysis serves as a starting point for characterizing key empirical patterns that begin to illustrate the framework outlined in Section **??**. We next seek to formalize the microeconomic behavior of software project maintainers in an effort to explain how dependency networks evolve over time and deliver benefits to users. Using the network formation model suggested by **hsieh2022structural** as a basis, our structural approach models the coevolution of both individual software projects and the dependency network. The structural model allows us to conduct two distinct types of counterfactual policy analysis and assess changes to equilibrium welfare, which we measure as the aggregate time cost for software developers. First, we can perturb structural parameters such as the distribution of maintainer risk aversion or variation in project quality. Second, we can simulate the removal of "key projects" (**ballester2006s**).

3.6.1 Setup

sec:structural-setup

The setup of the structural model follows the framework from Section **??**. We specify a project quality relation, contribution costs, preferences, and information available to each maintainer. [65]

[65] While the model structure captures a sequence of static equilibria over a number of periods, we will suppress the time subscript throughout most of Sections 3.6.1 and 3.6.2 for the sake of streamlined notation. This should not affect any implications of the model. Assumption 6 in Section 3.6.2.1 discusses the specific sequence which these static equilibria follow.

3.6.1.1 Project Quality

Project quality y_i is a function of (1) contribution effort $x_i > 0$ and (2) dependency relationships summarized by the directed network:

$$y_i(x_i, y_{-i}, G) = b_i x_i + \beta \sum_{j \neq i} G_{ij} y_j + \xi_i \quad \forall i \in \mathcal{N} \tag{3.9}$$

eq:quality

Here, b_i is the marginal product of manager i's contribution and β captures the attenuation factor over quality derived from project i's upstream dependencies.[66] The term ξ_i are unobservable influences that partially determine project quality.

In subsequent sections, it will be convenient to assume that $y_i > 0$ for all $i \in \mathcal{N}$.[67] This is done without loss of generality as it is still assumes $y_i > y_j$, then project i is of relatively higher quality when compared with project j.

3.6.1.2 Contribution costs

Contribution costs are assumed to be a convex function of effort level $x_i > 0$:

$$c_i(x, G) = \frac{1}{2} x_i^2 - \left(a_i + \alpha \sum_{j \neq i} G_{ij} x_j \right) x_i \tag{3.10}$$

eq:cost

[66]During estimation, we include a constant term in Equation (3.9), similar to the reduced form analog in Equation (3.6), which we omit here to simplify notation.

[67]For the purposes of our study, this is guaranteed by the construction of project quality: log complexity plus log contributors, scaled to the interval $[0, 1]$. This assumption is simple enough for other measures of quality as well. If, for example, project quality is defined as the inverse of the number of detected vulnerabilities or faults.

Notice that c_i is decreasing in a_i and α, which capture manager i's own productivity and any productivity spillovers from contribution in upstream dependencies, respectively.[68] Compared with parameters (b_i, β) in (3.9) which capture the marginal productivity of contribution effort in terms of project quality, parameters (a_i, α) in (3.10) allow us to distinctly characterize contribution productivity in terms of marginal costs. As discussed in Section **??**, however, if upstream and downstream contribution are gross complements, then $\alpha < 0$ and dependency usage imposes a net costs on the maintainer.

3.6.1.3 Preferences

Project managers derive utility from their expected private valuation of their project.

eq:utility

$$u_i(x, y, G) = \mathbb{E}\left[v_i\left(y_i\right)\right] \tag{3.11}$$

By allowing variation in the Bernoulli function $v_i(\cdot)$, we capture the idea that maintainers will differ with respect to how much dependency risk they are willing to take on. In particular, we will assume some level of concavity in the function $v_i(\cdot)$:

assum:utility

Assumption 4 (Exponential Utility)**.** *$v_i(z; r_i)$ is an exponential or constant absolute risk aversion (CARA) utility function*

eq:cara

$$v_i(z; r_i) = -e^{-r_i z} \tag{3.12}$$

[68]While contribution costs in Equation (3.10) are in expressed in rather arbitrary terms, we can derive a mapping between contribution costs implied by the structural model and time allocation (hours) to project development, ω_i, observed in the empirical sample. Given estimates for a_i, α and data x, G, we can estimate parameters γ_0, γ_1 from a simple linear specification:

$$c_i(x, G) = \gamma_0 + \gamma_1 \omega_i + \epsilon_i$$

where the absolute risk aversion parameter, $r_i > 0$, varies across maintainers $i \in \mathcal{N}$.

Under Assumption (4), the parameter $r_i \in \mathbb{R}$ captures project manager i's relative level of risk aversion: maintainer i is said to be more risk averse as $r_i \to \infty$. Since a portion of project quality in Equation (3.9) uncertain and unobservable, the expected quality preferences under Equation (3.11) and Assumption 4 with imply that as a project manager becomes more risk averse, she suffers greater disutility with increased volatility of both her own package and the inherited volatility of upstream dependencies. We make an assumption over the specific form of this uncertainty in the following section.

3.6.1.4 Information Sets

To introduce uncertainty over project quality, we assume that stochastic quality disturbances ξ_i are unobservable to maintainers *ex ante*.

assum:uncertainty

Assumption 5 (Uncertainty in Package Quality)**.** *Assume the following*

1. $\xi \overset{iid}{\sim} N(0, \Sigma)$ *where* $\Sigma = I\sigma^2$ *and* $\sigma^2 = (\sigma_i^2)_{i \in \mathcal{N}}$ *are known only in distribution by project maintainers.*
2. ξ_i *is independent between projects*
3. ξ *is independent and identically distributed across time periods.*
4. *Observables* (x, y, G) *and parameters* $\theta = (a, \alpha, b, \beta, r, \Sigma)$ *are public information to all maintainers* $i \in \mathcal{N}$.

Therefore maintainers are uncertain about the quality of all projects, including their own. The risk averse maintainer will enjoy greater utility when she takes actions to minimize exposure of her project to any sources of quality risk. Since the distribution ξ is common knowledge, the setting is characterized by a shared level of uncertainty rather than information asymmetries between agents (e.g. **akerlof1978market**).

3.6.2 Equilibrium

sec:structural-equilibrium

With the basic elements of the structural model now established, we now discuss how maintainers are expected to behave in equilibrium. Assume that each project maintainer $i \in \mathcal{N}$ chooses a tuple $(y_i^\star, x_i^\star, \{G_{ij}^\star\}_{j \neq i})$ to minimize development costs $c_i(x, G)$ while keeping their private utility of expected project quality[69] $\mathbb{E}\left[v_i\left(y_i\right)\right]$ above a threshold $\underline{u}_i$. We can express the maintainer's static cost minimization problem as follows:

eq:ump

$$\begin{aligned} \min_{x_i \geq 0, y_i, \{G_{ij}\}_{j \neq i}} \quad & c_i(x, G) \\ \text{s.t.} \quad & u_i(x, y, G) \geq \underline{u}_i \end{aligned} \tag{3.13}$$

where c_i, y_i, and u_i are defined in Equation (3.10), Equation (3.9), and Assumption 4, respectively. We analyze the equilibrium of this system is several distinct phases.[70] First, we must make an assumption over the sequence of project development choices for each individual maintainer. Second, we characterize the equilibrium choices of the continuous quantities $(y^\star, x^\star)$. Third, we derive an expression for the project maintainers expected utility over the quality of their project in equilibrium, $\mathbb{E}\left[v_i(y_i^\star)\right]$. Fourth, we characterize factors influencing the formation and dissolution of software dependencies and derive an expression for

[69]Or put more precisely, maintainer i's private valuation of expected project quality.

[70]It should be noted that the maintainer's optimal choice over contribution levels and dependencies can be represented with alternative formulations. We present some of these alternatives in Appendix 3.C.1 and discuss how we use them to map between the exposition here in Section 3.6.2 and structural estimation covered in Section 3.6.3.

$\Pr(G_{ij} = 1)$, the likelihood that maintainer i imports project j. Finally, we conclude with some comparative statics for equilibrium quantities.

3.6.2.1 Timing

sec:structural-timing

Maintainers myopically best respond to solve the development cost minimization problem in Equation (3.13), conditional on both the state of the system at the beginning of the period, $\mathcal{D}_{t-1}$ and the optimal strategies of other maintainers.

assum:timing

Assumption 6 (Timing)**.** *At the beginning of each period t, a single maintainer $i \in \mathcal{N}$ is presented with the opportunity to change the dependency relationships of their software project. Next, all agents $i, j \in \mathcal{N}$ adjust their contribution levels under the new network. These developments unfold according to the following sequence:*

(S1) Maintainer i chooses a set of optimal dependencies $\{G^{\star}_{ijt}\}_{j \neq i}$, conditional on the state of the ecosystem in the last period, $\mathcal{D}_{t-1}$. This updates the network to $G_{t-1} \rightarrow G^{\star}_{t}$.

(S2) In accordance with the response functions derived in Equation 3.13, all agents $i, j \in \mathcal{N}$ determine their best response contribution levels $x^{\star}_{it}$ under the new network $G^{\star}_{t}$. This updates the remaining observables in the ecosystem: $x_t \rightarrow x^{\star}_{t}, y_{t-1} \rightarrow y^{\star}_{t}$, and $W_{t-1} \rightarrow W_t$.

Therefore by the end of (S2), $\mathcal{D}_{t-1} \rightarrow \mathcal{D}_t$.

The purpose of Assumption 6, beyond providing some structure to the game, is to connect the data generating process of the observed data to an estimation strategy that we outline in Section 3.6.3.[71] Observing the disaggregated evolution software dependency networks over

[71] I the empirical sample, a software project and its dependencies when a new version is registered with the `npm` registry. Hence the timing of our model associates each sample moment $t \in \mathcal{T}$ with a single agent

time allows us to model network formation as a sequence choices made by individual agents in each period, greatly simplifying the estimation procedure compared situations often found in the literature[72] in which only a single network observation is observed. In the following two sections, we derive a characterization of the equilibrium data generating process via backwards induction of the maintainer's game.

3.6.2.2 Optimal contribution decision (x_i)

We first derive equilibrium contribution effort $x^\star$ and project quality $y^\star$, taking the dependency network G as given. The first order necessary conditions for maintainer i's optimal choice of $x_i > 0$ imply[73]

eq:xstar

$$x_i^\star = a_i + \alpha \sum_{j \neq i} G_{ij} x_j^\star \tag{3.14}$$

In matrix form, the system described in Equation (3.14) becomes $x^\star = Aa$ where $A \equiv (I - \alpha G)^{-1}$ and $a = (a_i)_{i \in \mathcal{N}}$. For A to exist, it must be that $|\alpha| < 1$. Therefore $x_i^\star = \sum_j A_{ij} a_j$. In equilibrium, this implies project quality will be given by

eq:ystar

$$y_i^\star = b_i \left(a_i + \alpha \sum_{j \neq i} G_{ij} x_j^\star \right) + \beta \sum_{j \neq i} G_{ij} y_j^\star + \xi_i \tag{3.15}$$

i who then makes $j \neq i$ linking decisions, $\{G_{ijt}\}_{j \neq i}$. This updates the network $G_{t-1} \to G_t$. Then we allow all agents to optimally adjust their contribution levels conditional on the new network G_t so that $(x_{t-1}, y_{t-1}, W_{t-1}) \to (x_t, y_t, W_t)$.

