Granular knowledge spillovers: Evidence from software developers*

Aaron Lohmann[1]

[1]Univeristy Bielefeld & IfW Kiel

2025-02-27

Abstract

What drives the technology choices of knowledge workers? Recent models of endogenous growth emphasize the role of interactions between agents in shaping aggregate productivity. This paper provides new evidence using detailed microdata on open-source software developers contributing to the programming language Rust. Leveraging granular data on the adoption of 613 distinct technologies, I find that developers are influenced by the technologies used by their peers. However, the strongest predictor of adoption is a developer's own past experience with a technology, highlighting a significant path dependence in technological choices. Knowledge spillovers are particularly strong among younger and less experienced developers, suggesting that peer effects play a crucial role in the early stages of skill formation.

Keywords: Knowledge diffusion, software, open source software, productivity
JEL: O33, O12, L17

*Comments, questions, and feedback are welcome and can be directed to: aaron.lohmann@uni-bielefeld.de.

1 Introduction

Economic agents frequently make technology choices, particularly in knowledge-intensive industries, where such decisions often rest with individuals. The recent endogenous growth literature, following Lucas (2009), highlights the importance of peer interactions in shaping productivity, though the precise nature of productivity remains somewhat ambiguous. One interpretation, supported by Sandvik et al. (2020), defines productivity as knowing the most effective way to perform specific tasks.

This paper examines detailed micro-data on software developers, their peer networks, and their small-scale technology choices. The findings reveal that peer influence significantly enhances technology adoption, with this effect being strongest among less experienced developers. Additionally, developers exhibit considerable stickiness in their technology choices, suggesting that once a technology is adopted, they are unlikely to switch.

I study technology choices of software developers. More specifically Open Source Software (OSS) developers working on the programming language Rust. OSS powers most modern, digital applications from database management, to websites and the latest machine learning models. A key feature of many OSS systems is that developers do not reinvent the wheel every time but rather rely on code written by others. This code is usually stored in so-called packages and can be used free of charge. The natural economic inepretation is that of a technology choice. In contrast to other contexts, this technology choice is observed at an individual level.

This project relies on several data sets to study the technology choices and peer effects of developers. First, I use data shared by Schueller et al. (2022) which allows to track contributions of individual developers, tracks the technology choices of projects and allows me study which developers are connected to each other. Second, I incorprate additional data from cartes, the package manager of Rust, to categorize software packages. Third, libs.rs allows me to give quality measures to projects. Finally, from GHtorrent and GHarchive I get additional information on developers.

To summarise my data, I provides some general information. The time frame is from 2014 to 2022. Throughout the paper, I measure everything in quarters. In total, there are $46,265$ developers who contribute to Rust packages. Though, only $16,335$ own at least one of those packages. In total, we have $91,437$ projects which are stored in $51,657$ repositories. This means that some repositories have multiple pckages. Still, the ownership within the pacakge is independent of repository ownership allowing us to identify all key variables. Of the $91,437$ packages I can identify owners for $85,200$ of the projects. of the total amount pakcages, $29,072$ are used as an input at least once in the sample. The average project imports 3.58 others projects – usese these as technologies. This number is steadily rising. The projects are organized in 71 different categories.

Empirically, I assume that developers choose technologies according to a thresholf rule. As long as the marginal utility gain, conditional on knowledge, outweighs the costs, technology is adopted. Knowledge of a technoloyg is increased through two main channels. For one own experience with this specific technology, second peers experience with this technology. To

estimate the effects, I employ a linear probabiliyt model. Both channels are quantitatively meaningful. Though, the developers' own past technological decisions are generally more important. This suggests that knowledge workers are very sticky in their technological choices. Finally, heterogeneity analysis reveals that

To address the endogeneity concern, I use an IV strategey based on the Firends of Friends idea in Bramoullé, Djebbari, and Fortin (2009). I recoonstruct the peer exposure variable used in the baseline regression based on peers' of peers who are not peers of the focal developer. The 2SLS results confirm the insight that peer effects are an important predictor.

Related Literature

This paper contributes to two main strands of the literature.

First, endogenous growth and evolution of productivity. Lucas (2009) has argued for an important role of interactions between individuals who exchange ideas. A range of theoertical and empirical contribution have build on this intuition. R. E. Lucas and Moll (2014) study the allocation of time on meeting individuals and producing, Perla and Tonetti (2014) show the importance of firms across the etnrire distribution of productivity for aggregate growth. More closely related are the contributions by Jarosch, Oberfield, and Rossi-Hansberg (2021) and Herkenhoff et al. (2024) who both estimate learning functions based on the productivity of coworkers. Similiarly, Akcigit et al. (2018) show that among patent innovators, learning from individuals others can have an positive effect. What is common to the above mentioned papers, is that the sources of productivity are ambigous. Jarosch, Oberfield, and Rossi-Hansberg (2021) and Herkenhoff et al. (2024) base their productivity estimates on the marginal wage levels. Akcigit et al. (2018) base productivity on received citations. Though, Sandvik et al. (2020) show in an experiment that is is concrete ways of doing things which have the strongest impact on productivity growth. My contribution to this literature, is to provide more micro evidence on the knowledge evolution of high skilled individuals.

