The geography of production and sourcing in the weightless economy: Evidence from open-source software[*]

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

September 2023

Abstract

Trade costs, broadly defined, limit interaction between far-away economic agents and motivate them to concentrate in close proximity to one another. The gravity equation, which assigns a large role for geographic distance as a determinant of trade costs, has been remarkably successful in explaining patterns of international and international goods trade between firms, services trade, individual mobility and migration, and even scientific collaboration. It is unclear, however, why distance is relevant for sectors in which direct trade costs are zero. We study patterns of interaction when the good in question is "weightless" using data from the open-source software industry. We find that importing other software packages as dependencies is at most weakly correlated with distance. By contrast, collaboration between different developers on the same piece of software is strongly decaying with distance. These results suggest that geography continues to matter for goods and services where collaboration is frequent and deep.

Keywords: Software industry, open source, gravity, global production networks, international sourcing
JEL Classification: F10, F14, L86, F23

[*]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 the modern economy, most products are created using in-house production (within the firm or establishment), local inputs (neighborhood, city, country), and imports (global supply network). We are interested in understanding the drivers of production and input decisions: whether to make an input in house, or source it from elsewhere. These decisions will depend on production and direct purchase costs, transport costs, and transaction costs related to search and adaptation.

This paper describes a special yet important industry, software, focusing on open-source software (OSS). OSS has emerged as a fundamental component of the global economy, powering modern digital infrastructure in various domains and industries. The software industry as a whole represents about 1 % of global GDP, growing rapidly at twice the rate of the rest of the economy.[1] Furthermore, 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.[2]

Yet, despite being "unbound" by geography due to its very nature, its production tends to concentrate heavily in certain parts of the world. Consider the 100 most valuable companies in the software sector by market capitalisation: 75 are in the US, 5 are in Israel and 3 in Canada with other countries having at most 2 in the top 100.[3] The aim of this paper is to investigate the production and sourcing decisions in the open source software industry. While we focus on this industry we hope to gain a more general understanding for the weightless economy.

Before moving on, it is useful to pin down some key terms. Open source software refers to software packages shared freely with the public not only for end users, but also for modification and other reuse by other developers. The most frequent form of software reuse involves developers "importing" code written by others into their own software package. This creates a "dependency": the using software depends on the imported one and may partially or fully use its codebase. Because OSS allows for sharing and modification, most OSS packages are developed by multiple developers in collaboration.

These activities have clear parallels in the economics of supply chains. The developer of a software package is a producer. Importing a dependency is akin to purchasing intermediate inputs from another producer. Collaboration is similar to employment in an

[1]See https://www.gartner.com/en/newsroom/press-releases/2023-04-06-gartner-forecasts-worldwide-it-spending-to-grow-5-percent-in-2023 and https://www.mckinsey.com/industries/technology-media-and-telecommunications/our-insights/reversal-of-fortune-how-european-software-can-play-to-its-strengths. For comparison, the automotive sector makes up about 3 % of global GDP (https://www.ilo.org/wcmsp5/groups/public/ed_dialogue/sector/documents/publication/wcms_161519.pdf.

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

[3]www.companiesmarketcap.com, data retrieved on 11/09/2023

often multinational team. Our focus will be sourcing decisions for a software package: the decision whether to write code for a certain feature or import it as a dependency.

Studying OSS allows us a closer look at transaction costs. In OSS, transport costs as well as the direct costs of using an input are zero. The decision to make or source an input (a piece of software providing some functionality) will then be based on transaction costs alone. Are suitable inputs easy to find? Can they be easily evaluated whether they are fit for purpose? Is the source (the creator or provider of the piece of software) trustworthy? What modifications are necessary to use this software? These questions are also relevant in other industries and other products, but they can be directly studied in OSS.

We think of an OSS producer who wants to add functionality to their software. They can write code themselves, or they can use code written by others. Sourcing from others can take two forms: collaboration or importing a dependency. Collaboration involves other coders writing code directly for the producer with appropriate planning, discussion and review. By contrast, importing a dependency is a unilateral decision of the producer: because OSS code is publicly available, it can simply be reused with a few programming statements. Such a use still requires adaptation of existing code, and may require occasional contact with the original producer of the dependency, to fix bugs, for example (Békés et al., 2023). However, the frequency and intensity of collaboration with the original producer are orders of magnitude smaller than under joint coding.

We use data from GitHub and Libraries.io to measure patterns of collaboration between coders and patterns of dependencies between software packages. We find that collaboration is very much shaped by geography: coders who are closer in space, reside in the same country, share a common language and a time zone are much more likely to collaborate. By contrast, the role of distance and other geographic frictions are much more limited for choosing to import a dependency. Geographic distance, in particular, is often uncorrelated with dependency importing decisions.

These results suggest that in the weightless economy, transactions costs become much less geography-dependent. All coders see a very similar menu in Berkeley, Budapest, or Bangalore. Their choice is less dependent on geography and more on quality. At the same time, collaboration remains very localized, affecting producers looking for specialized inputs and frequent interaction. The finding has implications on the spatial concentration of coding.