[72]For example, **leung2015two**, **mele2017structural**, **christakis2020empirical**, and **ridder2020estimation**, to name a few.

[73]Technically, since the first-order necessary conditions for the maintainer's problem in (3.13) imply $\frac{\partial c_i(x^\star, G)}{\partial x_i} = \lambda_i \frac{\partial u_i(x^\star, y^\star, G)}{\partial x_i}$ for $x_i^\star > 0$ and a Lagrange multiplier $\lambda_i \geq 0$, the parameter a_i in equilibrium condition Equation (3.14) does not exactly equal the parameter a_i from Equation (3.10). We take a simplifying approach to represent equilibrium effort choice in Equation (3.14) to simplify both exposition and estimation. We discuss these details in Appendix 3.C.1, providing alternative characterizations for the maintainer's problem in (3.13) and provide conditions under which the equilibrium allocations $(x^\star, y^\star)$ across these characterizations coincide. See Proposition 1 in Appendix 3.C.1.

Similarly, $y^\star = B\left(b \circ x^\star + \xi\right) = B\left(b \circ (Aa) + \xi\right)$ where $B \equiv (I - \beta G)^{-1}$, $|\beta| < 1$, and $b \circ x^\star$ denotes the Hadamard product of the vectors b and $x^\star$: $(b \circ x^\star)_i = b_i x_i^\star$ for $i \in \mathcal{N}$.

Remark (Leontief Inverse for Software Dependency Networks)**.** *The matrices A and B would be known as the "Leontief inverse" in the literature on input-output modelling and capture the extent to which the effects, such as increased contribution or fluctuations in quality, percolate through the dependency network G to affect the welfare of dependents. This is the source of network externalities in this framework. Notice that if the spectral radius of αG is less than 1, then $A = I + \sum_{k=1}^{\infty} \alpha^k G^k$.*[74] *Roughly speaking, if maintainer j increases her contribution effort by 1%, then maintainer i will be induced to increases her contribution effort by $A_{ij} a_j$%. This is the source of productivity and quality externalities: the use of dependency j influences the marginal cost of contribution for maintainer i in her own project. For project quality, the influence of dependencies can operate in different directions since the relative influence of dependency j on project i, $B_{ij} = b_j x_j + \xi_j$, is the sum of an upstream contribution effect $b_j x_j$ and unobservable fluctuations or uncertainty ξ_j.*

Finally, these simplifications together imply that equilibrium project quality for project i in Equation (3.15) can also be expressed as follows:

$$y_i^\star = \sum_j B_{ij} \left(b_j \left(\sum_k A_{jk} a_k \right) + \xi_j \right)$$

Notice that equilibrium project quality is therefore simply a function of equilibrium contribution and fluctuations ξ. The advantage of this characterization of project quality implied by Equations (3.14) and (3.15) is that a maintainer's utility over the expected quality of

[74]This concept of attenuating indirect influence from dependencies further and further upstream is captures in Example 3.3.1.

their project, $u_i(x, y, G)$, can now be expressed as simply a function of the network G and parameters. We derive an expression for maintainer utility in the following section.

3.6.2.3 A Maintainer's Utility over Expected Project Quality

Before discussing optimal dependency formation behavior, we first seek to simplify the project manager's expected utility of their project $u_i(x, y, G) = \mathbb{E}\left[v_i(y_i)\right]$ under the dependency graph G and equilibrium contribution $x^\star$. We will use the derived expression in the subsequent section to evaluate maintainer i's expected incremental utility from forming a dependency relationship with project j.

Conditional on optimal choices of contribution effort, $x^\star$, resulting project quality $y^\star$, and the dependency graph G, by the normality of ξ_i stipulated by Assumption 5, $u_i(x^\star, y^\star, G)$ becomes

$$\begin{aligned} u_i(x^\star, y^\star, G) &= -\exp\left(-r_i \sum_j B_{ij}\left(b_j x_j^\star - \frac{r_i}{2} B_{ij}\sigma_j^2\right)\right) \\ &= -\exp\left(-r_i \sum_j B_{ij}\left(b_j\left(\sum_k A_{jk} a_k\right) - \frac{r_i}{2} B_{ij}\sigma_j^2\right)\right) \end{aligned} \tag{3.16}$$

eq:eu

Details for this simplification can be found in Appendix 3.C.2. Equation (3.16) makes clear the notion that maintainer preferences are shaped by both (1) there relative degree of risk tolerance r_i and (2) the extent to which upstream dependents vary with respect to quality, σ_j^2 for $j \in \mathcal{N}$.

3.6.2.4 Optimal dependency formation decision (G_{ij})

In the previous section, we saw how preferences reflecting risk aversion influences a manager's expected utility of their project in equilibrium. In this section we explore expected incremental utility changes under the formation of new links. Intuitively, manager i ought to only form a dependency with project j if and only if conditional on the realization of linking disturbances, their expected utility $u_{ij}^{+} \equiv u_i(x^{\star}, y^{\star}, G + ij)$ is greater than the utility they expect without using project j, $u_{ij}^{-} \equiv u_i(x^{\star}, y^{\star}, G - ij)$.

The non-linearity of $u_i(\cdot)$ is designed reflect the fact the maintainers may vary with respect to the amount of risk they are willing to introduce into their own project by importing dependencies. This is a notable departure from much of the literature on strategic network formation, which typically employ linear utility specifications to simplify the calculation of incremental changes to utility under different links.

To address this complication, we first make an assumption over the way in which stochastic and unobserved utility shocks, realized when either forming or not forming the dependency, enter into this decision. This assumption will then allow us to adopt a change-of-variable technique suggested by **fosgerau2009discrete** in an effort to facilitate easier estimation of parameters within the maintainer's discrete choice problem, exploiting the fact that by Assumption 4, $u_i(\cdot) = \mathbb{E}\left[v_i(\cdot)\right] < 0$ over its domain. Assume that ε_{ij}^{+} and ε_{ij}^{-} are stochastic and unobserved utility shocks realized by maintainer i from either forming (ε_{ij}^{+}) or not forming (ε_{ij}^{-}) the dependency with j.

assum:linking

Assumption 7 (Link Formation with Multiplicative Disturbances)**.** $\varepsilon_{ij}^{+}, \varepsilon_{ij}^{-} \in (0, +\infty)$ *are independent and identically distributed across project pairs and enter the dependency formation problem of the maintainer multiplicatively. Upon learning a realization for* $(\varepsilon_{ij}^{+}, \varepsilon_{ij}^{-})$*, project manager* i *forms uses project* j *as a dependency according to the following rule:*

$$G_{ij} = 1 \iff u_{ij}^{+}\varepsilon_{ij}^{+} \geq u_{ij}^{-}\varepsilon_{ij}^{-} \tag{3.17}$$

and $G_{ij} = 0$ *otherwise.*

Following the linearization transformation of **fosgerau2009discrete**, we can show that under Assumption 7, the equilibrium likelihood that maintainer i import project j as a dependency becomes

$$\Pr(G_{ij} = 1) = F_{\epsilon}\left(r_i \underbrace{\left(\sum_j \Delta B_{ij} b_j \left(\sum_k \Delta A_{jk} a_k \right) \right)}_{Z_{0ij}} - \frac{1}{2} r_i^2 \underbrace{\left(\sum_j \Delta B_{ij}^2 \sigma_j^2 \right)}_{Z_{1ij}}; \theta_{\epsilon} \right) \tag{3.18}$$

eq:link-likelihood

where θ_{ϵ} are a vector of parameters for the random variable $\epsilon_{ij} \equiv \epsilon_{ij}^{+} - \epsilon_{ij}^{-} \sim F_{\epsilon}$. Complete details for this derivation can be found in Appendix 3.C.3.[75]

In Equation (3.18), we define $\Delta B_{ij} \equiv B_{ij}^{+} - B_{ij}^{-}$ and $\Delta A_{jk} = A_{jk}^{+} - A_{jk}^{-}$ as the difference between elements of the Leontief inverse matrices for project quality and contribution effects

[75]Briefly, this simplification works as follows. First, define $\overline{u}_{ij}^{+} \equiv -\lambda \ln(-u_{ij}^{+})$ and $\overline{u}_{ij}^{-} \equiv -\lambda \ln(-u_{ij}^{-})$. Second, define $\epsilon_{ij}^{+} \equiv -\lambda \ln(\varepsilon_{ij}^{+})$, $\epsilon_{ij}^{-} \equiv -\lambda \ln(\varepsilon_{ij}^{-})$. Third, define $\epsilon_{ij} \equiv \epsilon_{ij}^{-} - \epsilon_{ij}^{+} \overset{\text{iid}}{\sim} F_{\epsilon}(z; \theta_{\epsilon})$ where $\lambda > 0$ and $u_{ij} \equiv \overline{u}_{ij}^{+} - \overline{u}_{ij}^{-}$, $\epsilon_{ij} \equiv \epsilon_{ij}^{-} - \epsilon_{ij}^{+}$. Ultimately, $\Pr(G_{ij} = 1) = \Pr(u_{ij} \geq \epsilon_{ij}) = F_{\epsilon}(u_{ij}; \theta_{\epsilon}/\lambda)$ connects equilibrium linking behavior in Assumption (7) with the empirical likelihood of observing a dependency relationship in Equation (3.18).

under $G+ij$ and $G-ij$.[76] Intuitively, ΔB_{ij} and ΔA_{ij} will reflect the change in exposure to net quality and contribution cost influences that arise when maintainer i import project j as a dependency.

For the purposes of exposition, we rearrange u_{ij} into a quadratic function of r_i and label the coefficients Z_{0ij} and Z_{1ij} in the last equality of Equation (3.18). These coefficients are functions of the network G and parameters $(\alpha, a, \beta, b, \Sigma)$. In Section 3.6.3 and Appendix **??**, we show that since $(\alpha, a, \beta, b, \Sigma)$ can be estimated using moment conditions (3.9) and (3.14) via generalized method of moments, the result in Equation (3.18) will form the basis for a likelihood function for the remaining unknown parameters, r and θ_ϵ. Hence under the assumptions outlined in the body of the paper, r and θ_ϵ can be estimated via maximum likelihood.

Finally, a distributional assumption over ϵ_{ij} to refine Assumption 7 helps inform comparative statics in Section 3.6.2.5 and the estimation procedure described in Section 3.6.3.

assum:linking-final

Assumption 8. *ϵ_{ij} is a logistic random variable, independent and identically distributed across potential links and time.*

Note that Assumption 8 can be supported if conditional on Assumption 7, ϵ^+_{ij} and ϵ^-_{ij} are are assumed to be mutually (i.e. across both project pairs and time) independent Gumbel random variables.