Second, Open Source Software. The economics of OSS has received some attention in the early 2000s. More recently, research on OSS has shifted to an empirical field. Wachs et al. (2022) show that development of software is geograhically concentrated. A similiar result can be found in Goldbeck (2023). More recently the attention has also shifted to the technical side of OSS. Hoffmann, Nagle, and Zhou (2024) provide estimates for the value of OSS. In another paper Lohmann et al. (2025) we show the global production of OSS. The contribution this paper makes is to highlight the determinants of software dependency choices.

The remainder of this paper is structured as follows. In Section 2 I introduce OSS, Rust and GitHub. Thereafter, in Section 3 I discuss the used data sources with which I give first empirical evidence in Section 4. After this, the empirical appraoch is introduced in Section 5 with the results presented in **?@sec-results**. Finally, Section 7 concludes.

2 Open Source Software, Rust and GitHub

Open Source Software (OSS) is a fundamental component of the modern digital infrastructure, supporting a wide range of technologies across different domains. It is the backbone of operating systems (e.g., Linux, FreeBSD), database management (e.g., PostgreSQL, MySQL), web development (e.g., Apache, Nginx, Node.js), cloud computing (e.g., Kubernetes, Docker), and machine learning (e.g., PyTorch, TensorFlow, Scikit-learn). OSS is widely used in both consumer and enterprise applications, playing a key role in web browsers (e.g., Firefox, Chromium) and cybersecurity tools (e.g., OpenSSL, Wireshark). Still, the features of its production have only received limited attention from the economic discipline. This is dispite OSS development is observable at a detailed level and usually conducted by high skilled individuals.

Reports from Synopsys and the Linux Foundation highlight the critical role OSS plays in software development.[1][2] emphasizing its widespread adoption across industries. A recent study by Hoffmann, Nagle, and Zhou (2024) further demonstrates the high estimates of the demand-side and supply-side value of OSS.

In another study (Lohmann et al. (2025)), we show that OSS production processes follows a modular structure, driven by the package paradigm. OSS is built using small, reusable packages (or libraries), allowing developers to share their code and integrate existing solutions. This eliminates the need to develop everything from scratch and allows relying on the code of others. A natural way of thinking about these decisions is as technology choices.

Take this paper as an example. The underlying code is largely written in the programming language *R*. To make visually appealing plots, I use the package *ggplot2*, to estimate regressions with high dimensional fixed effects I use *fixest* and to deal with big sparse matrices I use the package *Matrix*. All of these packages are contributed by software developers and shared freely with the world. For an economist this set of packages (and many more) are the technology to craft research results. In software development the set of imported libraries and packages is often fittingly described as the "technology stack". Without efficient matrix multiplication or estimation techniques already implemented this project might otherwise be an incredibly tedious or even hopeless endeavor.

Nowadays, most packages are shared via so-called package managers, with each programming language having its own respective system. For *Python*, it is *PyPI*; for *JavaScript*, *npm*; and for *Rust*, *crates*. The number of available packages on these platforms has grown significantly. Take *crates*, which is the focus of this study, as an example. It is the youngest of the three, yet by the end of September 2022, it hosted a total of 93,000 packages. As of writing this paper in February 2025, that number has increased to 171,867. All of these packages could in principle be reused by developers as their technology.

This amount of packages comes with a choice for software developers. Making this choice given this size of the choice set is not easy. This paper will aim to study these decisions and highlight certain determinants.

[1] Synopsys OSSRA Report (2024)
[2] Linux Foundation Census II Report

The package paradigm described above is not specific to any programming language. With some deviations most use a similiar logic. The programming language under scrutinity in this paper *Rust* is no exception. Rust is a low-level general purpose programming language. It is a relatively young language (first stable version in 2014) with the first development having started in 2008. In recent years Rust has gained quite a bit of popularity. Based on surveys of software developers with high participation rate, the Stack Overflow surveys, it is often voted the *most loved language*. The Biden administration has endorsed the use of Rust. For most, the main motif behind this are some new features in Rust wich enhance safety and speed. This popularity has also given rise to the strong growth of software packages from a handful to the above mentioned $170,000$ today. More firms in manafucturing or online datbase management start to adopt Rust. Maybe on of the most valid signals that Rust is becoming a key technology comes from the fact that the Linux Kernel started to rewrite some parts in Rust. This suggests that Rust is in fact a relevant new technology, and studying its ecosystem is potentially insightful.

Given the high skill requirements of Rust, the developers which I study here should be compared to R&D workers in firms. Furthermore, given that many software development processes are organised similiarly, I expect the results in this study to extend to other languages.

The final part to the background to this paper is GitHub, the largets online collaboration platform for software development. The key function of GitHub is to provide a convenient platform to work on software. Potentially in teams. While GitHub has competitiors with GitLab, Bitbucket, GitHub is arguably in a monopoly position of development platform. In my dataset, 94% packages are developed on GitHub. Developers on GitHub work in so called repositories. Think of a repository as a place to store the code which constitues a package. Developers locally write lines of code which they then *commit* to the repository which is hosted online. *Commits* are usually the smallest unit of code change which we can measure. Subject to approval developers can commit to all repositories but in most cases the core team of the repository is small.

Taken together, I observe and study developers (workers), who potentially interact and work on packages (projects) in which they choose technologies. Nothing in that sounds specific to development of software, the key part of software is simply the possibility to observe next to everything.

3 Data

This project relies on multiple data sources. In Section 4, I provide descriptive evidence to give an overview of the data and discuss key processing choices.

Rust Collaboration.

