Geography of Open Source Software Production*

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

June 2023

Preliminary and incomplete — please do not cite or circulate

Abstract

This paper analyses the production function of open source software and highlights its international aspects. While the modern digital infrastructure does rely on open source software, the production of it is not well understood. We propose a production function which understands open source software as the outcome of two interrelated networks. The *within* and the *between* network, where the former relates to collaboration and the later to inter-project dependencies. Thus, we describe the production of open source software as the outcome of a team production function with intermediary inputs. By using novel data sources we empircally describe properties of the `JavaScript` open source community. Subsequently, we use this data to investigate the drivers of those networks. Our results suggest that geographical distance does not matter for the *between* but the *within* network. Thus, collaboration is driven by geographical proximity. Having a common language seems to be an important driver for both styles of networks.

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

*Funded by the European Union. 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

The modern digital infrastructure relies partially on open source software (OSS). Maching Learning in Python is often performed using `TensorFlow`; Developing web applications in Python is facilitated by `Flask` and `React` helps develop User Interfaces (UI) in `JavaScript`. Though, OSS is often an integral part of software supply chains, the underlying production networks are not well studied. We posit that the production of OSS can be understood as the outcome of a team production function consisting of two networks. One network describes the team composition and as such builds the links between individual developers and software projects. The other network describes links between projects itself. While the first network seems intuitively clear. The second is driven by the observation that OSS is typically not standalone but rather builds on already existing software in the form of `dependencies`. We explicitly appreciate this fact and propose to understand dependencies in the context of value chains. A defining feature of OSS is that anyone with a laptop and an internet connection may contribute to it. Importantly, this also means that participation is not necessarily defined by geography. However, given the success of the gravity model, we aim to assess here as well how geography might shape either of the two networks. We consider several novel data sources which allow us to tackle this question. As `JavaScript` is the biggest programming language world wide, we analyse its production network in detail. Before turning to regression results, we provide descriptive statistics. Our regression results reveal that in fact geography does not seem to play a role for links between projects. However, within projects (i.e. who collaborates with whom), we find singificant negative distance effects. The remainder of this paper is structured as follows. In section 2 we describe the production of OSS and provide the team production function, in section 3 we provide an overview of used data sources, section 4 provides descriptives statistics, the empirical design and results are discussed in section 5 and **??**. After some robustness checks in section 6 we conclude in section 7.

2 Background

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.

2.1 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 software widely used today include:

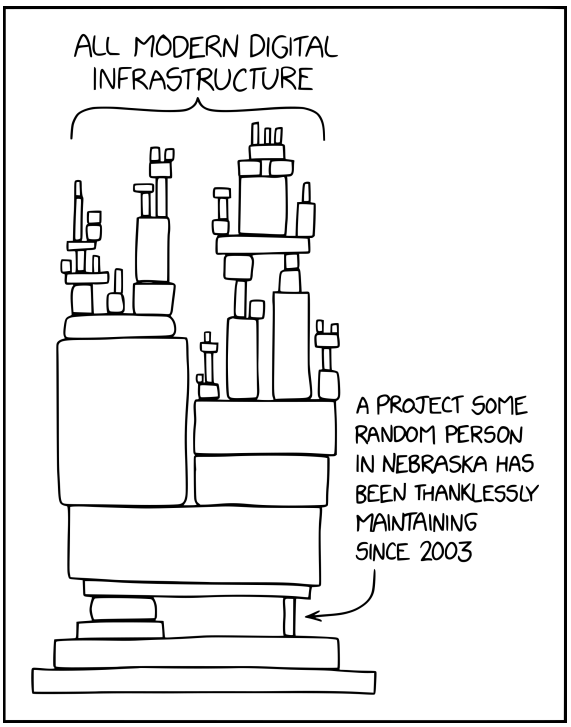


Figure 1: Dependency (Source: XKCD, https://xkcd.com/2347/)

- 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[1] 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. From an economic point of view, OSS poses an interesting application. A good which is subject to high skill requirement but costless consumption. Furthermore, it is common of open source projects

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

to rely on each other, so called dependencies. That is a software project may import all or part of the functionalities another project has. This idea can be related to traditional models of Global Value Chains and is illustrated by Figure 1. We will try to capture this in a production function with team collaboration and network links between projects.