[76] Let $\Delta B_{ij} \equiv B^+_{ij} - B^-_{ij}$ where $B+ij = [B^+_{ij}]_{i,j\in\mathcal{N}} = (I-\beta(G+ij))^{-1}$ and $B-ij = [B^-_{ij}]_{i,j\in\mathcal{N}} = (I-\beta(G-ij))^{-1}$. Equivalently, let $\Delta A_{jk} = A^+_{jk} - A^-_{jk}$ where $A+ij = [A^+_{jk}]_{j,k\in\mathcal{N}} = (I-\alpha(G+ij))^{-1}$ and $A-ij = [A^-_{jk}]_{j,k\in\mathcal{N}} = (I-\alpha(G-ij))^{-1}$.

3.6.2.5 Comparative Statics

sec:comparative-statics

How do various parameters influence the dependency formation in equilibrium? Consider the effect of slight perturbations to parameters on the likelihood of dependency formation (Equation (3.18)):

1. **Risk Aversion (r_i):** Using the likelihood that maintainer i imports project j, one can show that

$$\frac{\partial \Pr(G_{ij} = 1)}{\partial r_i} \begin{cases} < 0 & \text{if } Z_{ij} < r_i \\ \geq 0 & \text{if } Z_{ij} \geq r_i \end{cases}$$

   since $\frac{\partial F_\epsilon}{\partial z} > 0$ and $r_i > 0$ and $Z_{ij} \equiv \frac{Z_{0ij}}{Z_{1ij}}$. Therefore if maintainer i becomes less likely to import dependency j as her risk aversion r_i increases beyond a threshold, Z_{ij}. We can interpret Z_{ij} as a net benefit threshold for maintainer i. If $Z_{ij} < r_i$, the maintainer i's is risk averse to the point that they consider the additional risk of depending upon project j to outweigh the net benefits in terms of (dis)utility.

2. **Project Quality Variance (σ_k^2):**

$$\frac{\partial \Pr(G_{ij} = 1)}{\partial \sigma_k^2} \leq 0$$

   Notice also that, since we have ruled out risk seeking preferences by restricting $r_i > 0$, increased project volatility will deter all maintainers to some extent. This effect will be stronger for more risk averse maintainers: $\frac{\partial^2 \Pr(G_{ij}=1)}{\partial r_i \partial \sigma_k^2} \leq 0$.

In other words, more risk averse maintainers are less likely to rely on dependencies, *ceteris paribus*, once they are beyond a certain threshold.[77] Similarly, increased volatility in project quality reduces the likelihood of dependency formation. Overall, while intuitive, these comparative statics reveal a level sophistication embedded in the dependency management choice that confronts the project maintainer. These complexities ought to guide the design and interpretation of simulated counterfactuals

3.6.3 Estimation

sec:estimation

In general, estimating strategic network formation models is complicated. Two broad classes of approaches deal with estimating network formation when only a single network observation is available: (1) non-iterative estimation using link formation strategy is assumed to take place under incomplete information (**leung2015two**; **ridder2020estimation**) and (2) iterative strategic network formation where opportunities to form or dissolve links arrive according to some specified sequence (**mele2017structural**; **christakis2020empirical**; **badev2021nash**; **hsieh2022structural**). While our model falls into the latter class, we can further exploit the fact that the dependency network is effectively observed in continuous time. In this case, the complete sequence of linking decisions is known to the econometrician, an advantage not present in empirical settings where only the final equilibrium network is observed. By Assumption 6, we leverage this feature to model the data generating process as a Markov chain: in each period, a single maintainer makes new or revises existing dependency decisions based on the current state of the network. Immediately after, all agents subsequent adjust their contribution levels under the new network, and the process repeats

[77]In other words, as $r_i \to \infty$.

with another maintainer's link decision. Our estimation framework is therefore quite similar to the approach described in **snijders2010maximum**, as the empirical setting bears a closer resemblance to a network panel and the model falls into a broad class of "stochastic actor-oriented models". Consequently, our model of the coevolution of both contribution actions and dependency formation decisions is actually much simpler to similar approaches with access to only a single network observations (**badev2021nash**; **hsieh2022structural**). In these cases, a high-dimensionality state space of potential networks and action profiles must be tracked[78] in order to explain observed equilibria.

We relegate full details and discussion of our structural estimation strategy (i.e. the moment conditions and likelihood function) to Appendix **??**. In broad strokes, the procedure can be described as follows. The observed data is $\mathcal{D} = (x_t, y_t, G_t, W_t)_{t \in \mathcal{T}}$. Structural parameters to be estimated are $\theta = (a, \alpha, b, \beta, \Sigma, r, \theta_\epsilon)$.[79] We break estimation into two phases. In the first phase described in Steps 1 through 3, we use moment conditions and identities from the structural model to recover estimates for $(a, \alpha, b, \beta, \Sigma)$ using the generalized method of moments (GMM).[80] In the second phase described in Step 4, we combine observed data with the parameter estimates in the first phase to estimate (r, θ_ϵ), using equilibrium dependency formation characterized in Equation (3.18) in maximum likelihood estimation (MLE).

[78]**badev2021nash** and (**hsieh2022structural**) use approach in which behavior is determined by an exact potential game, the equilibrium of which can be characterized by a Gibbs measure. Both the presence externalities and the non-linear nature of our structural model would make the use of a similar method quite complicated.

[79]While not a part of the discussion of the structural model in the main body of Section **??**, we introduce the parameter γ in Appendix **??**.

[80]We also describe how to use simpler methods such as ordinary least squares (OLS) to recover these parameters in sequence.

3.6.3.1 Dimensionality Reduction

sec:structural-dim-reduce

As discussed previously in the description of the empirical sample, we take steps to reduce the dimensionality to facilitate and simplify structural estimation. First, we limit the size of the empirical sample by beginning with a "seed" of just 10 core projects, sample upstream[81], and restrict the set of observed sample moments to only minor versions of package releases. Second, we restrict the range of the risk aversion parameter r to the half-interval $(0, 1]$ as it is somewhat of a nuisance parameter which is really only of interest in distribution so as to characterize the DGP. From a computational perspective, we discuss additional simplifications to ease structural estimation in Appendix 3.D.1.

3.7 Counterfactual Analysis

sec:dsc-counterfactual

Developing a structural model allows of to completely characterize the data generating process for the decision-making process of project maintainers and the evolution of the software dependency network. Importantly, it allows us to explore the effect of counterfactual interventions on the resulting equilibrium of the system. Specifically, we can either (1) perturb parameters or (1) remove certain key projects, re-simulate the data generating process[82], and analyze the impact of the counterfactual on contribution, project quality, contribution costs, and project maintainer welfare (i.e. utility).

[81]As opposed to sampling downstream. The top 10 most depended upon packages in the NPM ecosystem features tens of thousands of downstream dependent packages each.

[82]That is, we can use the arrival sequence of projects observed in the sample moments and use the equilibrium conditions of the structural model to simulate contribution decisions, project quality evolution, and dependency formation decisions from for the sample period from start to finish.

To evaluate the impact of each counterfactual, we first must define a social welfare function for the software dependency graph G, conditional on the set of project $\mathcal{N}$ and parameters θ:

eq:welfare-utility

$$u_{\mathcal{N}}(G;\theta) = \sum_{i \in \mathcal{N}} u_i(G;\theta) \tag{3.19}$$

Notice several notational simplifications.[83]. We define similar functions for aggregate contribution, project quality, and contribution costs. For each counterfactual, we specify a change to parameters or the set of projects and then simulate the data generating process under this new system, beginning at the beginning of the sample period.[84] Using the aggregate welfare functions, we compare the counterfactual equilibrium for the final sample period[85] with a baseline based on the observed data.

3.7.1 Reducing Fluctuations in Project Quality

Our discussion of structural model comparative statics in Section **??** establishes that risk averse project maintainers are less likely to use packages highly volatile in quality as dependencies. Moreover both conventional wisdom and empirical evidence suggest developers are reluctant to import dependencies that are either immature or subject to frequent, backwards-incompatible changes (**zerouali2018empirical**). Innovations in software best practices can promote stability in the quality of OSS projects, such as including testing frameworks to ensure intended functionality (**ellims2006economics**), using automation and continuous integration to efficiently and safely integrate contributions from the wider

[83]Since each y_i is ultimately just a function of x, we can write $u_i(x, G;\theta)$ and $u_{\mathcal{N}}(x, G)$. Furthermore, in equilibrium $x^\star$ is really just a function of the network G and parameters θ, we could even go one step further a write $u_{\mathcal{N}}(G;\theta)$.

[84]We take the average of 10 counterfactual simulations.

[85]September 2022

community (**vasilescu2015quality**; **hilton2016usage**), keeping the design scope of the project focused and succinct[86], and using systems like semantic versioning to release software often but in a manner respectful of downstream dependents (**raemaekers2017semantic**; **decan2019package**). For the purposes of the counterfactual analysis, we specifically consider alternative levels of project quality volatility $\Sigma \to \Sigma'$ for $\Sigma' \in \{0.5\Sigma, 2\Sigma, 4\Sigma\}$.

3.7.2 Increasing Developer Risk Aversion

While some theoretical (**walsh2002role**) and experimental (**kina2016analyzing**) analyses of risk aversion for software developer exist, there is comparatively little empirical evidence of how risk tolerance influences project maintainer decision-making and the resulting equilibrium. In our structural model, the likelihood to import dependencies increases with a maintainer's level of risk tolerance, which has the potential to improve package quality and reduce development costs. On the other hand, increased risk aversion may prevent risky dependency relationships from being formed, albeit at increased contribution costs. In our counterfactuals, we modify the profile of maintainer risk aversion $r \to r'$ for $r' \in \{r + \sigma_r, \mathbf{1}, \min(r)\}$.[87]

3.7.3 Key Projects

In the spirit of the "key player analysis" described by **ballester2006s**, **lee2021key**, and **hsieh2022structural**, we define a key software project $i^\star$ such that

eq:key-project

$$i^\star = \arg\max_{i \in \mathcal{N}} u_{\mathcal{N}}(x, G; \theta) - u_{\mathcal{N}\setminus i}(x, G; \theta) \tag{3.20}$$

[86]Recall the UNIX philosophy: "Make each program do one thing well."

[87]That is, when risk aversion for each project maintainer is increased by one standard deviation, equal to 1, and equal to the minimum value of the estimated from the sample.

conditional on a set of parameters θ. We could determine the key project $i^\star$ by iteratively simulating aggregate welfare by Equation (3.20). However simulating 1,263 counterfactuals entails a significant computational burden. We therefore opt for a simpler approach of removing the package with the most downstream dependents in the final period observed in the sample.[88] The package with the most downstream dependents in our sample is `babel`, a "tool that helps you write in the latest version of JavaScript" (**babel2022**).[89]. Additionally, we estimate counterfactual equilibria after removing the 10 packages with (1) the largest Katz-Bonacich and (2) betweenness centrality measures in the final sample period. For the key player analysis, we compare the welfare of the remaining packages under the baseline with their outcomes in a world in which the critical set of packages is removed. In this way, we are estimating the value of externalities the key packages generate for the software network as a whole.

