Abou El-Komboz, Lena; Fackler, Thomas

Conference Paper

Productivity Spillovers among Knowledge Workers in Agglomerations: Evidence from GitHub

Beiträge zur Jahrestagung des Vereins für Socialpolitik 2022: Big Data in Economics

Provided in Cooperation with:
Verein für Socialpolitik / German Economic Association

Suggested Citation: Abou El-Komboz, Lena; Fackler, Thomas (2022) : Productivity Spillovers among Knowledge Workers in Agglomerations: Evidence from GitHub, Beiträge zur Jahrestagung des Vereins für Socialpolitik 2022: Big Data in Economics, ZBW - Leibniz Information Centre for Economics, Kiel, Hamburg

This Version is available at:
https://hdl.handle.net/10419/264083

Productivity Spillovers among Knowledge Workers in Agglomerations: Evidence from GitHub

Lena Abou El-Komboz*, Thomas Fackler†

February 28, 2022

Abstract

How much do high-tech workers benefit from being physically close to each other? On the one hand, software engineers' work could almost entirely be done remotely. On the other hand there is an extensive literature on clustering and agglomeration effects on innovative behavior. Using a large data set with over 11 million observations covering the years 2015 to 2021 from the open source platform GitHub, we relate cluster size to a user's productivity. Our findings suggest that high-tech workers benefit from physical proximity to a large number of other workers in their field. In further analyses, we implement an event-study design and study cross-field spillovers and heterogeneities by cluster size, skill-level and project productivity.

Keywords: agglomeration effects, knowledge spillovers, open source, online collaboration
JEL: D62, J24, O36, R32

*ifo Institute and LMU Munich, Poschingerstraße 5, 81679 München, abou-el-komboz@ifo.de.
†ifo Institute, LMU Munich and CESifo, Poschingerstraße 5, 81679 München, fackler@ifo.de.

1 Introduction

During the COVID-19 pandemic, a large shift from office work to work from home was observed. In the past, workers had disadvantages when working from home such as lower wages in comparison to their coworkers working on-site (Emanuel and Harrington, 2020). Employers were of the opinion that productivity decreases when an employee was not physically present at the workplace. Now, as work from home is more common, these prejudices may decrease (Emanuel and Harrington, 2020).[1]

This shift to remote work prompts various follow-up questions regarding infrastructure and urban utilization: What to do with the now empty offices? Do they become redundant, if employees work remotely? Will employees move to the countryside to benefit from lower rents? In that case, cities as we know them will change and urban density may decrease.

Research suggests that this is unlikely: Cities with higher densities have benefits beyond the smaller distance of employees to their workplace. Urban density was found to positively affect wages as well as worker and firm productivity. One reason seems to be the better diffusion of knowledge with physical proximity. Knowledge spillovers occur between workers, i.e., workers benefit from the differing skills of their coworkers and by learning from them, they gain new skills and their productivity increases (Cornelissen et al., 2017). Especially in the innovative sector, strong clustering is observed because workers tend to locate near each other (Demsas, 2021).

Agglomeration economics captures this process. It conceptualizes the effects on workers and firms with an increase in urban density (Combes and Gobillon, 2015). Several findings suggest that proximity plays a big role for productivity spillovers. The effects rapidly decay with an increase in distance to each other (Baum-Snow et al., 2020).[2] Innovation clusters seem to locate near universities to benefit from the decreased costs of communication and increased knowledge flow (Andersson et al., 2009).

Patent data is often used to measure innovative activity and the effects of agglomeration on it. However, the data likely does not capture all productivity gains from an increase in urban density (Carlino and Kerr, 2015).

To capture more fine-grained productivity effects, we use an alternative data source. GitHub[3] is the world's biggest open source platform. Software developers collaborate on projects through the platform. The work on public projects is observable by everyone. A user in a larger cluster

[1] Emanuel and Harrington (2020) find, based on natural experiments, that before the COVID-19 pandemic workers were adversely selected into remote work. Less productive workers were more likely to work from home and wages for remote work were lower. Now in the COVID-19 pandemic, productivity gains of working from home for workers are observed. The researchers conclude that, before the COVID-19 pandemic, too little employees worked from home regarding the productivity gains, they would experience with a change of workplace.

[2] Baum-Snow et al. (2020) find that firm and productivity spillovers fully decline within 250 meters. They also show, that these spillover effects are higher for higher quality firms.

[3] `https://github.com/`

might also benefit from knowledge spillovers and become more active on the platform. So even smaller shifts in productivity with an increase in cluster size are likely captured by the data.

Open source software plays an important role for firms. Firms that incorporate open source software experience an increase in value-added productivity. These platforms are places where a lot of productive output is generated (Nagle, 2019).[4]

It is also part of the high-tech sector[5] and based on prior research (Casalnuovo et al., 2015), clusters should occur and matter as well for GitHub users' productivity.

To study agglomeration effects on the productivity of GitHub users in the USA and Canada, we build on the empirical approach by Moretti (2021). We use the exogenous variation in cluster size originating from users moving across cities, to estimate the impact of an increase in cluster size on the users' productivity. This can be via affecting the output's quantity as well as quality. Furthermore, is there heterogeneity in the effects, i.e., does the effect differ on several aspects as cluster size or productivity level of a user? The findings are important to foster open source activity and maximize the gains from it, but also to improve our understanding of agglomeration effects among knowledge workers more generally.

A concern when estimating agglomeration effects on productivity are simultaneity and unobserved productivity shocks. Therefore, we implement an event-study design based on movers. The setting allows us to study the dynamic response of a user to a move. We find positive productivity gains with the move, whereas lagged values of cluster size do not affect current productivity.

To begin with, we explain the research question in more detail and give an overview of the existing literature on innovation and agglomeration economics in section two. Then, in section three, the theoretical interplay of knowledge spillovers, productivity, and urban density is described. Thereafter, the empirical framework is presented in section four. First, GitHub is explained in detail, as well as the data gathering. Second, the data is described and possible limitations of it are discussed. Third, the estimation strategy to identify agglomeration effects on GitHub users' productivity is explained. The findings of the empirical analysis are presented in the following section five. Finally, we conclude in section six.

2 Motivation

In this section, the research question is presented to analyze the effects of cluster size on the productivity of GitHub users. Furthermore an overview of the existing literature on the effects of exposure to innovation and productivity as well as agglomeration effects and productivity is given.

[4] A firm incorporating one percent more free open source software raises the value-added productivity by 0.002 to 0.008 percent (Nagle, 2019)

[5] In line with Moretti (2021), we use the term high-tech to describe any firm generating innovations.

2.1 Research Question

Workers and firms relying on knowledge-intensive tasks spatially cluster more than, e.g., manufacturing employment (Carlino and Kerr, 2015; Audretsch and Feldmann, 1996). A higher urban density tends to positively affect the number of patents in that area (Carlino et al., 2007). If an inventor moves to such an area, her productivity, measured by e.g. number of patents, seems to increase (Moretti, 2021). This hints at productivity spillover effects of local inventors affecting the productivity of the moving inventor.

The increase in innovative activity as a result of a relocation to a denser area might be due to an increase in collaborations with local inventors (Catalini, 2018). Specifically, active exposure to innovation matters: For instance after an inventor dies, the positive exposure effect on productivity among co-authors of the departed inventor likely decreases (Azoulay et al., 2010).

The majority of studies on exposure to innovation and agglomeration effects measures productivity by patents. Yet, as studies suggest, using patent data to measure innovative activity has its downsides (Carlino and Kerr, 2015). Not all patents possess the same value. Rather, it seems that there are some very valuable patents containing important innovations, whereas a large number of patents are less valuable. The latter ones might be too specific or were filed for legal reasons (Carlino and Kerr, 2015).

Another shortcoming of patent data to analyze innovation is its lack of representativeness of innovation. A patent is the first step of the innovative process, namely, the invention, but not every innovative idea is patented (Carlino and Kerr, 2015). To patent or not is also field specific and whether innovations are commonly patented or not varies across domains (Cohen et al., 2000). Therefore, patent data does not capture all innovative activity.

In conclusion, measuring innovation based on patent data has its disadvantages. However, research finds positive cluster effects on patent recording and citation, suggesting spillover effects between innovators' productivity exist (Carlino and Kerr, 2015).

An alternative way to analyze these cluster effects, at least in the high-tech sector, is using GitHub data to measure productivity. GitHub is a code hosting platform based on the `git` version control system (Kalliamvakou et al., 2014; GIT, 2021). Developers can upload (*push*) their code changes onto the platform into a project repository. Others can then download (*pull*) the project and work on the code and modify it. Any code modification is called a *commit* (Laurentsyeva, 2019). The progress of a project depends heavily on the number of commits and hence a commit can be seen as a productivity unit (McDonald and Goggins, 2013). As a user, the main work interactions with other developers are commits. On the user profile pages, the commits by project are summarized to show how active a user is (Laurentsyeva, 2019).

Open Source platforms gain in importance, as for example, Microsoft even bought GitHub in 2018 for $7.5 billion (Microsoft, 2021). This shows, that for software firms these platforms are

important input for their work and they are willing to invest large amounts of money to incorporate them.

The field of software programming is rapidly expanding and emerging. A software evolves over time and new components are added or old components removed. Patents in this sector likely only capture the lower bound of productivity as well as innovative activity, because patent filing is a process that can take several years (Cohen et al., 2000).

In comparison to patent data, commits are not necessarily the first stage of innovation, but rather can also represent later stages in the innovative process. A project goes through several stages of development such as building, integration testing or system testing. With every commit, a project further evolves and becomes more innovative (Vasilescu et al., 2015). For a public GitHub project, these steps are observable by everyone. Furthermore, the values of commits are possibly more normally distributed in comparison to the high skewness of patents' values (Carlino and Kerr, 2015).

Even though programming is done on the computer and no real-world contact is necessary, interactions and knowledge diffusion in the real world might positively affect the advances of a project through an increase in commits or an increase in the quality of commits. Patent citations are very localized and the number of (local) patent citations also increases with cluster size, meaning in larger clusters, in comparison to smaller clusters, more citations are observed (Jaffe et al., 1993). The increase in local patent citations with cluster size points to better knowledge diffusion in bigger clusters caused by a more efficient way of knowledge flow (Moretti, 2021). GitHub users likely benefit from the increased knowledge flow as well.

Building on these advantages of GitHub, we use the empirical approach by Moretti (2021) to analyze cluster effects using GitHub data. Specifically, we take the number of commits of a developer as a measure of her productivity. Based on that, we estimate how the cluster size of a programming language affects her productivity in that programming language after a move to a larger cluster. With the data on hand, we find that cluster size positively affects a user's productivity as well as the quality of a user's output. The effects may differ on several aspects, e.g., the level of user or project productivity, or cluster size. So a heterogeneity analysis of the effects is carried out as well. The effects seem to be larger for more productive projects and users, and larger cluster. On the other hand, more skilled users, proxied by their number of followers, gain less from larger clusters than less-skilled users.

To test our results from the OLS regression using all variation, we estimate a dynamic response of productivity to a change in cluster size by including leads and lags of cluster size additionally to current cluster size. Productivity gains occur with a move and do not precede it. The first lead seems to affect current productivity, this can either be due to our data generating process or may suggest that productivity increases not only with the move but also in the next time interval. The

findings support our main results of productivity gains with an increase in cluster size.

Cluster effects on productivity, as estimated by Moretti (2021) using patent data, are observable with GitHub data as well. After a user moves to a larger cluster, she might start to interact with local users. With larger cluster sizes, more users of a given cluster reside in a city and the relocated inventor can interact with a greater number of users in her cluster and collaborate with them. Thus, she might start contributing more to projects on GitHub.

Commits as a proxy for productivity to analyze agglomeration effects has not been tried in the literature. As research suggests, agglomeration effects are quite profound and even more so for skilled workers and knowledge-intensive tasks and, as shown, occur for GitHub users as well (Combes et al., 2010; Carlino et al., 2007).

On GitHub, users are arguably more skilled as it requires a certain level of knowledge to modify existing code. Based on this research design, it is possible to analyze even smaller shifts in productivity due to changes in cluster size. Not every innovative idea is patented and, thus, some productive developments caused by agglomeration effects are likely not observed. Every progress in a public GitHub project measured by a commit is observed and productivity gains of a move to a larger high-tech cluster may be more accurately captured.

Further understanding of agglomeration effects are important as governments attract companies to foster the development of high-tech clusters (Moretti, 2021). The reason lies in the fact that cities with large high-tech clusters tend to have higher mean wages and mean incomes in comparison to cities with less high-tech clusters. High-tech firms such as Amazon or Tesla are offered a large amount of subsidies to locate an establishment in a city. Obtaining more detailed knowledge on even smaller shifts in productivity as a result of agglomeration and collaboration effects is important because the city's or state's governance hopes for those effects to take place with the newly attracted establishments (Moretti, 2021).