A team production function The following paragraphs introduce a very stylized production function of which the purpose for now mostly lies in guiding and structuring our line of thought. Open source software production can be thought of as the composition of two related networks. Throughout this paper, we will call the *within*-network the bipartite network formed by collaborators on different projects. This approach follows Hsieh et al. (2018) who study the production of knowledge goods, more specifically academic research. The *between*-network refers to connections of software projects irrespective of their collaborators. Thus, we extend Hsieh et al. (2018) by another dimension which can be thought of as intermediary products and accordingly relates to the Global Value Chain literature. Let $\mathcal{G}$ denote the bipartite network which considers collaborators $\mathcal{I} = \{1, \dots I\}$ and software projects $\mathcal{P} = \{1, \dots J\}$. Note, that $\mathcal{G}$ can be represented by a $I \times J$ matrix such that collaborators are in rows and projects are in columns. Let a single element of $\mathcal{G}$ be denoted by $g_{ij} \in \{0, 1\}$ and refer to whether developer i works on project j. Writing this into a simple production function:

$$P_j = \sum_{i=1}^{I} g_{i,j} e_{ij} \alpha_{ij} \tag{1}$$

, where now a software projects is simply the sum of effort e_{ij} the single contributors exert. The parameter α_{ij} is a simple factor augmentation describing different project-specific skills of contributors.

Dependencies - intermediary inputs We extend the previously proposed team production function by intermediary inputs. This is captured by another network that can in turn again be represented as a matrix. Let $\mathcal{D}$ be the *between*-network of software projects. This matrix has the dimensions of available software projects $J \times J$.[2] Note, that this network is a directed network since one software project may import another while the other direction is then excluded in our model. However, it is possible that dependency chains build cycles by indirect dependence. Let $d_{m,k}$ be the single elements of $\mathcal{D}$, where $m = k = 1, \dots J$. Further, $d_{mk} = 1$ then signals that P_m declares a dependency on P_k. Note that we impose $d_{mm} = 0$ as a dependency on itself is not a well-defined concept. Augmenting the production function in Equation 1 further allows us to write:

[2]Where the diagonal elements represent dependencies on itself.

$$P_j = \sum_{i=1}^{I} g_{i,j} e_{ij} \alpha_{ij} + \sum_{j=1}^{J} d_{jk} P_k \phi_{jk} - \sum_{j=1}^{J} d_{jk} c_{jk} \tag{2}$$

, where $\phi_{jk} \in \mathbb{R}$ represents compatibility. The intuition is that not all projects can be useful to all other projects. Especially, two packages which want to solve the same problem by different ways should be highly incompatible and thus have a negative ϕ_{jk}. Note, that the compatibility is non-symmetric such that $\phi_{jk} \neq \phi_{kj}$. Our main interest lies in empirically understanding the c_{jk} term.

Utility of developers The collaboration network is formed by developers. Each of the developers maximizes their individual utility which is symmetric for $\forall i \in \mathcal{I}$. Let the utility of a single developer be written as:

$$U_i = \sum_{j}^{J} g_{ij} P_j - \sum_{j=1}^{J} e_{ij}^2 - \sum_{j}^{J} g_{ij} \tau_{ij} \tag{3}$$

, where a developer receives utility for contributing to a good project but has a quadratic cost associated with exerted effort. Furthermore, τ_{ij} is a parameter which captures a network based cost to enter into collaboration on a particular proejct j. In our context, we think of this τ_{ij} being an important determinant in describing the network $\mathcal{G}$. Similar to the production function before, we are for now mostly interested in the τ_{ij} term.

Github - social media for collaborative efforts GitHub is a widely-used web-based platform for version control and collaboration, primarily designed for software development projects. It allows developers to host and manage their code repositories, track changes, and work collaboratively on projects with others. Users can create branches to work on specific features or bug fixes, and merge them back into the main codebase. GitHub provides tools for issue tracking, documentation, and code review, enabling effective project management and quality assurance. It also fosters an active open-source community by facilitating contributions, allowing developers to share their work, and encouraging collaboration among peers. Beyond everything that is related to the development of software, `Github` functions as a social network for developers. This means that developers often self-report their location, interests and companies they work for. This allows us to study this variables in more detail.

