Success and geography in the weightless economy: Evidence from open-source software*

Gábor Békés[†], Julian Hinz[‡], Miklós Koren[§], and Aaron Lohmann[¶]

28 February 2024. Preliminary version 0.2.

Abstract

Open source software (OSS) is a global, vast in scale, and extremely successful industry with millions of developers and millions of packages produced and with almost all software including OSS code. How can groups of independent and geographically dispersed producers work together and create globally used products? This papers describes the relationship between product success (adoption of software) and the geographical dispersion of its producers. Focusing on the largest language, JavaScript, we analyze contributions by 217 thousand developers in 300 thousand projects with 78 thousand unique projects imported as intermediate goods (called a dependency). We find that while OSS developers will collaborate more intensively when in close proximity, more widely adopted software packages are written by a more spatially diverse group of developers. We show that this is not driven by the geography of adoption, as developers only marginally prefer locally written packages as intermediate input (dependency). It is not driven by talent distribution across cities either or self-selection of best coders into best project. One explanation is selection of coder pairs into better projects.

Keywords: Open-source Software, Trade Costs, Collaboration, Global production networks, international sourcing
JEL Classification: F10, F14, L86, R12, O33

*This work was funded by the European Union under the Horizon Europe grant 101061123. Views and opinions expressed are, however, those of the author(s) only and do not necessarily reflect those of the European Union or the European Commission. Neither the European Union nor the granting authority can be held responsible for them.

[†]Central European University, KRTK and CEPR.
[‡]Bielefeld University and Kiel Institute for the World Economy.
[§]Central European University, KRTK, CEPR and Cesifo.
[¶]University Bielefeld and Kiel Institute for the World Economy.

1 Introduction

In this study, we explore the role of geographical distance in the formation and success of global collaboration teams within the context of open source software development. While digital goods are often perceived as being spatially unbound, our analysis investigates how geography still plays a significant role in shaping collaboration patterns and dependencies among software projects. We aim to shed light on the underlying mechanisms and determinants of these production networks, which have important implications for innovation and knowledge diffusion.

Software is everywhere in our digital world and open source software is key part of the industry. Open source software (OSS) refers to software packages shared freely with the public — not only for end users, but also for modification and reuse by other developers. OSS mostly free, but present in fee-based platforms such as Overleaf.

Open Source Software (OSS) has a vast landscape, `GitHub` hosts over 330 million repositories. It plays an important roles in Websites (JavaScript), Operating systems (Linux, Android), Data Science and analysis (R Tidyverse, Python Pandas, Julia), Machine Learning and AI (PyTorch, LLaMA). It is also a very important sector of the global economy. The software industry as a whole represents about 1 % of global GDP, and open source software is included in 98% of software and a large majority of all code written across all industries is in fact open source code.[1] Recently, Hoffmann et al. (2024), estimated the user value of open source to reach striking $\$8.8$ trillion globally.

Open source software are created by mostly independent developers. Their work is driven by an OSS ethos do do good in the community, create value for people. The intrinsic motivation of being helpful and self-accomplished plays a crucial role. Developers are drawing utility from the creation of useful code. Community recognition, future career concerns via gaining popularity is also important. Open source is also a method to create great software via collaboration and bugfixing – as discussed in our companion paper, Békés et al. (2023). In OSS developments, developers can learn from each other, review code, and collectively improve software quality. [2]

Because OSS allows for sharing and modification, most OSS packages are developed by multiple developers in collaboration. Indeed, a large share of OSS is produced by teams of developers who may be scattered around the globe. This is especially true for popular

[1] See https://www.synopsys.com/software-integrity/resources/analyst-reports/open-source-security-risk-analysis.html.

[2] Sources: Bing chat, `freecodecamp.org/news/what-is-great-about-developing-open-source-and-what-is-not`, `bcg.com/publications/2021/open-source-software-strategy-benefits`, `openaccessgovernment.org/open-source-technology/129261/`, `linux.com/news/why-do-programmers-write-open-source-software/`, `stratoflow.com/benefits-of-open-source-software/`

packages. In our data, we'll show developers from 4000 cities, collaborating on creating software, sometimes used by millions.

This paper focus on the dispersed aspect of development. How can geographically dispersed workers create great products? What is the impact of this dispersion on the quality of output?

Using data from the OSS sector to learn about spatial patterns of (joint) production has the benefit that its very nature — being open source — means it can be observed in great detail. OSS is shared on open platforms, such as GitHub,[3] where we can directly observe who works with whom, in what intensity, and how different pieces of software are used in the functionality of each other.

We start by looking at collaboration patterns in OSS asking how does geography affect the formation of such teams. Using a gravity model at city-pair level, we find the localization is very important, being in the same city or region greatly increases collaboration. This suggests the presence of search and matching costs. Then we consider success – popularity of packages measured as the number of users. We find that more dispersed teams create more successful software. We show that this is not driven by reverse causality or best coders self-selecting into good projects.

The remainder of this paper is structured as follows. We start by introducing open source software in section 2, followed by related literature in section 3. We describe data sources and key variables in section 4 and present some nice pictures about the geography of OSS in section 5. The two empirical parts presents evidence on gravity (section 6) and success (section 7). The paper ends with some discussion (section 8).

2 Open source software

OSS development, characterized by its collaborative and decentralized nature, has arguably revolutionized the software industry by enabling knowledge sharing, innovation, and community-driven software development.

The very nature of open source software makes it easy to study the inner workings of the software industry.

First, all source code is shared openly, including references to other software used in the development of a particular program. Second, the development of open source software is often managed by a version control system to ease collaboration among many contributors. The most popular version control system is called `git` and was invented for

[3] GitHub is a platform that fosters collaboration and innovation by allowing developers to host and review code, manage projects, and build software alongside millions of other developers.

the development of the Linux kernel. It allows tracking small changes to the code base, called “commits”, and attribute them to specific developers.

Our unit of observation is a project set up by a developer to solve a problem. A project is organized in a single repository. It often contains a single package but the project may be organized in multiple packages (scripts). In this paper, ’project’ and ’repo’ will be used interchangeably, repos referring to observations in the data.

In OSS, the *collaborative production* is thus writing code together, and the task is to find fellow developers. *Importing intermediate goods* is about finding software packages that can perform the necessary tasks instead of starting own code with the same functionality from scratch. With zero transport and “commuting” costs, we can focus on search, adaptation, and maintenance costs and understand the role of spatial frictions better.

The most frequent form of software reuse involves developers “importing” code written by others into their own software package. This creates a so-called “dependency”: the using software depends on the imported one and may partially or fully use its codebase.

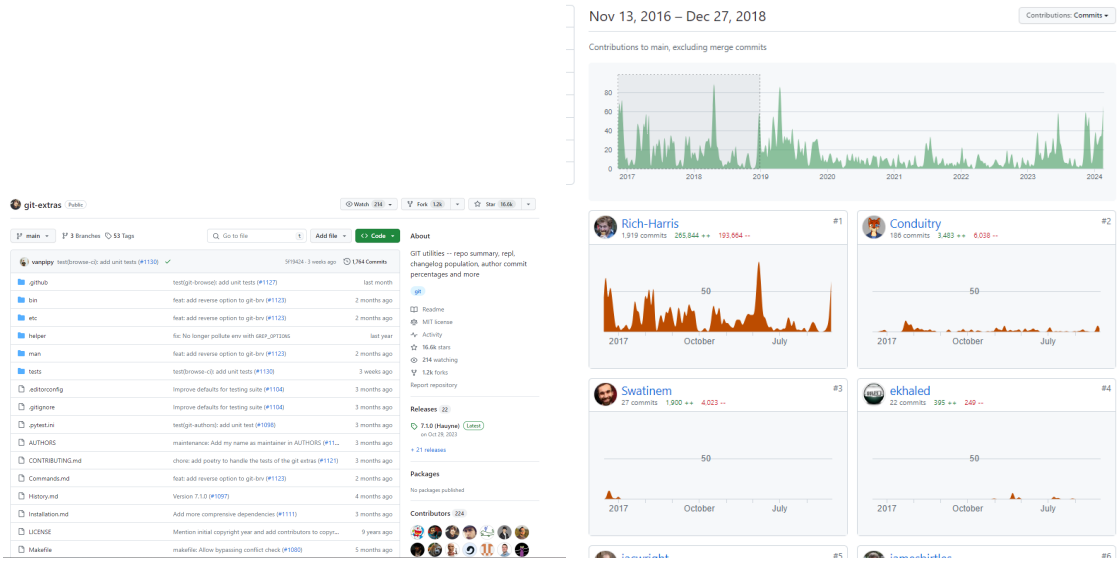
We hence have a unique view into both the “intermediate inputs” used in the production of software, as well as the own value added used to produce it. It is common for open source projects to rely on each others’ functionality, with so called “dependencies”. That is, a software project may import all or part of the functionalities another project has. We aim to capture this in a production function with team collaboration and network links between projects.[4]

Our focus in this paper is *JavaScript*, the most popular language for OSS. It has open source libraries that are essential for the operation of the World Wide Web. For instance, Node.js is a cross-platform, server environment that can run on many different OS, including Windows or Linux. It has over three thousand contributors, and over 300 releases. As of August 2022, jQuery another library, is used by 77% of the 10 million most popular websites[5]. Apart from other open source packages using it, companies like Microsoft rely on them (e.g. for Visual Studio).