GitHub offers a perfect place to tackle this question due to the fine-grained data availability of interactions on the platform. On the one hand, data on the history of interactions on GitHub as well as data on the users are provided by GitHub Torrent (Gousios, 2013) and is accessible for free via their website or the Google Cloud Platform. On the other hand, GitHub contains integrated social features because it was constructed as an online platform for collaboration (Laurentsyeva, 2019). Therefore, it offers a perfect surrounding to study productivity spillover effects as they seem to be strongest in collaborations (Azoulay et al., 2010).

2.2 Overview of the Existing Literature

The main body of the innovation literature measures innovative activity with patents (Carlino and Kerr, 2015). The advantage of this approach lies in the data availability of patents. After its digitization in the late 1990s an increasing share of researchers measures productivity with patent

data. The scope of this literature varies, among others, from the effects of exposure to innovation on productivity (Bell et al., 2019) to agglomeration effects on productivity (Moretti, 2021).

2.2.1 Effects of Exposure to Innovators and Productivity

Bell et al. (2019) analyzed the determinants of becoming an inventor by using patent data. They find that exposure to other inventors and network effects play an important role for children patenting later on in life. These effects seem to be gender- and technology class-specific, e.g., girls, which are exposed to women inventing in a specific field in their childhood, are more likely to patent in that field when grown up (Bell et al., 2019).[6] The findings of technology class-specific exposure effects show, that not the total number of inventors, or city size, matters for innovation but the field-specific cluster size.

Exposure to other inventors does not only affect innovation during childhood but also during adulthood. Azoulay et al. (2010) explore peer effects in the field of life sciences. They use quasi-random variation in knowledge flows caused by unexpected deaths of eminent scientists to estimate the effects of collaborations with co-authors.[7] They find a persistent decline in quality-adjusted publication output of co-authors after the death of the eminent scientists.[8] The decline was even more profound in the case of highly cited eminent scientists (Azoulay et al., 2010). Their estimates suggest knowledge spillovers occurring among scientists which are of considerable size.

Though, as Cornelissen et al. (2017) state, these spillovers do not translate into higher wages. They, besides others, specifically look at high-skilled occupations and to what extent peer quality affects workers' wages. On average, an increase in peer quality increases individual wages only to a small extent.[9] They conclude that productivity increases may not be one-to-one converted in wages (Cornelissen et al., 2017). Using wages as a proxy for productivity, thus, likely capture knowledge spillovers incorrectly, whereas innovative output might more accurately capture the peer effects on productivity.

Catalini (2018) further examines the importance of colocation on scientific collaborations. He exploits the exogenous colocation and separation of laboratories at the Paris Jussie campus, a leading scientific and medical complex in France, due to asbestos removal. The results imply that colocation increases the likelihood of collaborations and are persistent over future separation. Furthermore, new collaborations tend to work on riskier research as the articles published are

[6] Bell et al. (2019) estimate that the causal effects of the environment explain 75% of the correlation between children's propensity to become inventors and patent rates among adults in their commuting zone.

[7] Unexpected death is defined as an age of 67 years or less at the time of death. Furthermore, scientists needed to be active before death (Azoulay et al., 2010).

[8] Specifically, they estimate a decline of five to eight percent in quality-adjusted publication rates for co-authors of the eminent scientists (Azoulay et al., 2010).

[9] Their results suggest that a ten percent increase in peer quality raises individual wages by about 0.1 percent (Cornelissen et al., 2017).

either at the top or bottom of the citation distribution (Catalini, 2018).

The results imply exposure to other inventors matters and positively impacts innovative activity. The findings likely hold for GitHub users as well. Coding is a knowledge intensive task, and by working together on projects, knowledge spillovers can occur which then may raise the users' productivity. By being geographically close to each other these effects are enhanced, as especially the results by Catalini (2018) show.

2.2.2 Agglomeration Effects and Productivity

Agglomeration effects have been analyzed for a long time. A summary of the work in the field is, besides others, available by Carlino and Kerr (2015). The seminal paper by Jaffe et al. (1993) uses patent citations to measure knowledge spillovers. The researchers find a strong localization of patent citations, which is quite persistent over time. They conclude that, given inventors are more likely citing patents by other inventors that are geographically more close to them, knowledge spillovers are more localized as well. Though, Jaffe et al. (1993) did not further identify city characteristics affecting these spillover effects.

Carlino et al. (2007) build on the work of Jaffe et al. (1993) by analyzing the question of city characteristics determining the extent of productivity effects with an increase in urban density. Especially the employment density, i.e., jobs per square mile, has a positive impact on the number of patents per capita.[10] The competitiveness of the market structure and total employment additionally positively affect patent intensity. The results suggest strong agglomeration effects on the innovative activity of local workers. Living in a denser cluster fosters the knowledge process.

Moretti (2021) investigates how a move of an inventor to a larger cluster affects her number of patents produced and the number of citations received. Both experience an increase after a relocation, which is further confirmed by Instrumental Variable (IV) estimates. Local cluster size is instrumented by cluster size changes that originate elsewhere. This way he tries to tackle a possible bias in the estimates of cluster size due to unobservable productivity shocks.[11] Moretti (2021) concludes that cluster size positively impacts patenting and, based on that, productivity.

The findings show a positive relationship between urban density and productivity, especially for innovative activity. This likely applies for software engineers as well. With a move to a larger cluster, surrounded by a larger number of users in her programming language, the focal user might experience an increase in productivity caused by knowledge spillover effects.

Urban density, next to productivity, also affects other factors such as industrial employment, or firms' location choices (Combes and Gobillon, 2015). Firms should choose to settle where their

[10] Carlino et al. (2007) show that with a doubled employment density, the patent intensity increases by 20 percent.

[11] In detail, Moretti (2021) estimates an elasticity between number of patents produced and cluster size of 0.0662. The contemporaneous effect of a change in cluster size on productivity is 0.0162 and, when instrumented, 0.0307.

expected profit is maximized. As urban density affects productivity and higher productivity results in higher profits, urban density should have an impact on firms' location choices. Research in that area focuses on Foreign Direct Investments (FDI) and firm creations. Market size, measured by local total income or employment in the manufacturing or service sector as well as market access, measured by distance to the main cities in a country, are all found to positively affect firms' location choices (Combes and Gobillon, 2015).

In the context of employment growth, local total employment is used to estimate its effect on employment growth. In general, local market size has a positive effect on industrial employment growth. The effect on employment in the service sector, however, is mixed, depending on the country analyzed (Combes and Gobillon, 2015).

In the following, the focus will be on agglomeration economics affecting productivity and more so innovation.

3 Theoretical Framework

In this section, the theoretical interplay between peer effects, agglomeration effects and innovative activity in the literature is presented to understand the mechanism that may lead to productivity effects on GitHub with an increase in cluster size. To begin with, peer effects and productivity more generally are analyzed. Then, turning to the field of agglomeration economics, the links between peer effects and urban density are explained. To end, the difficulties for estimating agglomeration effects are discussed.

3.1 Peer Effects and Innovation

Invention and innovation are often regarded as similar things. However, as Carlino and Kerr (2015) point out by referring to Schumpeter (1939), invention means the creation of something non-existent to date. Innovation describes the process of putting that service or product up to sell. So the commercial gain of a new product will come to place where it was put in place, i.e., the location of the innovation, which might not necessarily be the same location of the invention.

An innovation has two parts: the creation of a new idea and its commercialization. Both are complementary to each other and necessary for innovative activity resulting in an increase in economic growth (Schumpeter, 1939). If an invention was made at a university in one city, but put in place in another, the immediate welfare effects are rather observed in the latter, where the commercial process took place (Carlino and Kerr, 2015).

These differences are also reflected by patents. Patents inherit by definition a novelty, however not every patent will ever result in a sellable good. Therefore, the rate of patents by location

does not necessarily imply higher monetary gains in the case of a higher patent rate (Carlino and Kerr, 2015). In most cases, a patent is never commercialized. For economic growth, though, commercially successful innovative ideas are of interest to analyze (Carlino and Kerr, 2015).

The majority of open source projects contain content for real-world applications, e.g., a project on an email client or a desktop application for SoundCloud, a online music streaming platform (Borges et al., 2016). Commits to these projects, thus, should inherit commercially important input.

Innovations can be distinguished in many dimensions. Either in their form of impact, incremental or radical, or in their form itself, as an improved product or rather an improved process. To find appropriate measures capturing these aspects is arguably difficult. Patents, despite their flaws, embody to a high degree innovative activities (Carlino and Kerr, 2015).

An increase in patents is often seen as an increase in productivity. In times of a rising number of non-routine tasks and learning on the job, peer effects and collaborations undergo an increase in importance as determinants of productivity. Knowledge flows among workers seem to raise their productivity and innovative activity. These knowledge spillover effects occur in collaborations, but not by simply working next to each other (Borjas and Doran, 2015). As expected, knowledge spillovers specifically matter in high-skilled and high-innovative occupations. In low-skilled occupations, it seems, peer effects increase the productivity of workers rather via the channel of social pressure (Cornelissen et al., 2017).[12] Though, as especially innovation tends to cluster to a greater extent, one source of agglomeration economics are knowledge spillovers between individuals (Carlino and Kerr, 2015).

Even though coding itself is a solitary task, collaborations and knowledge spillover effects among software developers matter as well. As in most non-routine tasks, individuals benefit from interactions with others, because, based on them, they can benefit from differences in cognitive frameworks and value sets. These interactions then lead to increases in their own skill set, e.g. by learning a new programming language or new commands in a programming language (Casalnuovo et al., 2015).

For new ideas, which then lead to innovations, especially collaborations matter. The advantages of teamwork for innovative activities are observed by an increasing share of multiple-authors journal articles across all research fields. Patents similarly are filed by a rising number of teams (Wuchty et al., 2007). A large number of factors can be the source of this pattern, varying in importance across research fields. An increase in capital intensity in research might lead the shift towards collaboration in laboratory sciences. Another factor might be the overall increase in researchers and thus, fostering specialization and more diverse teams. With a higher number of researchers,

[12]Social pressure in the context of peer effects describes the feeling of shame or guilt due to lower productivity in comparison to co-workers. Workers might act on that by increasing their productivity (Cornelissen et al., 2017).

the individual researcher is able to focus more on a narrow topic and acquire profound knowledge. Teams of such specialized researchers then cover a broader range of topics and hence, they are more diverse. Moreover, a reduction in communication costs possibly decreases the social network losses which in turn makes collaboration work more efficient (Wuchty et al., 2007).

These findings imply that individual effort and peer performance are complements. An individual only gains from a collaboration by raising her own efforts. The increase in individual effort in collaborations is based on the accumulation of new knowledge as a result of exposure to better peers. The extent of this increase describes the importance of knowledge spillovers as a link between the two (Cornelissen et al., 2017).

Knowledge spillovers in innovative processes depend on the proximity between the peers for which they occur. Proximity can refer to the closeness of intellectual content or geographic distance. Both seem to play a role. Peers that are geographically close to the individual might help her in the workplace, e.g., showing how something works. On the other hand, peers that are close in the intellectual space might foster new output creation within an individual by their ideas (Azoulay et al., 2010). In both cases, peers complement an individual's work via their proximity to her, and thus, increase her productivity, however in two different ways. Both are likely independent of each other, implying a greater increase in productivity as a result of both being present at the same time (Azoulay et al., 2010).

Additionally to the two factors, Borjas and Doran (2015) add the sphere of collaboration networks to the factors. It matters if individuals just work next to each other as colleagues, but might not interact very much or if they have high interactions as a result of collaborations.

These spillover effects via all spheres seem to be the highest if the peer inducing the effects is of high importance. Often, this is measured by the number of citations. A highly cited peer tends to have the highest spillover effects on her co-workers. These individuals might have very innovative ideas and explore fields that receive less attention. They are able to lead the focus of their surroundings to new topics (Azoulay et al., 2010).

3.2 Agglomeration Effects on Productivity

The sphere of geographical proximity is the focus of the literature on agglomeration effects. Agglomeration effects describe the benefits of a high urban density on several aspects as wages, productivity, or incomes. Cities with higher urban density are places of higher labor and production costs. More firms compete for workers, which in turn raises the wages. The increase in production costs is caused by sparse land, as more firms for production as well as individuals for living, demand land. As a result, the production costs in cities are higher than in locations with a lower urban density. Yet, firms are willing to accept these costs due to production benefits that seem to increase with urban density. Agglomeration economics tries to identify the underlying

determinants resulting in the advantages of cities (Rosenthal and Strange, 2020).