JavaScript and NPM `JavaScript (JS)` is one of the most popular programming languages commonly used in the development of web applications and mobile apps. According

to reports of w3techs.com[3] 98.7 percent of webpages employ JS in one form or another and a large share of mobile apps rely to some extent on it. Part of the JS infrastructure is NPM which is its dominant package manager. A package manager in the realm of software development can be thought of as a library. It organises, categorizes and provides access to different software projects. For our purpose, NPM lends itself nicely to identify a set of relevant software projects. Given the central role of JS, we deem it appropriate to limit ourselves to it for the first round of results and expect that these translate nicely to other languages and industries as well. Though, we will also explore robustness of our results by considering other languages.

Geography of production Geographical locations and distances have proven to be an important factor in understanding production networks of traditional goods as well as services. Digital goods, software in our case, has the potential to be geographically entirely unbound. We aim to test this for both styles of networks laid out before.

3 Data

In this paper, we make use of a range of different data sources. 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 "labor" used to produce it.

We combine a number of different data sources to construct a sample which allows us to estimate the determinants of network formation as discussed in the previous section. We use Libraries.io[4], and the API endpoints of Github[5], NPM[6] and Nominatim[7].

Libraries.io is a data collection project which provides access to 32 different package managers acrooss three different code repositories. It allows to identify github repositories of individual packages and maps dependencies for several versions of packages. It builds

[3] Link last accessed on 10/07/2023
[4] Libraries.io last accessed on 07/07/2023
[5] api.github.com last accessed on 07/07/2023
[6] npmjs.com last accessed on 07/07/2023
[7] nominatim.org last accessed on 07/07/2023

our main data source for the dependencies and is used to define our sample. The latest completely recorded data for the latest data dump is 2019.

Nominatim is an open source API endpoint which maps locations based on open street map. We use this to map the self-reported locations of users on `Github` to geographical locations.

Github The `Github`-API serves multiple purposes for us. We retrieve user location data, assign contributors to specific software projects and finally also determine the extent to which individual users contribute to a specific package.

NPM is the dominant package manager for `JavaScript`. Beyond that `NPM` also overs an API endpoint which allows us to access the download statistics of individual `JavaScript` projects which are hosted on `NPM`.

In the end we have a sample of 217618 contributors and 298973 software projects and their contributions to each of those projects.

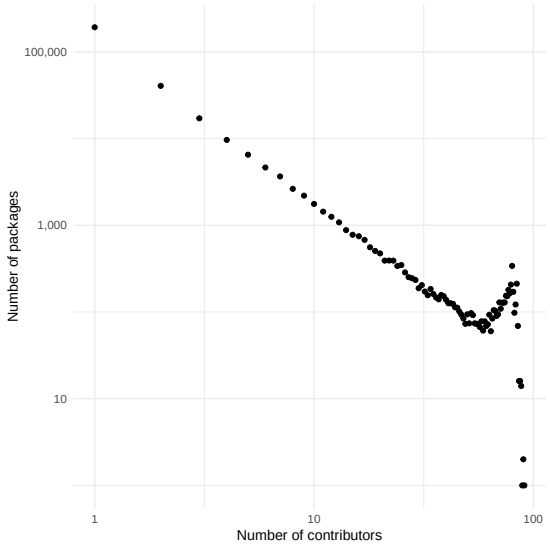
4 Descriptive statistics

We first use the data described in section 3 to plot some simple descriptive statistics about the composition of collaboration (the *within*-network) and dependencies (the *between*-network). Later in this section, we use the collected data to build synthetic locations for software projects to place them on the geographical landscape.