The remainder of this paper is structured as follows. In section 2 we discuss related work. In section 3 we provide an overview of the data sources and introduce the OSS industry.subsection 3.3 provides descriptives statistics for the data. We sketch an empirical framework in section 4. The empirical design and results are discussed in section 5. We conclude in section 6.

2 Related literature

This paper is related to previous work in international trade and economic geography focusing on services.

Physical distance is the defining determinant of trade in goods (Head and Mayer, 2014) — and services where transport costs (usually) do not enter: 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). Importantly, they also report such effects for so-called IT services. Anderson et al. (2018) also estimate gravity equations for different services categories. For *Computer services* the short distance effects are insignificant while long distance effects are significant but small [4].

Related to physical distance, but particular in the context of tradable services, are time zone differences. Stein and Daude (2007) use time zone differences as an explanatory variable for bilateral FDI flows. Similar to traditional geographical distance effects, they report negative significant time zone elasticities. More recently Bahar (2020) shows that time zone differences matter especially for knowledge intensive tasks by tracking subsidiaries of cooperations.

More generally, this paper is also related to the literature on global value chains, where software dependencies share characteristics with (physical) intermediate inputs. For a recent survey see Antràs and Chor (2022).

Our paper is also related to research in economic geography. Leamer and Storper (2017) hypothesize about agglomeration and dis-agglomeration as competing forces in the internet age. While routine tasks are seen to become more disaggregated, non-routine tasks would tend to agglomerate.

There is also related research on scientific collaboration and innovation. The seminal study by Jaffe et al. (1993) suggests that even in a weightless economy distance matters. This resonates with van der Wouden and Youn (2023), who show that geographical proximity among scientists is associated with knowledge spillovers. 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 development of OSS is also comparable to that of knowledge production in academia. Ductor et al. (2014); Hsieh et al. (2018), e.g., study networks of economists and its impact on output and optimal effort in team collaboration.

Finally, our work is also related to studies of the software industry itself.[5] A commonly found result resonates with questions about physical production: concentration is very

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

[5]A general introduction from an economic perspective emphasising relevant research questions can be found in Lerner and Tirole (2005).

important. For instance, Wachs et al. (2022) suggests that software is developed in geographically clustered areas. Related to this is also the work by Laurentsyeva (2019) who estimate gravity equations on open source software *commits*, reporting (small) negative estimates for the distance elasticity.

3 Data *on* Open Source Software

We now turn to our subject of study — open source software. We first introduce the peculiarities of the industry before describing the data used in the analysis.

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

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.[6]

3.2 Data sources

The very nature of the production of open source code allows us to study it in great detail. We make use of three main data sources.

Our first used data source is `Libraries.io`.[7] 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 collect metadata on published

[6]More background on open source software may be found in the A.1 section of the Appendix.
[7]`Libraries.io` last accessed on 07/07/2023

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`.[8] `JavaScript` is one of the fundamental technologies of the so-called "World Wide Web"[9] and thus the language with the most projects in the `libraries.io` database.

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 Githubs 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 starts in 2013 and ends in 2019. We use the latest data dump from June 2019.

Note that the descibred open source code is hosted on `Github`.[10] The website offers additional data on (self-reported) user location data, 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.

Finally, we make use of standard gravity measures for distance, language, and time zone information relying on Conte et al. (2022).[11]

3.3 Descriptive statistics

We now take a first peak at the data. We first look at the contributors to open source software, before describing stylized facts for imported code, i.e. dependencies.

3.3.1 Contributions

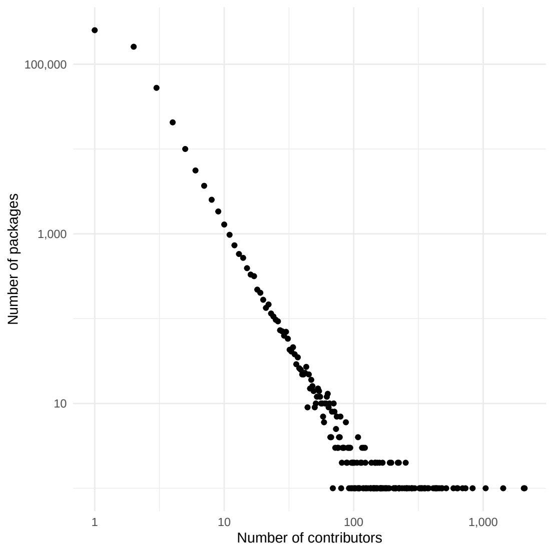
Software is developed in teams with contributions from a few coders, in some cases by hundreds even. As such it is an interesting first observation to see how the size of teams is distributed for our sample. Therefore, we plot the number of contributors per projects against number of projects which can be seen in Figure 1a. We observe that the relationship seems to represent a power law. That is, most software projects have only