3.2.1 Causes of Agglomeration Effects

Marshall (1890) states three factors for agglomeration economics, namely, input sharing, labor market pooling, and knowledge spillovers. The latter is described as an unplanned process and requires the highest proximity (Marshall, 1890). It seems, that in the case of knowledge spillovers, communication costs increase to a greater extent with distance. That causes knowledge spillovers to be more localized to foster the unplanned process. New information technologies that offer a reduction in communication costs in more distant interactions are complements to in-person interactions, not substitutes (Rosenthal and Strange, 2020). Theory predicts decreasing marginal returns in productivity gains as a result of agglomeration economics. The gains of an additional skilled worker likely decrease, the higher the number of already present skilled workers is (Combes and Gobillon, 2015).

Agglomeration effects also seem to matter to a varying degree for different types of workers. Rosenthal and Strange (2012) find that the gains of agglomeration are smaller for female entrepreneurs than for male entrepreneurs. Thus, the increase in communication costs with distance differs between workers. This implies heterogeneity in the productivity effects with an increase in cluster size.

Research clusters tend to evolve close to universities as they complement the knowledge creation and transmission. The closer, the better knowledge spillover can occur. Andersson et al. (2009) suggest that about half of the productivity gains due to agglomeration effects are located within eight kilometers of a newly founded university. Hence, agglomeration effects decay rapidly with distance. A great number of research underlines this hypothesis by providing evidence for knowledge spillovers at the metropolitan level (Moretti, 2021; Rosenthal and Strange, 2020).

This is in line with other findings regarding the heterogeneity in agglomeration effects. In most industries, positive agglomeration effects can be found, except agriculture. The results are as expected, as agriculture relies more on free land (Foster and Stehrer, 2009). Manufacturing on the other hand benefits more from agglomeration effects than the service sector (Melo et al., 2009). This in some aspects contradicts the findings of others about cognitive and social skills being rewarded more in denser cities than motor skills and physical strength (Bacolod et al., 2009). Some suggest that only non-routine occupations gain from agglomeration (Andersson et al., 2014). The manufacturing sector is rather seen as a more routine-intensive sector, whereas the service sector depends more on social skills and non-routine tasks (Autor et al., 2003). The higher benefits of the manufacturing sector due to agglomeration effects may be that the production process is enhanced by closely located suppliers.

The mechanisms resulting in higher productivity with larger city size range from task spe-

cialization, worker mobility between firms, labor pooling and training. The hypothesis of task specialization implies that with a larger local labor market, a finer division of labor is possible. Workers may become more specialized and therefore more efficient and productive in their tasks. This process is enhanced by a larger city and, as a result, a larger local labor market (Combes and Gobillon, 2015). Knowledge spillovers might evolve through worker mobility between local firms. A new more productive worker in a firm might increase the productivity in the given firm. Evidence supports this by less on-the-job-training in larger cities (Combes and Gobillon, 2015).

A further mechanism regarding learning as a cause for productivity gains with increases in city size may be that workers at the start of their career move to a bigger city to improve their skills via interactions with more experienced workers. The latter may stay in cities to pass on their knowledge to future generations (Glaeser, 1999).

Next to a better transmission of knowledge, the creation of new knowledge may be enhanced with an increase in city size. A theory by Duranton and Puga (2001) suggests that a larger city may be more diverse. It provides many opportunities for trial-and-error in the product development process and via that, results in more innovations. Therefore, young firms should settle in larger cities and, when being more mature and settled with their products, relocate to smaller towns. In France, this pattern of firm relocation could be found (Duranton and Puga, 2001).

Finally, communication as a mean of knowledge spillover seems to also be enhanced in cities. Via communication, knowledge can be transferred between workers, and as suggested, communicative activity is stronger with urban density (Charlot and Duranton, 2004).

It seems for coding, as an skill-intensive and non-routine task, positive agglomeration effects on productivity should take place. Furthermore, GitHub as a coding platform with social characteristics likely further fosters these effects.

3.2.2 Endogeneity Concerns in Estimating Agglomeration Effects

There are several difficulties when estimating agglomeration effects on productivity. One is regarding the quality and quantity of labor. High-skilled workers may self-select themselves to work and live in a larger city to a greater extent than low-skilled workers. This may be because they value the amenities a larger city offers more, or because, on the grounds of history, high-skilled workers tend to reside more in cities and pass on their skills to future generations. This could, next to an endogeneity concern, also be a case of reverse causality. As a result of more amenities and productivity gains in a larger city, urban density increases and a higher presence of high-skilled workers in cities may not be the result of agglomeration effects (Combes et al., 2010). Moreover, more productive workers may choose a larger city to have a higher benefit of the productivity gains of a city. In that case, individual ability and urban density would be correlated (Combes et al., 2010). Not taking these factors into account would lead to a bias in any estimation of the causal

effects of urban density on productivity. A way to overcome this bias may be by using individual panel data and introducing individual fixed effects. In that case for any time-invariant individual characteristics affecting their productivity is controlled for and, hence, the bias reduced (Combes and Gobillon, 2015).

Another concern are local productivity shocks that simultaneously affect urban density and productivity. For instance, a city might invest a lot in research and via that attract new inventors, increasing urban density and next to that increase productivity. As a result, this leads to a bias in the estimate (Combes and Gobillon, 2015). Different ways to address this concern were introduced in the literature. For instance, by introducing cluster or city fixed effects as well as interaction effects of city and time, these would capture time-invariant and time-variant cluster characteristics introducing a possible bias in the estimate (Combes and Gobillon, 2015). They would not, however, resolve a reverse causality issue, meaning an increase in productivity translating into more amenities in a city. This in turn would attract more high-skilled worker. Historical and geographical instrumental variable approaches were introduced to mitigate these biases. For instance, long lagged values of population or density are considered relevant as they likely affect urban density nowadays, however assumed to be exogenous to unobserved local productivity shocks. Other instruments consider the subsoil of a location (Combes et al., 2010).[13]

Alternatively, shift-share instruments are implemented by using exogenous aggregate shifts to predict local changes (Farhauer and Kröll, 2009). If, e.g., the employment in other software firms in other cities increases, the local employment in the focal software firm may also increase. The geographical variation in software firms is used to predict changes in local employment in a software firm. In that case, local productivity shocks likely do not affect the instrument, i.e., employment changes in other cities.[14]

The dynamics of agglomeration effects are a minor cause of bias. In most specifications, they are assumed to be contemporaneous. Although, it could be the case that density effects have not only an impact on productivity in the same period but result in an ongoing raise in productivity. Furthermore, agglomeration effects of one city may also affect neighbouring cities. Yet, as agglomeration effects seem to decrease rapidly with distance, this seems a minor concern (Combes et al., 2010).

[13]The considerations in that case are such, that soil composition or depth to rock were important determinants of agriculture in the past. The developments from agriculture to manufacturing and service occurred in places where people already settled. These instruments are possibly good predictors of urban density nowadays, however, are unlikely correlated to productivity gains in present times in a city (Combes and Gobillon, 2015).

[14]Moreover, natural experiments or Generalised Method of Moments (GMM) estimation are further alternatives used in the literature to mitigate the concern of biases in the estimates (Combes and Gobillon, 2015).

4 Research Design

Now turning to the empirical framework, first, the platform GitHub is explained in more detail. Next, the steps for the data preparation as well as the data limitations are discussed. Then, the data is described and the estimation strategy is presented.

4.1 GitHub

GitHub is the world's biggest code hosting site and is based on the `git` revision control system (GIT, 2021). The platform launched in 2007 and since then experienced an increase in popularity across software developers (Laurentsyeva and Fackler, 2020). Its attractiveness lies in the easy usage and, in its basic version, no costs. The platform exhibits features of a social network in line with its motto: "GitHub: social coding" (Lima et al., 2014). After registration, users can create a repository to which code can be *pushed*, i.e., uploaded. The platform supports every programming language. Each repository has one owner. After invitation, other users can become project members. They can modify the repository's content and approve or disregard submitted contributions by others (Lima et al., 2014). A repository can also be *forked*, i.e., copied, and in this case independently worked on. A *commit* represents the sum of code changes a user sends to the repository via a *pull request* in a session. Collaborators can review and *merge* it into the repository or discard it (Lima et al., 2014). Regarding the social features of GitHub, it is possible to *follow* other users and then be notified about their actions. Another trait is to *star* a repository. This way, it is bookmarked and can be found more easily later in time. The number of stars or forks per project is seen as a measure of a project's popularity among users (Lima et al., 2014).

When registering, users are able to provide their real name, location and other biographical information (Laurentsyeva and Fackler, 2020). Each repository can be set private or public. The data used in the analysis contains only commits to public projects. In this case, any actions taking place in a repository are observable by everyone (Laurentsyeva and Fackler, 2020). Furthermore, on every user page their actions are shown and everyone can see to which projects a user contributes as well as the timing of it (Laurentsyeva, 2019).

The social features of the platform allow users to develop impressions of other users' social and technical skills and behavior (Casalnuovo et al., 2015). For instance, Tsay et al. (2014) find that users next to forming opinions about the abilities of others also use these features to control their own online standing.

The motivation for contributions on open source platforms was analyzed in the literature and the findings suggest heterogeneity across individuals. The driving motives spread from career concerns in the sense of building reputation, paid work at software companies to working on own software projects or helping others (Belenzon and Schankerman, 2008; Hergueux and Jacquemet,

2015)

Positive productivity effects of co-workers on individuals seem to be mitigated in an environment of fixed wages. The source may be a free-rider problem. If a more productive co-worker enters the team, other individuals might decrease their effort because of the more productive worker taking up a greater share of the work (Herbst and Mas, 2015). The motivation on GitHub is mainly intrinsic and, hence, users likely won't fall into the trap of a free-rider problem, but rather knowledge spillovers lead to an increase in productivity.

The social characteristics of the platform affecting collaborations among software engineers was studied as well. Users are more likely to join the projects of users they have social connections with (Casalnuovo et al., 2015). In these cases, productivity, measured by a user's number of commits, is enhanced at the start of the collaborations as well as in the long run. The authors also find that tighter social connections lead to lower productivity in the beginning, but larger increases in the long-term (Casalnuovo et al., 2015). One reason might be, that in the beginning of a project, where the new member has to understand the project structure, more prior links lead to more communication and less output. After this initial phase, a user then might be able to focus on specific parts that raise output and productivity (Casalnuovo et al., 2015). These developments depend on the one hand on the programming language. If a user is more familiar with it, productivity increases are higher. On the other hand, it depends on the level of social connections with other project members. If they are stronger, the productivity increase is further enhanced (Casalnuovo et al., 2015).

A feedback and recognition system as well as a community infrastructure increase the contribution's quality (Wright et al., 2020). On GitHub, users can add comments to their commits, and via that send feedback to the other project members. Timely feedback was found as an important factor for innovation incentives (Manso, 2011).[15]

When forming new teams, individuals especially value previous collaborations in self-organized networks. In that case, they gain from past interactions, as they were able to build mutual trust and have a certain level of knowledge about the other's abilities. Similar to the findings of Azoulay et al. (2010) about greater productivity increases due to collaborations with eminent scientists, individuals also value more collaborations with more established users. They seem to raise their own motivation and their impression of possible project success (Casalnuovo et al., 2015).

Acquiring knowledge through the work with others on open source platforms was also found to positively impact entrepreneurship. Through collaborations, users make links which then may lead to starting entrepreneurial businesses (Wright et al., 2020).

Thus, agglomeration effects may also affect the productivity of software engineers. As a result

[15]Manso (2011) shows that timely feedback on performance especially matters for exploration. It gives the individual information to improve future work.

of a move to a larger cluster, a GitHub user can form more social connections with other users in the cluster. Based on these connections, the user will contribute to a greater number of projects with a higher number of commits and a productivity increase is observed. Especially in GitHub projects with a small number of project members, users tend to be geographically close to each other (Casalnuovo et al., 2015). This would be in line with the mechanism found by Jaffe et al. (1993). Instead of local inventors citing patents by other local inventors, now users commit to projects by other local users.

4.2 Dataset

In this section the generation of the data set is described. Furthermore, the limitations of the data are discussed and their possible effects on the results and the results' interpretation.

4.2.1 Commit Data

The full data used for the analysis is a combined version of several snapshots from GitHub Torrent (GHTorrent) (Gousios, 2013). GHTorrent creates snapshots of the activities on GitHub, e.g. user registration, projects and commits, and makes it accessible to everyone in a relational database. It is either available directly on their website or via the Google Cloud Platform. The commits recorded are only commits to public repositories.

The snapshots included in the data were taken on the 2015/09/25 (201509), 2016/01/08 (201601), 2016/06/01 (201606), 2017/01/19 (201701), 2017/06/01 (201706), 2018/01/01 (201801), 2018/11/01 (201811), 2019/06/01 (201906), 2020/07/01 (202007) and 2021/03/06 (202103). The activity stream of commits as well as data on the corresponding projects are taken from the snapshot 202103. The commits queried were limited to users that have a US or Canadian location stated in the user table of 202103. The user table contains several variables on the location of a user, namely the variables *location*, *long* (longitude), *lat* (latitude), *city*, *state* and *country_code*. Since the snapshot 201606 GHTorrent geocodes users via Open Street Map based on the self-stated *location* variable and creates the additional location variables. The commits are also limited to users that have some form of location information, either longitude (and latitude) or a non-empty location description in the 202103 snapshot.