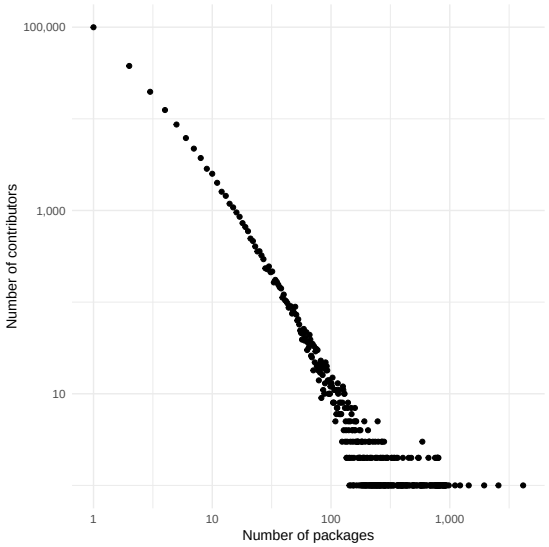
4.1 Descritptives of the *within*-network

Software is developed in teams. As such it is an intersting 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 2a. We observe that the relationship seems to represent a powerlaw. That is, most software projects have only a few contributors while there are some which have many contributors. Note, that this powerlaw is by no means perfect as for extreme values there is a reversal. This relationship merits further exploration especially corrected for the project quality. For the flipside of the previous question: To how many projects does each contributor contribute to. This is displayed in Figure 2b. Here, we observe a strong powerlaw like relationship.

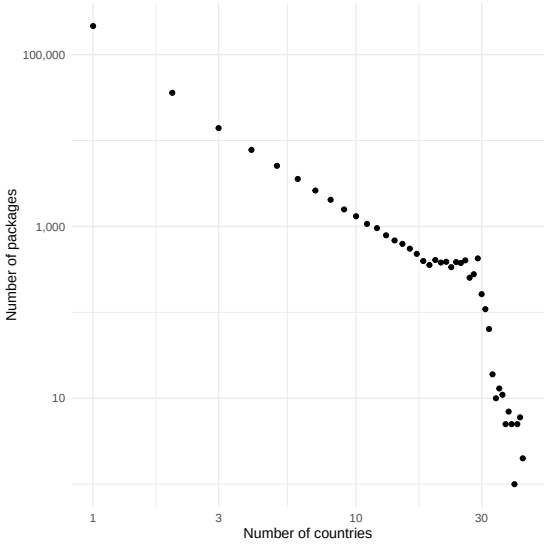
Having established the general size contribution of teams, we now move to their geographical locations. Before considering specific packages, we present the location of contributors displayed on a world map in Figure 3. What becomes immediately obvious is that the development of software is strongly dominated by North America and Europe. It is easy



(a) Number of contributors per repository



(b) Number of repositories per contributor



(c) Number of countries per repository

Figure 2: Distribution of the number of contributors and their home countries versus number of repositories

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 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.[8]

Moving beyond the distribution of developers world wide general, we consider the dis-