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

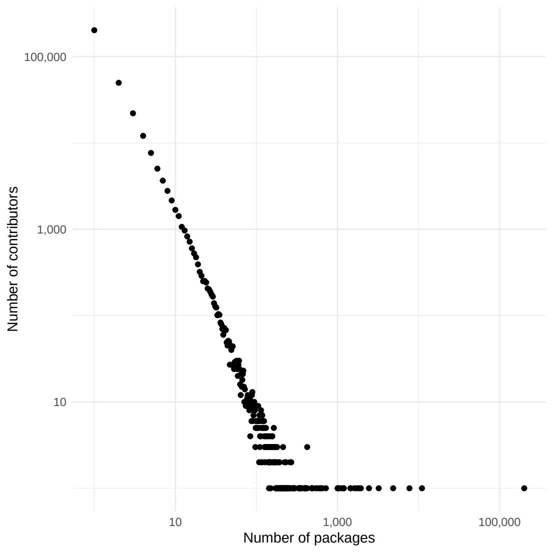
[9]Javascript, often abbreviated with JS, is used both on the client-side — rendered in web browsers to display websites — as well as server-side — to host website. See https://en.wikipedia.org/wiki/JavaScript for more information.

[10]`github.com`.

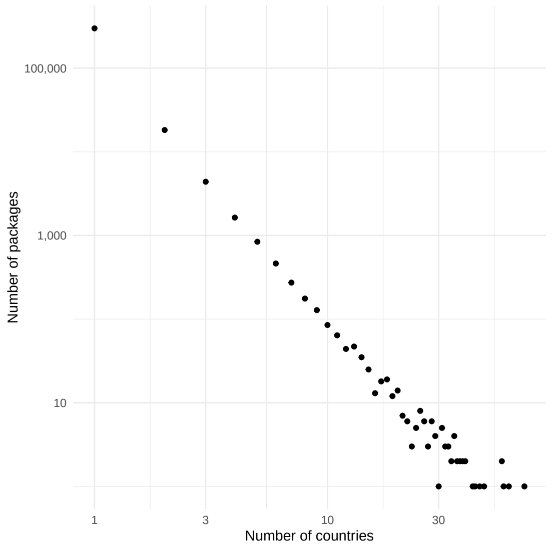
[11]Note that we modify the coding of the variable *common language* to be 1 for domestic flows.



(a) Number of contributors per repository



(b) Number of repositories per contributor



(c) Number of countries per repository

Figure 1: Distribution of the number of contributors and their home countries versus number of repositories

a few contributors while there are some which have many contributors. The flipside of the previous question is: To how many projects does each collaborator contribute. This is displayed in Figure 1b. Here, we observe a strong power law relationship.

Having established the general size contribution of teams, we now move to their geographical locations. Before considering specific packages, we present the location of contributors displayed on a world map in Figure 2. What becomes immediately obvious is that the development of software is strongly dominated by North America and Europe. It is easy to make out single clusters like Silicon Valley, the area around New York or Austin Texas. Looking beyond that it also seems that Brazil, China and India heavily contribute to open

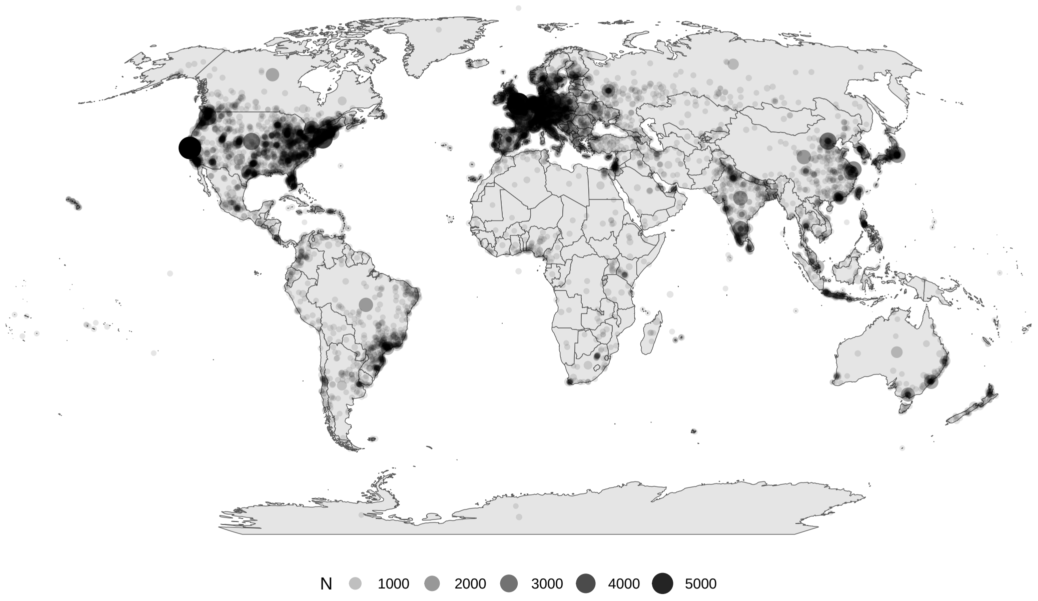


Figure 2: Developers per tenth of a square degree

source and finally it is possible to identify some of the tech hubs on the African continent like Cape town (SA), Lagos in Nigeria or Nairobi in Kenya. The individual self-reported locations of developers were geocoded and binned, such that each dot represents a rectangle of one tenth of a degree, latitude and longitude.[12]