Every user has a unique user id. Commits are matched via the author id, not the committer id, to the user id. The scope of the analysis is to observe productivity changes based on changes in cluster size, i.e., if a user produces more (new) output measured by more commits. This is more likely captured by more written commits then uploaded commits. Matching the commits by committer id might rather capture a higher activity on GitHub more generally and a link to higher productivity is less clear. The user might create a lot of pull requests with other users'

written content. Hence, matching commits by author id to users creates a better image of the user's knowledge output. Some users have several GitHub accounts (Casalnuovo et al., 2015). Unfortunately, we cannot account for these user aliases and might underestimate spillover effects as a result of less commits per user than she actually contributes.

The commit data contains all commits a user has ever generated after account creation until the date of the snapshot. Commits are matched via a project id to the project table to obtain information on the project's programming language. This way the commit's programming language is identified.[16] The programming language can be understood in a broader way and includes statistical programming languages such as R. For the analysis, only projects with a stated programming language are considered. Further information on the number of forks and the number of stars per project are added via the project and watcher table, respectively.

In total, there are 406 stated programming languages in the commit dataset. However, 17 programming languages cover 90 percent of all commits. Clusters will be defined as programming language $\times$ city. Including too many programming languages might result in a large number of zero clusters, i.e., in a city only one user uses a programming language. Thus, out of practical reasons, the data is limited to 17 programming languages. These are C, C#, C++, CSS, Go, HTML, Java, JavaScript, Objective-C, PHP, Python, R, Ruby, Rust, Shell, Swift, and TypeScript. To further aggregate cluster fields, CSS and HTML are combined as they are commonly used together.

Commits are aggregated to snapshot intervals. The first time interval comprises all commits to a project after the user account was created and up to 25 September 2015. The second interval contains all commits to a project between 26 September 2015 and 8 January 2016. This system follows for the other snapshot intervals and results in ten time intervals.[17]

If a user commits in a programming language other than the ones included in the analysis, the user is recorded with an empty number of commits for the programming languages used beforehand, given they are included in the sample. For example, if a user commits in Java, Python, and Ruby in time interval one, in FORTRAN, a programming language which is not included, in time interval two and in Ruby in time interval three, for time interval two zero commits are filled for the programming languages Java, Python and Ruby, the programming languages used in the previous time interval. If a user only commits in a programming language not included in the sample, she is excluded.The data set for the 16 programming languages considered contains 712,078 unique users and in total 15,485,608 observations.

To remove inactive accounts, the data was further limited. It was filtered for only users that commit in at least two time intervals or, if the account was created in the last time interval and the user committed in that time interval, those users are included as well. This results in 13,471,157

[16] In a project, files in several programming languages can exist. GHTorrent defines the project's programming language by the programming language that makes up the largest number of byte counts in the project.

[17] In the following, we will use the terms snapshots and time intervals as equivalents.

observations with 500,595 users.

4.2.2 User Data

Regarding the user accounts and their location, a snapshot contains only the currently stated location. To observe user location changes, the snapshots of the user accounts from 201509 until 202103 were combined. Accounts can be marked as fake, i.e., that a user appears only as a committer or author of a commit, but does not have own projects or creates other events as push or pull requests. Those users are included in the sample as we are only interested in commit events and not in other events such as project creations.[18]

There are two types of accounts, users and organizations. Organizations, a group of users that appear as meta users, can only own projects, but can not do any other actions. Therefore organization accounts are excluded.

The user table contains, as mentioned above, several variables on the location of a user, namely the variables *location*, *long* (longitude), *lat* (latitude), *city*, *state* and *country_code*. In the snapshots 201509 and 201606 only the *location* variable is available. In the other snapshots, where possible, users were geocoded by GHTorrent via Open Street Map and the additional location variables were created. However, in about 50 percent of cases, i.e. 115,707,489 observations, no location information is available at all. In the cases of no location information available at all, we use the non-empty location of the snapshot before or, in the case of no available before snapshot, next snapshot.

If no coordinates are available, but other location information, these are taken to geocode the user. In practice, combinations of the *location*, *city*, *state* and *country_code* variables are matched with data sets on "us.cities", "canada.cities" and "world.cities" provided by the R-package *maps* (Becker and Wilks, 2018), which contain coordinates on the cities. For US and Canadian cities, further, more comprehensive data sets provided for free by simplemaps was used (Simplemaps, 2021).

Users are further matched to one of the 179 US "Economic Areas" defined by the Bureau of Economic Analysis (BEA) or the Canadian equivalent, namely one of the 76 economic regions defined by Statistics Canada. In many cases, "Economic Areas" are comparable to Metropolitan Statistical Areas (MSA). However, the San Francisco Bay Area or New York, i.e. in the case of larger areas, the "Economic Area" covers the entire economic region and, thus, is larger than the corresponding MSA. Economic Areas are in the following called "cities". Finally, the user data contains 1,004,139 users with always US or Canadian locations and matched to economic areas, with 7,002,495 user snapshot observations.

[18] Accounts marked as fake are about 22.8 percent of all user observations.

4.2.3 Full data

The combined commits and user data results in 11,364,475 observations with 445,230 unique users. The difference in the number of users between commits data with 500,595 and the full data with 445,230 stems from the fact that in the commit data, there are users not always living in the USA or Canada or users that could not clearly be matched to a location. On the other hand, in the user data, there are users that only commit in programming languages not included in the analysis or commit never or only once. As mentioned, these user observations are excluded from the sample. The full data set is used to calculate the cluster size. For the regression, only users that are observed in all snapshots, i.e. geocoded and with non-zero commits in all time intervals, are used. These are 2,095,978 observations and 17,302 unique users.

In section 4.3 both, the full data and the regression data are described in detail. They are very similar regarding the distribution of commits per programming language. On the other hand, the projects users commit to tend to have more stars and users included are more active regarding their number of commits. This suggests that those users are more likely to experience productivity effects in larger clusters and that we are able to observe this by their high, and possibly even higher activity on GitHub. For users that generally commit less, it is harder to identify an increase in their commits on GitHub. They might experience positive productivity spillover effects from denser local clusters, though this might not result in a higher number of commits, as they were less active on GitHub to start with.

Calculating cluster size using the regression data might result in clusters actually too small. If a user in our data does not commit in a time interval, it might be the case that she commits to a private project. Even if the user does not commit at all in a time interval, she might still have positive productivity spillover effects on the other active users.

For robustness, we estimate the elasticity between cluster size and productivity loosening the length of time intervals with non-zero commits per user. In this specification, the elasticity becomes less significant.[19]

4.2.4 Clusters

The clusters are constructed as programming language $\times$ city. Cluster size S for user i in time t in programming language f in city c is the number of users in programming language f in city c excluding user i relative to all users in a programming language f in time t. More formally speaking, cluster size is calculated as:

$$S_{-ifct} = \frac{\sum_{j \neq i} N_{jfct}}{\sum N_{jft}}$$

[19] See section 5.5 for the discussion of the results.

where the summation of users N is across all users j in city c in programming language f in time t but user i. Cluster size is defined in relative terms by dividing the sum of users in city c in programming language f excluding user i by the total number of users N in programming language f in time t.

The programming language per user in a snapshot is determined by the projects a user commits to. In practice, a user, that commits to projects in the programming languages Python and JavaScript in the first time interval, is assigned to the clusters Python $\times$ city and JavaScript $\times$ city in time interval one.

4.2.5 Data Limitations

(1) The first main limitation of the data are user locations. Location changes are only observed if a user decides to change the location in her account. Hence, it depends on how thoroughly and regularly one cultivates her account. Users that might be less active might also put less emphasis on updating their account on a regular basis. However, these users might also benefit less from denser clusters or at least it would be more difficult by only looking at their commits. If they are less active and so also in the case of a move, a change might not be observed. On the other hand, if they become more productive, i.e., commit more after a move, even though they did not update their location, it would lead to a bias in our estimates. Cluster size would be measured incorrectly, as a user is assigned to a city she actually is not currently living in.

A large number of users in our full data, that is, 27,881 users, have in nine snapshots no location information at all and are filled by their before or after location. In these cases, we assume no location changes and the measurement error is further magnified. This measurement error in the independent variable, cluster size, should lead to a bias towards zero, also called attenuation bias (Pischke, 2007).

On the other hand, users used in the regression are users, that are active in all time intervals and, hence, use GitHub on a regular basis, what makes them more likely to update their accounts. The measurement error in the independent variable, namely, cluster size, possibly occurs for less active users that are not part of the analysis.

(2) A second limitation stems from the fact that the data contains only commits to public projects. We do not observe shifts from public to private projects due to a larger cluster, but only a decrease in commits. Although this decrease can be caused by different reasons, not only a shift to commit more to private projects. Another possible reason might be that a user spends less time committing as a result of a new job, and hence, an actual decrease occurs and not a shift. Yet, we cannot distinguish the two with the data given. The reason to contribute to open source projects is very broad and there are different kinds of public projects. Some are leisure projects, users work

on next to their main job. Other projects are work projects, either for users' work or as their main job. These work projects can also be a way for a user to show her skills and, via labor market signalling, attract job offers. Then again a decrease in commits might be because the worker has a new job and is less active on GitHub and cannot be interpreted as a decrease in productivity.

(3) The third limitation is the content of commits. Commits are any change of code. This can be correcting a typo, but also making major changes in the code. A commit could also be uploading files to a project to save them on GitHub. In that case, a user might commit several times a day. Therefore, the extent of knowledge creation might vary significantly across commits. However, we do not analyze the content of the commit and might compare a typo correcting commit with a major code changing commit. To reduce this possible concern, in a further analysis we restrict the sample to users with commits to only projects with at least one star. The number of stars or the number of forks per project are quality indicators for a project. A commit to a project with more stars or forks is more likely to be of higher quality as it is more difficult to commit to such a project. Nevertheless, we still can not completely rule out the case of minor code correction in these commits neither.[20]

A possible solution for future work could be to further examine the content of the commit. Casalnuovo et al. (2015) take the number of files touched with commits or number of changed code lines, either added or deleted, as productivity measures of GitHub users.[21]

4.3 Descriptives

In this section, we are going to describe the full data, i.e., the data of all users on which cluster calculation is based on. This data set will be compared to the regression data, in which only users with commits in all time intervals are included.

4.3.1 Commits, Users and Projects

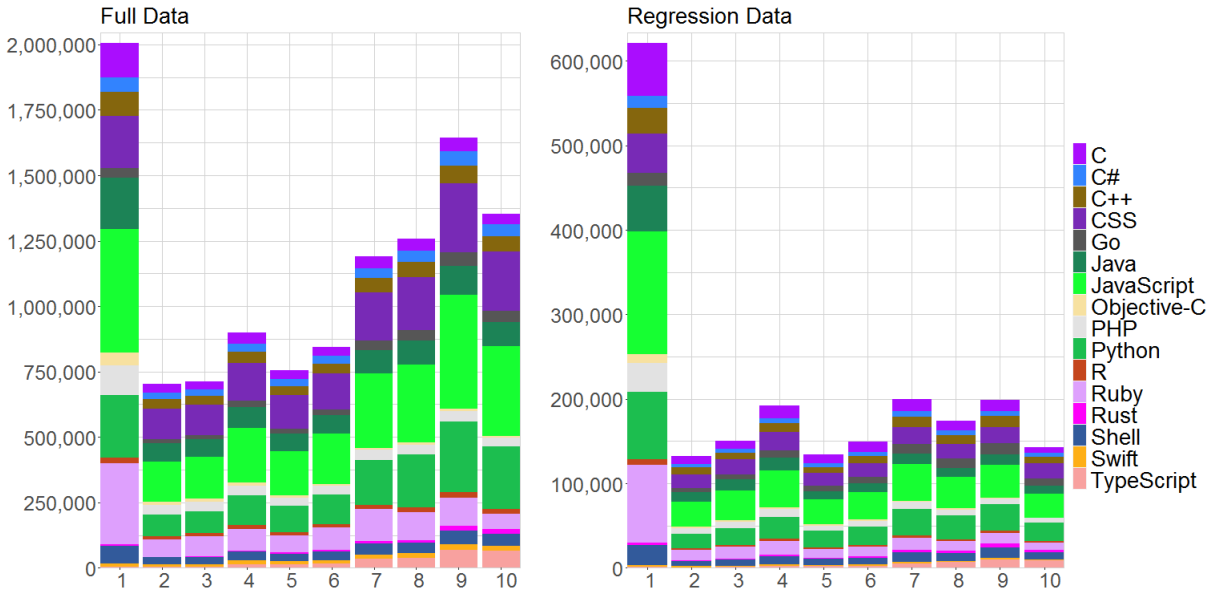
The left graph of Figure 1 shows the number of observations per time interval and per programming language in the full data.