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

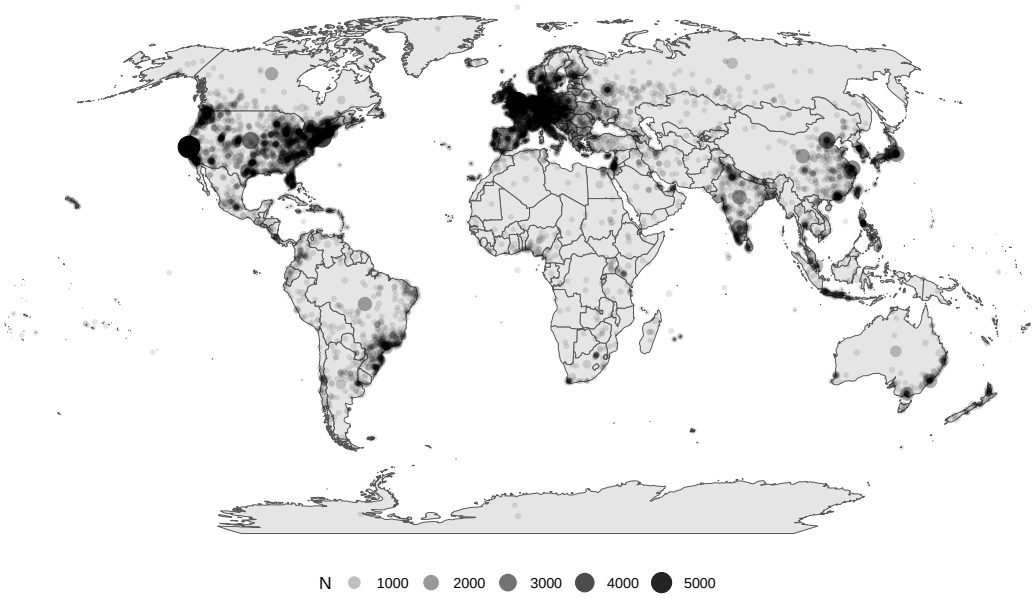


Figure 3: Developers per tenth of a square degree

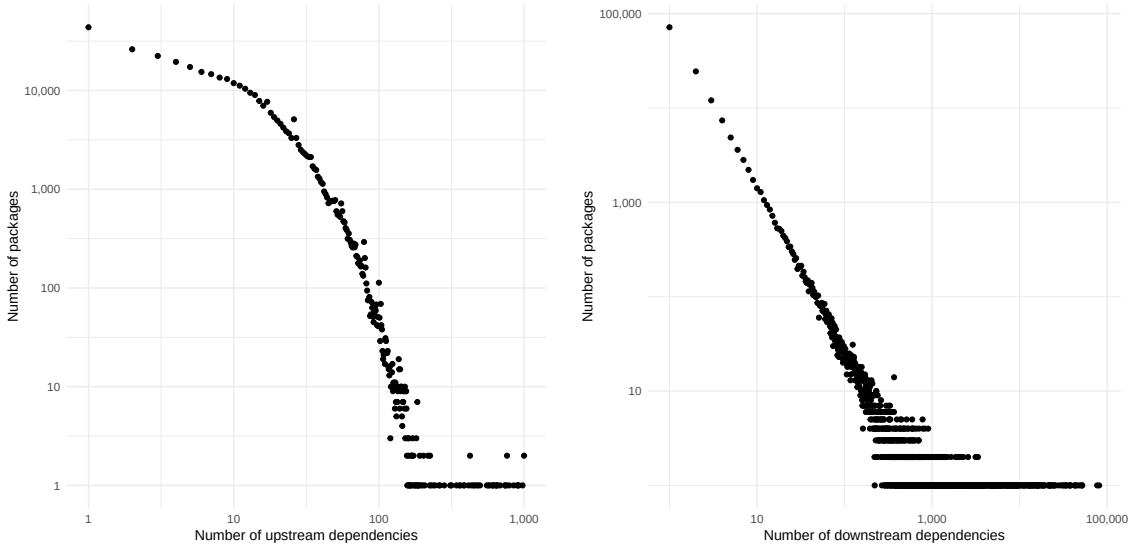
tribution of developers per package. This is displayed in Figure 2c. We observe again a relationship which might resemble a powerlaw like decline with a plateu in the middle. The main takeway is that most packages are produced in a single country while there is a relevant amount of projects developed jointly across borders.

4.2 Descriptives of the *between*-network

Moving on to the *between*-network perspective, we first of all consider the distributional properties of the dependcy structures. In Figure 4a we depict the amount of other projects one project typically depends upon. We observe that most packages have atleast 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 4b again depcits a strong powerlaw like relationship. A plausible channel of explanation could be the idea of preferential attachment often observed in network like structures.

4.3 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



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

Figure 4: Distribution of dependencies

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.
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 table Table 1, we compare the amount of developers to the amount of projects which we assign to each country. The ordering is based on the amount 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 amount of developers and amount of packages however coincide. Finally, India and Brazil show an intersting pattern in that even though they have a strong developer base, only few amount of projects.

4.4 Constructing trade and contribution flows

The synthetic locations of projects now allows 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

Table 1: Top countries for developers of NPM software. 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.

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

Notes: This is where authors provide additional information about the data, including whatever notes are needed.

located in Germany, we say there is a contribution flow from the United States to Germany.