The primary dataset comes from Schueller et al. (2022), offering a comprehensive account of the Rust ecosystem from *2014* to September *2022*. It includes information on approximately *90,000* packages and *45,000* developers. Given Rust's relative youth, this dataset effectively

captures the entire history of the Rust ecosystem. Developer activity is recorded at the repository level, with most repositories hosted on GitHub. The dataset also contains GitHub login names, allowing for integration with external data sources.

Crates Data.

Crates.io, Rust's official package manager, publishes detailed information about uploaded packages. Two key pieces of information are particularly relevant:

1. Package ownership, which allows tracking of developer contributions.

2. Categorization of packages, where Rust defines a set of pre-determined categories. In addition, developers can assign up to five custom keywords to their crates, providing further classification.

Libs.rs.

Libs.rs is a service designed to categorize and facilitate the discovery of Rust crates. I leverage its underlying database to complement category assignments where Crates.io does not provide them. Additionally, Libs.rs offers a quality ranking for packages—a composite measure that can be used to assess the relative quality of different crates within categories.

Additional GitHub Data.

While the dataset from Schueller et al. (2022) primarily focuses on Rust development, I also incorporate broader GitHub activity for robustness checks. To achieve this, I use GHtorrent and GHarchive, which provide information on:

1. The other programming languages developers engage with beyond Rust.

2. The teams and repositories they contribute to outside the Rust ecosystem.

These additional data sources help assess how Rust developers interact with other technologies and whether their behavior within Rust reflects broader programming trends.

4 Descriptive evidence

This section provides some first descriptive evidence on the used data.

In Figure 1, I plot the quarterly adoption rates of four different software packages, illustrating how developers choose and transition between technologies over time. The left panel focuses on logging libraries, while the right panel tracks time-handling packages.

In the left panel, we observe a clear shift in adoption patterns between two competing logging packages. Initially, one package dominates, but over time, a newer alternative gains traction and eventually surpasses the incumbent. This pattern suggests that developers gradually migrate toward a newer solution, likely due to improvements in features, performance, or ease of integration.

A similar trend emerges in the right panel, where two packages designed for handling time in software development exhibit differentiated adoption trajectories. The package chrono, which offers a more comprehensive feature set, steadily overtakes time, reflecting a broader shift toward more advanced tools as the ecosystem matures. Unlike the logging libraries, where the transition appears more abrupt, the time-handling packages exhibit a smoother and slower diffusion process, suggesting that developers might be more reluctant to switch due to compatibility constraints or the complexity of date-time handling.

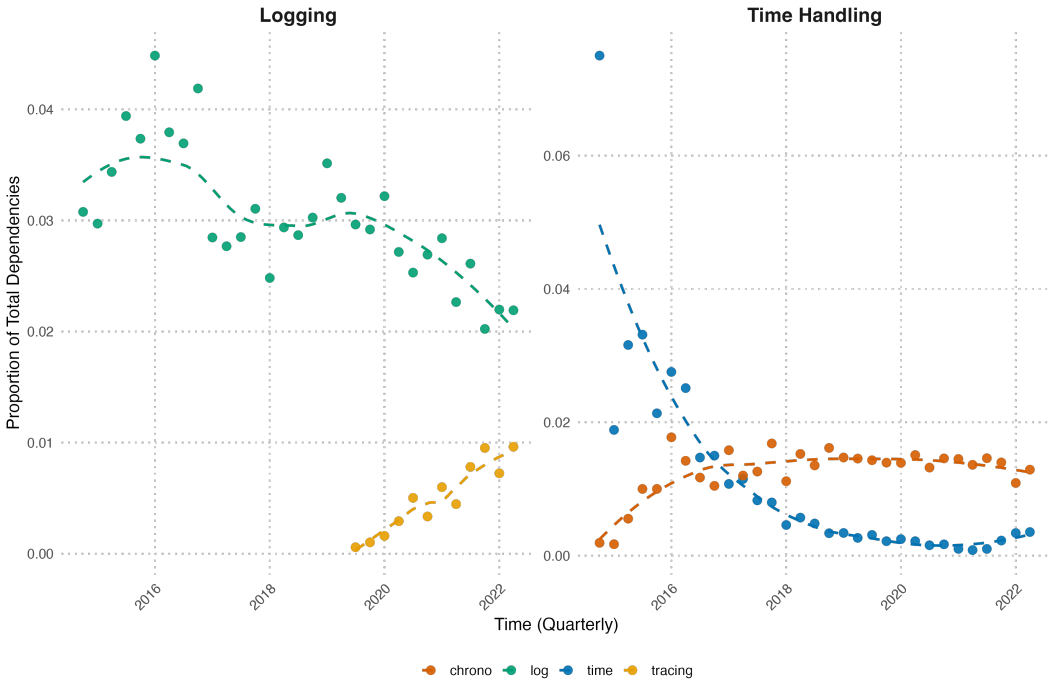

Figure 1: Diffusion processes in Rust.

Notes: This figure plots adoption rates for four different Rust packages grouped by the categories. Adoption is defined on a quartely basis and computed as the share of packages which use this respective packages over all dependency decisions made in this quarter. Left panel shows logging packages, log and tracing. Right panel shows packages which are for handling time.