3.7.4 Summary

We present the results of the counterfactual exercises in Table 3.B.5. Overall, the overall impact of our counterfactuals is comparatively small in percentage terms. This result is largely driven by the fact that none of our counterfactuals significantly alters the network formation path. We argue this pattern is driven by the fact that individual project features largely drive project management decisions, which are in turn robust to marginal perturbations to project quality volatility, risk aversion, and the removal of core packages.

[88]We could also remove the package with the largest Katz-Bonacich centrality. Coincidentally, the package with the largest Katz-Bonacich centrality measure in our sample also has the most dependencies.

[89]As of October 2022, the `babel` package has over 15,450 commits, 1,033 distinct contributors, 5,500 forks, and 41,000 stars on GitHub.

In particular, contribution is virtually unchanged under different levels of risk aversion. On the other hand, the results in Table 3.B.5 seem to indicate that significantly increasing maintainer risk aversion ($r_i' = 1$) can increase aggregate package quality (2.11%) enough to effectively offset the direct welfare effects[90] (-0.04%). In other words, upstream maintainers can create value for downstream dependents by exercising more discipline when choosing what software to rely on that in turn offsets increased disutilty from quality uncertainty in aggregate.[91]

The effect of increasing package volatility is slightly more puzzling. We can see that reducing volatility actually has a small increase on aggregate project quality (0.19%). Increasing volatility has a larger positive influence on aggregate quality. There is virtually no change to contribution patterns or welfare. We argue this is a result of strong package fixed characteristics: maintainers seem less concerned with uncertainty over package unobservables compared with their immediately appreciable benefits.

Finally, our counterfactuals that remove key packages reveal the most interesting results. Core packages with the highest Katz-Bonacich centrality measures create significant value for the dependency network: removing the top 10 packages, which number less than 0.8% of the sample, reduces aggregate package quality by -5.73% for their remaining peers. Moreover, aggregate contribution falls by -1.3%, suggesting that maintainers find contribution in their own packages complementary with these upstream core packages.

[90] $\frac{\partial u_i}{\partial r_i} < 0$ for $r_i > 0$.

[91] One could also interpret this as shifting the burden of risk from dependents to upstream maintainers.

3.8 Discussion

sec:dsc-discussion

In an effort to understand the dynamics and value created by software dependency networks, we have studied micro-founded decision making from the perspective of the maintainer of an OSS project. We have developed both reduced form and structural methodologies and brought them to bear on an empirical sample of 1,263 Node.js packages observed over time. Overall, we find that individual project features are largely responsible for driving maintainer decisions. In our reduced form approach, we find that upstream projects have relatively limited effect on downstream quality and contribution levels on average. Complementarity between upstream and downstream contribution was greatest in the earlier periods of the dependency network. Our structural approach, to the best of our knowledge, is the first attempt to micro-found the cost minimization decision of a risk averse software maintainer within a strategic network formation model. In doing so we can characterize aggregate network evolution as the aggregation of individual decision-making over time. Our counterfactuals reveal that while the network formation process is relatively robust to perturbing individual parameters like maintainer risk aversion and project quality volatility, removing highly critical core dependencies can have outsized influences on package quality downstream.

A better understanding of software dependency formation and its impact on downstream users concerns a broad population of stakeholders and has even garnered attention at the public policy level (**eo14028**). Our study develops a framework that would clearly benefit from extension and further inquiry. In particular, while we have endeavored to cast welfare effects of dependency networks in terms of production costs, estimates of the consumption value of OSS remains an ongoing challenge. Additional research is needed to connect the implications of software dependency management to value created with respect to labor markets, firm profitability, and innovation.

Finally, while we have attempted to characterize the salient features of this setting in our framework, our results seem to indicate that individual project features, as well as features of their maintainers, are critically important to understand welfare effects in detail. A major innovation in this line of inquiry would be to integrate individual characteristics of maintainers themselves. These characteristics have historically been difficult to observe in aggregate but recent efforts are underway to collect more refined information about OSS collaboration communities (**augur2022**; **duenas2021grimoirelab**).

Appendices

3.A Figures

app:dsc-figures

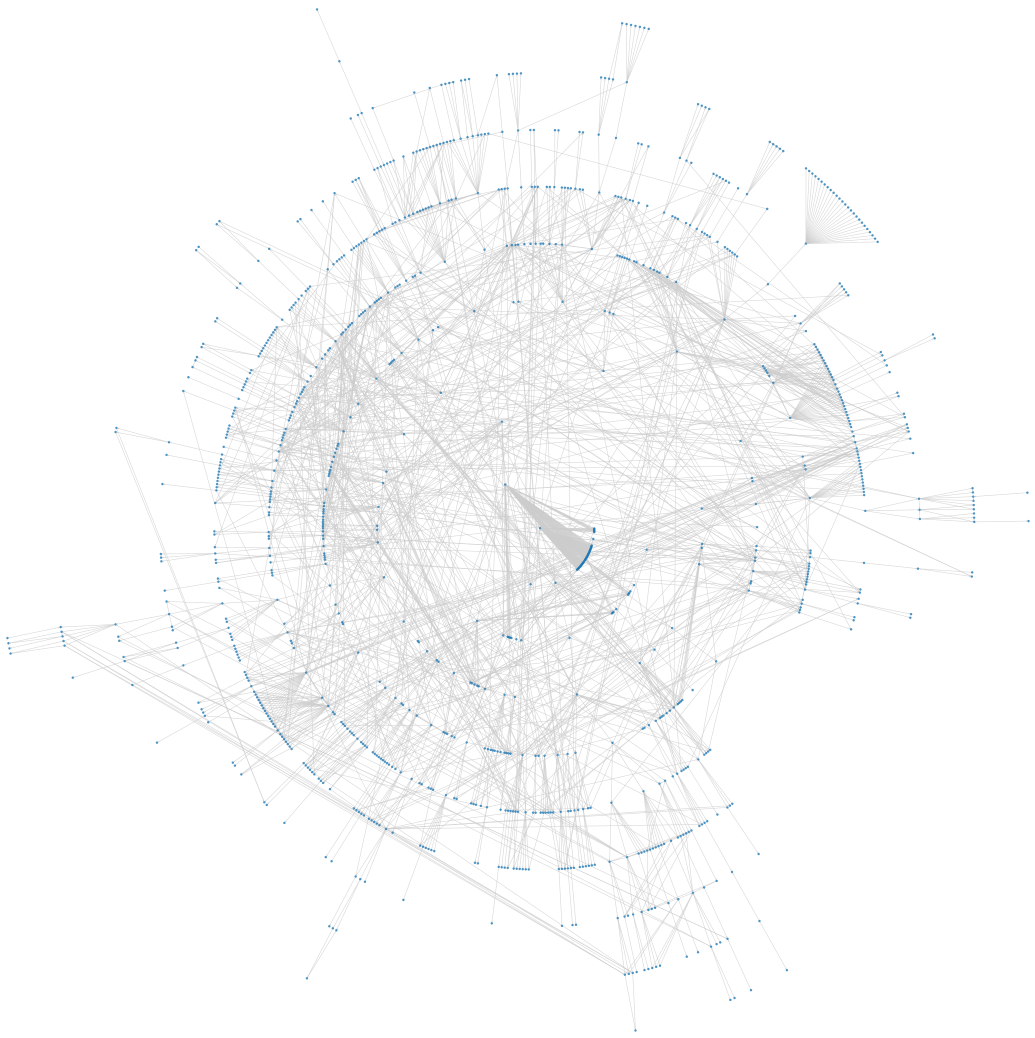


Figure 3.A.1: Empirical Node.js Dependency Network Sample (fig:dsc_empirical_network