[20] Additionally, a commit might be assigned to a different programming language than it is actually written in. The commit's programming language is determined by the project the commit is to, but a project can contain files in several programming languages. The 'main' programming language of a project is based on the largest number of byte counts. However, this should cause a minor measurement error, as the majority of commits to a project are assigned to the correct programming language.

[21] As Casalnuovo et al. (2015) note, using the number of changed code lines might come with noise. Users could copy code from other files and add this to a file or merge codes. This leads to imprecisely measuring productivity. However, they compare their results based on line changes and file changes and find that the results are consistent across measures.

It varies based on the different lengths of the time intervals. For example, the first, seventh and ninth time interval capture a longer time period in comparison to the other time intervals. The peak in observations in the first time interval is due to the fact, that it contains all commits to projects after an account was created until 25 September 2015. This time interval captures the longest period of time and results in the greatest number of observations per time interval. The seventh and ninth time intervals are about a year long, whereas the other time intervals are about six month long and, thus, represent a larger number of commits. For the regressions, time fixed effects are included to take this variation of commits into account. The distribution of observations in the regression data, shown on the right of Figure 1 is similar to the full data, but the level of observations, as expected, is a lot smaller. The number of observations per time interval is cut in half or decreased even more when restricting the sample to only users observed in all time intervals.

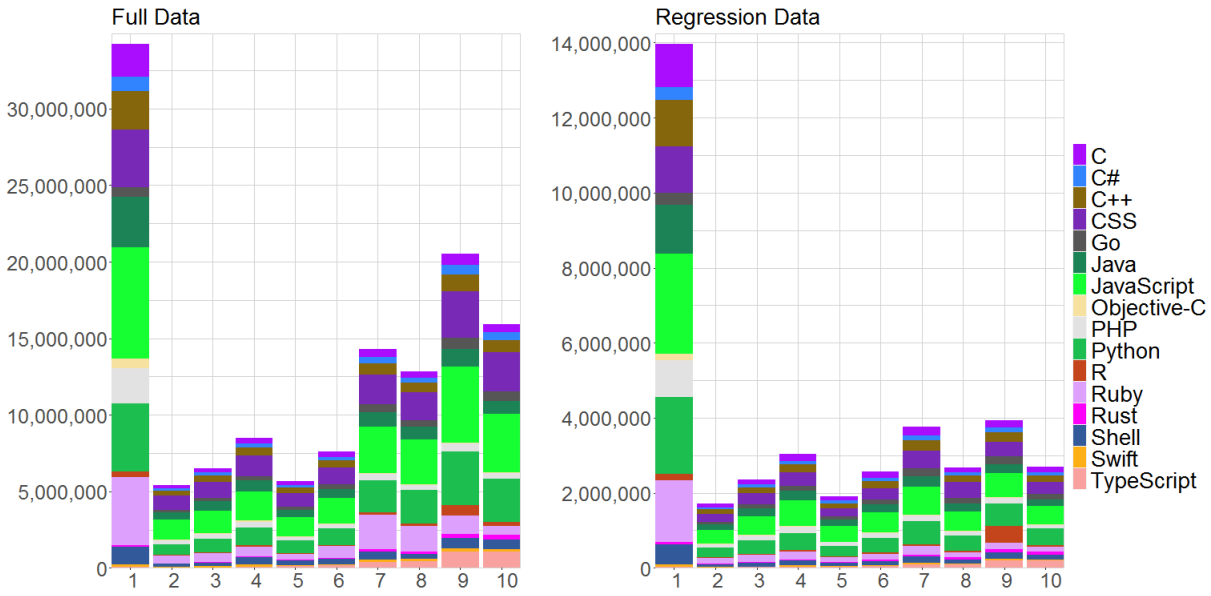
Figure 1: Number of Observation per Snapshot and per Programming Language



In both figures, there is noticeable an overall increase in users, projects and commits over time, similarly shown in Figures 2 and 3. The distribution of programming languages in the observations is comparable to the distribution of the programming languages in the number of commits as shown in Figure 2.

It plots the sum of commits per programming language for all 16 programming languages. To take into account differences in the number of commits per programming language, programming language fixed effects as well as programming language $\times$ time fixed effects are introduced in the regression. The latter controls for time trends in the usage of programming languages.

Figure 2: Sum of Commits per Programming Language per Snapshot



Comparing the regression data with the full data in Figure 2, the distribution of commits per programming language is also very similar, but levels are lower.

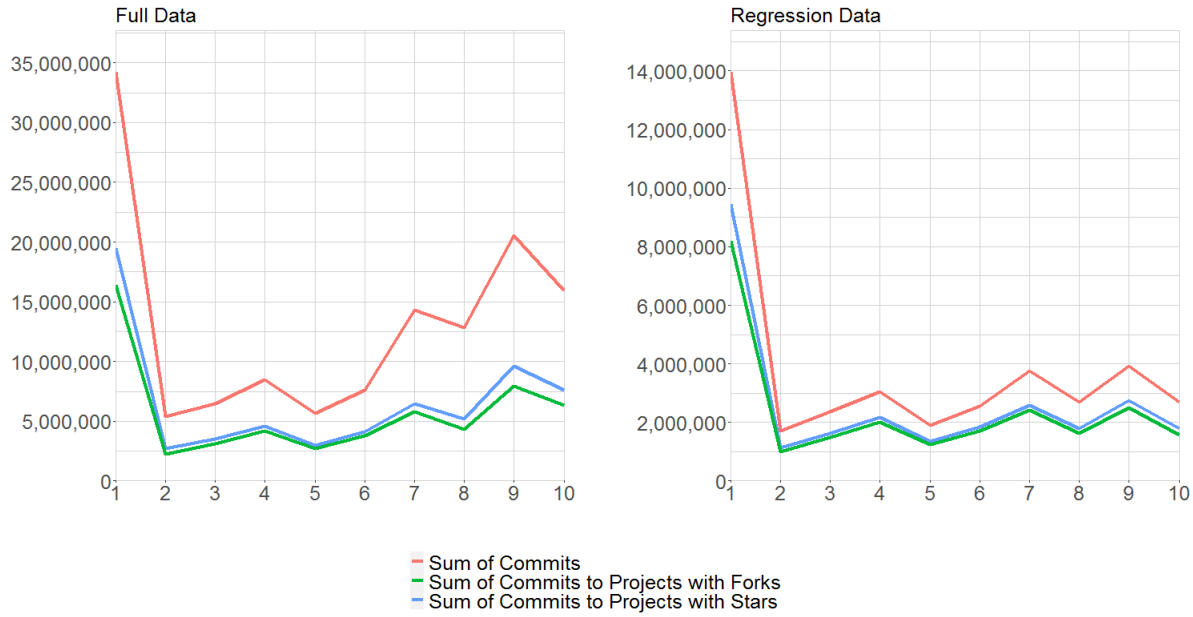
Table 11 shows the share of commits per programming language for the full data and Table 12 for the regression data. The programming language effects in the regression control for the differing popularity of programming languages. It shows the minimum, median, mean, and maximum total number of commits per user for the 16 programming languages in the full data. JavaScript has the highest median total number of commits per user with 22, followed by CSS with 17 commits per user.

The total number of commits per programming language per user is very broad. The maximum total number of commits per user is 406,159 in R, followed by Shell with 281,997. An explanation for these high numbers for users might be that they are very active on the platform because they use GitHub for their work.

Regarding the regression data in Table 12, first of all, the mean increases for all programming languages. The increase in the minimum is a result of the definition of the data. The regression data contains only active users, i.e. with non-zero commits in all time intervals as compared to the full data, that also contains users not committing in a time interval.

The fact that the maximum number of total commits for most programming languages remains unchanged shows, that the regression data includes more active users.

Figure 3: Sum of Commits per Snapshot



If one looks at the sum of commits shown in Figure 3, it is increasing over time. The difference between the total sum of commits and the sum of commits to projects with at least one star in Figure 3 is a result of the great number of projects with zero stars. The lines for the sum of commits to projects with stars or forks show only the commits to non-forked projects with stars or forks.[22]

The number of stars and forks are an indicator for the quality of a project (Laurentsyeva, 2019). Commits to projects with zero stars or zero forks might be the user's own project in which she possibly only saves files. These commits might not necessarily be an indicator for productivity. The distribution of stars, shown in Table 13, is highly skewed towards zero. For non-forked projects with at least one star, the median number of stars is two while the highest value of stars per project is 259,118. 75 percent of all non-forked projects with at least one star have between 7 and one star. The distribution of the number of forks per project is very similar. The median number of forks for non-forked projects with at least one fork is two and about 75 percent of those projects have six forks. The maximum number of forks is 145,997.

This suggests, there are a lot of 'small' projects regarding the number of forks and stars. To commit to a project with a great number of stars or forks demands a higher quality commit. Though, this might take more time and the absolute number of commits might be smaller than the number of commits to a project with less stars or forks. To account for this quality concern of commits, the baseline model is used with a sample restricted to users that commit in all time intervals to only projects with at least one star and also only those commits are considered. Hence

[22]Non-forked projects are the projects at the root of forks, one could say the original project. They are not forked from another project.

the sample is restricted to potentially more high-quality users with more high-quality content.

General popularity of projects, which the number of stars or number of forks are proxies for, likely affects the total number of commits to a project. Project effects included in the regression account for this aspect.

In the full data, the total number of commits per user, considering all programming languages, shown in Table 13, ranges from zero commits to 411,048 commits. Again, it might be that the user with a total number of 411,048 commits, uses GitHub for work, although we can not confirm this by the data at hand.[23]

The most active user with 411,048 total commits is also included in the regression data, as the maximum stays the same in Table 14. Concerning the average total number of commits in the full data, it is 295.22 compared to 2,232.14 in the regression data. The median is 70 total commits per user in the full data and 1,092 in the regression data. This strong increase in the median is partly a result of the data definition. As users with zero commits in a time interval are removed, the minimum and first quartile already shift from zero and 22 to 36 and 535, respectively. User effects in the regression model control for the differing activity levels across users.

Not only the distribution of commits is higher in the regression data, but also the number of forks and stars of the projects included. While 75 percent of projects with at least one star in the full data have between one and 7 stars, it is between one and 16 in the regression data. Similar occurs for the number of forks, given a project was at least once forked. The number of forks for 75 percent of non-forked projects with at least one fork in the full data ranges from one to six in comparison to one and 11 in the regression data. Next to the more active users remaining in the regression data, also projects with higher quality stay in the data. Likely, these two belong together. The quality of more active users' commits is higher, as they, with higher activity level, possibly have more experience on GitHub and the programming languages. Then, they are able to commit to projects that demand higher quality of the commits.

Out of the 445,230 users in the full data, 92,510 users moved in sum 39,617 times.[24] In the regression data, 5,402 of the 17,302 users moved a total of 4,971 times.[25] This suggests sufficient variation in cluster size.

On average, users use a total of 3.3 (median: 3) programming languages and similarly, on

[23] In some research (Casalnuovo et al., 2015) large commits are excluded as they might not capture typical developer behavior. As Hindle et al. (2008) show, both small and large commits are important steps in a project development and capture productive output.

[24] 70,862 users moved once, 18,577 users moved twice, 2,963 users moved three times, 106 users moved four times and two users moved five times. Moves occurred in the second time interval 8,324 times, 2,299 times in the third time interval, 16,079 times in the fourth time interval, 24,956 times in the seventh time interval and 65,681 times in the tenth time interval.

[25] 3,831 users moved once, 1,300 users moved twice, 245 users moved three times, 25 users moved four times and one user moved five times. Moves occurred in the second time interval 1,353 times, 251 times in the third time interval, 1,153 times in the fourth time interval, 1,759 times in the seventh time interval and 2,755 times in the tenth time interval.

average per time interval 2.09 (median: 2) programming languages in the full data. In the regression data, again users are active in more programming languages. The average number of programming languages used in total is twice as high as in the regression data with 6.36 (median: 6). Per time interval, the average number of programming languages is slightly higher with 3.21 (median: 3) programming languages.

The overall higher activity of users in the regression data suggests that they experience the largest spillover effects. They likely interact the most with other GitHub users and profit the most of the different skill sets. Furthermore, shifts in productivity should also be best observed for these users.

4.3.2 Clusters

Now we turn to describing the clusters, i.e., changes in cluster size over time. Clusters are calculated on the basis of the full data, and hence, the cluster sizes in the full data and the regression data are the same.