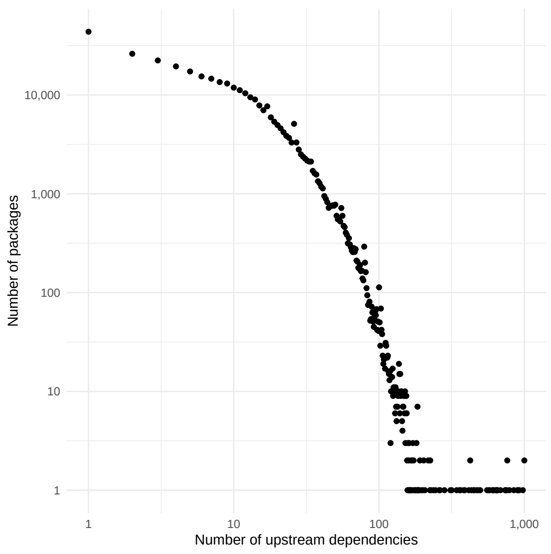
Moving beyond the distribution of developers world wide general, we consider the distribution of countries per package. This is displayed in Figure 1c. We observe again a relationship which might resemble a powerlaw.

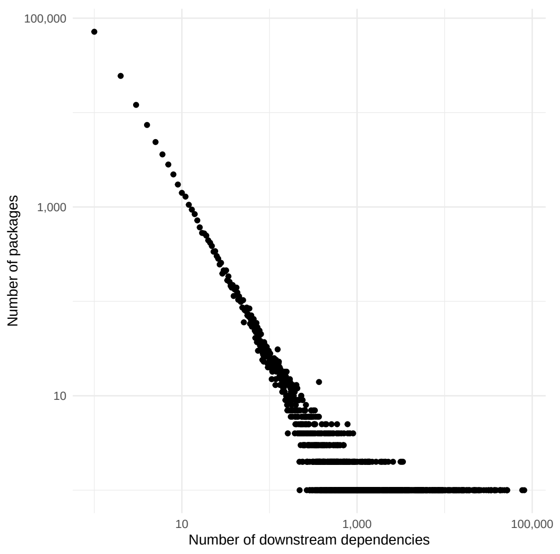
The main takeway is that 67.1% packages are produced in a single country while 28.5% share of projects developed in two countries, only 4% are produced over many sites.

3.3.2 Dependencies

Next, we turn to dependencies, starting with its distributional properties. In Figure 3a we depict the amount of other projects one project typically depends upon. We observe that most packages have at least one other project they depend upon which is followed by a decrease of amount of dependencies. Though, the concave decrease suggests that it might be beneficial to depend on a certain amount of projects but managing many at the same time becomes infeasible. This is a result which is consistent with the concept of *dependence hell* which refers to exactly this situation that occurs where dependencies cannot be efficiently managed anymore. The flipside in Figure 3b again depicts a strong

[12]At the equator this equal roughly a rectangle of 11km by 8km. The rectangle is smaller closer to the poles.





(a) Distribution of upstream dependencies **(b)** Distribution of downstream dependencies

Figure 3: Distribution of dependencies

Description	Value
Contributors (all)	320,721
Contributors (with country)	117,734
Mean amount projects per contributor	3.57
Mean amount projects per contributor (with country)	4.04
Median amount projects per contributor	1
Median amount projects per contributor (with country)	1
Unique countries among contributors	179
Average amount of contributors per country	2659.54
Median amount of contributors per country	88
Amount of software projects	517,779
Amount of software projetcts (atleast one country)	325460

Table 1: Descriptive Statistics

power law relationship. A plausible channel of explanation could be the idea of preferential attachment often observed in network like structures. This suggests that the more projects use a specific dependency, the likelihood of the next project to use the very same dependency is ceteris paribus higher.

3.4 Constructing synthetic locations

To address multinational software projects, we build synthetic locations which allow us to assign a dominant contributor to each of the projects. The dominant contributor can be thought of as representing an owner to the package and thus deriving the highest amount of utility from it. The algorithm to assign a synthetic location is:

1. Identify the contributors to each project.

Table 2: Top countries for developers of NPM software

Country	Share of developers	Share of projects	Log difference
United States	**0.272**	**0.330**	-0.194
Germany	0.069	0.069	-0.006
United Kingdom	0.063	0.068	-0.063
China	0.054	0.059	-0.091
Canada	0.043	0.042	0.027
France	0.042	0.041	0.023
India	**0.034**	**0.018**	0.639
Brazil	**0.029**	**0.019**	0.432
Netherlands	0.026	0.027	-0.055
Australia	0.025	0.029	-0.136
Japan	0.020	0.022	-0.076
Spain	0.018	0.017	0.056
Poland	0.017	0.015	0.127
Russia	0.016	0.012	0.327
Sweden	0.016	0.017	-0.081

Note: Ordering is based on the share of developers. Share of projects refers to amount of software projects in the country as identified by the synthetic locations. Note, that this table only considers projects with at least two contributors and only contributors who contribute to at least two projects.

2. From the identified contributors identify the set of contributors for which self-reported locations are available.
3. Assign the project to the contributor with the highest amount of contributions and thus place the synthetic location there as well.