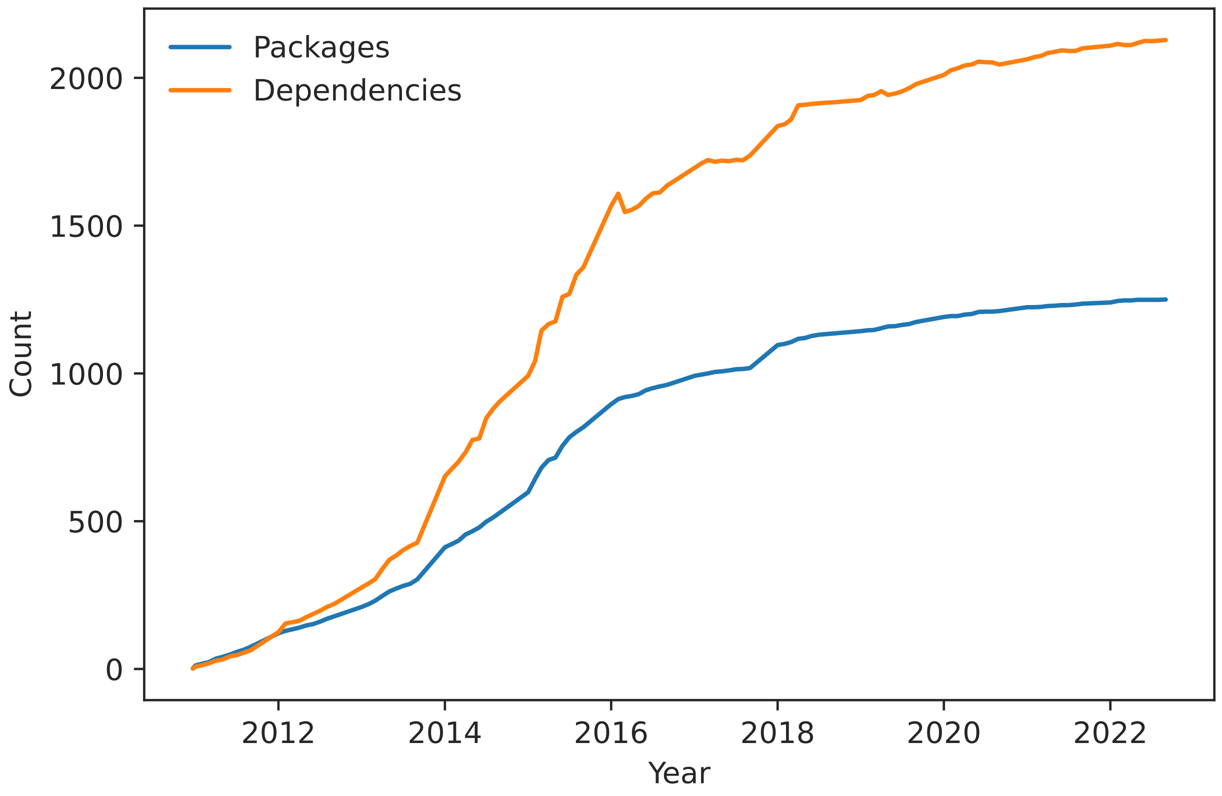


Figure 3.A.2: Empirical Dependency Network Sample (growt fig:dsc_network_size

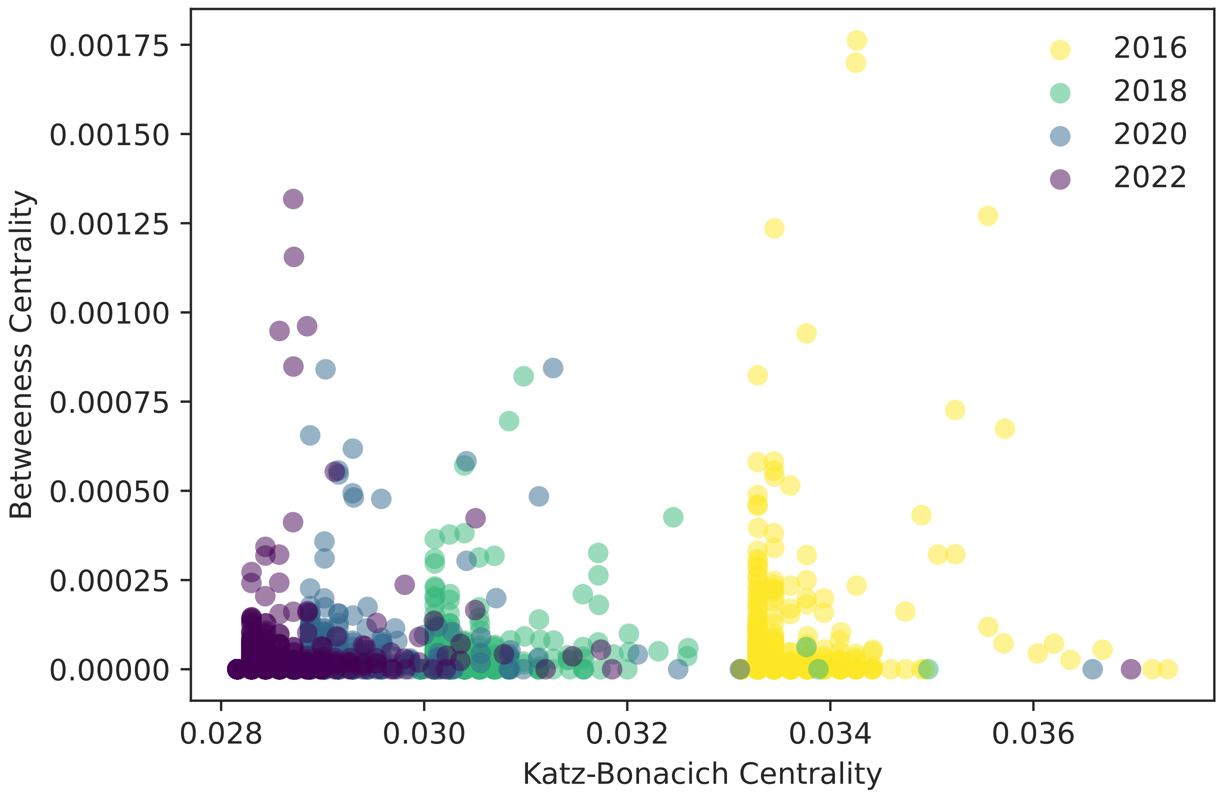


Figure 3.A.3: Empirical Dependency Network Sample (relationship between Katz-Bonacich and Betweenness centrality at the project level) fig:dsc_network_kb_between_scatter

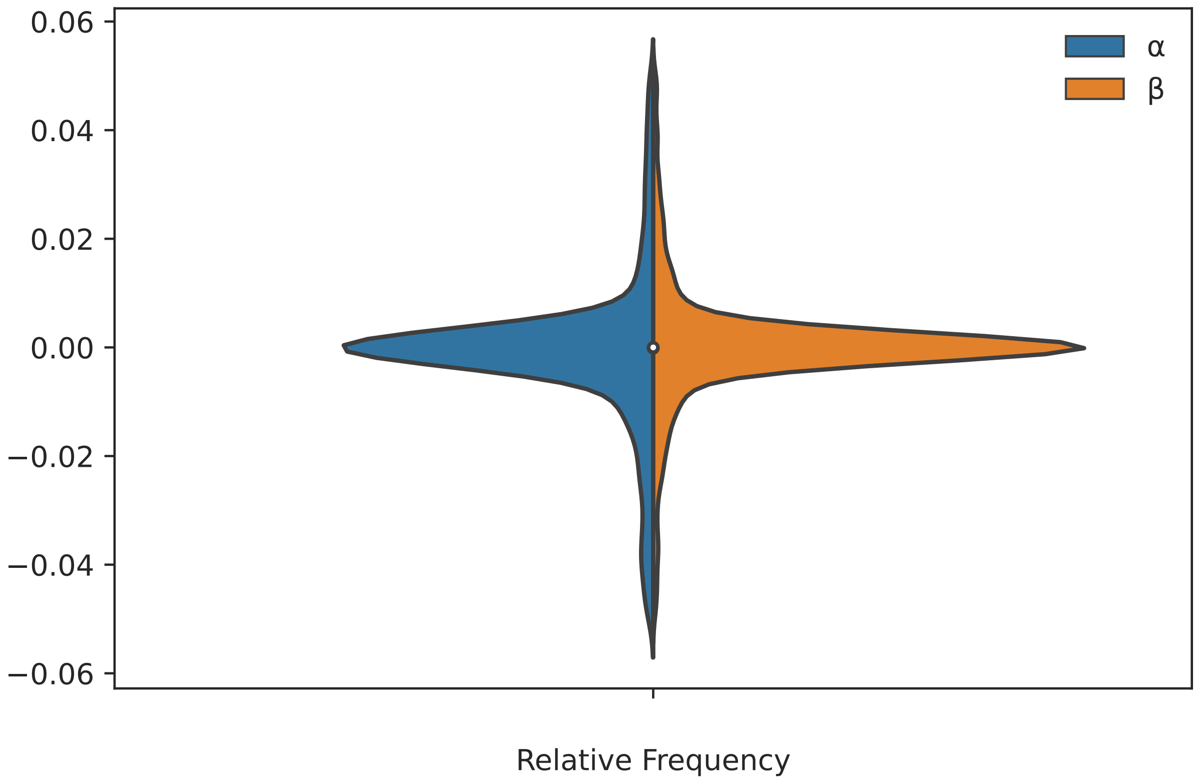


Figure 3.A.4: Project-level Heterogeneity in Reduced Form Estimates for Equation (3.5) and Equation (3.6).

fig:reduced_proj_het

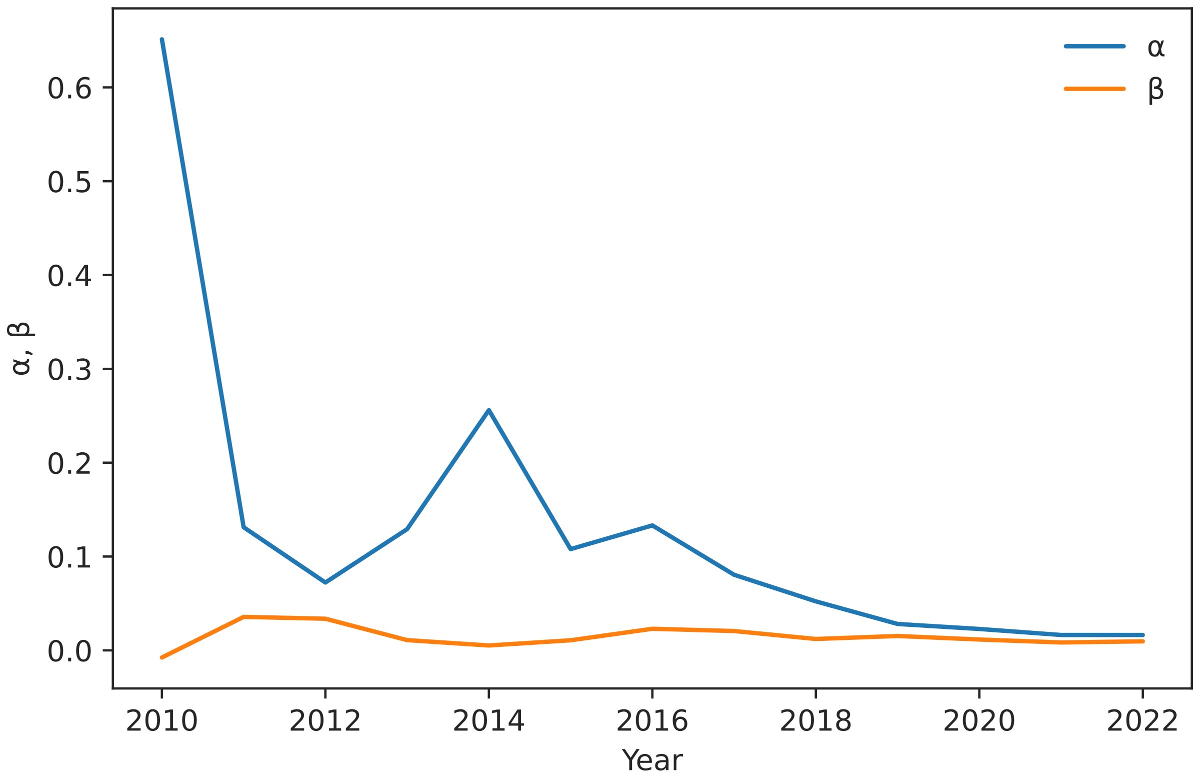
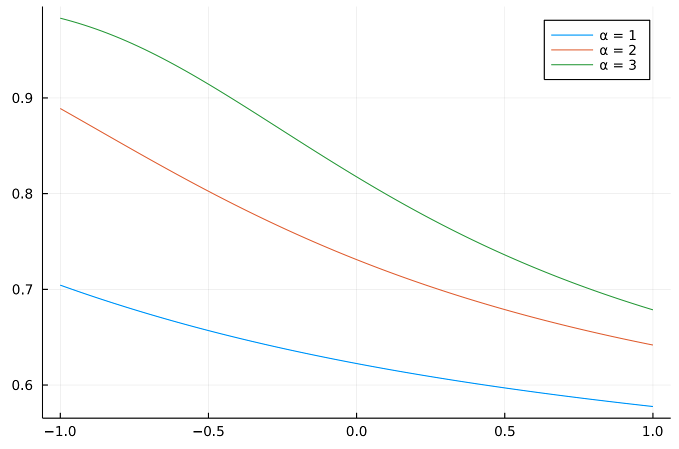
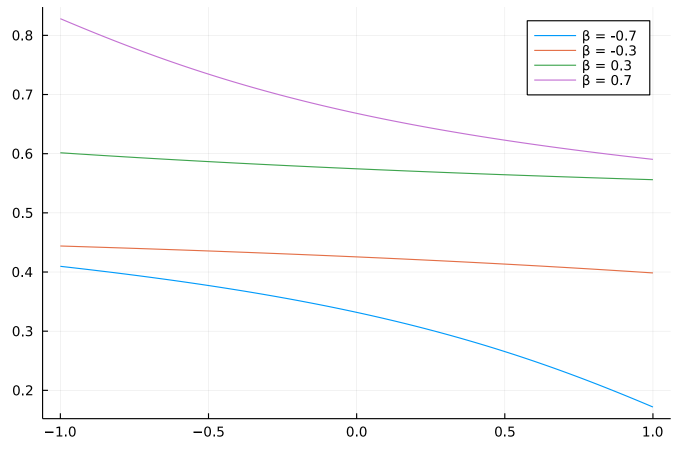