Finding partners and intermediate products is a difficult task, however, as the pool of potential co-workers and possible dependencies is huge: Globally, there are up to 27 million programmers[6], and the pool of packages is also large, e.g. with approximately 1.3 million packages[7] for the JavaScript language alone.

Developers collaborate online on software development platforms. Github is a free platform to build software packages, and allows tracking the process as shwon in 1.

[4]More background on open source software may be found in the A.1 section of the Appendix.
[5]`https://en.wikipedia.org/wiki/JQuery`, `https://en.wikipedia.org/wiki/Node.js`
[6]Source: `https://qubit-labs.com/how-many-programmers-in-the-world/`
[7]Source: `https://en.wikipedia.org/wiki/Npm`



(a) GitHub Collaboration **(b)** Tracking of user activity (2017-2019)

Figure 1: Collaboration is done mostly online

Notes: Two screenshots of GitHub repositories. In Panel (a) displayed the a repository with the corresponding files, some repository level information and crucially a list of contributors. In panel (b) contributor level activity graphs based on the amount of commits per user in a given repository.

While OSS development is done in collaborative teams mostly online, finding partners remain partially in the real world. These interactions may take place in a variety of settings, from workplace gatherings at institutions like CEU and companies like Oracle to local community events hosted by organizations such as Xaccelerator. In addition, regional meetups and groups like R Ladies Auckland and PyData Berlin provide platforms for developers to connect and share their work within their local spheres.

Conferences play a pivotal role in the open source community, offering numerous opportunities each month for developers to convene and discuss new technologies. Examples include CityJS in Berlin and React Summit in the US. More specialized events like Node-js fwdays23 in Kyiv allow developers to present and learn about the latest packages. Alongside these in-person events, online forums, Stack Overflow, Reddit, and Twitter serve as virtual spaces where developers can learn about new packages and connect with others in the field.

Open source software vs patents and academia R&D and patenting

- Need machines, secrecy, often top-down
- Distance matters in collaboration
- More cited patents – geographically focused authors

Science (math, academic papers)

- Similar, but often longer projects, not open, F2F important to think and discuss
- Distance matters in collaboration
- Major role of top Universities / Centers

The development of open-source software (OSS) and other knowledge-intensive goods, such as patents and scientific research, share notable similarities. Collaboration among research teams or within general work teams is prevalent across all three domains. Additionally, citation patterns in science or patents can be analogized to dependencies in software development.

However, significant differences exist, particularly regarding the nature of dependencies and citations. Dependencies in software development are direct observations, unaffected by personal relationships or biases, unlike citations, which in scientific research or patents may be influenced by affiliations or social connections.

Moreover, patents and scientific research tend to exhibit geographical clustering, indicating regional concentrations of innovation. Similarly, citation patterns in these fields also demonstrate clustering, suggesting thematic or disciplinary groupings within the broader knowledge landscape.

3 Related literature

This paper is related to multiple strands in the economic literature.

First, this work relates to the broader trade literature which makes the role of geographical distance specific in their models. Those models are typically models on network formation and accordingly arddess the extensive margin of trade. The extensive margin of trade is interpreted as the outcome of a network formation model.

Chaney (2014) proposes a dynamic model to study the formation of a social network of importers and exporters. Firms may either meet at random or through the already existing network. When using the existing already existing network, firms use there already established foreign contacts to meet close - that is geographically close - firms. The model is estimated and supported by data using French exports in the years 1986 till 1992.

Another approach to network formation is proposed by Bernard et al. (2019). Using data for detailed Japanese Buyer-Supplier relationships, they argue that geographical distance plays an important role in shaping this network. To further validate their findings they use the opening of a passenger-only train line which allows for easier social network formation while leaving costs for shipment untouched. The results support the idea that search costs are significant for describing in this case a domestic trade network.

Again, using Japanese data Bernard et al. (2020) now explore the role of social networks

for the formation of research teams which are working on patents. Using the building of bridges to lower costs of meeting and face to face contact. The findings suggest that people who were strongly affected by this infrastructure improvement had the biggest effects on the production of knowledge.

The above papers are only a selection of contributions which show that it is important to study the formation of social networks and their role in the economy.

Our work is related to understanding the geography of services. Head et al. (2009) estimate gravity equations for services trade in the period from 1992 till 2006 and find strong negative distance effects across sectors (also see Walsh, 2008)[8].

Within services, The development of OSS is a knowledge intensive task of typically highly skilled individuals. Thus, our work also relates to the work on geographical aspects of knowledge production as is common in the study of academic research teams or inventor groups working on patents. Lychagin et al. (2016) analyze US firm productivity spillovers and R&D activities, and find that the exact geographic location of a firm's researchers matters for knowledge spillovers. The aspect though we focus on is the citations of research papers or patents. In OSS we observe the dependencies which can plausibly interpreted as related to citations. Starting with the seminal paper Jaffe et al. (1993) they argue that following citations is an important dimension to understand knowledge production. They provide early evidence that citations are often geographically close. This result has been confirmed by many other papers over the course of time. In terms of patents,

Another related field is academia. Head et al. (2019) show that distance also matters here. Though, they also find that ties that bind - past colocation, coauthership, university links - strongly reduce the distance friction suggesting a big informational friction channel. The results are driven strongly by lower quality research papers , i.e. those for which it is harder to hear randomly about. Taste heterogeneity will also play a role shaping geography and trade. Blum and Goldfarb (2006) studying the role of distance in website traffic argue that that distance effects typically occur for taste dependent goods.

In terms of collaboration, a paper closely related to us AlShebli et al. (2018) who study the success of academic papers as a function of *ethnic* diversity of authors, and find a positive correlation. At the same time negative correlation between diversity and intensity of collaboration is found by Békés and Ottaviano (2022) in a very different setting: professional soccer.

More recently, a surge of papers is using OSS data and content with related questions. For example Wachs et al. (2022) show that OSS contributors - even though most work can be conducted entirely remote - often cluster in locations like Berlin, New York or Tokyo. The idea of clusters is also explored by Abou El-Komboz et al. (2022) who

[8]For more on services trade and its link to goods trade, see Ariu et al. (2019) and Breinlich et al. (2018).

study the effect of cluster on single developers productivity. They propose to use the amount of commits of a developer as a measure of productivity and find positive effects for the effects of clusters on productivity. Moving beyond clusters and appreciating the whole distance distribution is a stream of the literature which looks at gravity-type regressions. For instance, Laurentsyeva (2019) argue that distance in different non-parametric specifications negatively affects collaboration. Common language on the other hand is important for collaboration. Using also a gravity-type regression Goldbeck (2022) uses a sample for developers living in the United States and shows that co-located developers work more intensely together. In this vain another branch of literature aims to quanitify the value of OSS as for example Hoffmann et al. (2024) do. Chelkowski et al. (2021) talk about the role of large organizations after analyzing activity in 1300 projects for 21 years. They also argue for an important role in fostering collaboration. They argue that as projects mature, organizations as intermediaries are less needed.

4 Data sources, variables

Let us start with identifying objects in the data.

Developers

In our analysis, we model three entities. Individual software developers ("users" or "contributors") are identified as individual users of GitHub. Because GitHub identifies its users with short, unique user IDs that are persistent over time, this is a good way to identify individual persons working on GitHub.[9]

Repositories

A software project typically has a single page on GitHub, called a "repository." This is where source code is shared and bugs and other issues are discussed. Even though each project will have many versions over time, we treat a repository as the unit of analysis for software.

Organizations

The third entity is the "organization." This is a permission management feature of GitHub. Users can create an organization and invite other users to it. This helps manage several repositories together.

Github users can be part of organisations which might be aligned with companies or can simply be loosely defined interest groups. Of our sample of $320,721$ developers $65,750$ share at least one organisation with some other developer in the sample. This

[9]There is some, but very limited activity on GitHub done by bots. These are automated steps in the software development process that can be done by machines rather than humans. We exclude the user accounts corresponding to bots.

yields $2,553,882$ individual developer-developer links which are also part of the same organisation. Each developer may belong to multiple organisations.

Organisations may correspond to real-world businesses (e.g. Microsoft, or Posit/RStudio or Overleaf are GitHub organizations), or other organizations that aim at coordinating the work of many contributors to related projects (such as the Tidyverse organization hosting several R packages, or CEU Econ department hosting several course projects).

For most exercises, we'll omit them to avoid collaboration that are promoted by a company.

In what follows we describe our data sources.

Libraries.io data

Our first used data source is `Libraries.io`.[10] The website's main goal is to collect and track all dependencies of repositories hosted on one of 32 popular package managers, software that manages code dependencies. The website collects metadata on published versions of tracked code, the respective `GitHub` repositories and, most importantly, on the inter-dependencies between projects.