The next table, provides some general information about the data. The time frame is from 2014 to 2022. Throughout the paper, I measure time in quarters. In total, there $46,265$ developers who contribute to Rust packages. Though, only $16,335$ own at least one of those packages. In total, we have $91,437$ projects which are stored in $51,657$ repositories. This means that some repositories have multiple pckages. Still, the ownership within the pacakge is independent of repository ownership allowing us to identify all key variables. Of the $91,437$ packages I can identify owners for $85,200$ of the projects. Of the total amount pakcages, $29,072$ are used as an input at least once in the sample. The projects are organized in 71 different categories. More details on the categories can found in the appendix.

Table 1: Basics

Variable	Value
Time Frame	2014 - 2022, September (Quarterly)
Developers	46,265, project owners: 16,335
Projects	91,437 packages in 51,657 repositories.
Inputs	29,072 packages used as input at least once.
Categories	71

How many inputs/technologies does each package import? In Figure 2 I plot the average amount of inputs used in each of the observed projects. Clearly, the amount of used inputs is growing over time. This suggests that as time progresses, over time more complex project can be build because better technology is available.

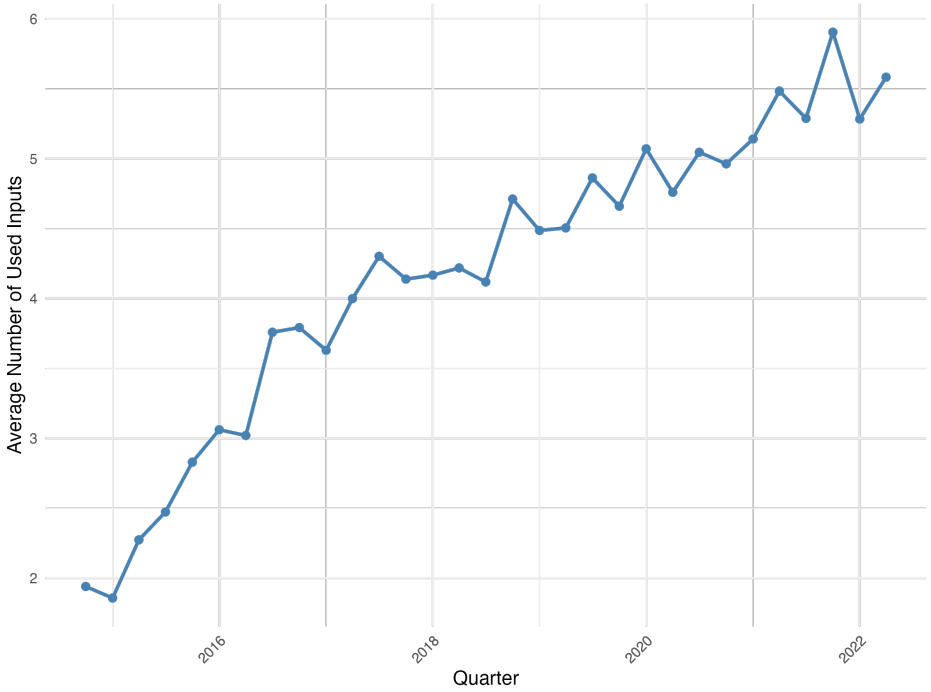

Figure 2: Amount of imported technologies.

Notes: This figure plots the average amount of used inputs per project over time. The amount of inputs used in each project is increasing over time. Time is measured in quarters.

The left panel of Figure 3 plots the distribution of co-workers of developers in the sample. Co-working relationships are defined by being active in the same repository in a given quarter. Most developers have at least one co-worker throughout, many have more. This suggests that the development of Rust can be a good case study for learning from co-workers.

In the right panel of Figure 3 I plot the distribution of downstream dependencies for all packages hosted on crates during the sample period. The red dotted lines indicate the dependencies which I consider later in the regressions. I include packages which have at

least 50 unique input choices and are below 2000. The lower bound is motivated by wanting to choose packages which really act as technology as opposed to being something extremely niche. The upper bound is chosen to exclude the all-star packages which will be well-known no matter what. In total I herewith include 37% of choices, the lower bound excludes 23% of decisions and the upper bound excludes another 39%. The average project in the sample imports 3.58 other packages with substantial heterogeneity.

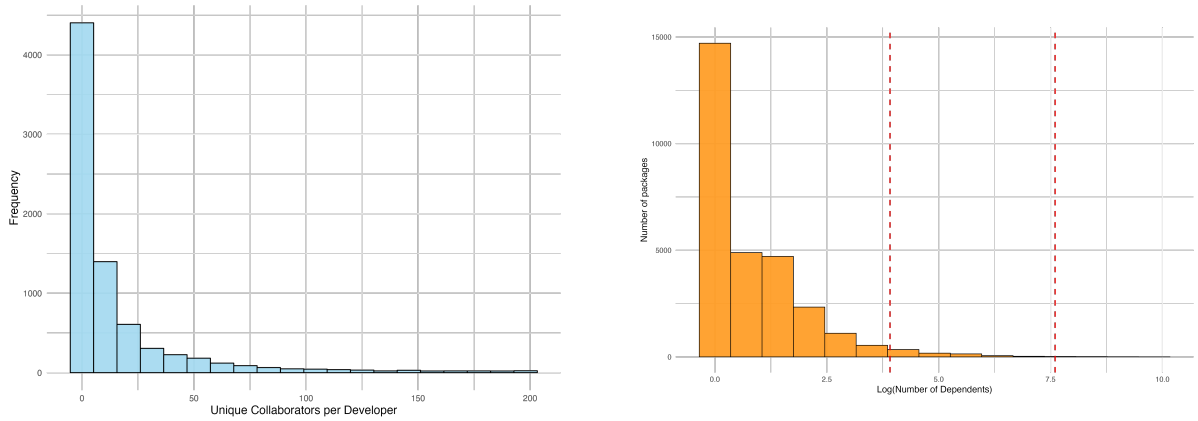

Figure 3: Distribution of co-workers and input adoption

Notes: The left panel plots the distribution of co-workers for developers. The right panel, plots the amount of technological adoption per project. The red-dotted lines show which technologies will contribute to the regressions.