Figure 3.A.5: Temporal Heterogeneity in Reduced Form Estimates for Equation (3.5) and Equation (3.6). fig:reduced_time_het

(a) $\Pr(G_{ij} = 1)$ as α_j varies



(b) $\Pr(G_{ij} = 1)$ as β_j varies



(c) $\Pr(G_{ij} = 1)$ as σ_j^2 varies

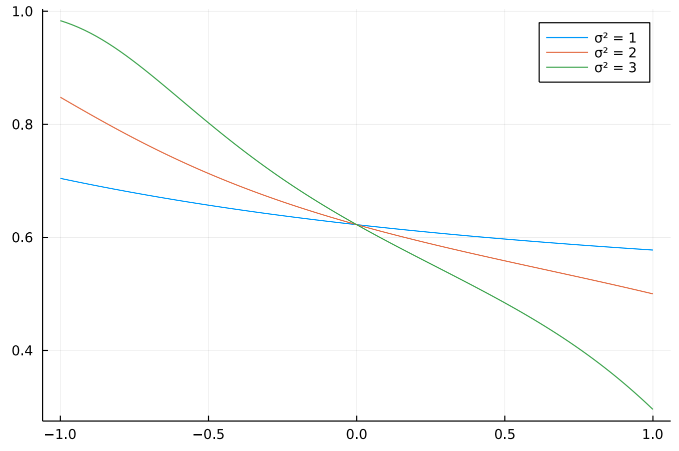


Figure 3.A.6: Comparative Statics – Probability of dependency formation, $\Pr(G_{ij} = 1)$, as risk aversion, r_i, varies.

fig:p__ij__comparative__statics

3.B Tables

app:dsc-tables

Table 3.B.1: Empirical Node.js Dependency Network Sample Descriptive Statistics

tab:stats

Notation	Measure	Obs.	Mean	SD	Min	Median	Max
x_{it}	**Project Contribution:** Cumulative total of commits to project i at time t.	206,598	2,703	6,573	1	195	81,641
$\sum_{j \neq i} G_{ijt} x_{jt}$	**Dependency Contribution:** Cumulative total of commits to project i's dependencies.	206,598	1,451	19,427	0	0	1,098,604
y_{it}	**Project Quality:** Sum of (log) complexity and cumulative contributors (log), scaled to $[0, 1]$.	206,598	0.363	0.199	0	0.334	1
$\sum_{j \neq i} G_{ijt} y_{jt}$	**Dependency Quality:** Sum of quality for project i's dependencies.	206,598	0.225	1.236	0	0	55.2
ω_{it}	**Time Allocation**: Cumulative project labor hours.	206,598	2,909	7,571	2	202	105,504
W_{it}	**Contributors**: Cumulative number of contributors.	206,598	233	983	1	20	17,195
W_{it}	**Core Contributors**: Smallest number of cumulative contributors with $\geq$80% of total contribution.	206,598	20	233	1	1	4,421
W_{it}	**Bus Factor**: Ratio of core contributors to total contributors.	206,598	0.209	0.260	0.003	0.121	1
W_{it}	**Files**: Number of files in codebase.	206,598	2,692	7,109	1	26	65,724
W_{it}	**SLOC**: Single lines of code in codebase.	206,598	129,556	326,412	2	2,717	2,974,829
W_{it}	**Modularity**: Ratio of file count to SLOC.	206,598	222	2,528	1.87	50.5	74,349
W_{it}	**Documentation**: Ratio of commented lines to SLOC.	206,598	0.110	0.172	0	0.046	3.18
W_{it}	**Number of languages**: Count of distinct programming languages.	206,598	7.87	3.34	1	7	29
W_{it}	**Project Age**: Days since first commit.	206,598	1,240	987	0	1,019	4,278
$d_{it}^{out} \equiv \sum_{j \neq i} G_{ijt}$	**Upstream Dependencies:** Number of external project dependencies project i declares.	161,211	2	3.76	0	1	74
$d_{it}^{in} \equiv \sum_{j \neq i} G_{jit}$	**Downstream Dependents:** Number of external projects that depend on project i.	161,211	5	7.94	0	2	66

Table 3.B.2: Reduced Form – Effect of Upstream Dependencies on Project Contribution

tab:reduced-contrib

	Project Commits							
	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
Constant	2,653***	-68.19***						
	(14.51)	(9.744)						
Dependency Commits	0.034***	0.000***	0.003***	0.007***	0.002	0.000*	0.000	-0.000
	(0.002)	(0.000)	(0.001)	(0.001)	(0.001)	(0.000)	(0.000)	(0.000)
Project Quality		156.2***				959.5	126.9***	790.1***
		(22.26)				(518.5)	(19.57)	(322.0)
# Contributors		0.101***				0.546*	0.211***	1.542*
		(0.017)				(0.276)	(0.034)	(0.742)
Bus Factor		18.96***				68.50*	7.401*	6.286
		(3.851)				(32.56)	(2.928)	(21.80)
SLOC		0.000***				0.000	0.000***	0.000
		(0.000)				(0.000)	(0.000)	(0.000)
Documentation		-6.203				108.5	9.413***	136.5*
		(3.201)				(67.99)	(3.578)	(61.60)
Modularity		-0.001***				-0.001	-0.001***	-0.002
		(0.000)				(0.001)	(0.000)	(0.001)
# Languages		6.423***				7.306	5.836***	2.513
		(0.996)				(11.41)	(0.973)	(11.57)
Age		-0.012***				-0.002	-0.017***	-3.617
		(0.002)				(0.026)	(0.004)	(2.415)
Controls		✓				✓	✓	✓
Project FE			✓		✓	✓		✓
Time FE				✓	✓		✓	✓
R^2	0.010	0.999	0.964	0.674	0.988	0.999	0.999	0.999
Observations	206,598	202,905	206,575	201,336	201,308	202,869	196,930	196,894

Note: This table contains coefficient estimates from ordinary least squares (OLS) and fixed effect (FE) estimates for the reduced form relationship between project contribution and upstream dependency contribution in Equation (3.5). Specification variations are reported in columns. Standard errors, heteroskedasticity-robust for OLS and clustered by project for FE models, are reported in parentheses below each coefficient. Additional covariate controls not reported in the table include 3 lags each of project commits and dependency commits and the square of project age. All terms rounded to four significant figures. Statistical significance indicators: *** $\Rightarrow p \leq 0.001$, *** $\Rightarrow p \leq 0.01$, and * $\Rightarrow p \leq 0.1$ where p is the p-value for the coefficient estimate.

Table 3.B.3: Reduced Form Equation (3.5) – Effect of Upstream Dependencies on Project Quality

tab:reduced-quality

	Project Quality							
	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
Constant	0.302***	0.002***						
	(0.001)	(0.049)						
Dependency Quality	0.017***	0.000	0.004***	0.009***	0.002	0.001	0.000**	0.000
	(0.001)	(0.000)	(0.002)	(0.001)	(0.001)	(0.001)	(0.000)	(0.001)
Project Commits	0.000***	0.000	0.000*	0.000***	0.000***	0.000	0.000	0.000
	(0.000)	(0.000)	(0.000)	(0.000)	(0.000)	(0.063)	(0.047)	(0.052)
# Contributors		0.000				-0.000	0.000	0.000
		(0.000)				(0.000)	(0.000)	(0.002)
Bus Factor		-0.003***				-0.007***	-0.003***	-0.007***
		(0.000)				(0.001)	(0.000)	(0.001)
SLOC		-0.000				0.000	-0.000	-0.000
		(0.000)				(0.000)	(0.000)	(0.001)
Documentation		0.001***				0.002***	0.001***	0.003
		(0.000)				(0.000)	(0.000)	(0.003)
Modularity		0.000				0.000	0.000	0.000
		(0.000)				(0.000)	(0.000)	(0.000)
# Languages		0.000***				0.002***	0.000***	0.002
		(0.000)				(0.000)	(0.000)	(0.002)
Age		0.000				0.000	0.000	0.000
		(0.000)				(0.000)	(0.000)	(0.000)
Controls		✓				✓	✓	✓
Project FE			✓		✓	✓		✓
Time FE				✓	✓		✓	✓
R^2	0.514	0.999	0.966	0.764	0.987	0.999	0.999	0.999
Observations	206,598	202,905	206,575	201,336	201,308	202,869	196,930	196,894

Note: This table contains coefficient estimates from ordinary least squares (OLS) and fixed effect (FE) estimates for the reduced form relationship between project quality and upstream dependency quality in Equation (3.6). Specification variations are reported in columns. Standard errors, heteroskedasticity-robust for OLS and clustered by project for FE models, are reported in parentheses below each coefficient. Additional covariate controls not reported in the table include 3 lags each of project quality and dependency quality and the square of project age. All terms rounded to four significant figures. Statistical significance indicators: *** $\Rightarrow p \leq 0.001$, *** $\Rightarrow p \leq 0.01$, and * $\Rightarrow p \leq 0.1$ where p is the p-value for the coefficient estimate.

Table 3.B.4: Reduced Form Equations (3.7) and (3.8) – Effect of Project Features on Dependency Formation

tab:reduced-dep-formation

	# of Upstream Dependencies				# of Downstream Dependents			
	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
Constant	0.720***				4.720***			
	(0.121)				(0.147)			
Project Commits	-0.000	-0.000	0.000	-0.000	0.001	0.000	0.001	0.001
	(0.027)	(2.493)	(0.126)	(0.000)	(0.038)	(0.001)	(0.210)	(0.001)
Upstream Commits	-0.000	-0.000	-0.000	-0.000	0.000	0.000**	0.000	0.000**
	(0.001)	(0.205)	(0.028)	(0.000)	(0.017)	(0.000)	(0.038)	(0.000)
Project Quality	3.073***	4.898***	5.180***	4.306*	3.935***	7.444	5.850***	5.350
	(0.029)	(0.255)	(0.002)	(1.986)	(0.098)	(4.481)	(0.044)	(3.444)
Upstream Quality	3.000***	1.227***	2.818***	1.176***	-1.683***	-0.313***	-1.585***	-0.327***
	(0.005)	(0.037)	(0.030)	(0.222)	(0.012)	(0.089)	(0.035)	(0.080)
# of Contributors	-0.000	0.000	-0.001	0.002	-0.009***	-0.002	-0.008	-0.004
	(0.001)	(0.019)	(0.000)	(0.001)	(0.003)	(0.003)	(0.005)	(0.004)
Bus Factor	-0.181***	0.078***	-0.227***	0.215	-1.098***	0.595	-0.787***	1.040*
	(0.010)	(0.003)	(0.000)	(0.243)	(0.018)	(0.667)	(0.001)	(0.526)
SLOC	-0.000	-0.000	-0.000	0.000	-0.000	-0.000	-0.000	-0.000*
	(0.004)	(0.037)	(0.005)	(0.000)	(0.016)	(0.000)	(0.001)	(0.000)
Documentation	-0.941***	-0.979***	-1.170***	-0.873	4.325***	2.607*	4.640***	2.513***
	(0.058)	(0.042)	(0.001)	(0.544)	(0.124)	(1.047)	(0.005)	(0.697)
Modularity	-0.000	-0.000	-0.000	-0.000	0.000	0.000	-0.000	0.000
	(0.005)	(0.010)	(0.002)	(0.000)	(0.012)	(0.000)	(0.009)	(0.000)
# of Languages	-0.055***	-0.036	-0.061	0.002	-0.575***	-0.272	-0.427**	-0.280
	(0.002)	(0.693)	(0.063)	(0.001)	(0.005)	(0.217)	(0.156)	(0.166)
Age	-0.000	-0.000	-0.000	-0.000	0.002	0.002***	0.004	-0.056
	(0.002)	(0.004)	(0.001)	(0.000)	(0.004)	(0.000)	(0.003)	(0.067)
Project FE		✓		✓		✓		✓
Time FE			✓	✓			✓	✓
R^2	0.535	0.875	0.622	0.899	0.143	0.946	0.384	0.965
Observations	161,211	161,183	161,211	161,175	161,211	161,183	161,211	161,275