In the following, we focus our analysis on code from a single language called `JavaScript` and its dominant package manager `NPM`.[11] `JavaScript` is one of the fundamental technologies of the web and thus the language with the most projects in the `libraries.io` database.[12]

It is very common for JavaScript developers to import someone else's software. NPM eases the discovery and installation of other JS software and records metadata about which software uses which other one. It is this metadata that libraries.io collects and we use in our analysis.

GHTorrent data

To map collaboration in software production, our second set of data comes from `GHtorrent` (see Gousios and Spinellis, 2012), a data collection effort which stores large chunks of GitHub activity data. Recorded activities are — among others — commits, issues and pull requests, i.e. code contributions, questions and change suggestions. Furthermore, some characteristics of single projects, called "repositories", and users are stored. Data coverage ends in 2019. We use the latest data dump from June 2019 which is the latest in the regular uploading schedule. While data on user characteristics like the location is specifc to the point in time it was recorded, the contributions to projects also covers past contributions and is time stamped.

[10] `Libraries.io` last accessed on 07/07/2023

[11] `NPM` itself offers data on the download statistics of individual `JavaScript` projects which are hosted on `NPM`.

[12] JavaScript, often abbreviated with JS, is used both on the client side — rendered in web browsers to display websites — as well as on the server side — to host the website.

Note that the described open source code is hosted on GitHub. The website offers additional data on (self-reported) user location, identifies contributors to specific software projects, and allows us to measure the extent to which individual users contribute to a specific package.

Combining the different data sources, we can track the development of open source software of 217,618 developers ("contributors") and 298,973 software projects.

Project complexity

Additional aspects relevant to our analysis are the complexity of repositories as well as user characteristics which may define certain group memberships. It is common in the firm level economic literature to differentiate between different firms based on size or productivity metrics. Here, we achieve something similar by considering the complexity of a repository. To proxy this, we count the amount of commits and perform the our analysis on different cuts of the sample. Considering all $517,779$ repositories in our sample, we find that the average repository has 13, the 90% quantile is at 91 commits while the five repositories with the most commits have above $80,000$ commits.

4.1 Data processing

This subsection outlines the procedures for creating variables from the initial raw data sources mentioned earlier.

Libraries.io

We obtain the most recent data dump of Libraries.io, which was released in January 2020. From this we filter out projects hosted on NPM and their versions. Versions in software developments should be understood as iterations of the same software project. Each software project might release multiple versions each year. Therefore, we consider only the latest version published in 2019 for each project. First, this ensures that no projects are double counted and gives us a sample of projects that were actively maintained in 2019. For this sample, we collect all the projects on which these projects depend. Note that these dependencies might have been established prior to 2019 and might also include projects without a version update in 2019. The last step involves obtaining repository URLs for projects that are available on Github as for those we can obtain more detailed repository level data.

Finally, this first data processing step yields a sample of actively maintained JavaScript projects in 2019, including their dependencies as well as Github address.

GHtorrent

The next step is to assemble more detailed information about the developers of the respective software projects. To do this, we consider GHtorrent and here download the

Table 1: How many coders develop a package?

Number of developers	Share of Repositories
1	83.39%
2	9.72%
3 - 5	5.05%
6 - 10	1.26%
11-100	0.56%
100+	0.01%

Notes: Based on the full sample of repositories, the amount of developers per repository which we identify in the GHtorrent data.

latest available dump in the regular uploading cycle which is from June 2019. These data contain information on users, projects, and the extent to which such users are active in each project. We filter the GHtorrent data to consider only such repositories which have been identified as relevant from Libraries.io either as dependent or as dependency. For those projects, we find the developers who have committed to these projects and aggregate the data over the available time horizon. The GHtorrent is cumulative in the sense that the dump from 2019 also contains data from 2018 and previous years. Therefore, our unit of observation is then a contributor-repository pair and the sum of commits until June 2019 to this project. An additional key variable is the user-reported location data which is available for ca. 30% of developers. These self-reported location data are used to allocate developers to cities. Furthermore, we capture in which `Github`-organisations each user is a part of. Organizations on `Github` may capture firm compositions, research clusters, or just a loosely connected group of people with similar interests.

Sample design

We focus on collaborating partners, who are likely to make joint decision and can be understood as core members. Therefore, we exclude developers which we label *bug-fixers*. One interpetation of these bug-fixers are external consultants which work on a project for a very limited amount of time and help solve a particular part. Instead we want to capture the core team. Thus, we exclude developers in project where they do less than 4 commits or have only 1% of the commits.

Further we only focus on developers who are part of the core team and have committed in the first 2 years to the project.

4.2 Descriptive statistics

Let us start with developers. We aggregated data at the repository (project level). As Table 1 suggests 83 % of projects have only a single developer. After this the amount of developers is steeply declining.

We observe the city locations of developers. Therefore, we look next at the number of

Table 2: In how many cities are developers in a project located?

Number of cities per repository	Share of Repositories
1	38.31%
2	38.20%
3 - 5	17.98%
6 - 10	3.89%
11-100	1.57%
100+	0.03%

Table 3: How many repos developers are involved in?

Number of repositories per developer	Share of developers
1	62.62%
2	15.61%
3 - 5	13.13%
6 - 10	4.82%
11-100	3.68%
100+	0.11%

Notes: Within our sample the amount of repositories each contributor has sent at least one commit to.

cities which we can identify in each project (see 2). Naturally, the share of projects for which we can only identify one city is very high with 88 %. The fact that this is higher than the share of single developed repository is consistent with the idea that within cities developers will work on some of the same projects. Still a non-trivial share of projects are developed in multiple cities.

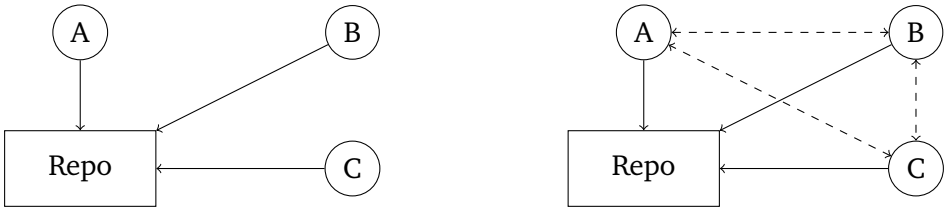
Next we map out how many repostirories each contributor works in. We see that a good third of the developers only contribute to one project. Though, some of the contributors work in many projects at the same time with a few contributing to over a hundred.

5 Description of the geography

Typically, developers will report their location not finer than the city level. Therefore, the meaningful variation of distance and agglomeration metrics will occur at the city level which is why we aggregate the above variables to the city level. Formal descriptions of this aggregation can also be found in **??**. Here, follows a verbal description of the city level aggregation which is used to estimate gravity equations later on.

5.1 Aggregation to city level

Typically, developers will report their location not finer than the city level. Therefore, the meaningful variation of distance and agglomeration metrics will occur at the city level which is why we aggregate the above variables to the city level. Formal descriptions of this



(a) Contribution: Observables. **(b)** Contribution: Inferred links.

Notes: Stylized figures depicting observables and inferred links. In panel (a) what is observable in the data: Developers sending commits to a repository. In panel (b) what we infer, resulting in undericted links between developers.

aggregation can also be found in **??**. Here, follows a verbal description of the city level aggregation which is used to estimate gravity equations later on.

Aggregating contributions

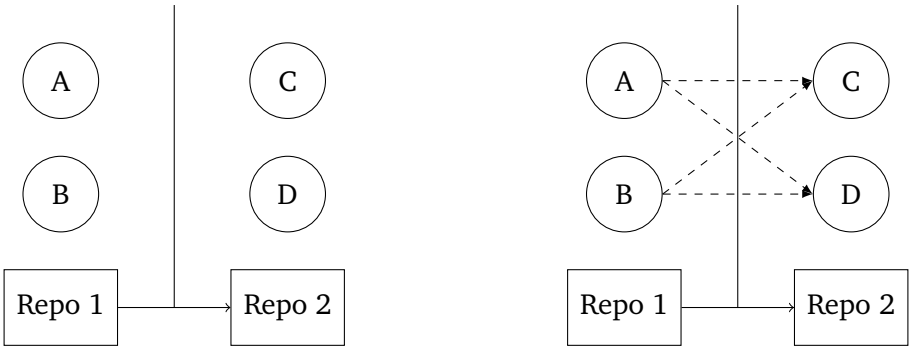
In terms of contributions, the aggregation should be understood as counting symmetric links between cities. We collect the contributors for each repository including their personal city location. To aggregate to the city level we count the city-city links. Let for example project A have 5 developers: Two from Vienna (V), two from Budapest (B) and one from Kiel (K). This would yield the following city-city links: two from V to B, two from V to K, two from B to V, two from B to K and one from K to B and V. Counting all connections builds our base case. For other specification where we consider organisations as an important variable, we count those connections separately. Once only considering those connections where two contributors are in the same repository and in the same organisation. Once, only the same repository without being in the same organisation.

The aggregation is illustrated in the following figure:

Aggregating dependencies

Let us turn to dependencies. In essence this follows the same logic as for contributions. Key difference is however that the dependency links are symmetric. Let us again consider an example as this illustrates the variable in a straight forward way. Take project A as described above. Now introduce project D which has two developers from Paris (P) and one developer from London (L). If project A now depends on project D, we record the following links: two times two links from P to V and B, i.e. 4 links from P to V and B, two links from L to B and V and one link from L to K as well as two links from P to K.