In our dataset, we have $6,206$ unique combinations of longitude and latitude to which we assign repositories. Aggregating this to a country level leaves us with assigning repositories to 172 unique countries.

In table Table 2, we compare the amount of developers to the amount of projects that we assign to each country. The ordering is based on the number of developers. What is striking at first is that the US seems to have way more repositories than the amount of developers would suggest. Often the amount of developers and amount of packages however coincide. Finally, India and Brazil show an interesting pattern in that even though they have a strong developer base, they only few amount of projects.

3.5 Constructing trade and contribution flows

The synthetic locations of projects now allow us to build national-level dependency and contribution flows. For now, we focus on the extensive margin of both of these flows. These flows are conceptually quite easy to grasp with an example:

1. A dependency flow: If a project in the United States depends on a project located in

Hungary, we say there is a dependency flow from Hungary to the United States.

2. A contribution flow: If a contributor in the United States contributes to a project located in Germany, we say there is a contribution flow from the United States to Germany.

In our dataset, we will have $329,487$ nonzero-observations regarding flows of dependencies from countries and $150,821$ nonzero-observations for contribution flows on a collaborator-country level. Aggregated at the national level, we observe contribution for $3,165$ country pairs, and use of dependencies between $4,548$ pairs.

4 Estimation framework

Before we turn to our empirical exercise, let us introduce some structure to think about dependencies and collaboration in the process of development of OSS projects.

4.1 Theory of production and sourcing

Let i denote an individual software producer. This may be an individual developer or an organization developing and maintaining a piece of software. For simplicity, we assume single-product producers.

Given a need for some new functionality z, this producer has three major options. First, she can write the code herself to provide the needed functionality. Second, she can collaborate with other developers. Third, she can import an existing piece of software as a dependency.

We can study the first and second options jointly, as working on her own is a special case of not collaborating with anyone else. Dependency importing is special, however, and it will be studied separately.

The utility from custom written code, whether by the producer or by other collaborators, is

$$U_{iz,\mathrm{c}} = \max_j Q_j - \beta' \mathbf{d}_{ij} + \varepsilon_{izj}.$$

This depends on the quality of developer j, the vector of distance metrics between i and j, $\mathbf{d}_{ij}$, and an idiosyncratic matching cost ε_{izj}. We assume that ε_{izj} is distributed as Type-I extreme value, independently across producers, developers and functionality. This matching cost is unknown to the producer when making the choice. Note that j may be equal to i, in which case the producer will write the code herself.

The utility from importing a dependency k is

$$U_{iz,\mathrm{d}} = \max_k \alpha_k - \gamma' \mathbf{d}_{ik} + \xi_{izk}.$$

This depends on the quality of developer k, the vector of distance metrics between i and k, $\mathbf{d}_{ik}$, and an idiosyncratic matching cost ξ_{izk}. We assume that ξ_{izk} is also distributed as Type-I extreme value, independently across producers, coders and functionality, and independently from ε_{izj}.

Given the distributional assumptions, the probability of choosing a developer j is given by the logit formula,

$$\pi_{izj} = \frac{\exp(Q_j - \beta'\mathbf{d}_{ij})}{\sum_l \exp(Q_l - \beta'\mathbf{d}_{il})}.$$

Aggregating across different functionalities z, this formula also provides the *share* of functionalities that are assigned to developer j.

Highly skilled developers (high Q_j) that are in close proximity to i (low $\mathbf{d}_{ij}$) will write a larger fraction of the code, or otherwise collaborate more intensively.

By similar reasoning, the share of functionalities solved by importing dependency k is given by

$$\Pi_{izk} = \frac{\exp(\alpha_k - \gamma'\mathbf{d}_{ik})}{\sum_l \exp(\alpha_l - \gamma'\mathbf{d}_{il})}.$$

Good quality packages (high α_k) and those developed in close proximity of the producer (low $\mathbf{d}_{ik}$) will be imported in higher shares.

The ultimate make-or-use decision then compares $U_{iz,\text{c}}$ to $U_{iz,\text{d}}$. The quality of the product is then

$$\alpha_i = \int_z \max\{U_{iz,\text{c}}, U_{iz,\text{d}}\} dz.$$

In this paper we only study direct, one-step collaboration and dependencies, but see Békés et al. (2023) for a study of the full network.

We are particularly interested in the role of geographic frictions on the decision to use inputs, captured by the coefficient vectors β and γ. Given the functional form assumptions we made, these can be estimated in traditional log-linear gravity equations.

4.2 Estimated equation

We estimate the same standard gravity equation for dependency flows and collaboration flows. Dependency flows are binary in the sense that we either observe that one project depends on another or not. Collaboration flows, which we measure by the number of commits, also capture the intensity of collaboration. We measure both flows at the country-level as well as the granular, software package level.

Take a software package i whose developer resided in country d (remember that we model software package developers as producers). The equations above characterize the probability of this package using input from other packages k or developers j. The dummy variable D_{ik} captures the dependency on package k. The count variable q_{ij} captures the number of commits contributed by developer j.