Note: Columns (1) through (4) report coefficient estimates for the specification in Equation (3.7), a regression of the number of upstream dependencies for a given project (i.e. out-degree) on observable project characteristics. Columns (5) through (8) report coefficient estimates for the specification in Equation (3.8), a regression of the number of downstream dependents for a given project (i.e. in-degree) on observable project characteristics.

Table 3.B.5: Counterfactual Analysis

tab:counterfactual-results

Counterfactual	Details	Welfare (Δ%)	Project Quality (Δ%)	Contribution (Δ%)	Costs (Δ%)
	Minimize Risk Aversion ($r_i' = \min(r_i)\,\forall i \in \mathcal{N}$)	0.22	0.40	0.00	0.00
$r \to r'$	Increase risk aversion ($r_i' = r_i + \sigma_r\,\forall i \in \mathcal{N}$)	-0.07	-0.09	0.00	0.00
	Maximize Risk Aversion ($r_i' = 1\,\forall i \in \mathcal{N}$)	-0.04	2.11	0.00	0.00
	Reduce quality volatility ($\Sigma' = 0.5\Sigma$)	0.00	0.19	0.00	0.00
$\Sigma \to \Sigma'$	Increase quality volatility ($\Sigma' = 2\Sigma$)	0.00	1.51	0.00	0.00
	Increase quality volatility ($\Sigma' = 4\Sigma$)	0.05	1.14	0.00	0.00
	Remove top package (Katz-Bonacich)	0.00	0.03	0.00	0.00
Key Package Analysis	Remove top 10 packages (Katz-Bonacich)	-0.07	-5.73	-1.30	-0.87
	Remove top 10 packages (Betweenness)	-0.08	0.57	-0.07	0.00

Note:

3.C Mathematical Details

app:dsc-math

3.C.1 Alternative Representations for the Maintainer's Problem

app:maintainers-problem

We note that the maintainer's cost minimization problem presented in Equation 3.13 can have alternative representations that under certain conditions, can deliver an equivalent set of equilibria. Alternative representations can be helpful during estimation. First, the cost minimaztion problem of Equation 3.13 is presented here in Equation P1:

$$\begin{aligned} \min_{x_i \geq 0, y_i, \{G_{ij}\}_{j \neq i}} \quad & c_i(x, G) \\ \text{s.t.} \quad & u_i(x, y, G) \geq \underline{u}_i \end{aligned} \tag{P1}$$

eq:app-cost-min

Alternatively, a project maintainer could equivalently modelled as maximizing expected project quality subject to a cost constraint.

$$\begin{aligned} \max_{x_i \geq 0, y_i, \{G_{ij}\}_{j \neq i}} \quad & u_i(x, y, G) \\ \text{s.t.} \quad & c_i(x, G) \leq \omega_i \end{aligned} \tag{P2}$$

eq:app-util-max

Notice that by assumptions over u_i and c_i, both P1 and P2 are convex problems in x_i. Finally, the decision problem can be reframed to make it more amenable to established network formation estimation procedures (**hsieh2022structural**). Consider P3, in which contribution cost is embedded into the maintainer's utility function:

$$\max_{x_i \geq 0, y_i, \{G_{ij}\}_{j \neq i}} \quad \mathbb{E}\left[v_i\left(\gamma y_i(x, G) - c_i(x, G)\right)\right] \tag{P3}$$

eq:app-net-util-max

Note that the parameter γ can be interpreted as converting project quality into the value the maintainer places on the project in terms of time saved for some computing task. Hence $\gamma y_i - c_i$ is the net value of project i in measured in hours.

The following proposition establishes an equivalence between the solutions of P1, P2, and P3.

prop:equiv-problems

Proposition 1 (Equivalence between P1, P2, and P3)**.** *Assume $x_i^\star > 0$.*

1. *The solutions to P1 and P3 coincide if the minimum project quality threshold binds: $u_i(x, y, G) = \underline{u}_i$.*
2. *The solutions to P2 and P3 coincide if the contribution cost constraint binds: $c_i(x, G) = \omega_i$.*

Proof. The Lagrangian for P1

$$\mathcal{L}_i(x_i, G; \lambda_i) = c_i(x, G) + \lambda_i \left(\underline{u}_i - u_i(x, G)\right)$$

First Order Necessary Conditions (FONCS) for P1:

$$\begin{aligned}
\frac{\partial c_i}{\partial x_i} - \lambda_i \frac{\partial u_i}{\partial x_i} &\leq 0 \\
x_i \left(\frac{\partial c_i}{\partial x_i} - \lambda_i \frac{\partial u_i}{\partial x_i} \right) &= 0 \\
x_i &\geq 0 \\
u_i(x, G) &\geq \underline{u}_i \\
\lambda_i(\underline{u}_i - u_i(x, G)) &= 0 \\
\lambda_i &\geq 0
\end{aligned} \tag{3.21}$$

eq:foncs-cost-min

The Lagrangian for P2

$$\mathcal{L}_i(x_i, G; \mu_i) = -u_i(x_i, G) + \mu_i \left(c_i(x_i, G) - \omega_i\right)$$

First Order Necessary Conditions (FONCS) for P2:

$$\begin{aligned}
-\frac{\partial u_i}{\partial x_i} + \mu_i \frac{\partial c_i}{\partial x_i} &\leq 0 \\
x_i \left(-\frac{\partial u_i}{\partial x_i} + \mu_i \frac{\partial c_i}{\partial x_i}\right) &= 0 \\
x_i &\geq 0 \\
c_i(x_i, G) &\leq \omega_i \\
\mu_i(c_i(x_i, G) - \omega_i) &= 0 \\
\mu_i &\geq 0
\end{aligned} \tag{8.22}$$

eq:foncs-util-max

Finally, the FONCs for P3:

$$\begin{aligned}
-\gamma \frac{\partial y_i}{\partial x_i} + \frac{\partial c_i}{\partial x_i} &\leq 0 \\
x_i \left(-\gamma \frac{\partial y_i}{\partial x_i} + \frac{\partial c_i}{\partial x_i}\right) &= 0 \\
x_i &\geq 0
\end{aligned} \tag{8.23}$$

eq:foncs-net-util-max

If $x_i > 0$, then $\frac{\partial c_i}{\partial x_i} = \lambda_i \frac{\partial u_i}{\partial x_i}$, $\mu_i \frac{\partial c_i}{\partial x_i} = \frac{\partial u_i}{\partial x_i}$, and $\frac{\partial c_i}{\partial x_i} = \gamma \frac{\partial y_i}{\partial x_i}$ for P1, P2, and P3 respectively.

Case 1: Suppose the minimum utility threshold constraint from P1 binds. Then $\lambda_i > 0$ and since $\frac{\partial u_i}{\partial x_i} > 0$ under the exponential utility Assumption 4, then $\frac{\partial c_i}{\partial x_i}$. Then further assuming that $\gamma \frac{\partial y_i}{\partial x_i} > 0$, then the equilibrium solutions to P1 and P3 coincide if and only if for each $x_i^\star$

$$\frac{\partial c_i}{\partial x_i} = \lambda_i \frac{\partial u_i}{\partial x_i} = \gamma \frac{\partial y_i}{\partial x_i} > 0$$

Case 2: Suppose the contribution cost constraint in P2. Then $\mu_i > 0$. Similar to the argument in Case 1, the equilibria of P1 and P3 coincide if and only if for each $i \in \mathcal{N}$

$$\frac{\partial c_i}{\partial x_i} = \frac{1}{\mu_i}\frac{\partial u_i}{\partial x_i} = \gamma \frac{\partial y_i}{\partial x_i} > 0$$

■

Proposition 1 implies that in the edge case where both the cost and minimum quality constraints bind, all representations of the maintainer's problem result in the same solution and how the Lagrange multipliers of P1 and P2 correspond to the parameter γ in P3. We use the result of this proposition along with its assumptions to simplify estimation.

Remark (Equilibrium Contribution in Equation 3.14)**.** *The exposition in Proposition 1, although tedious and perhaps a bit excessive, makes clear the relationship between observed equilibria (e.g. $x_i > 0$ or $c_i(x, G) = \omega_i$) and the equilibrium conditions implied by each of the FONCs of P1, P2, and P3. The parameter a_i, representing maintainer i's intrinsic marginal cost of contribution, one would obtain from estimating Equation 3.14 does not exactly, equal a_i from Equation 3.10. Instead, for the purposes of estimation in Section 3.6.3, we assume that the cost constraint binds. Then by Proposition 1, we replace a_i in Equation 3.14 with $\tilde{a}_i = a_i + \gamma b_i$ for the purposes of estimation. This should have no affect on the exposition in Section* **??**.

3.C.2 Expected Project Quality

app:eu-details

Under Assumption (5), maintainer utility becomes

$$
\begin{aligned}
u_i(x^\star, y^\star, G) &= \mathbb{E}\left[v_i\left(y_i^\star\right)\right] \\
&= \mathbb{E}\left[-\exp\left(-r_i\left(\sum_j B_{ij}(b_j x_j^\star + \xi_j)\right)\right)\right] \\
&= -\exp\left(-r_i \sum_j B_{ij} b_j x_j^\star\right) \mathbb{E}\left[\exp\left(-r_i \sum_j B_{ij}\xi_j\right)\right] \\
&= -\exp\left(-r_i \sum_j B_{ij} b_j x_j^\star\right) \exp\left(\frac{r_i^2}{2}\sum_j B_{ij}^2 \sigma_j^2\right) \\
&= -\exp\left(-r_i \sum_j B_{ij}\left(b_j x_j^\star - \frac{r_i}{2} B_{ij}\sigma_j^2\right)\right)
\end{aligned}
\tag{3.24}
$$

eq:eu-details

The third line of (3.24) follows from Assumption 5 on maintainer information sets. By the normality of ξ established by Assumption (5), the fourth line of Equation (3.24) uses the moment generating function for a linear combination of the normally distributed random vector ξ.