Now, we are equipped with city-pair level of contribution and dependency flows. To estimate gravity equations, control variables are also needed. As control variables, we consider distance, common language, time zone differences and travel restrictions. Whereas common language, time zone differences and travel restrictions are taken on a national level, distance is computed on a city pair wise level. To compute greater city pair-wise distances, we compute greater circle distances based on longitude and latitude information



(a) Dependency: Observables. **(b)** Dependency: Inferred links.

Notes: Stylized figures depicting observables and inferred links. In panel (a) what is observable in the data: Developers A and B contributing to Repo 1, C and D contributing to Repo 2. There is a dependency link from Repo 1 to Repo 2 i.e. Repo 2 depends on Repo 1. In panel (b) directed developer to developer links which are inferred from that.

of the respective cites.

Stacked data by organizations

There is an additional setup to consider. We aggregated individual links at the city-pair level. As organizations play an important role in mediating frictions, we need to deal with them carefully. Thus we first aggregated only links that happen outside organizations, calculated flows and the exposure variable. Then, we repeated this exercise aggregating links that are outside of organizations, calculated flows.

The dataset we run our regressions on is a stacked (appended) version of these two datasets, subset by aggregating links outside or inside organizations.

5.2 Role of cities

Taking a first look at the data. Figure 4 shows that there are some large centers worldwide such as San Francisco, London or New York. However, there are great many places with many developers spread out beyond the US and Europe, in Asia as well as South America. Indeed most countries have JavaScript developers.

The chart in Figure 5 is representing the locations of developers in the top 25 cities in 2019, the color coding is represents different geographic regions. Cities in North America lead in terms of size and presence in top 25. They are followed by European centers like London and Berlin with Eastern European cities like Moscow and Kiyv also making it. Asian centers play a comparable role with Beijing and Mumbai coming on top.

The concentration is visible in the map, but OSS is actually less concentrated than applied research such as physics[13]. The full distribution of 4000 cities in our data may be well approximated with a Pareto distribution.

[13] TO ADD

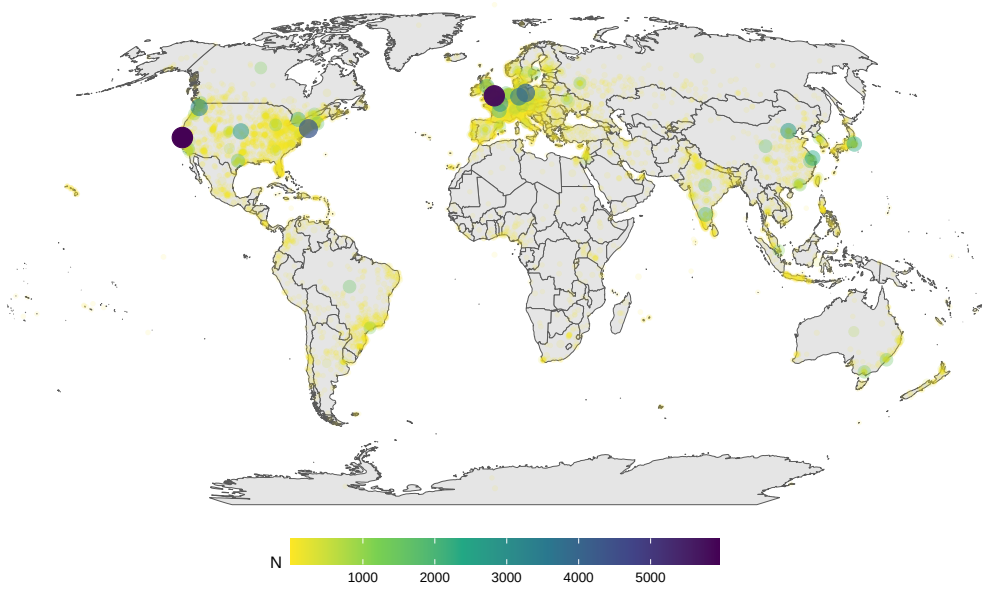


Figure 4: Developers per tenth of a square degree

Notes: World map depicting identified locations of Open Source JavaScript developers. Colour coding indicates clustering intensity with at most 5000 developers identified in San Francisco. Other technological hubs like New York, Berlin, London are clearly visible.

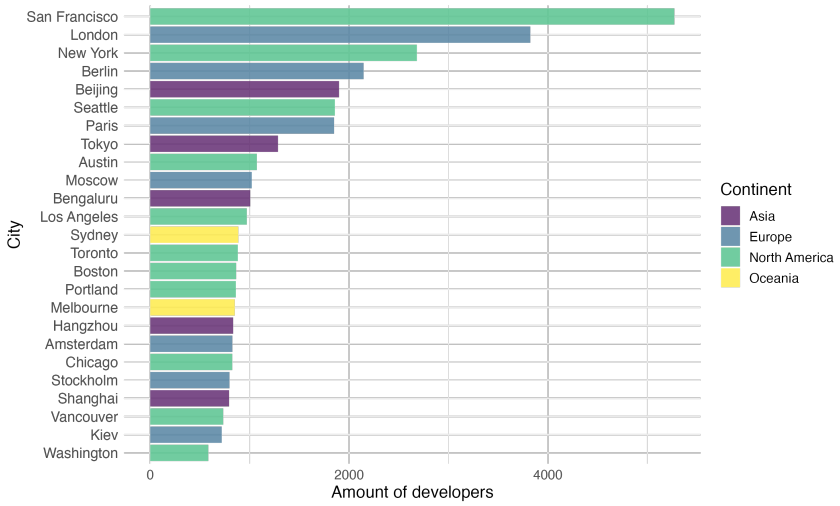
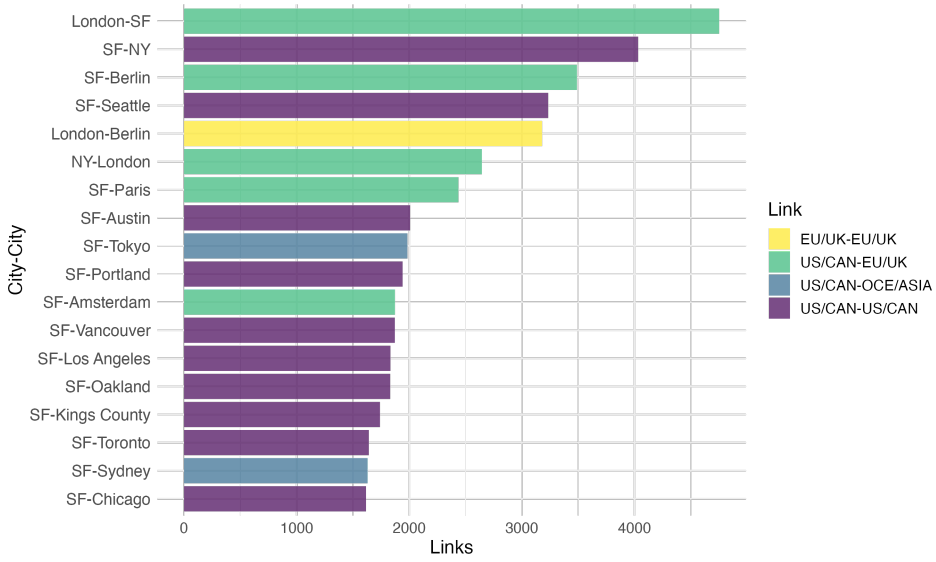


Figure 5: Contributors for Top 20 Cities

Notes: Barplot illustrating the amount of JavaScript developers in the top 20 cities with the most developers. Colour coding based on continents.



Notes: Barplot illustrating links between cities as measured by being active in the same repository. Same city links are excluded in this figure. Colour coding indicates world region pairs. Only one of the highest linked city pairs does not include the North American region.

Figure **??** depicts a horizontal bar chart that illustrates the number of shared projects between JavaScript developers residing in different city pairs in 2019. The chart illustrates the dominance of the London-San Francisco axis in the JavaScript developer community, suggesting a significant concentration of collaborative projects between these two tech hubs. The prevalence of San Francisco in multiple top-ranking city pairs underscores its pivotal role in the global JavaScript ecosystem. It is noteworthy that traditional economic centers like New York and London also appear frequently.

Most city pais are either with North-America or linking US and European hubs. The overall take is that links typically link key clusters in global North, leaving less room for Nort-South links.

6 Team formation for collaboration

We now turn to one of the key aspects of OSS: Work in teams. As software development does not require meeting in person, teams may be geographically quite dispersed. By considering a matching model, we first aim to understand how developers start joint projects.

6.1 Empirical setup

Guided by the theoretical arguments above, we develop our estimation framework.