Table 15 shows the largest clusters for the top five most used programming languages, JavaScript, Python, CSS, Ruby and Java, as of the 202103 snapshot and Table 16 shows the distribution of cluster size per programming language.

For the five most used programming languages, San Francisco-Oakland-San Jose is in all cases by far the largest cluster. Between 10 to 14 percent of all users in the respective programming language in the snapshot 202103 stem from this area.

In the case of JavaScript, San Francisco-Oakland-San Jose makes up about 11.49 percent of all users in that programming language, followed by a profound gap by Washington-Baltimore of 6.3 percent and next Los Angeles-Riverside with 6.2 percent. The top ten cities cover 49.17 percent of all users in JavaScript in the latest time interval. Median cluster size for JavaScript with about 0.066 percent is Baton Rouge, LA-MS. The ratio between the largest cluster and the median cluster is 745.25. This means about 745 times more users stem from San Francisco-Oakland-San Jose than from Baton Rouge, LA-MS. The ratio between the 90th percentile and the median cluster size for JavaScript is 12.9. This shows that cluster size decreases rapidly moving away from the largest cluster.

For Python, San Francisco-Oakland-San Jose is also by far the largest cluster with 13.06 percent of all users stemming from this area. The ratio between largest cluster and the median cluster is 190.77. Therefore 190 times more users stem from San Francisco-Oakland-San Jose than from Tallahassee, FL-GA, the median cluster, with 0.068. A lot smaller with 12.22 is the ratio between the 90th percentile and the median cluster. The difference between largest cluster and 90th percentile cluster is even more extreme for Python than for JavaScript.

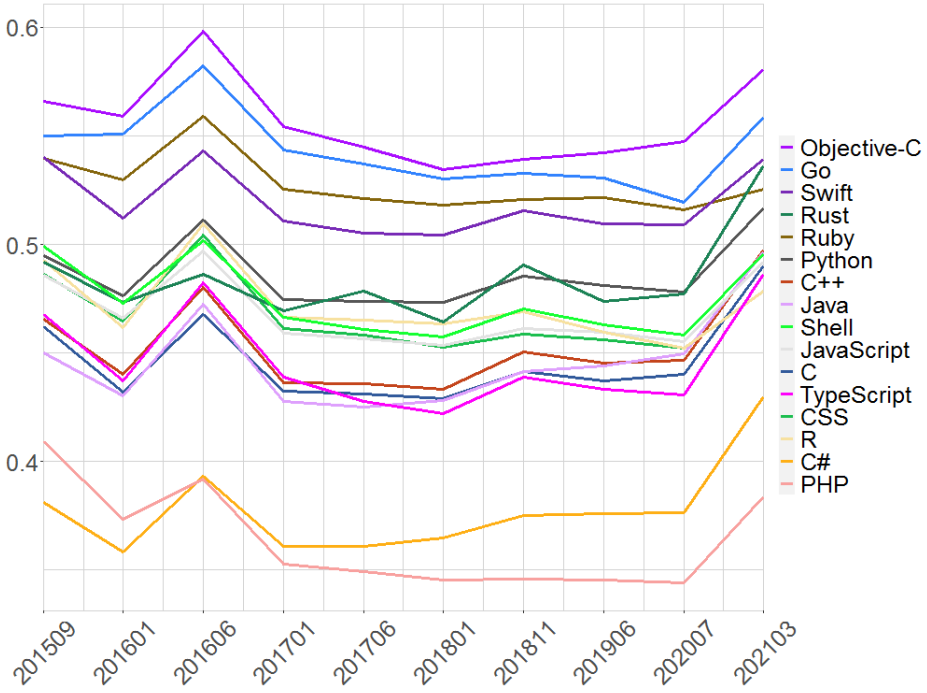
In the case of CSS there is a smaller difference between the largest cluster San Francisco-

Oakland-San Jose and Washington-Baltimore, DC, second largest cluster, with almost 10.66 percent to 7.62 percent. The ratio between the largest and median cluster is the smallest after Java with 156.65. This means that users from San Francisco-Oakland-San Jose commit about 157 times more than users from Kingston-Pembroke, the median cluster with 0.006 percent.

For Ruby, again the largest cluster is San Francisco-Oakland-San Jose with a cluster size of 13.85 percent and Washington-Baltimore, DC with 6.12 percent is the second largest cluster. Thus, the distance in cluster size between the largest and second largest is even more pronounced than for JavaScript. The ratio between largest cluster and median is 189.67 and 14.12 between the 90th percentile and median. Hence, again cluster size decreases quite rapidly.

Java is very similar to Ruby regarding cluster sizes and ratio. The largest cluster is again in San Francisco-Oakland-San Jose with 12.13 percent followed by Washington-Baltimore with 7.65. The ratio between largest cluster and median is 179.91 and 15.69 between 90th percentile and median.

Figure 4: Share of Top 10 Cities for All Programming Languages



As noticeable by Figure 4, users in the programming languages are already quite concentrated to start with. The figure plots the share of users originating from the top ten cities relative to all users in the respective programming language for all 16 programming languages. Especially for Objective-C with 58.07 percent of all users in that programming language stemming from only ten cities and Go with 55.84 percent in the tenth time interval, the clustering seems to be profound. Even though in no programming language the share is monotonically increasing, in general, though,

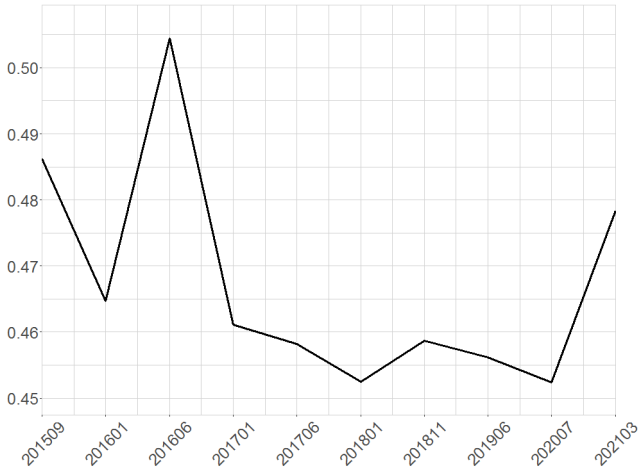
the share increases.

Looking more into the detail, for the top five programming languages in Figure 5, the increase in the share between the first and the last time interval is rather small with 0.56 (JavaScript), 2.2 (Python) and 4.62 (Java). It even decreased for Ruby with 1.41 and 0.78 for CSS. However, one has to take into account that the time period only spans about seven years, therefore greater changes might also take more time. Nevertheless, the figures suggest that over time users, possibly especially very active users, tend to cluster.[26]

Figure 5: Share of Top 10 Cities for Top Five Programming Languages

[26]See Figure 6 in the Appendix for the change in the share of top 10 cities for all programming languages.

(e) CSS



4.4 Estimation Strategy

To study the relationship between cluster size and productivity, we implement the following regression equation:

$$ln(y_{ijfct}) = \alpha\ ln(S_{-ifct}) + d_{cf} + d_{ft} + d_{ct} + d_c + d_f + d_i + d_j + \mu_{ijfct} \tag{1}$$

where y_{ijfct} is the number of commits of user i in time interval t to project j located in city c in the programming language f; S_{-ifct} is the cluster size in city c of the programming language f in time interval t, excluding user i; d_{cf} are city $\times$ programming language effects, controlling for city specific effects in programming languages; d_{ft} are programming language $\times$ time effects, accounting for trends in programming languages; d_{ct} are city $\times$ time effects, taking into account changes in cities over time; d_c are city effects, controlling for time-invariant city characteristics; d_f are programming language effects, accounting for time-invariant programming language characteristics; d_i controls for time-invariant individual effects and d_j for time-invariant project effects. Standard errors are clustered on the city $\times$ programming language level to take into account serial correlation. The error terms for users in a city active in a programming language are likely to be correlated with each other, hence clustering should be done on this level. Additionally, treatment, i.e., the change of cluster size, also occurs on the programming language $\times$ city level. Variation in cluster size stems from users moving.

For α being positive, it would imply a positive elasticity between cluster size and productivity. An increase in cluster size, either by more local users committing in the programming language or by moving to a larger cluster, a user would become more productive via an increase in her number of commits. This would hint at spillover effects raising the focal user's productivity. The more users positively impact the focal user. In the case of a negative α, interpretation is more difficult.

It implies that a user commits less to public projects with an increase in cluster size. Therefore, it does not necessarily imply a decrease in productivity. It could be the case that the user commits more to private projects or that the decrease in commits is due to a new job after moving. In both examples, a negative α does not necessarily imply a decrease in productivity or the nonexistence of productivity spillovers. Hence, interpretations of the results have to be done with caution.

Clusters are defined as the number of users in a city relative to all users in a programming language and, thus, the effect of cluster density on productivity is estimated. By including city $\times$ time fixed effects, we take into account changes in city size, and hence the elasticity estimates are similar to holding constant city area (Moretti, 2021).

An endogeneity concern when estimating agglomeration effects are unobserved determinants in the error term μ_{ijfct} simultaneously affecting productivity and cluster size (Combes and Gobillon, 2015). For time-invariant characteristics of a city biasing the estimates of α, e.g. location of the city, is controlled for by city effects. The attractiveness of a city or its size, which may change over time, is also controlled for by city $\times$ time effects. For trends in the popularity of programming languages is taken account for by programming language and programming language $\times$ time fixed effects. Moreover, differences in commits due to the popularity of projects are not affecting the estimates by including project fixed effects.

A possible concern in our case could be, that users move in expectation to be more active to a larger cluster. The user fixed effects control for users' inherent level of activity, although not for changes in their ability. In that case, unobservable time-varying productivity shocks could both affect the cluster size and the user's number of commits. Either via sorting, i.e., endogenous quality of labor, or simultaneity (endogenous quantity of labor), the OLS results might be biased (Moretti, 2021).

The latter concern regarding OLS identification of α are unobserved productivity shocks at the individual level. GitHub might foster a programming language in a city by promoting local projects in that programming language. Users may start to commit in that programming language, both cluster size and productivity would increase, however caused by unobserved productivity shocks at the city programming language level.

The first concern of sorting, i.e. a user chooses her location in the prospect of a future increase in productivity would violate the orthogonality assumption of μ_{ijfct} and α. A general tendency of more productive users locating in larger clusters is not a threat to identification, as time-invariant user productivity is accounted for by user effects. However, if a user moves to Silicon Valley in expectation to be more productive, user productivity might be affected by unobserved productivity shocks biasing α.

To tackle possible biases because of sorting, we implement an event study design by estimating the productivity gains of a user where treatment is the move. In this setting, variation comes

from only movers. Additionally we also estimate a dynamic response of productivity with changes in cluster size. In both settings, we can test if past or future cluster size is correlated current productivity. If we find significant effects for future cluster size, this might imply larger clusters attracting users with raising productivity.

5 Findings

In this section, the baseline estimates of equation (1) are presented. To account for possible endogeneity concerns an event study design is carried out. Additionally, a heterogeneity analysis and robustness checks to test the validity of the results are conducted and shown.

5.1 Baseline Estimates

Table 1 shows the estimates for the OLS regression of equation (1). For this regression, only users which commit in all time intervals are included. The estimated elasticity in the first column, only conditioning on city, time and programming language fixed effects is 0.045 (0.0170) and significant at the one percent level. After adding user fixed effects in column two, taking into account time-invariant user ability, the coefficient for log size becomes 0.0296 (0.0098) and is still statistically significant at the one percent level. The decrease from column one to column two suggests that larger clusters attracted users with higher mean unobserved productivity.

The estimates stay significant at the one percent level after adding dummies for the interaction of city $\times$ time effects in column three. The elasticity increases from 0.0296 (0.0098) in column two to 0.0391 (0.0127) in column three. City-specific productivity shocks and selection due to local amenities especially seem to matter for larger clusters.

Trends in programming languages or productivity shocks for certain programming languages captured by programming language $\times$ time fixed effects, only slightly decreases the coefficient for log size from 0.0391 (0.0127) to 0.0366 (0.0125) in column four of Table 1. The coefficients in columns three and four both are statistically significant at the one percent level.

Regarding R^2, it increases from column one with 0.015 to 0.1 in column two. This shows, that user fixed effects are important to control for, as they explain a lot of variance of log commits. Not including them would bias the estimate of log size.