In the aggregate analysis, we aggregate both dependency and collaboration flows by origin country o and destination country d. That is, we define the number of dependencies imported by producer i from country o as

$$x_{io} = \sum_{k \in o} D_{ik} = \nu_i \cdot \mu_o \cdot \exp(-\gamma' \mathbf{d}_{io}), \tag{1}$$

where the second equation follows from the logit choice model. The fixed effect ν_i captures the availability of other dependencies and $\mu_o = \left[\sum_{k \in o} \exp(\alpha_k)\right]$ measures the total quality of available software packages in country o (akin to a "multilateral resistance term" Anderson and van Wincoop, 2003). Aggregating over producers in the same destination country d gives our estimable equation

$$X_{do} = \sum_{i \in d} x_{io} = \left[\sum_{i \in d} \nu_i\right] \cdot \mu_o \cdot \exp(-\gamma' \mathbf{d}_{do}), \tag{2}$$

and we have made use of the fact that distance is only country specific but does not depend on individual producers. This is a standard gravity equation which we estimate with a Poisson Pseudo Maximum Likelihood estimator (Santos Silva and Tenreyro, 2006).

Similarly, the intensity of collaboration also follows a gravity equation,

$$Q_{do} = \sum_{i \in d} \sum_{j \in o} q_{ij} = N_i \cdot M_o \cdot \exp(-\beta' \mathbf{d}_{so}). \tag{3}$$

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.

The difference in time zones captures the ability to collaborate. It is measured as common business hours. We assume there are 8 working hours everywhere, and we counted how many hours overlap. This would be 8 between Bielefeld and Budapest (both GMT+1), 2 hours between Budapest and Boston (GMT-5), and it's 0 between Bielefeld and Berkeley (GMT-8) that is 9 hs behind Central Europe. Similarly, it is also 0 between the US East Coast and China (GMT+8).

5 Estimation results

We now estimate equations (2) and (3) to understand how distance might influence the formation of production networks described earlier. We do this with the previously constructed contribution- and dependency flows. We first present estimates at a nationally aggregated level, followed by project (repository) level estimates.

5.1 National level

We have 27,384 potential flows between countries regarding contributions, and 15,510 on dependencies. There is a gap because for some country pairs in the dependency flows all observations are zero and accordingly uninformative and drop out of the analysis when including fixed effects.

Let us start with intermediate good imports (dependencies). The results can be found inTable 3. Column (1) has the (log) distance between country pairs and a same country dummy, (2) add a common language indicator, (3) repeats the exercise excluding same country imports. Finally, column (4) replaces physical distance with the number of common business hours. All models have both origin and destination country level fixed effects.

In the baseline model of column (1), being in the same country more than doubles ($\exp(0.9) - 1 = 1.45$) the number of dependencies imported. Beyond that, distance has a marginal role. In fact, we observe that the distance parameter is for multiple specifications close to zero and non-significant. This seems to suggest that the distance between projects, as measured by the synthetic locations, does not introduce frictions. Rather, the determinants of dependencies need to be driven by other variables. Adding common language actually makes a difference, it increases flows. However, most of this comes from being in the same country (column 3), but shared language still adds 10 percent when considering flows across different countries (column 3). Finally, common business hours has a small but negative point estimate – it is less likely to import a dependency when coders share more business hours.

It turns out that one key issue with these results is that the role of the United States is overwhelming. To investigate we repeated all estimations with the projects based in the US, and without US as a country of production. Results are shown in Table 4. Indeed, without US, it is really the same country that matters, increasing flows by more than four (!) times ($\exp(1.68) - 1 = 4.36$). Estimates of the distance elasticity in miles or shared common hours are now zero, and so is common language.

Next, we estimate the same gravity model with collaboration flows as the dependent variable. Contribution is measured as the total number of commits done by developers in country j, contributing to a repository maintained in country i.

Table 3: Dependency flows on a national level.

	(1)	(2)	(3)	(4)
log(distance)	-0.06*	-0.03	0.01	
	(0.03)	(0.03)	(0.02)	
Same country	0.90***	0.79***		1.07***
	(0.30)	(0.26)		(0.24)
Common language		0.24***	0.09**	0.31***
		(0.08)	(0.04)	(0.09)
Common Business hours				-0.04***
				(0.01)
Origin country FE	Yes	Yes	Yes	Yes
Destination country FE	Yes	Yes	Yes	Yes
Observations	15,510	15,510	14,420	15,510
Pseudo R^2	0.98387	0.98420	0.99336	0.98470

Clustered (iso_o & iso_d) standard-errors in parentheses
*Signif. Codes: ***: 0.01, **: 0.05, *: 0.1*

Table 4: Dependency flows on a national level excluding USA.