6.2 Sparse network approximation

Given the large number of potential collaborators and dependencies, the probability of relying on any single one of them is very small. Hence the number of links, whether collaboration links or dependency links, within any group of users will be well approximated by the Poisson distribution. In particular, we will take users in one city and users in another city, and form groups of such city-pairs. The number of links for a city pair is then approximately distributed Poisson, with a mean proportional to $n_{ij} \exp(-\beta' \mathbf{d}_{ij})$ and $n_{ij} \exp(-\gamma' \mathbf{d}_{ij})$ for dependencies., where n_{ij} is the number of user pairs in the city pair.

For example, if there are 5,000 developers in San Francisco, and 4,000 in Berlin, there are 20 million potential links, $n_{SF,Berlin} = 20,000,000$. By contrast, if Bielefeld has 200 developers, $n_{SF,Bielefeld} = 1,000,000$. The number of potential links serves as an *exposure* variable in the Poisson regression. City pairs with higher n are expected to have more links and also receive a larger weight in the regression. As a special case, if there are no developers in a city, $n = 0$, and that city does not contribute to the regression analysis.

Importantly, exposure variables change if we subset the original data before aggregating. For instance, we will create the pairwise city-level data both for link that happen outside organizations as our core dataset and for links that happen within them. To continue the example above, if there are no links within organizations between Paris and Bielefeld, it will be zero. See Appendix A.4 for more details.

We estimate the regressions in a city-level gravity equation, but note that the coefficients are approximately equal to the logit coefficients in the individual decision equations described above. In other words, our coefficients can be read as proportional changes in the probability of collaborating with someone or importing their dependency.

6.3 Estimated equation for distance

We estimate city-pair gravity equations for two different outcome variables. One is links that are established by collaboration, the other is established by dependencies. First, consider collaboration. Take developer j with $j = 1, \ldots, J$ who are contributing to a software package i. We consider two developers j and k to be connected by collaboration if they contribute to at least one software package together. Formally, let $\mathbf{C}$ be a $J \times J$ matrix. The jkth element of this matrix is 1 if there is at least one software project on which developer j and k collaborate, and 0 otherwise. We set the diagonal $\mathbf{C}_{jj} = 0$ to rule out self-connections. Note that this matrix is symmetric.

Second, we consider dependencies. The logic follows closely the one outlined above. Let software project i depend on an outside software project b. We say that developer j depends on the work of developer k, if there exists a project i developed by j that depends on a project b developed by k. Accordingly, we obtain here a matrix of dependency links $\mathbf{D}$

which has the dimensions of $J \times K$. By contrast to collaborations, the dependency matrix is directed and not necessarily symmetric. Developer j may import projects developed by developer k, but the reverse may not be true.

If there are M cities, we can use an $M \times J$ city-aggregator matrix to add up all the links of users belonging to the same city. Denote this aggregator matrix by $\mathbf{M} = \mathbf{I}_M \otimes \mathbf{1}_{1\times J}$.

Our outcome variables will be the $M \times M$ matrix

$$\mathbf{Y} = \mathbf{MCM}',$$

capturing the *number* of collaboration links between city pairs. We can similarly capture the number of dependency links with $\mathbf{MDM}'$.

Take the element of this matrix corresponding to the origin city o and destination city d. By the Poisson approximation,

$$Y_{od} = \exp(\beta\tau_{ij} + \nu_o + \nu_d) + \varepsilon_{od} \tag{1}$$

where Y_{od} are contribution links from city o to city d. Fixed effects are at the city level o and d and relevant distance metrics are captured in τ_{ij}. Finally, ε_{od} represents an orthogonal error term.

The right hand side for both styles of regressions are structurally the same. This is a standard gravity equation which we estimate with a Poisson Pseudo Maximum Likelihood estimator (Santos Silva and Tenreyro, 2006).

As introduced earlier, in addition to the extensive margin (number of collaborators), we can also estimate Q_{do} as the intensity of collaboration. While the extensive margin counts links between a contributor and a package, the intensive margin measures how many commits flow from contributor to Packages.

Q_{do} also follows a gravity equation,

$$Q_{do} = N_i \cdot M_o \cdot \exp(-\beta'\mathbf{d}_{so}) + \epsilon_{od}. \tag{2}$$

The variables of interest are given by the distance vector $\mathbf{d}_{do}$ and its corresponding coefficients β and γ. We consider standard frictions like geographic distance, common languages and differences between time zones. We will show several functional form estimates. Regressions will include an exposure variable as described earlier. For more on the Poisson approximation, see Section A.2 in the Appendix.

6.4 Results - Collaboration

We now estimate equations (2) to understand how distance might influence the formation of production networks described earlier. We do this with the previously constructed contribution- and dependency flows. All models are estimated on a city level.

To understand the role of spatial frictions we estimate the gravity model as described in 1. The dependent variable is the number of collaborator links between cities, where one link is being a committer to other developers in the repository. Distance will be measured in different functional forms, and the the structural gravity model will include origin and destination fixed effects.

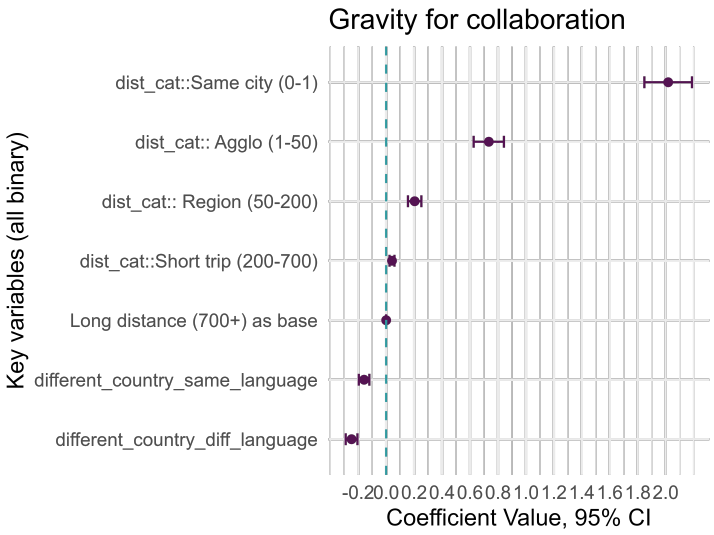
First we model spatial friction as a combination of co-location in the same city, and ln distance when they reside separately. Country frictions are: cities being in the same country (vs other), having a common language (one in common for multi-language countries).

Then, we have spatial frictions are measured with physical distance transformed into a categorical variable, as well as a set of variables measuring cross country frictions. Physical distance categories were shaped various approximate cutoffs for personal meeting: (i) being in the same city, being in the same agglomeration (but not same city) as measured with a 50km radius, being in the same region (between 50 to 200km), being available with a short trip (200 to 700 km) and beyond (as base). [14].

For our core models, we'll disregard very small cities where the number of contributors is less than 20 – this is half the city-pair level sample. We'll show robustness in Appendix. Figure 6 shows the core coefficients, while Table 4 shows the details.

[14] In extensions, we also include having a large time zone difference, and travel difficulty as needing a visa. (see details in the Appendix) To be added

Figure 6: Contribution and development links, core sample



Notes: Dependent variable is count of links between cities. Sample: Exclude very small cities. Exclude within-organization links. PPML estimates for gravity equation

Core results (Column 3 and Figure 6) show strong localization and a steep decline by distance. Contributors who reside in the same city have 6.5 times (exp(2.018)-1) higher chance to work together than those who live in far away (700km +) segments of the same country. This suggest a massive co-location benefit. Agglomeration is also very high, suggesting a more 2x higher chance. There is a sharp spatial decay and even 200km vs 700 gives only a 4% bump (and zero beyond). When physical trade cost is zero, and only face to face meeting costs count, the gravity model suggests massive co-location benefits, but basically no proximity gains beyond the commute distance. National borders also matter: Crossing a country is a 15% drop in probability, which rises to 25% if the countries don't share a language. Results are similar if we include pairs in organizations (column 2).

Table 4: Contribution and development links, core sample

Dependent Variable:	N of links between contributors		
Model:	(1)	(2)	(3)
Different city	-1.261***		
	(0.1055)		
ln distance — not same city	-0.0539***		
	(0.0060)		
dist_cat = Same city(0-1)		1.746***	2.018***
		(0.0772)	(0.0858)
dist_cat = Agglomeration(1-50)		0.6351***	0.7344***
		(0.0724)	(0.0873)
dist_cat = Region(50-200)		0.1905***	0.2039***
		(0.0319)	(0.0307)
dist_cat = Short-trip(200-700)		0.0245*	0.0416***
		(0.0127)	(0.0101)
different country, same language	-0.0792***	-0.1749***	-0.1581***
	(0.0229)	(0.0215)	(0.0184)
different country, diff language	-0.1910***	-0.2856***	-0.2476***
	(0.0369)	(0.0369)	(0.0322)
In same organization (0-1)	5.565***	5.556***	
	(0.0858)	(0.0855)	
Fixed-effects			
city_destination	Yes	Yes	Yes
city_origin	Yes	Yes	Yes
Pseudo R^2	0.84371	0.84385	0.86084
Observations	3,636,122	3,636,122	3,478,716

Notes: Dependent variable is count of links between cities. Sample: Exclude very small cities. Exclude within-organization links. PPML estimates for gravity equation. Standard errors, clustered at city_destination & city_origin level are in parentheses, ***: 0.01, **: 0.05, *: 0.1.