After further adding project fixed effects, the estimate becomes insignificant in column five with 0.0641 (0.0420). When including all controls, i.e. also the dummies for city $\times$ language, the estimated elasticity is 0.103 (0.0566) and significant at the ten percent level. Especially project effects explain a lot of the variance in the number of commits as R^2 increases when adding them from 0.11 to 0.7. This may not be very surprising as the type of project determines how active a

Table 1: Baseline Estimates

	Log(Commit)					
	(1)	(2)	(3)	(4)	(5)	(6)
Log(Size)	0.0450***	0.0296***	0.0391***	0.0366***	0.0641	0.1030*
	(0.0170)	(0.0098)	(0.0127)	(0.0125)	(0.0420)	(0.0566)
Fixed-effects						
City	Yes	Yes	Yes	Yes	Yes	Yes
Time	Yes	Yes	Yes	Yes	Yes	Yes
Language	Yes	Yes	Yes	Yes	Yes	Yes
User		Yes	Yes	Yes	Yes	Yes
City x Time			Yes	Yes	Yes	Yes
Language x Time				Yes	Yes	Yes
Project					Yes	Yes
City x Language						Yes
R^2	0.015	0.105	0.106	0.107	0.700	0.702
Observations	2,095,978	2,095,978	2,095,978	2,095,978	2,095,978	2,095,978

*Signif. Codes: ***: 0.01, **: 0.05, *: 0.1.*

Notes: Standard Errors are clustered by city x (programming) language. Every column presents a regression.

user is. If it is the user's own project or a work project, she likely commits more.

The final estimate implies a positive elasticity, meaning a user commits 1.03 percent more in a time interval in a programming language with a ten percent increase in cluster size of the respective programming language. Precisely, if the share of users in the user's city in her programming language relative to all users in her programming language increases, she commits more. A user's number of commit in JavaScript would increase by 1.18 percent, if she moves from the median cluster in JavaScript, Baton Rouge, LA-MS, to the largest cluster, San Francisco-Oakland-San Jose. Hence, there is a positive relationship between cluster size and productivity. This is in line with the findings of others (Moretti, 2021; Combes et al., 2010), that similarly estimate a positive elasticity between cluster size and productivity. Though, the results have to be taken with caution. We only observe commits to public projects. If a user's cluster increases, she might move to committing more to private projects. Therefore, the results may provide a lower bound of the elasticity. [27]

[27]Using the absolute cluster size instead of cluster density, results in exactly the same elasticity of 0.1030(0.0566) with all covariates included. Additionally, we test if the results are stable for excluding large commits and large projects, i.e. commits bigger than 100 and projects with more than 40 users committing to. The elasticity increases to 0.1114 (0.0559) and becomes significant on the five percent level. See Table 20 and 21 in the Appendix for the regression results, respectively.

5.2 Quality of Commits: Project Stars

To analyze if the quality of a user's commits increases with cluster size, we restrict the sample to users that are observable over the whole period of analysis with commits to projects with at least one star and only consider those commits. As the number of stars are a measure for the quality of a project, committing more to those high quality projects suggests an increase in the commit's quality itself. Table 2 shows the results of a regression of log commit on log size with the restricted sample.

Table 2: Baseline Estimates - Excluding Projects with Zero Stars

	Log(Commit)					
	(1)	(2)	(3)	(4)	(5)	(6)
Log(Size)	0.0524*	0.0307*	0.0653***	0.0640***	0.0857*	0.1245*
	(0.0270)	(0.0177)	(0.0223)	(0.0223)	(0.0467)	(0.0642)
Fixed-effects						
City	Yes	Yes	Yes	Yes	Yes	Yes
Time	Yes	Yes	Yes	Yes	Yes	Yes
Language	Yes	Yes	Yes	Yes	Yes	Yes
User		Yes	Yes	Yes	Yes	Yes
City x Time			Yes	Yes	Yes	Yes
Language x Time				Yes	Yes	Yes
Project					Yes	Yes
City x Language						Yes
R^2	0.022	0.130	0.133	0.134	0.623	0.627
Observations	736,828	736,828	736,828	736,828	736,828	736,828

*Signif. Codes: ***: 0.01, **: 0.05, *: 0.1.*

Notes: Standard Errors are clustered by city x (programming) language. Every column presents a regression. Sample includes only projects with at least one star.

The coefficients in the first four columns are all significant at the ten and one percent level, varying from 0.0524 (0.0270) to 0.0640 (0.0223). In the final two columns, the elasticity between commit and cluster size stays significant at the ten percent level. Conditional on all controls the elasticity is 0.1245 (0.0642).

Regarding the positive elasticity between cluster size and number of commits of 0.1245 (0.0642), it implies that the user commits 1.245 percent more to projects with at least one star if her cluster increases by ten percent. This suggests a positive impact of cluster size on the quality of commits.

5.3 Heterogeneity in Elasticity

Cluster Size The results for the elasticity between number of commit and cluster size might differ depending on the cluster size.

Table 3: Heterogeneity in Elasticity by Cluster Size

	Log(Commit)	
	(1)	(2)
First Quartile (Smallest)	0.0062	0.1049*
	(0.0261)	(0.0568)
Second Quartile	0.0025	0.1011*
	(0.0267)	(0.0582)
Third Quartile	0.0069	0.1020*
	(0.0274)	(0.0586)
Fourth Quartile (Largest)	0.0276	0.1116*
	(0.0287)	(0.0614)
Fixed-effects		
Project		Yes
R^2	0.110	0.702
Observations	2,095,977	2,095,977
Wald (joint nullity), p-value	0.001	0.398

*Signif. Codes: ***: 0.01, **: 0.05, *: 0.1.*

Notes: Standard Errors are clustered by city x (programming) language. In all regressions, fixed effects for city, time, programming language, city × programming language, programming language × time, city × time and user are included.

It could be the case that productivity spillovers require a certain cluster size to occur. In smaller clusters, the benefits of the existence of other users, as they are less, might also be smaller. In this case, it may depend on a certain threshold agglomeration effects occur. Contrary, in larger clusters, a one percent increase in cluster size might result in smaller productivity gains in relative terms, compared to a one percent increase in a smaller cluster.[28] Finally, both could be true, implying an S-shaped elasticity between cluster size and productivity. Therefore, we let the effect of cluster size on commits vary with respect to cluster size. Table 3 shows the results of a regression of log commit on log size where the size is interacted with dummies for cluster size quartiles. In column one, all controls but project effects are included, in column two project effects are added. The coefficient in both columns is the greatest, in absolute terms, for the largest clusters, i.e., the absolute elasticity is the highest with 0.1116 (0.0614) in the largest clusters, conditional on

[28] Au and Henderson (2006), e.g., estimate a bell-shaped relation between productivity and city size for Chinese cities.

all covariates. The estimates for different size quartiles range from 0.1049 (smallest size quartile; 0.0568) to 0.1116 (largest quartile; 0.0614) in column two, hence the variation in elasticity across quartiles is considerably small.

They suggest a mildly linear elasticity between cluster size and productivity, as the estimates increase with size quartile. A Wald test for testing that all coefficients are zero in the model with all controls included, cannot be rejected with a p-value of 0.398. Hence, the elasticity between cluster size and productivity does not seem to vary with respect to cluster size. This is in line with the findings of Moretti (2021), which also does not find a heterogeneity in elasticity by cluster size.

Project Productivity Another cause of heterogeneity in elasticity could be project productivity. More productive projects, i.e., projects receiving a higher number of commits, might benefit more from productivity spillovers. Findings suggest that in the case of firms, more productive firms gain the most from agglomeration effects (Combes et al., 2012).

Table 4: Heterogeneity in Elasticity by Project Productivity

	Log(Commit)	
	(1)	(2)
First Quartile (Least Productive)	0.3712***	0.0862
	(0.0272)	(0.0556)
Second Quartile	0.1439***	0.0933*
	(0.0259)	(0.0560)
Third Quartile	-0.0026	0.0997*
	(0.0257)	(0.0563)
Fourth Quartile (Most Productive)	-0.1653***	0.1100*
	(0.0260)	(0.0576)
Fixed-effects		
Project		Yes
R^2	0.391	0.702
Observations	2,095,978	2,095,978
Wald (joint nullity), p-value	0	0.273

*Signif. Codes: ***: 0.01, **: 0.05, *: 0.1.*

Notes: Standard Errors are clustered by city x (programming) language. Project productivity is measured by the total number of commits a project received. The most productive projects are projects that are in the fourth quartile of the distribution of total commits per project. In all regressions, fixed effects for city, time, programming language, city × programming language, programming language × time, city × time and user are included.

For GitHub projects, the same might be the case. Projects receiving a large number of commits likely have a higher quality demand on their received commits. They profit more if users acquire

more knowledge from other users. In that case, we would expect an increase in the elasticity with respect to project productivity, measured by the project's total number of commits received.

Table 4 presents the estimates of a regression of log commit on log size, letting the slope of log size vary by project productivity quartile. The first column shows the coefficients conditional on all fixed effects as in the baseline model but project effects. In column two, project fixed effects are added. The estimates for the lower three quartiles are smaller than the baseline estimate of 0.103 (0.0566), with all covariates included. They lay between 0.0862 (least productive; 0.0556), 0.0933 (second quartile; 0.0560) and 0.0997 (third quartile; 0.0563) and significant at the ten percent level for the second and third quartile. The elasticity for the most productive projects is significant at the ten percent level and larger than the baseline estimate with 0.1100 (0.0576). A Wald test for testing that all coefficients are zero cannot be rejected with a p-value of 0.273.

This shows, that the elasticity between cluster size and number of commits increases with project productivity. The projects in the fourth quartile might be work projects and therefore experience a larger increase in commits with an increase in cluster size. However, based on the Wald test, we cannot reject the null hypothesis of no heterogeneity in cluster size with project productivity.

User Follower The elasticity could also vary by the user's popularity. Users on GitHub can *follow* each other, i.e. the followers are notified about the followed user's activity. The number of followers are a measure of a user's popularity. More followers means more users are interested in the activity of the respective user. The higher interest in the user's action on GitHub might be due to her high quality work and they are perceived as especially skilled (Lee et al., 2013).

Again it is unclear if especially skilled users benefit more from larger cluster and via that knowledge spillover effects, or if contrary, unskilled users' productivity gains are larger with an increase in cluster size. By letting the elasticity vary with follower quartile, we study heterogeneity in that respect.

The estimates in column (2) of table 5 conditional on all controls suggest that productivity effects are largest for the users with least popularity. For users in the first quartile of followers the effect is 0.119 (0.0569) and significant on the five percent level. The effect for the second highest and highest follower quartile are 0.117 (0.0575) and 0.1057 (0.0576), respectively and significant on the ten percent level. The estimate for the second lowest follower quartile is insignificant. Users with a lower skill set seem to benefit more from a change in cluster size. They might be able to acquire more knowledge in contrast to higher-skilled users.

Even though the difference in estimates are rather small, a Wald test can be rejected with a p-value of 0.053. Elasticity seems to vary with user follower.

Table 5: Heterogeneity in Elasticity by Followers

	Log(Commit)	
	(1)	(2)
First Quartile (Least Followed)	0.0037	0.1119**
	(0.0264)	(0.0569)
Second Quartile	-0.0068	0.0771
	(0.0266)	(0.0575)
Third Quartile	0.0040	0.1117*
	(0.0271)	(0.0578)
Fourth Quartile (Most Followed)	-0.0029	0.1057*
	(0.0265)	(0.0576)
Fixed-effects		
Project		Yes
R^2	0.110	0.702
Observations	2,095,978	2,095,978
Wald (joint nullity), p-value	0.666	0.053

*Signif. Codes: ***: 0.01, **: 0.05, *: 0.1.*

Notes: Standard Errors are clustered by city x (programming) language. Followers are measured by the number of other users following the activity stream of the user. The users with the most followers are users that are in the fourth quartile of the distribution of total followers per user. In all regressions, fixed effects for city, time, programming language, city × programming language, programming language × time, city × time and user are included

Alternative User Samples Lastly, we estimate the elasticity between cluster size and the number of commits for different user samples. In detail, we calculate a user's share of commits to all commits, and restrict the sample to users in the upper 25 percent, upper 50 percent and upper 75 percent of the distribution of commits per user. More productive users, i.e., higher in the distribution of total commits per user, might benefit more from larger clusters. Table 6 presents the estimates for the three subsamples of users, in decreasing order, and the baseline estimate in the final column.

Table 6: Alternative User Samples

	Log(Commit)			
	Upper 25%	Upper 50%	Upper 75%	All
	(1)	(2)	(3)	(4)
Log(Size)	0.1300	0.1191*	0.1110*	0.1030*
	(0.1257)	(0.0718)	(0.0590)	(0.0566)
R^2	0.720	0.704	0.702	0.702
Observations	739,636	1,516,258	1,990,068	2,095,978

*Signif. Codes: ***: 0.01, **: 0.05, *: 0.1.*

Notes: Standard Errors are clustered by city x (programming) language. Every column presents a regression. Controls for city, time, language, city × language, language × time, user, city × time and project are included. Users are measured by their share of commits to all commits. Hence, users in the upper 25% sample cover the upper 25% of all commits by their commits.