3.C.3 Optimal Dependency Formation

app:details-link-likelihood

Following **fosgerau2009discrete**, let $\epsilon^+_{ij} = -\lambda \ln(\varepsilon^+_{ij})$ and $\epsilon^-_{ij} = -\lambda \ln(\varepsilon^-_{ij})$ where $\lambda > 0$. Under Assumption 7, we can show

$$\begin{aligned}
\Pr(G_{ij} = 1) &= \Pr\left(u^+_{ij}\varepsilon^+_{ij} \geq u^-_{ij}\varepsilon^-_{ij}\right) \\
&= \Pr\left(-\ln(-u^+_{ij}) - \ln(\varepsilon^+_{ij}) \geq -\ln(-u^-_{ij}) - \ln(\varepsilon^-_{ij})\right) \\
&= \Pr\left(\overline{u}^+_{ij} + \epsilon^+_{ij} \geq \overline{u}^-_{ij} + \epsilon^-_{ij}\right) \\
&= \Pr\left(u_{ij} \geq \epsilon_{ij}\right)
\end{aligned} \tag{3.25}$$

eq:linking-details1

where $(\overline{u}^+_{ij}, \overline{u}^-_{ij}) \equiv (-\lambda \ln(-u^+_{ij}), -\lambda \ln(-u^-_{ij}))$, $u_{ij} \equiv \overline{u}^+_{ij} - \overline{u}^-_{ij}$, and $\epsilon_{ij} \equiv \epsilon^-_{ij} - \epsilon^+_{ij}$. The advantage of this approach is that now the equilibrium link formation decision can be represented as a random utility with additive disturbances and is linear-quadratic in the parameter of interest, r_i. If we substitute the expression for expected project quality under both $G + ij$ and $G - ij$ from Equation (3.24) to form u^+_{ij} and u^-_{ij}, the likelihood that maintainer i imports project j in Equation (3.25) becomes

$$\begin{aligned}
\Pr(G_{ij} = 1) &= \Pr\left(-\ln\left(\mathbb{E}\left[\exp\left(-r_i y^+_{ij}\right)\right]\right) + \ln\left(\mathbb{E}\left[\exp\left(-r_i y^-_{ij}\right)\right]\right) \geq \epsilon_{ij}/\lambda\right) \\
&= \Pr\left(r_i\left(\sum_j \Delta B_{ij}\left(b_j\left(\sum_k \Delta A_{jk} a_k\right) - \frac{r_i}{2}\Delta B_{ij}\sigma^2_j\right)\right) \geq \epsilon_{ij}/\lambda\right) \\
&= F_\epsilon\left(r_i \underbrace{\left(\sum_j \Delta B_{ij} b_j \left(\sum_k \Delta A_{jk} a_k\right)\right)}_{Z_{0ij}} - \frac{1}{2} r_i^2 \underbrace{\left(\sum_j \Delta B^2_{ij}\sigma^2_j\right)}_{Z_{1ij}}; \theta_\epsilon\right)
\end{aligned} \tag{3.26}$$

eq:linking-details2

where $u_{ij} \equiv \overline{u}_{ij}^{+} - \overline{u}_{ij}^{-}$, $\epsilon_{ij} \equiv \epsilon_{ij}^{-} - \epsilon_{ij}^{+}$, $y_{ij}^{+} \equiv y_i(x, y_{-i}, G+ij)$ and $y_{ij}^{-} \equiv y_i(x, y_{-i}, G-ij)$. Finally, we define $\Delta B_{ij} \equiv B_{ij}^{+} - B_{ij}^{-}$ as the difference between elements of the Leontief inverse matrices for project quality under $G + ij$ and $G - ij$: $B + ij = [B_{ij}^{+}]_{i,j \in \mathcal{N}} = (I - \beta(G + ij))^{-1}$ and $B - ij = [B_{ij}^{-}]_{i,j \in \mathcal{N}} = (I - \beta(G - ij))^{-1}$. Equivalently, we define $\Delta A_{jk} = A_{jk}^{+} - A_{jk}^{-}$ for the Leontief inverse matrices for project contribution: $A + ij = [A_{jk}^{+}]_{j,k \in \mathcal{N}} = (I - \alpha(G + ij))^{-1}$ and $A - ij = [A_{jk}^{-}]_{j,k \in \mathcal{N}} = (I - \alpha(G - ij))^{-1}$.

For notational convenience, in the last equality of Equation (3.26), we rearrange u_{ij} into a quadratic function of r_i and label the coefficients Z_{0ij} and Z_{1ij}. These coefficients are functions of the network G and parameters $(\alpha, a, \beta, \Sigma)$. In Section 3.6.3 and Appendix **??**, we show that since $(\alpha, a, \beta, \Sigma)$ can be estimated using moment conditions (3.9) and (3.14) via GMM, the result in Equation (3.26) will form the basis for a likelihood function for the remaining unknown parameters, r and θ_ϵ. Hence under the assumptions outlined in the body of the paper, r and θ_ϵ can be estimated via maximum likelihood. We will assume that ϵ_{ij} is a logistic random variable, independent and identically distributed across potential links. This can arise if ϵ_{ij}^{+} and ϵ_{ij}^{-} are independent Gumbel random variables. Therefore $F_\epsilon(\cdot)$ is the logistic function which has the convenient property that $F_\epsilon'(\cdot) = F_\epsilon(\cdot)(1 - F_\epsilon(\cdot))$. Furthermore, θ_ϵ is a vector of two parameters for a logistic distribution (i.e. location and scale). We make no specific assumption on the value of the free parameter λ other than $\lambda > 0$. Therefore the value of λ will simply influence the estimated scale parameter for the distribution of ϵ_{ij}.

3.D Estimation Details

app:dsc-estimation

Data is $\mathcal{D} = (x_t, y_t, G_t, W_t)_{t\in\mathcal{T}}$. Parameters are $\theta = (a, \alpha, b, \beta, \Sigma, r, \gamma, \theta_\epsilon)$.

1. Estimate b, β given $\mathcal{D}$ using the project quality specification Equation (3.9) in separate OLS regressions for each project $i \in \mathcal{N}$.

$$y_{it} = b_i x_{it} + \beta \sum_{j\neq i} G_{ijt} y_{jt} + \tilde{\xi}_{it}$$

   where $\tilde{\xi}_{it} = \delta' W_{ijt} + \xi_{it}$ is an are other influences partitioned in observable $\delta' W_{ijt}$ and unobservable x_{it} components. Furthermore, we can use the residuals of each OLS regression to estimate Σ. Therefore the structural estimation of b, β is equivalent to the reduced form estimation of project quality influences in Equation 3.6. Furthermore, our consideration of Heterogeneity beyond the framework of the model matches the structural estimation approach outlined by **hsieh2022structural**. In our setup this may allow us to control for technical aspects of projects that at least partially determine quality or fixed costs of project contribution that are absent from our structural discussion in Section **??**.

2. Estimate a, α, γ given $\mathcal{D}$ and estimates for b using Proposition 1 and a modified version of the project contribution specification in Equation 3.14 using OLS regressions for each project

$$\begin{aligned} x_{it} &= \tilde{a}_i + \alpha \sum_{j\neq i} G_{ijt} x_{jt} + \tilde{\nu}_{it} \\ &= a_i + \gamma b_i + \alpha \sum_{j\neq i} G_{ijt} x_{jt} + \tilde{\nu}_{it} \end{aligned}$$

where, as before, $\tilde{\nu}_{it} = \delta' W_{ijt} + \nu_{it}$ and ν_{it} is independent and identically distributed and mean zero in expectation.

3. (Optional) If we assume that contribution costs $c_{it}(x, G)$ exactly equal, conditional on some noise or measurement error d_{it}, estimates of time allocation ω_{it}, we can estimate fixed costs of contribution for each project $i \in \mathcal{N}$ using the following specifications to back out a residual d_{it}:

$$d_{it} = \omega_{it} - \frac{1}{2}x_{it}^2 + a_i x_{it} + \alpha \sum_{j \neq i} G_{ijt} x_{jt}$$

where ω, x, G are observed in $\mathcal{D}$ and a, α were recovered in previous steps. Taking the average of for each project gives an estimate of the fixed costs of contribution: $\overline{d}_{it} = \frac{1}{|T_i|} \sum d_{it}$ where T_i is defined as the number of time periods in which project i appears in the empirical sample. These estimates will help refine the estimation of welfare effects under counterfactual analysis.

While we suggest that the parameters in 1-3 above can be estimated with simple OLS, it might be more prudent to organize Equation (3.15), Equation (3.14), and Equation (**??**) described above into a set of moment conditions ans subsequently estimate $(a, \alpha, b, \beta, \Sigma, \gamma, d)$ using the generalized method of moments (GMM) with constraints: $\alpha, \beta \in (-1, 1)$, $\sigma_i > 0$

4. Estimate r, θ_ϵ by means of MLE, maximizing a likelihood function based on equilibrium link formation described in Equation (3.18).

$$L(\theta \mid \mathcal{D}) = \prod_{t\in\mathcal{T}} \Pr(\mathcal{D}_t | \mathcal{D}_{t-1}, \theta) = \prod_t \prod_{j\neq i} \Pr(G_{ij} = 1 \mid \mathcal{D}_{t-1}, \theta) \tag{3.21}$$
$$= \prod_t \prod_{j\neq i} F_\epsilon \left(u_{ijt}\right)^{G_{ijt}} \left(1 - F_\epsilon(u_{ijt})\right)^{1-G_{ijt}}$$

eq:likelihood

As mentioned previously, this likelihood function forms a Markov chain of likelihoods for the observed sequence of linking decisions. MLE estimates for r and θ_ϵ minimize $-\ln L(\theta \mid \mathcal{D})$.

3.D.1 Additional simplifications to reduce computational burden

app:estimation-more

Given the size of our empirical sample, the estimation procedure as specified remains a time intensive task on available hardware. In conjunction with the dimensionality reduction we discuss in Section 3.6.3.1, we take a few computation shortcuts to calculate the coefficients Z_{0ijt} and Z_{ijt} for each sample moment $t \in \mathcal{T}$, the project $i \in \mathcal{N}_t$ associated with that particular moment t and each potential dependency $j \neq i \in \mathcal{N}_t$. First, we approximate the true matrix inverse using $A = I + \sum_{k=1}^{K} \alpha^k G^k$ where $K = 5$ provides a decent approximation. Second, following the suggestion of **hsieh2022structural**, we use the Sherman-Morrison formula to efficiently calculate a new proposal Leontief inverse A or B to calculate ΔA_{ij} and ΔB_{ij}. Third, instead of considering proposal links for all $j \neq i$ in the current network, we consider a set of randomly selected potential dependencies from $\mathcal{N}_t$ that is (1) equal in size to the set of current dependencies for project i and (2) contains potential dependencies not in the current set of dependencies for project i. Finally, instead of using all sample

Bibliography