When using a single distance metric (column 1), the point estimate is well below that of estimations in trade or even below that of patents. However, non-linearity is super important: this average masks a steep localization premium and basically flat relationship beyond commute distance.

In terms of robustness, we looked at two potential issues (see details in the Appendix)[15]. First, maybe a few very large repositories dominate and flatten the curve. After excluding them, coefficients indeed decline somewhat but the overall picture is unchanged. Second, despite city fixed effects, cities like San Francisco may dominate the results. Looking at heterogeneity by city size, we see some but no huge difference excluding few largest cities

[15] To be added

6.5 Intensive margin

As introduced earlier, links measure the extent of flows between cities. As we have information about the commits, we can also look at the intensity of relationships. Let us consider city pairs with positive links, and compare the intensity of collaboration by distance. Here our dependent variable will be $Commit\_share_{ij} = N\_commitsij/N\_linksij$, and we repeat the same regression as before.

As shown in Table 5 the intensity is also higher for pairs co-located in the same city but does not vary beyond that. Developers located in the same city will commit 2x as much in a give project than those in different cities.

Table 5: Contribution and development links, core sample

Dependent Variables:	N links	commit share
Model:	(1)	(2)
Variables		
dist_cat = Samecity(0-1)	2.018***	0.7564***
	(0.0858)	(0.1309)
dist_cat = Agglo(1-50)	0.7344***	0.1838
	(0.0873)	(0.1410)
dist_cat = Region(50-200)	0.2039***	0.0906
	(0.0307)	(0.0795)
dist_cat = Shorttrip(200-700)	0.0416***	-0.0192
	(0.0101)	(0.0399)
Fixed-effects		
city_destination	Yes	Yes
city_origin	Yes	Yes
Fit statistics		
Pseudo R^2	0.86084	0.52444
Observations	3,478,716	451,423

Origin, destination city FE, Clustered (city_destination & city_origin) standard-errors in parentheses

Notes: Dependent variable is count of links between cities and ratio of commits to links between city pairs. Sample: Exclude very small cities. Exclude within-organization links. PPML estimates for gravity equation. Standard errors, clustered at city_destination & city_origin level are in parentheses, ***: 0.01, **: 0.05, *: 0.1.

7 Success and dispersion

We now turn to the main results looking at success. The core measure of success is the number of packages that has ever imported it as a dependency. Greater number of imports is signaling the extent of a repository's integration into the broader software development environment.

7.1 Estimated regression for success

For our main estimation, we estimate two models, one where the dependent variable is count of all dependents and another where we only focus on dependencies from NPM.

They key difference is that package managers are environments made for package as intermediate product use. As we describe in Section A.1.3, NPM offers technical support, version control and even quality control to help adoptions. Packages that are not on NPM are typically final use projects not created to be used as inputs by others.

We estimate the number of packages that import our package as dependency as a function of spatial dispersion: number of countries per developer, number of cities per country and number of developers.

$$\Pr(Y_i|.) \approx \text{Poisson}[\exp(\beta_1 f(cities_i) + \beta_2 g(countries_i)) + \gamma \mathbf{Z}]$$

To empirically investigate this, we employ a Poisson regression model where the dependent variable, Y_i, represents the number of repositories importing a given repository i. Dispersion – number of cities and countries will be included in different parametric and non-parametric ways, with ($cities_i$) and countries ($countries_i$) denoting the number of places from which the repository is contributed to. We also have a vector of control variables $\mathbf{Z}$: the number of developers, categorized by their contribution level, and the age of the project, also treated categorically. These controls are designed to account for project-specific characteristics that might influence the repository's popularity.

In assessing the quality of coding contributions, we control for the geographical size of cities where the top three contributors reside, under the premise that urban agglomeration might correlate with resource availability and talent pools. The quality of contributors is proxied by the number of stars garnered by the top three contributors, a proxy for peer recognition in the open source community.

Furthermore, our model will later incorporate the volume of code contributions through the sum of commits and the squared sum of commits to control for both the extent and intensity of development activity.

7.2 Results

The table 7 presents the results of a regression analysis where the dependent variable is the number of dependents on NPM (which is a measure of a package's popularity). Three different models are shown with different functional form on city and country count. Model 3 is also shown in Figure 7. We find that more popular packages are produced with higher spatial dispersion.

The first two columns show the linear functional form with second column including controls of number of developers and age. The number of cities and countries has a positive are positively correlated with the number of NPM dependents across both models, with coefficients of 0.4075 and 0.3431 for cities and 0.2306 and 0.3057 for countries respectively. The effect sizes decrease slightly in Model 2 when controlling for project age and number of developers, suggesting these controls account for some of the variance in the number of dependents. The third model show that the pattern is fairly linear for 2-4 cities and 1-3 countries. Finally the graph suggest that when combining cities and countries, there is indeed an interaction, with adding a country being really important.

Table 6: Contribution and development links, core sample

Dependent Variable:	N Dependents (NPM)		
Model:	(1)	(2)	(3)
Count of cities	0.4075***	0.2306***	
	(0.0403)	(0.0491)	
Count of countries	0.3431***	0.3057***	
	(0.0628)	(0.0637)	
City cat × CI2 × 2cities			0.3851***
			(0.0813)
City cat × CI3 × 3cities			0.4925***
			(0.1242)
City cat × CI4 × many cities(4+)			0.6543***
			(0.1642)
Country cat × CO2 × 2 countries			0.2461***
			(0.0813)
Country cat × CO3 × many countries(3+)			0.6269***
			(0.1462)
Constant	1.745***	1.148***	1.673***
	(0.0548)	(0.0857)	(0.0674)
Age, N_Dev	No	Yes	Yes
Commits	No	No	No
Coders	No	No	No
Pseudo R^2	0.05100	0.11532	0.11586
Observations	36,491	36,491	36,491

Notes: Dependent variable is count of imports as dependents. Sample: Exclude very small cities. Exclude within-organization links. PPML estimates. Standard errors, clustered at city_destination & city_origin level are in parentheses, ***: 0.01, **: 0.05, *: 0.1.

When we consider all user not just those on NPM – including final users such as websites – we find very similar results. Even point estimates are very close once we include coder and commit controls (column 3 vs 4).

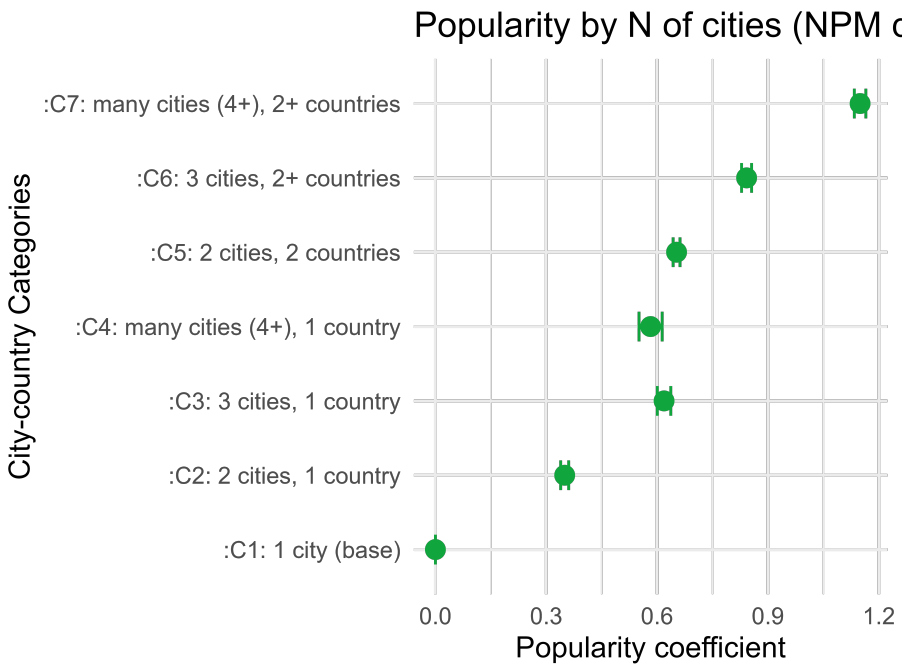


Figure 7: Dependents of packages by amount of cities engaged in the project

Notes: Dependent variable is count of imports as dependents. Sample: Exclude very small cities. Exclude within-organization links. PPML estimates.