5 Empirical approach

This section describes the empirical approach used to study software dependency choices of developers.

Baseline

Developers i work on a project p in time period t (measured in quarters). To produce the projects p, they can rely on available inputs $\mathcal{J}_t$ with single inputs denoted by j. Using these inputs requires knowledge over them. For each available input, developers i evaluate a threshold condiation:

$$u_j \tau_{ijt} - c > 0$$

where u_j is a generic term capturing the internal quality of input j. The term tau_{ij} captures the knoweldge of developer i over input j. Naturally, knowledge over an input can be scaled between 0 and 1, therefore I assume that $\tau_{ij} = \exp(-x)$. The x generally captures everything that makes a developer more or less knowledgable of a certain input. More specifically, I will consider the past experiences with the input and the experiences of the peers. From this, it is easy to write a basic linear probability model as:

$$D_{ijpt} = \beta_1 \mathrm{E}_{ij,t-q} + \beta_2 \mathrm{S}_{ij,t-q} + \delta_{c(p)c(j)} + \gamma_{it} + \lambda_{jt} + \varepsilon_{ijpt}$$

where D_ijpt captures whether developer i uses input j in the project p which was started in time period t. The main explaintory variables are about the developers' use of this input in the past ($\mathrm{E}_{ij,t-q}$) and the use the developers' peers ($\mathrm{S}_{ij,t-q}$). The expectation is that both increase the knowledge of the specific input and thereby make it more probable. The q indicates the lags considered. Fixed effects capture different aspects. There is substantial heterogeneity of developers, including their overall experience levels which will be captured by γ_{it}. Not every input can be used in every project, for that reason I include a fixed effect based on the categories of project and inputs. Finally, the internal quality of the input is usually unobserved. Thus, I include a input-time specific fixed effect λ_{jt}. Estimation will be based on a linear probability model with OLS.

Most input decisions for projects are done within the first 6 months. In Figure A1 I plot the distribution of the difference between the package creation and the input decision. About 90% of decisions are done within the first 6 month. This means that having a proper panel structure with multiple observations per project over time is not necessary. Also this facilitates the interpretation as this clearly defines which period to consider for the prior experience of developers. Accordingly, I will consider only decisions within the first 6 months.

Moreover, a question might relate to the team size of the projects. In figure I show that in the vast majority of projects, it is the owner who contributes over 90% of the commits. This is also true for projects with multiple developers. Therefore, it seems like a plausible assumption that the owner will be responsible for the technology choices.

The two variables, Experience $\mathrm{E}_{ij,t-q}$ and Spillovers $\mathrm{S}_{ij,t-q}$, both depend on a lag parameter q , which captures past usage history. 1. Experience $E_{ij,t-q}$ measures how frequently developer i has used input j in the past q periods relative to the current period t :

$E_{ij,t-q} = \sum_{s=t-q}^{t-1} D_{ijps}$

where D_{ijps} is an indicator variable that equals 1 if developer i used input j in project p at time s , and 0 otherwise. 2. Spillovers ($S_{ij,t-q}$) capture the indirect knowledge gained from peers' past usage of input j . To define peer relationships, we introduce an adjacency matrix $M_{ii',t-q}$:

$$M_{ii',t-q} = \begin{cases} 1, & \text{if developers } i \text{ and } i' \text{ collaborated in the same repository in period } t-q, \\ 0, & \text{otherwise.} \end{cases}$$

Given this peer adjacency matrix, the spillover variable is constructed as:

$$S_{ij,t-q} = \sum_{i' \neq i} M_{ii',t-q} \cdot E_{i'j,t-q}$$

where $E_{i'j,t-q}$ represents the past experience of developer i' with input j . This means that developer i is more likely to adopt input j if their past collaborators have previously used it.

Instrumental Variable (Friends of Friends)

There is a valid endogeneity concern if developer i selects into collaboration with i' because this allows learning about a certain input j . To address this concern, I construct an instrument centered around the identification in Bramoullé, Djebbari, and Fortin (2009) by using friends of friends. Using these friends of friends, I reconstruct the spillover variable and use this as an instrument for the baseline spillover variable. The construction is straightforward. Recall that I define the matching matrix $M_{ii'}$, where $M_{ii'} = 1$ if developers i and i' are directly connected and 0 otherwise. From this, I construct the friends of friends matrix $F_{ii'}$. To do so, I first compute $G = M^2$, where $G_{ii'} = \sum_k M_{ik} M_{ki'}$ counts the number of two-step connections between i and i' . Next, I convert G into a binary variable such that $F_{ii'} = \mathbb{1}\{G_{ii'} > 0\}$, ensuring that it captures only the presence of indirect connections. From this matrix, I subtract both the diagonal elements and the original matching matrix to remove direct friendships and self-connections, ensuring that $F_{ii'} = \mathbb{1}\{G_{ii'} > 0\} - M_{ii'} - \mathbb{1}\{i = i'\}$.