	(1)	(2)	(3)	(4)
log(distance)	-0.09	-0.09	0.03	
	(0.06)	(0.06)	(0.03)	
Same country	1.68***	1.68***		1.94***
	(0.31)	(0.28)		(0.22)
Common language		-0.01	0.05	0.03
		(0.08)	(0.04)	(0.06)
Common Business hours				0.00
				(0.01)
Origin country FE	Yes	Yes	Yes	Yes
Destination country FE	Yes	Yes	Yes	Yes
Observations	14,796	14,796	13,464	14,796
Pseudo R^2	0.96788	0.96788	0.97987	0.96733

Clustered (iso_o & iso_d) standard-errors in parentheses
*Signif. Codes: ***: 0.01, **: 0.05, *: 0.1*

The results are displayed in Table 5. The most striking difference is that now the distance coefficient is negatively associated with contribution flows with actually a large point estimate. This time common business hours also work as a meaningful metric – one additional common business hour is associated with 7% more commits. This implies that even in the production of this entirely digital good, distance matters in forming who works with whom. Furthermore, we find the expected positive signs for the use of a common language.

The same-country dummy is now huge, actually, suggesting that collaboration happens mostly within country. The point estimate of 3.06 corresponds to a whopping 20 (!) times higher activity for same-country coders. Without the US, it's actually over 50 times

Table 5: Contribution flows on a national level.

	(1)	(2)	(3)	(4)
log(distance)	-0.59***	-0.51***	-0.12***	
	(0.04)	(0.04)	(0.02)	
Same country	3.06***	2.60***		4.32***
	(0.38)	(0.30)		(0.36)
Common language		0.92***	0.21***	1.16***
		(0.12)	(0.03)	(0.03)
Common Business hours				-0.01
				(0.03)
Origin country FE	Yes	Yes	Yes	Yes
Destination country FE	Yes	Yes	Yes	Yes
Observations	27,384	27,384	16,265	27,384
Pseudo R^2	0.98174	0.98328	0.95878	0.97999

Clustered (iso_o & iso_d) standard-errors in parentheses
*Signif. Codes: ***: 0.01, **: 0.05, *: 0.1*

higher.[13]

5.2 Project level estimation

In this subsection, we follow a more granular approach. Since we have contributor and project-level data, we only aggregate one dimension to the national level. The underlying questions remain the same however. The granular flow of dependencies considers projects as the importers which decide to import from given countries.

Here we estimate a very similar model to the previous one, with dependency then collaboration flows as dependent variables. On the right hand side, we now have:

$$X_{od} = \exp\left([\boldsymbol{\tau}_{od}]' \boldsymbol{\beta} + \xi_{po} + \nu_d\right) + \varepsilon_{od}. \tag{4}$$

What is different to (2) is that with (ξ_{po}) we now have project-level fixed effects for a dependency estimation and contributor (person) level for the contributor model. That means we first exploit variation in dependencies imported by a project, followed by variation across the projects a person contributes to.

The results are displayed in Table 6 as well as Table 7. We actually see similar results to the aggregate models. This suggests that there may be no weird selection across projects within a country to be responsible for what we see. Once again distance matters for contributions only. Being in the same country helps pick a dependency but is absolutely essential for software development.

[13] See Appendix subsubsection A.2.1.

Table 6: Dependency flows on a granular level.

	(1)	(2)	(3)	(4)
log(distance)	-0.02**	-0.01	0.01	
	(0.01)	(0.01)	(0.01)	
Same country	0.38***	0.34***		0.43***
	(0.04)	(0.03)		(0.03)
Common language		0.09***	0.03	0.11***
		(0.03)	(0.02)	(0.03)
Common Business hours				-0.01**
				(0.00)
D FE	Yes	Yes	Yes	Yes
Destination country FE	Yes	Yes	Yes	Yes
Observations	329,487	329,487	290,194	329,487
Pseudo R^2	0.45666	0.45687	0.42752	0.45698

Clustered (proj_repo_name & iso_d) standard-errors in parentheses
*Signif. Codes: ***: 0.01, **: 0.05, *: 0.1*

Table 7: Contribution flows on a granular level.

	(1)	(2)	(3)	(4)
log(distance)	-0.66***	-0.59***	-0.25***	
	(0.07)	(0.07)	(0.08)	
Same country	3.51***	3.13***		4.97***
	(0.29)	(0.24)		(0.33)
Common language		0.76***	-0.03	1.06***
		(0.20)	(0.18)	(0.20)
Common Business hours				0.01
				(0.03)
Developer FE	Yes	Yes	Yes	Yes
Destination country FE	Yes	Yes	Yes	Yes
Observations	19,186,078	19,186,078	4,339,145	19,186,078
Pseudo R^2	0.92642	0.92696	0.64597	0.92447

Clustered (login & iso_d) standard-errors in parentheses
*Signif. Codes: ***: 0.01, **: 0.05, *: 0.1*

6 Conclusion

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.

Using data from a software code versioning system and a software package manager in a novel empirical framework, we find that for sourcing intermediate inputs (software dependencies), physical or time distance does not play a role beyond using code form coders in our own country. However, for collaboration — as shown in other settings — the home bias is huge, and distance matters as well.

References

Anderson, James E and Eric van Wincoop, “Gravity with Gravitas: A Solution to the Border Puzzle,” *Am. Econ. Rev.*, March 2003, *93* (1), 170–192.

_ , **Ingo Borchert, Aaditya Mattoo, and Yoto V Yotov**, “Dark costs, missing data: Shedding some light on services trade,” *European Economic Review*, 2018, *105*, 193–214.