Table 7: Contribution and development links, NPM vs all users

Dependent Variables:	N Dependents (NPM)	N Dependents (all)	N Dependents (NPM)	N Dependents (all)
Model:	(1)	(2)	(3)	(4)
Variables				
City cat × CI2 × 2cities	0.3851***	0.5120***	0.4527***	0.4494***
	(0.0813)	(0.1266)	(0.0858)	(0.1309)
City cat × CI3 × 3cities	0.4925***	0.6288***	0.5993***	0.5982***
	(0.1242)	(0.1761)	(0.1262)	(0.1805)
City cat × CI4 × many cities(4+)	0.6543***	0.9042***	0.8047***	0.8391***
	(0.1642)	(0.2069)	(0.1688)	(0.2166)
Country cat × CO2 × 2countries	0.2461***	0.2602**	0.2308***	0.2980**
	(0.0813)	(0.1177)	(0.0810)	(0.1166)
Country cat × CO3 × many countries(3+)	0.6269***	0.4851**	0.5960***	0.6214***
	(0.1462)	(0.1916)	(0.1452)	(0.1892)
Constant	1.673***	4.577***	-0.9267***	2.437***
	(0.0674)	(0.1533)	(0.3057)	(0.4277)
Age, N_Dev	Yes	Yes	Yes	Yes
Coders	No	No	Yes	Yes
Commits	No	No	Yes	Yes
Pseudo R^2	0.11586	0.16712	0.18424	0.24202
Observations	36,491	36,344	36,491	36,344

Notes: Dependent variable is count of imports as dependents. Sample: Exclude very small cities. Exclude within-organization links. PPML estimates. Standard errors, clustered at city_destination & city_origin level are in parentheses, ***: 0.01, **: 0.05, *: 0.1.

7.3 What is behind this correlation

What could explain this correlation? Let us first consider possible arguments.

1. Reverse causality

Regarding dependencies, we see distance in any form plays a much smaller role. In fact, double fixed effects explain an overwhelming variation in the data, leaving little to be explained by distance. A popular package will be used no matter what. Having said that we see that being in proximity to each other is associated with a 8% higher probability of importing the partner's dependency.

Table 8: Gravity for dependencies

Dependent Variables:	contr_n_links	dep_value
Model:	(1)	(2)
Variables		
dist_cat = Samecity(0-1)	2.018***	0.0754***
	(0.0858)	(0.0138)
dist_cat = Agglo(1-50)	0.7344***	0.0805***
	(0.0873)	(0.0127)
dist_cat = Region(50-200)	0.2039***	0.0254***
	(0.0307)	(0.0095)
dist_cat = Shorttrip(200-700)	0.0416***	0.0045
	(0.0101)	(0.0036)
different country same language	-0.1581***	-0.0222***
	(0.0184)	(0.0082)
different country diff language	-0.2476***	-0.0499***
	(0.0322)	(0.0115)
Pseudo R^2	0.86084	0.98866
Observations	3,478,716	3,202,202

Origin, destination city FE, Clustered (city_destination & city_origin) standard-errors in parentheses

Notes: The dependent variable is the number of dependency imports from the origin to the destination city. Sample: Exclude very small cities. Exclude within-organization links. PPML estimates for gravity equations. Standard errors, clustered at city_destination & city_origin level are in parentheses. ***: 0.01, **: 0.05, *: 0.1.

2. Self-selection and FC of meeting

In A.3 we describe a model of selection. Once we allow for a higher cost to form relationships across cities, we'll find that better coders will self-select into better projects and in order to find a better match will seek partner across cities.

To shut down this channel we will partial out coder quality: Number of repos, number of followers for the 2-3 most important developers as well as the size of their city (N of total coders).

To condition on the size of the project we also include the number of projects.

Table 9: Selection? Partialing out coder quality and commits

Dependent Variables:	N Dependents (NPM)			N commits
Model:	(1)	(2)	(3)	(4)
Count of cities	0.2306***	0.1906***	0.2818***	-0.1087***
	(0.0491)	(0.0488)	(0.0509)	(0.0358)
Count of countries	0.3057***	0.3243***	0.2904***	0.0355
	(0.0637)	(0.0634)	(0.0637)	(0.0583)
Sum of commits (ln)			0.3330***	
			(0.0202)	
Constant	1.148***	-0.0641	-1.746***	5.090***
	(0.0857)	(0.1140)	(0.1507)	(0.0854)
Age, N. Developers	Yes	Yes	Yes	Yes
Coders	No	Yes	Yes	Yes
Commits	No	No	Yes	–
Pseudo R^2	0.11532	0.15056	0.18345	0.26074
Observations	36,491	36,491	36,491	36,491

Notes: Dependent variable is count of imports as dependents. Sample: Exclude very small cities. Exclude within-organization links. PPML estimates.

Table 9 suggest that results are broadly unchanged when we compare coders of similar quality based in similar locations, exclude success driven by bigger spatial reach of developers. They remain qualitatively the same even conditioning on the intensity of collaboration (commits).

When we compare coders of similar quality based in similarly sized cities working on comparable projects in terms of activity, the group of diverse coders will create more successful projects.

7.4 What can drive the results

Before making a causal argument, selection remains an issue. Given high costs of collaboration across cities, coders will only carry out projects with highest expected success. This will yield to cross-city (and cross-country) projects to be better on average.

The causal argument would posit that diversity in location is correlated with diversity in skills (and approaches) and a combination of diverse skills will allow coders create a better product.

8 Discussion

What is the impact of distance on patterns of production and sourcing in a weightless economy? In this paper we study collaboration and importing of inputs in the software industry, using data from open source software development.

We found that while OSS developers will collaborate more intensively when in close proximity, more widely adopted software packages are written by a more spatially diverse group of developers. We show that this is not driven by the geography of adoption, as developers only marginally prefer locally written packages as intermediate input (dependency). It is not driven by talent distribution across cities either or self-selection of best coders into best project. One explanation is selection of coder pairs into better projects.

Our results share similarities with evidence from other areas. First the role of distance to reduce transaction frequency is well known from international trade. Our gravity results, even in terms of point estimates, are pretty close to math papers in Head et al. (2019) (high role of shared location, small distance elasticity beyond, some role of crossing borders.). Albeit in a very different setup, AlShebli et al. (2018) showed evidence of a positive correlation between ethnic diversity and success. In that case, magnitudes were one order smaller.

What makes these results special is the fact that collaboration here is almost fully online. Yet results are similar to cases where personal contacts are essential (academia). This suggest that we have in past underestimated search and matching costs.

References

AlShebli, Bedoor K., Talal Rahwan, and Wei Lee Woon, "The preeminence of ethnic diversity in scientific collaboration," *Nature communications*, 2018, *9*, 1–10.

Ariu, Andrea, Holger Breinlich, Gregory Corcos, and Giordano Mion, "The interconnections between services and goods trade at the firm-level," *Journal of International Economics*, 2019, *116* (C), 173–188.

Bernard, Andrew B, Andreas Moxnes, and Yukiko U Saito, "Production networks, geography, and firm performance," *Journal of Political Economy*, 2019, *127* (2), 639–688.

_ , _ , and _ , "The Geography of Knowledge Production: Connecting Islands and Ideas," Technical Report, Working paper 2020.

Blind, Knut, Mirko Böhm, Paula Grzegorzewska, Andrew Katz, Sachiko Muto, Sivan Pätsch, and Torben Schubert, "The impact of Open Source Software and Hardware on technological independence, competitiveness and innovation in the EU economy," *Final Study Report. European Commission, Brussels, doi*, 2021, *10*, 430161.

Blum, Bernardo S and Avi Goldfarb, "Does the internet defy the law of gravity?," *Journal of international economics*, 2006, *70* (2), 384–405.

Breinlich, Holger, Anson Soderbery, and Greg C. Wright, "From Selling Goods to Selling Services: Firm Responses to Trade Liberalization," *American Economic Journal: Economic Policy*, November 2018, *10* (4), 79–108.

Békés, Gábor and Gianmarco Ottaviano, "Cultural Homophily and Collaboration in Superstar Teams," Technical Report, CEPR Discussion Paper No 17618 2022.

_ , Julian Hinz, and Miklós Koren, "Bugs 🐞," Technical Report, Unpublished 2023.

Chaney, Thomas, "The network structure of international trade," *The American Economic Review*, 2014, *104* (11), 3600–3634.

Chelkowski, T., D. Jemielniak, and Kacper Macikowski, "Free and Open Source Software organizations: A large-scale analysis of code, comments, and commits frequency," *PLoS ONE*, 2021, *16*.

El-Komboz, Lena Abou, Thomas Fackler et al., "Productivity Spillovers among Knowledge Workers in Agglomerations: Evidence from GitHub," in "VfS Annual Conference 2022 (Basel): Big Data in Economics" number 264083 Verein für Socialpolitik/German Economic Association 2022.

Goldbeck, Moritz, "Bit by bit: colocation and the death of distance in software developer networks," Technical Report, ifo Working Paper 2022.

Gousios, Georgios and Diomidis Spinellis, "GHTorrent: GitHub's data from a firehose," in "2012 9th IEEE Working Conference on Mining Software Repositories (MSR)" IEEE 2012, pp. 12–21.

Head, Keith, Thierry Mayer, and John Ries, "How remote is the offshoring threat?," *European Economic Review*, 2009, *53* (4), 429–444.

_ , Yao Amber Li, and Asier Minondo, "Geography, Ties, and Knowledge Flows: Evidence from Citations in Mathematics," *The Review of Economics and Statistics*, 10 2019, *101* (4), 713–727.

Hoffmann, Manuel, Frank Nagle, and Yanuo Zhou, "The Value of Open Source Software," *Harvard Business School Strategy Unit Working Paper*, 2024, (24-038).