Note again that the first point refers to the *within*-network and the second to the *between*-network.

5 Empirical approach

We estimate gravity equations to understand how geographical distance might influence the formation of propduction networks described earlier. We do this with the previously constructed contribution- and trade flows. Thus, we have one *within* and one *between* gravity equation.

Gravity equation A standard gravity equation of international economics relates bilateral flows to characteristics of the origin and destination countries, as well as bilateral frictions. Our focus is on the latter, as our interest is in uncovering those bilateral forces that shape the global production of software. We therefore estimate a standard structural gravity equation

$$X_{od} = \exp\left([\boldsymbol{\tau}_{od}]'\,\boldsymbol{\beta} + \xi_o + \nu_d\right) + \varepsilon_{od}.$$

The dependent variable is the flows from origin (o) to destination (d). Equation (5)

includes fixed effects ξ_o and ν_d to purge all origin and destination specific factors. The actual variables of interests are given by the vector $\boldsymbol{\tau}_{od}$ and its corresponding coefficients $\boldsymbol{\beta}$. We consider standard friction like bilateral distances, common languages, but also policy variables such as joint membership in the EU or WTO that generally make bilateral interactions, especially commercial ones, more convenient. We estimate equation (5) with a (Pseudo-) Poisson Maximum Likelihood estimator, as is common in the related literature. *Comment: We will explore other distance measures like hour differences which should play a role in terms of collaboration.*

Between gravity To measure the determinants of the *between*-network determinants, we consider the previously constructed bilateral trade flows on a national level. The results can be found inTable 2. 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 location does not introduce frictions. Rather, the determinants of dependencies need to be driven by other variables. These could include the quality of the package, the compatibilities and the market structure as a whole driven by preferential attachement. *Comment: This is still to study and rationalize.*

Table 2: Gravity regression for extensive margin

	(1)	(2)	(3)	(4)
log(distance)	0.01			0.00
	(0.01)			(0.01)
WTO		0.36		0.37
		(0.57)		(0.57)
EU		-0.12*		-0.11*
		(0.07)		(0.06)
FTA		0.04*		0.03
		(0.03)		(0.03)
Common language			0.06***	0.05**
			(0.02)	(0.02)
origin FE	Yes	Yes	Yes	Yes
destination FE	Yes	Yes	Yes	Yes
Observations	17,670	17,670	17,442	17,442
Pseudo R^2	0.98913	0.98919	0.98915	0.98920

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

Within gravity Similar to the estimation of the *between*-network, we now estimate the *within* counterpart exploring similiar contol variables. The results are displayed in Table 3. The most striking difference is that now the distance coefficient is negatively associated with contribution flows. This implies that even in the production of this entirely digital good, distance matters in forming who works with whom. Furhtermore, we find the expected positive signs for the use of a common language.

Comment: TODO: Explore the negative coefficients of the policy variables.

Table 3: Within gravity equation for extensive margin

	(1)	(2)	(3)	(4)	(5)
log(distance)	-0.09***			-0.10***	-0.07***
	(0.01)			(0.02)	(0.01)
WTO		0.05		0.01	
		(0.34)		(0.33)	
EU		-0.08		-0.14***	
		(0.05)		(0.05)	
FTA		0.14***			
		(0.03)			
Common language			0.23***	0.16***	0.17***
			(0.03)	(0.03)	(0.04)
origin FE	Yes	Yes	Yes	Yes	Yes
destination FE	Yes	Yes	Yes	Yes	Yes
Observations	27,602	27,602	26,642	26,642	26,642
Pseudo R^2	0.98109	0.98086	0.98097	0.98131	0.98124

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

6 Robustness

We explore several robustness checks:

- More granular approach than the national level.
- Different algorithm to map the software packages which is based on majority voting not only of the highest bidder but rather all contributors.
- A different software language like `Python`

7 Conclusion

To be added.

References

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.

Hsieh, Chih-Sheng, Michael D Konig, Xiaodong Liu, and Christian Zimmermann, "Superstar Economists: Coauthorship networks and research output," 2018.

A Appendix

B Data processing