With this F matrix, I recompute the spillover variable from before as $\tilde{S}_{ij,t-q}$ and use this as an instrument for the original spillover variable.

6 Results

The next table shows the key results. First of all, we find that that past decisions and peer decisions are positive predictiors of software input adoption. Quantitatively, own past

decisions are more important though. Still, taking into an account the low unconditional probability of observing an input decision (mean at 0.002) both estimates are quantitatively important. All observations are based on a conservative set of fixed effects as described in the previous section. The results are robost to different specifications to deal with zeros in the explanatory variables.

Table 2: OLS Regression Results

Dependent Variable:		depends	
Model:	(1)	(2)	(3)
Variables			
Log(Peers Value + 1)	0.0135***		
	(0.0011)		
Log(Self Value + 1)	0.1003***		
	(0.0084)		
Arcsinh(Peers Value)		0.0105***	
		(0.0008)	
Arcsinh(Self Value)		0.0784***	
		(0.0065)	
value_peers			0.0082***
			(0.0007)
value_self			0.0596***
			(0.0039)
Arcsinh(Peers)			0.9202
			(36.24)
Log(Cumulative Repos)			-0.1527
			(184.7)
Fixed-effects			
user_id-time	Yes	Yes	Yes
category_pacakge-category_input	Yes	Yes	Yes
input-time	Yes	Yes	Yes
Fit statistics			
Observations	28,168,422	28,168,422	26,832,801
R^2	0.03512	0.03516	0.03215
Within R^2	0.01156	0.01160	0.01150

Clustered (user_id-time) standard-errors in parentheses
*Signif. Codes: ***: 0.01, **: 0.05, *: 0.1*

Notes: OLS regressions with binary dependent variable indicating whether a project uses an input. Observations are project-developer-input specific. Key explaintory variables are the value of peers which is an aggregate measure of peer decicions with respect to this input in the past 4 quarters and the own past decisions with respect to this input.

Next, I presenet the results based on the instrumental variables. Here, the peer value is instrumented based on the procedure described before. The general insights do not change also with this empirical strategy. Though, quantitatively, I find somewhat smaller estimates suggesting that some part is subject to selection.

Table 3: 2SLS Regression Results

Dependent Variable:		depends	
Model:	(1)	(2)	(3)
Variables			
Log(Peers Value + 1)	0.0070**		
	(0.0034)		
Log(Self Value + 1)	0.1012***		
	(0.0083)		
Arcsinh(Peers Value)		0.0057**	0.0055**
		(0.0027)	(0.0024)
Arcsinh(Self Value)		0.0791***	0.0735***
		(0.0064)	(0.0050)
Arcsinh(Peers)			0.9175
			(36.21)
Log(Cumulative Repos)			-0.1575
			(184.7)
Fixed-effects			
user_id-time	Yes	Yes	Yes
category_pacakge-category_input	Yes	Yes	Yes
input-time	Yes	Yes	Yes
Fit statistics			
Observations	28,168,422	28,168,422	26,832,801
R^2	0.03497	0.03502	0.03121
Within R^2	0.01140	0.01146	0.01055

Clustered (user_id-time) standard-errors in parentheses
*Signif. Codes: ***: 0.01, **: 0.05, *: 0.1*

Notes: OLS regressions with binary dependent variable indicating whether a project uses an input. Observations are project-developer-input specific. Key explaintory variables are the value of peers which is an aggregate measure of peer decicions with respect to this input in the past 4 quarters and the own past decisions with respect to this input.

Finally, I show heterogeneity analysis which reveals that more experienced developers are not as affected by the value of peers. This is consistent with the view that a technological stack is chosen early in the career and is then carried forward. To meassure experience, I use the amount of past repositories the developer has contributed to and the age measured in quarters based on the first commit ever observed. The interaction with either of these experience measures is negative. The general peer effect however still remains positive.

Table 4: Heterogeneity with experience

Dependent Variable:	depends	
Model:	(1)	(2)
Variables		
Arcsinh(Peers Value)	0.0126***	0.0153***
	(0.0016)	(0.0016)
Arcsinh(Self Value)	0.0784***	0.0787***
	(0.0065)	(0.0066)
Arcsinh(Peers Value) × dev_age	-0.0003**	
	(0.0001)	
Arcsinh(Peers Value) × cum_repos		-0.0003***
		(7.1×10^{-5})
Fixed-effects		
user_id-time	Yes	Yes
category_pacakge-category_input	Yes	Yes
input-time	Yes	Yes
Fit statistics		
Observations	28,168,422	28,168,422
R^2	0.03518	0.03523
Within R^2	0.01162	0.01167

Clustered (user_id-time) standard-errors in parentheses
*Signif. Codes: ***: 0.01, **: 0.05, *: 0.1*

Notes: 2SLS regressions with binary dependent variable indicating whether a project uses an input. Observations are project-developer-input specific. Key explaintory variables are the value of peers which is an aggregate measure of peer decicions with respect to this input in the past 4 quarters and the own past decisions with respect to this input. The value of peers is instrumented based on on the peers of peers decisions.

7 Concluding remarks

This paper uses detailed micro data on software developers, their co-working relationships and detailted technology choices. Developers adopt technologies their peers. This effect is stronger for less experienced and younger developers. Though, the results also show strong persistence of developers in their choices.