Jaffe, Adam B, Manuel Trajtenberg, and Rebecca Henderson, "Geographic localization of knowledge spillovers as evidenced by patent citations," *Quarterly journal of Economics*, 1993, *108* (3), 577–598.

Laurentsyeva, Nadzeya, "From friends to foes: National identity and collaboration in diverse teams," Technical Report, CESifo Discussion Paper 2019.

Lychagin, Sergey, Joris Pinkse, Margaret E. Slade, and John Van Reenen, "Spillovers in Space: Does Geography Matter?," *The Journal of Industrial Economics*, 2016, *64* (2), 295–335.

Silva, J M Santos and Silvana Tenreyro, "The Log of Gravity," *Rev. Econ. Stat.*, 2006, *88* (4), 641–658.

Wachs, Johannes, Mariusz Nitecki, William Schueller, and Axel Polleres, "The Geography of Open Source Software: Evidence from GitHub," *Technological Forecasting and Social Change*, 2022, *176*, 121478.

Walsh, Keith, "Trade in services: Does gravity hold?," *J. World Trade*, April 2008, *42* (2), 315–334.

A Appendix

A.1 Background of OSS

In this section, we provide more background information on the development of open source software and describe the model that builds the basis of our theoretical understanding. The theoretical model then, in turn, also motivates our regression equations.

A.1.1 Principles and history of open source

OSS is a model of software development that is characterized by its collaborative and decentralized nature. It is based on the idea of making the source code of software publicly available, allowing anyone to inspect, modify, and enhance it.[16] The open source model is often contrasted with the traditional proprietary model, where the source code is kept private and only the compiled software is distributed. The open source model has been widely adopted in the software industry, with many companies and organizations contributing to open source projects. Major companies such as Alphabet, Meta, IBM, Microsoft, and Red Hat actively contribute and use open source projects. Examples of open source software widely used today include operating systems such as Linux,[17] web browsers such as Mozilla Firefox,[18] database systems such as MySQL,[19] machine learning and AI frameworks such as TensorFlow and PyTorch,[20] and web development platforms such as WordPress.[21]

The open source movement has its roots in the free software movement of the 1980s, which was led by Richard Stallman and the Free Software Foundation.[22] The free software movement was based on the idea of making software free to use, modify, and distribute. The term "open source" was coined in 1998 by a group of developers who wanted to promote the idea of free software to the business world.[23] The term was chosen to emphasize the practical benefits of open source software, such as the ability to inspect and modify the source code.

A.1.2 Open source software in the modern infrastructure

Open source software plays a vital role in the modern digital infrastructure, powering various domains and industries. Major companies such as Alphabet, Meta, IBM, Microsoft, and Red Hat actively contribute and use open source projects. Examples of open source

[16] See https://opensource.org/osd.
[17] See https://www.linuxfoundation.org/.
[18] See https://www.mozilla.org/en-US/firefox/new/.
[19] See https://www.mysql.com/.
[20] See https://www.tensorflow.org/ and https://pytorch.org/.
[21] See https://wordpress.com/.
[22] See https://www.gnu.org/philosophy/free-sw.en.html.
[23] See https://opensource.org/history.

software widely used today include:

- Operating Systems: Linux is a prominent open source operating system used in servers, embedded devices, and smartphones.
- Web Browsers: Mozilla Firefox is an open source web browser that offers secure and customizable internet browsing.
- Database Systems: MySQL is an open source database system used for managing large amounts of data.
- Machine Learning and Artificial Intelligence: Open source frameworks like TensorFlow and PyTorch enable the development and deployment of machine learning and AI models.
- Web Development: Content management systems like WordPress provide open source platforms for creating and managing websites.

Two recent reports by the EU commission (Blind et al. (2021))and the Linux foundation[24] have shown the importance of OSS. The EU report estimates that OSS contributes €95 billion to European GDP. Beyond that, it is also assessed that OSS is an important driver of innovation. The report by the Linux Foundation is largely a survey among software firms and mostly asserts that the benefits of open source outweigh its costs.

A.1.3 Suppliers of code: Package managers

Open Source Software Package Managers, such as npm and PyPI, are tools that facilitate the distribution and installation of software packages[25]. NPM stands for Node Package Manager, primarily used in the JavaScript ecosystem, notably with Node.js. PyPI, the Python Package Index works with python. They help managing libraries that your project depends on, allowing users to download and install JavaScript / Python packages.

They play several important roles. Package managers simplify dependency management. They track all the libraries your code depends on and their respective versions, ensuring compatibility and simplifying installation processes. They are open, so developers can publish projects. They help in versioning packages, allowing easy updates. An important feat is them serving as central repository: They act as centralized repositories where packages are hosted, making it easy for developers to find and install the software they need.

One important consequence is that package managers also help quality control. First both npm and PyPI enforce semantic versioning (semver), which is crucial for managing

[24] linux foundation report last accessed on 09/07/2023

[25] Builds on ChatGPT, on role of and reason for OSS package managers, and their role in quality assurance. 2024-02-12

dependencies' versions in a way that avoids breaking changes.[26]. Second, they offer mechanisms to report security vulnerabilities within packages. This allows the community to be aware of potential risks and for package maintainers to address these issues in subsequent releases. Third, they come with tools to resolve dependency conflicts and audit dependencies for known vulnerabilities. Fourth, npm and PyPI both support mechanisms to verify the integrity of packages via checksums or cryptographic signatures. This ensures that the packages have not been tampered with since their publication. Fifth, the open source nature allows for community feedback and review. Finally, package managers have Automated Testing and Continuous Integration Hooks: Many packages hosted on npm and PyPI use continuous integration services to automatically run tests on various versions of dependencies and different environments.

A.2 Poisson approximation

We have a binary outcome $y_i \in \{0, 1\}$ for $i = 1, 2, ..., N$, with N very large. The probability of success is given by logit,

$$\Pr(y_i = 1|X_i) = \Pi(\beta X_i),$$

with $\Pi(z) = e^z/(1 + e^z)$.

The explanatory variable is vector valued and may take G distinct values with $G \ll N$. For example, it may be a same city dummy ($G = 2$) or the pairwise distance between K cities ($G = K^2 \ll N$).

The log likelihood contribution of observation i is

$$\ln \mathcal{L}_i = y_i \ln \Pi(\beta X_i) + (1 - y_i) \ln[1 - \Pi(\beta x_i)]$$

so that the total log likelihood in the sample is

$$\ln \mathcal{L} = \sum_{i=1}^{N} y_i \ln \Pi(\beta X_i) + (1 - y_i) \ln[1 - \Pi(\beta X_i)].$$

Consider a change of variables. The values in the sum will be repeated very frequently, because there are only G distinct values that X_i can take. Let Z_g denote the gth possible value of X_i. For example, for a same-city dummy, $Z_1 = 0$ and $Z_2 = 1$.

Let $n_g := \|i : X_i = Z_g\|$ denote the number of observations for whom X_i takes its gth possible value. We will also call this group g.

Within this group, some observations have $y_i = 1$. Their count will be $n_{g1} := \|i : X_i =$

[26]Semver is a convention that uses version numbers to convey meaning about the underlying changes. This includes major changes (which may break backward compatibility), minor changes (additions that should not break existing functionality), and patches (bug fixes)

$Z_g, y_i = 1\|$. Similarly, we denote $n_{g0} := \|i : X_i = Z_g, y_i = 0\|$. Trivially, we have $n_g = n_{g0} + n_{g1}$.

For example, suppose X_i is a same-city dummy. Then $G = 2$, with $g = 1$ denoting not in the same city. $g = 2$ being the same city. Then n_{11} denotes the number of user pairs that are *not in the same city* and *have a link*.

Given this change of variables, we can rewrite the sum of log likelihood as

$$\ln \mathcal{L} = \sum_{g=1}^{G} n_{g1} \ln \Pi(\beta Z_g) + n_{g0} \ln[1 - \Pi(\beta Z_g)].$$

Maximizing with respect to β yields

$$0 = \sum_{g=1}^{G} n_{g1} \frac{\Pi'(\beta Z_g)}{\Pi(\beta Z_g)} Z_g - n_{g0} \frac{\Pi'(\beta Z_g)}{1 - \Pi(\beta Z_g)} Z_g = \sum_{g=1}^{G} \frac{\Pi'(\beta Z_g)}{\Pi(\beta Z_g)[1 - \Pi(\beta Z_g)]} Z_g [n_{1g} - \Pi(\beta Z_g) n_g].$$

The term in brackets is the error term in a regression trying to predict the number of links on group g. That error term is orthogonal to Z_g. The fraction at the beginning provide some weighting across groups,

$$\frac{\Pi'}{\Pi(1 - \Pi)} = 1 - \Pi(\beta Z_g).$$

When predicting links in a sparse graph, the probability of $y_i = 1$ will be very small, so that $\Pi \approx 0$ and the weight is ≈ 1.

Then the first-order condition for ML can then be approximated by

$$0 = \sum_{g=1}^{G} Z_g [n_{1g} - e^{\beta Z_g} n_g].$$

This is *exactly* the FOC for a Poisson ML, with n_g as an exposure variable.

A.3 Team formation - A matching model

[TO BE ADDED]