Antràs, Pol and Davin Chor, “Global value chains,” *Handbook of international economics*, 2022, *5*, 297–376.

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.

Bahar, Dany, “The hardships of long distance relationships: time zone proximity and the location of MNC’s knowledge-intensive activities,” *Journal of International Economics*, 2020, *125*, 103311.

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.

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, Julian Hinz, and Miklós Koren, “Bugs ❁,” Technical Report, Unpublished 2023.

Conte, Maddalena, Pierre Cotterlaz, Thierry Mayer et al., *The CEPII gravity database*, CEPII, 2022.

Ductor, Lorenzo, Marcel Fafchamps, Sanjeev Goyal, and Marco J Van der Leij, “Social networks and research output,” *Review of Economics and Statistics*, 2014, *96* (5), 936–948.

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 and Thierry Mayer, “Gravity equations: Workhorse, toolkit, and cookbook,” in “Handbook of international economics,” Vol. 4, Elsevier, 2014, pp. 131–195.

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

Hsieh, Chih-Sheng, Michael D Konig, Xiaodong Liu, and Christian Zimmermann, "Superstar Economists: Coauthorship networks and research output," Technical Report, CEPR Discussion Paper No. DP13239 2018.

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.

Leamer, Edward E and Michael Storper, "The economic geography of the internet age," in "Economy," Routledge, 2017, pp. 431–455.

Lerner, Josh and Jean Tirole, "The economics of technology sharing: Open source and beyond," *Journal of Economic Perspectives*, 2005, *19* (2), 99–120.

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.

Stein, Ernesto and Christian Daude, "Longitude matters: Time zones and the location of foreign direct investment," *Journal of International Economics*, 2007, *71* (1), 96–112.

van der Wouden, Frank and Hyejin Youn, "The impact of geographical distance on learning through collaboration," *Research Policy*, 2023, *52* (2), 104698.

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.[14] 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,[15] web browsers such as Mozilla Firefox,[16] database systems such as MySQL,[17] machine learning and AI frameworks such as TensorFlow and PyTorch,[18] and web development platforms such as WordPress.[19]

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.[20] 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.[21] 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

[14] See https://opensource.org/osd.
[15] See https://www.linuxfoundation.org/.
[16] See https://www.mozilla.org/en-US/firefox/new/.
[17] See https://www.mysql.com/.
[18] See https://www.tensorflow.org/ and https://pytorch.org/.
[19] See https://wordpress.com/.
[20] See https://www.gnu.org/philosophy/free-sw.en.html.
[21] 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[22] 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.2 Additional results

A.2.1 Excluding USA

Appendix Table A1: Contribution flows on a national level excluding USA.

	(1)	(2)	(3)	(4)
log(distance)	-0.51***	-0.50***	-0.16***	
	(0.06)	(0.06)	(0.03)	
Same country	4.07***	3.75***		5.30***
	(0.30)	(0.32)		(0.27)
Common language		0.41**	0.15***	0.53***
		(0.20)	(0.02)	(0.16)
Common Business hours				0.07**
				(0.03)
iso_o FE	Yes	Yes	Yes	Yes
iso_d FE	Yes	Yes	Yes	Yes
Observations	26,568	26,568	15,050	26,568
Pseudo R^2	0.98612	0.98627	0.88175	0.98415

Clustered (iso_o & iso_d) standard-errors in parentheses
*Signif. Codes: ***: 0.01, **: 0.05, *: 0.1*

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

Appendix Table A2: Dependency flows on a granular level excluding USA.

	(1)	(2)	(3)	(4)
log(distance)	-0.03***	-0.03***	0.01	
	(0.01)	(0.01)	(0.01)	
Same country	0.49***	0.48***		0.55***
	(0.06)	(0.05)		(0.04)
Common language		0.02	0.01	0.02
		(0.02)	(0.01)	(0.02)
Common Business hours				0.00*
				(0.00)
proj_repo_name FE	Yes	Yes	Yes	Yes
iso_d FE	Yes	Yes	Yes	Yes
Observations	196,711	196,711	175,390	196,711
Pseudo R^2	0.15358	0.15358	0.10868	0.15347

Clustered (proj_repo_name & iso_d) standard-errors in parentheses
*Signif. Codes: ***: 0.01, **: 0.05, *: 0.1*

Appendix Table A3: Contribution flows on a granular level excluding USA.

	(1)	(2)	(3)	(4)
log(distance)	-0.71***	-0.70***	-0.37***	
	(0.09)	(0.09)	(0.09)	
Same country	4.11***	3.98***		5.99***
	(0.42)	(0.45)		(0.42)
Common language		0.18	-0.09	0.40**
		(0.20)	(0.24)	(0.20)
Common Business hours				0.13***
				(0.03)
login FE	Yes	Yes	Yes	Yes
iso_d FE	Yes	Yes	Yes	Yes
Observations	12,260,646	12,260,646	1,997,761	12,260,646
Pseudo R^2	0.94049	0.94050	0.61259	0.93858

Clustered (login & iso_d) standard-errors in parentheses
*Signif. Codes: ***: 0.01, **: 0.05, *: 0.1*