8 References

Akcigit, Ufuk, Santiago Caicedo, Ernest Miguelez, Stefanie Stantcheva, and Valerio Sterzi. 2018. "Dancing with the Stars: Innovation Through Interactions." Cambridge, MA. https://doi.org/10.3386/w24466.

Bramoullé, Yann, Habiba Djebbari, and Bernard Fortin. 2009. "Identification of Peer Effects Through Social Networks." *Journal of Econometrics* 150 (1): 41–55.

Goldbeck, Moritz. 2023. "Colocation and the Death of Distance in Software Developer Networks."

Herkenhoff, Kyle, Jeremy Lise, Guido Menzio, and Gordon M. Phillips. 2024. "Production and Learning in Teams." *Econometrica* 92 (2): 467–504. https://doi.org/10.3982/ECTA16748.

Hoffmann, Manuel, Frank Nagle, and Yanuo Zhou. 2024. "The Value of Open Source Software," January. https://doi.org/10.2139/ssrn.4693148.

Jarosch, Gregor, Ezra Oberfield, and Esteban Rossi-Hansberg. 2021. "Learning From Coworkers." *Econometrica* 89 (2): 647–76. https://doi.org/10.3982/ECTA16915.

Lohmann, Aaron, Gábor Békés, Julian Hinz, and Miklós Koren. 2025. "Code Without Borders? Global Value Chains in Open Source Software Development." University of Bielefeld, Kiel Institute for the World Economy, Central European University, HUN-REN KRTK, CEPR, CESifo.

Lucas. 2009. "Ideas and Growth." *Economica* 76 (301): 1–19. https://doi.org/10.1111/j.1468-0335.2008.00748.x.

Lucas, Robert E., and Benjamin Moll. 2014. "Knowledge Growth and the Allocation of Time." *Journal of Political Economy* 122 (1): 1–51. https://doi.org/10.1086/674363.

Perla, Jesse, and Christopher Tonetti. 2014. "Equilibrium Imitation and Growth." *Journal of Political Economy* 122 (1): 52–76. https://doi.org/10.1086/674362.

Sandvik, Jason J, Richard E Saouma, Nathan T Seegert, and Christopher T Stanton. 2020. "Workplace Knowledge Flows*." *The Quarterly Journal of Economics* 135 (3): 1635–80. https://doi.org/10.1093/qje/qjaa013.

Schueller, William, Johannes Wachs, Vito D. P. Servedio, Stefan Thurner, and Vittorio Loreto. 2022. "Evolving Collaboration, Dependencies, and Use in the Rust Open Source Software Ecosystem." *Scientific Data* 9 (1): 703. https://doi.org/10.1038/s41597-022-01819-z.

Wachs, Johannes, Mariusz Nitecki, William Schueller, and Axel Polleres. 2022. "The Geography of Open Source Software: Evidence from GitHub." *Technological Forecasting and Social Change* 176 (March):121478. https://doi.org/10.1016/j.techfore.2022.121478.

A Additional figures

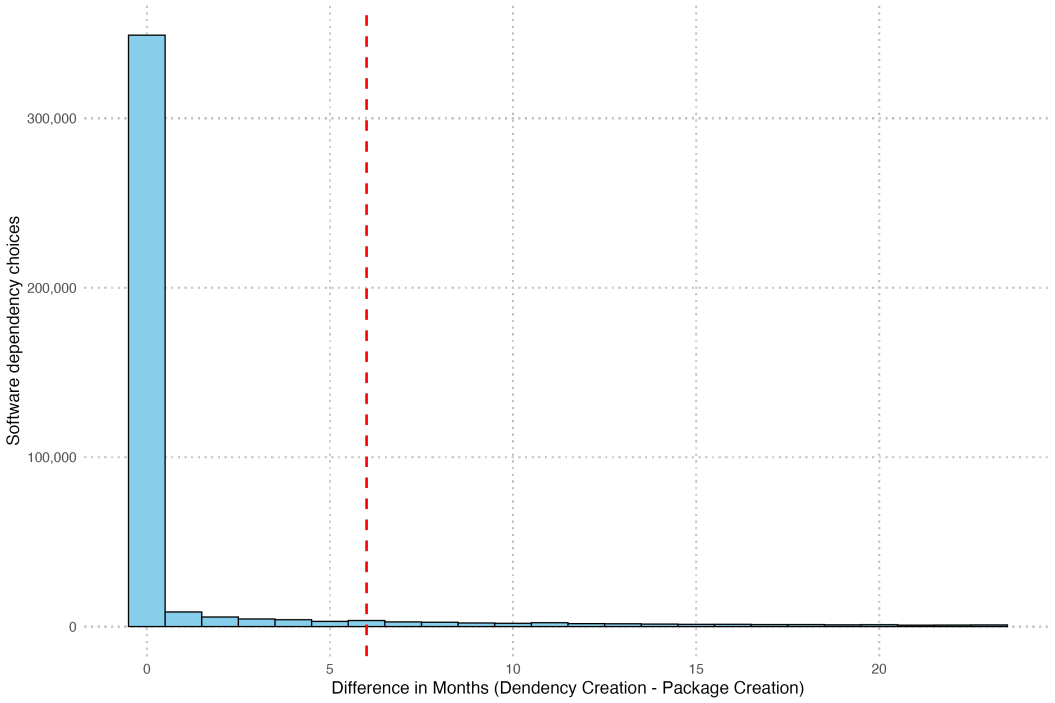

Figure A1: Month difference package creation and input decision.

Notes: This figure illustrate the distribution of. The figure combines information on 417,190 unique dependency decisions. That is the first time a package imports another package. The vast majority of decisions are done within the month of creating the package. Source: Schueller et al. (2022), own caluclations.

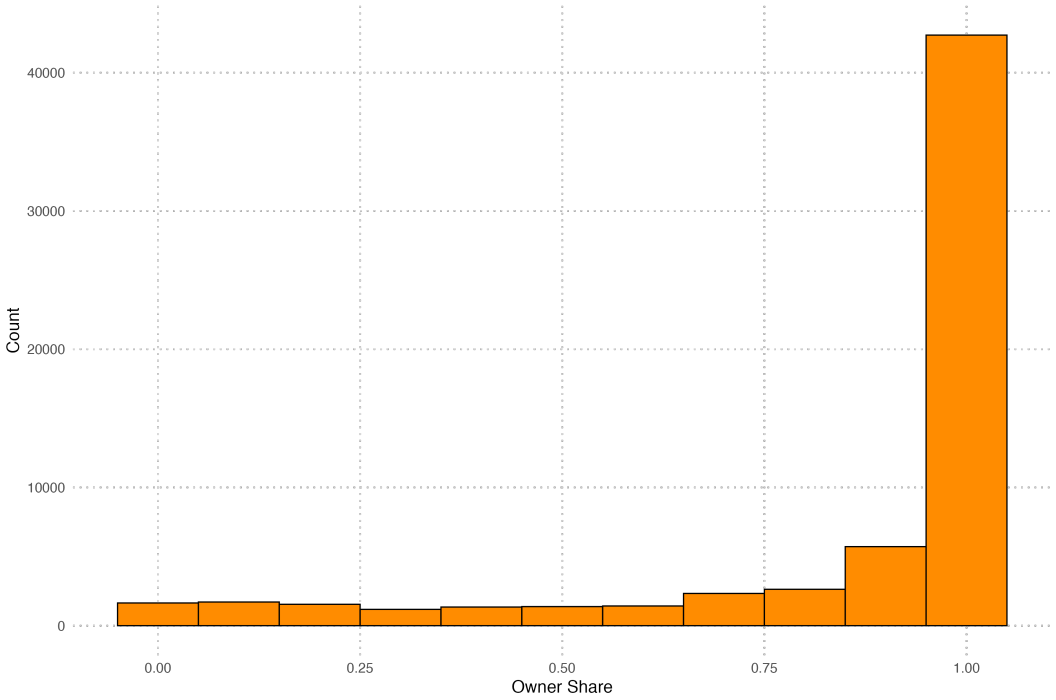

Figure A2: Test.

Notes: This figure plots the distribution of commits done by the owner, as registered on crates, over the total amount of commits in the repository where the package is stored. The share is heavily skewed to the right suggesting that in most packages the owner is the developer exerting the most effort. Source: Schueller et al. (2022), own caluclations.

B Additional tables

Category	Category.cont.
text-processing	development-tools::ffi
concurrency	memory-management
network-programming	web-programming
algorithms	config
cryptography	compression
asynchronous	internationalization
accessibility	science::math
rendering::graphics-api	simulation
parser-implementations	games
development-tools::procedural-macro-helpers	hardware-support
game-development	text-editors
compilers	database
embedded	os
os::windows-apis	finance
science::bio	science
command-line-utilities	date-and-time
encoding	os::unix-apis
development-tools::build-utils	development-tools::testing
data-structures	template-engine
multimedia::audio	no-std
web-programming::http-client	filesystem
wasm	development-tools::debugging
development-tools	multimedia
web-programming::http-server	os::macos-apis
multimedia::images	web-programming::websocket
rendering::data-formats	rendering
multimedia::video	value-formatting
caching	visualization
parsing	emulators
science::ml	database-implementations
authentication	rendering::engine
gui	email
science::robotics	development-tools::cargo-plugins
command-line-interface	science::geo
cryptography::cryptocurrencies	development-tools::profiling
rust-patterns	

Notes: Table displays the available 71 categories. Source: Libs.rs.

Table A1: Heterogeneity with experience

Dependent Variable:	depends	
Model:	(1)	(2)
Variables		
Arcsinh(Peers Value)	0.0126***	0.0153***
	(0.0016)	(0.0016)
Arcsinh(Self Value)	0.0784***	0.0787***
	(0.0065)	(0.0066)
Arcsinh(Peers Value) × dev_age	-0.0003**	
	(0.0001)	
Arcsinh(Peers Value) × cum_repos		-0.0003***
		(7.1×10^{-5})
Fixed-effects		
user_id-time	Yes	Yes
category_pacakge-category_input	Yes	Yes
input-time	Yes	Yes
Fit statistics		
Observations	28,168,422	28,168,422
R^2	0.03518	0.03523
Within R^2	0.01162	0.01167

Clustered (user_id-time) standard-errors in parentheses
*Signif. Codes: ***: 0.01, **: 0.05, *: 0.1*

Notes: OLS regressions with binary dependent variable indicating whether a project uses an input. Observations are project-developer-input specific. Key explaintory variables are the value of peers which is an aggregate measure of peer decicions with respect to this input in the past 4 quarters and the own past decisions with respect to this input. Heterogeneity with resepct to the effects of peers and the experience levels. Experience measured in the age of the develoepr and the amount of past projects contributed to.