All estimates are conditional on all fixed effects. The elasticity for the upper 25 percent users is the largest with 0.1300 (0.1257) and insignificant. The coefficient for the upper 50 percent is 0.1191 (0.0718) and significant at the ten percent level. For the upper 75 percent of users, the elasticity is significant at the ten percent level with 0.1110 (0.0590) and similar to the baseline estimate of 0.1030 (0.0566).

The largest elasticity for the upper 25 percent might match up with the results for heterogeneity in elasticity with respect to project productivity. Users in the highest quartile might use GitHub for work reasons, hence a change in cluster size might increase their commits the most. Users in the lower quartiles possibly use GitHub for leisure and private projects, which explains the slightly smaller elasticity. This is only hypothetically, which cannot be tested by the data on hand and requires more knowledge on the users themselves and the projects they commit to.[29]

[29]With more data on users on hand, one could also investigate if gender plays a role on the effects of cluster size on productivity. Rosenthal and Strange (2012) find smaller agglomeration effects for women than for men. Similarly might be the case for female users. They possibly benefit less from larger clusters than male users.

5.4 Dynamic Response

To test the validity of the baseline estimates we analyze the dynamic response of productivity with a change in cluster size. That way, we can study if future cluster size has an effect on current productivity. If that is the case, sorting may play a role. Individuals in expectation of future productivity gains would systematically sort into larger clusters. For example firms in larger clusters may attract programmers, anticipating increases in their productivity. We estimate the following model, a version of equation (1) which additionally to current cluster size includes leads and lags of cluster size:

$$ln(y_{ijfct}) = \sum_{s=-n}^{-1} \beta_s ln(S_{-ifc(t+s)}) + \beta_0 ln(S_{-ifc(t)}) + \sum_{1}^{s=n} \beta_s ln(S_{-ifc(t+s)}) + d_{cf} + d_{ft} + d_{ct} + d_c + d_f + d_i + d_j + \mu_{ijfct}$$

where the leads and lags $S_{-ifc(t+s)}$ refer to the cluster where the focal inventor i is at time $t+s$. The coefficients on the lead terms, β_s with $s \in [1, 4]$, show how a user's productivity in a given snapshot responds to future changes in cluster size. Similarly, the lag terms, β_s with $s \in [-4, -1]$, show how a change in cluster size propagates over time. We only have ten time intervals on hand, and thus, vary with the number of leads and lags included. Including four leads and five lags may result in overfitting the data.

Table 7 shows the estimates for the fitted models. In column (1) only one lead and one lag is included. In the following columns are always one lead and one lag added, i.e. in column (4), four leads and four lags are included in addition to current cluster size. Only in column (1) the estimate for β_0 is significant. With 0.1142 (0.0654) it is very similar to our baseline estimates with 0.103 (0.0566). The lead and lag terms are never significant, suggesting that there seems no sorting to be going on. When adding further leads and lags, β_0 becomes insignificant but its magnitude changes only minimally. Only when including four leads and four lags, β_0 becomes negative. Though, this might be due to overfitting. The large standard errors support this hypothesis.

We replicate this model with users that move once at $s = 0$ and cluster size interacted with relative timing to the move. Hence, the sample is limited to movers. This way, we can estimate how a move affects a user's productivity. The estimates for this regression are shown in Table 8. In Table 7 variation comes from stayers and movers, i.e. it uses all variation in cluster size. With the restricted sample including only movers, variation of stayers is excluded. The treatment variable is average cluster size before and after the move. Thus, cluster size before and after the move is the *average* cluster size before and after the move. This results in within-city variation in cluster size over time not being used to identify the effects.

Table 7: Dynamic Response - All Users

	Log(Commit)			
	(1)	(2)	(3)	(4)
β_{-4}				-0.0087
				(0.0226)
β_{-3}			0.0069	0.0101
			(0.0140)	(0.0446)
β_{-2}		-0.0033	0.0099	0.0159
		(0.0123)	(0.0188)	(0.0388)
β_{-1}	0.0103	0.0212	0.0282	-0.2119
	(0.0083)	(0.0135)	(0.0255)	(0.3397)
β_0	0.1142*	0.1142	0.1107	-0.2039
	(0.0654)	(0.0740)	(0.1276)	(0.2988)
β_1	-0.0035	0.0065	0.0193	0.0145
	(0.0091)	(0.0139)	(0.0152)	(0.0260)
β_2		-0.0070	-0.0005	0.2437
		(0.0111)	(0.0179)	(0.3147)
β_3			-0.0071	-0.1757
			(0.0155)	(0.2837)
β_4				-0.0032
				(0.0256)
R^2	0.711	0.740	0.779	0.857
Observations	1,209,331	814,764	484,813	175,303

*Signif. Codes: ***: 0.01, **: 0.05, *: 0.1.*

Notes: Standard Errors are clustered by city x (programming) language. In all regressions, fixed effects for city, time, programming language, city × programming language, programming language × time, city × time, project and user are included.

Table 8: Dynamic Response - Once Movers

	Log(Commit)			
	(1)	(2)	(3)	(4)
β_{-4}				-0.0070
				(0.0085)
β_{-3}			-0.0049	-0.0058
			(0.0079)	(0.0082)
β_{-2}		-0.0052	-0.0059	-0.0059
		(0.0076)	(0.0081)	(0.0083)
β_{-1}	0.0038	0.0045	0.0039	0.0036
	(0.0071)	(0.0077)	(0.0081)	(0.0088)
β_0	0.0142**	0.0157**	0.0161**	0.0169**
	(0.0066)	(0.0069)	(0.0080)	(0.0082)
β_1	0.0135*	0.0153*	0.0167*	0.0177*
	(0.0076)	(0.0083)	(0.0089)	(0.0094)
β_2		0.0106	0.0116	0.0133
		(0.0081)	(0.0088)	(0.0097)
β_3			0.0057	0.0079
			(0.0089)	(0.0097)
β_4				0.0098
				(0.0105)
R^2	0.782	0.782	0.782	0.782
Observations	459,830	459,830	459,830	459,830

*Signif. Codes: ***: 0.01, **: 0.05, *: 0.1.*

Notes: Standard Errors are clustered by city x (programming) language. The sample includes only users that moved once, and that is at time t = 0. In all regressions, fixed effects for city, time, programming language, city × programming language, programming language × time, city × time, project and user are included.

The estimates for β_0 are always significant at the five percent level, though smaller in size ranging from 0.0142 (0.0066) to 0.0169 (0.0082). Additionally, the first lead, β_1 is significant at the ten percent level and in size between 0.0135 (0.0076) and 0.0177 (0.0094). The other leads and lags are always insignificant.

First it seems that productivity gains occur with the move, i.e. with a change in cluster size. This supports our baseline estimates, and suggest they are not just due to spurious correlation but rather productivity gains due to knowledge spillovers among users.

Second, the significant lead terms may be a result of our data construction. We observe a user location change in the next snapshot after a user updates her GitHub profile. Hence, there is a lag between the profile update and the change in our data. In addition, users may take time to update their profile after the move. These delays might cause the lead term to be significant. The insignificant lag terms suggest that most of the productivity gains happen soon after the move.

In general the findings are in line with our baseline estimates. Especially the effect size of β_0 when including all users is almost the same as our baseline estimate.

5.5 Robustness

Cross-Field Spillover In the baseline regression, we assume that productivity depends only on the user's own cluster size. However, users in our analysis can be part of several clusters per time interval. It is likely that a user can translate skills, e.g., commands, from one programming language to another programming language. Next to that, some programming languages are more similar to each other than others. As a result, a user's productivity in a programming language might not only depend on the cluster size of that programming language but also on the cluster size of other programming languages.

Therefore we include, next to the own-field cluster size, the cluster sizes of other programming languages in the respective city $\times$ time interval as explanatory variables. Furthermore, we split the sample by programming languages. Column one in Table 17 contains all C-users, column two, all C#-users, column three all C++-users, and so on. Then in column one, the first line, the effect of log C-cluster size on log commit represents the own-field elasticity. The next lines, i.e., where programming language is not equal to the programming language in the column header, reflect cross-field spillovers. For all estimates, all covariates used in the baseline regression but the interaction of city $\times$ time are included.[30]

In general, the estimates for own-field elasticity become larger, though only for the programming languages C, own-field elasticity is statistically different from zero with 0.8655 (0.3332). This implies a 8.655 percent increase in productivity with a ten percent increase in cluster size. In all

[30]The interaction of city $\times$ time is not included in the regressions for cross-field spillover effects to circumvent multicollinearity.

cases but C#, C++, CSS, Go, Objective-C and R, own-field elasticity is positive.

For most programming languages, there are no significant estimates for cross-field elasticities. A negative cross-field elasticity is estimated for C and Go. For a negative elasticity the programming languages might be substitutes to each other. For example, the cross-field elasticity of C with respect to Java is similar in size with -0.7931 (0.3481) to own-field elasticity with 0.8655 (0.3332) and significant at the five percent level, respectively. Both programming languages might offer similar functions. If more users in a city start to commit in Java, C-users might switch to Java to benefit from the greater Java knowledge in their city. On differing levels, productivity in a programming language depends not only on the own cluster size but also on other programming languages' cluster size.

Alternative User Samples In the main analysis, only users are included that commit in all time intervals. They represent the more active users and as a result, the absolute elasticity may be the largest for them. Furthermore, as Casalnuovo et al. (2015) show, productivity increases are the largest in collaboration for users with greater knowledge in a programming language. Users with commits in all time intervals likely have a good level of knowledge in a programming language which may lead to having the largest elasticity between productivity and cluster size. Therefore, we loosen the restriction on the time intervals with non-zero commits per user. If the assumption is true, that the users with non-zero commits in all time intervals are the most active, we would expect the absolute elasticity between cluster size and productivity decrease by decreasing the number of time intervals with non-zero commits. Table 9 presents the regression results for different subsamples with all covariates from the baseline model included.

Table 9: Different Lengths of Observation Period

	Log(Commit)				
	(1)	(2)	(3)	(4)	(5)
Log(Size)	0.0654	0.0647	0.0660	0.0773*	0.0826*
	(0.0459)	(0.0456)	(0.0441)	(0.0436)	(0.0438)
R^2	0.770	0.769	0.744	0.727	0.712
Observations	5,672,198	5,612,706	4,612,410	4,016,109	3,476,721

*Signif. Codes: ***: 0.01, **: 0.05, *: 0.1.*

Notes: Standard Errors are clustered by city x (programming) language. Every column presents a regression of equation 1. In column (1), users are included that in at least one time interval had commits. In column (2), users are included that commit in at least two time intervals, and so on. Controls for city, time, language, city × language, language × time, user, city × time and project are included.

In the first column, users are included that commit in at least one time interval. In column

Table 10: Different Lengths of Observation Period - Continued

	Log(Commit)				
	(6)	(7)	(8)	(9)	(10)
Log(Size)	0.0905**	0.0903**	0.0917*	0.0908*	0.1030*
	(0.0442)	(0.0454)	(0.0470)	(0.0483)	(0.0566)
R^2	0.702	0.699	0.697	0.697	0.702
Observations	3,174,080	2,987,360	2,745,317	2,541,432	2,095,978

*Signif. Codes: ***: 0.01, **: 0.05, *: 0.1.*

Notes: Standard Errors are clustered by city x (programming) language. Every column presents a regression of equation 1. In column (1), users are included that in at least one time interval had commits. In column (2), users are included that commit in at least two time intervals, and so on. Controls for city, time, language, city × language, language × time, user, city × time and project are included.

two, the sample consists of users that commit in at least two consecutive time intervals. Column three shows the regression results of users with commits in at least three consecutive time intervals. This way, the column number represents the number of consecutive time intervals with non-zero commits per user. The elasticity is insignificant in the first three columns and decreases from 0.0654 (0.0459) to 0.066 (0.0441). This indicates that the sample of users with non-zero commits between at least one to at least three time intervals contains users for which no significant relationship between cluster size and productivity can be found. One reason might be, due to their lower activity on GitHub, productivity changes are not possible to be observed, or at least not in a statistical sense.

The coefficients increase in size in the following columns five to ten, as well as becoming significant at the ten percent level (column four, five, eight, nine and ten) to five percent level (columns six to seven). The elasticity in column ten, which presents the baseline estimate, is the largest in size with 0.1027 (0.0564). Hence, the assumption of largest elasticity for the most active users is confirmed by the results given.

It is further worth noting that the results may also reflect a possible bias towards zero. The results represent the intensive margin, i.e., an increase in commits with an increase in cluster size, given a user commits. In the case of zero commits, we do not observe the user's productivity. If a larger cluster also affects the probability to commit (extensive margin), and given this effect goes in the same direction as the effect on the intensive margin, the estimates would be biased towards zero. In column ten, only users with non-zero commits are included, and hence the bias towards zero should be the least pronounced. The estimate is the largest in size, which supports the hypothesis of a bias towards zero in the other columns.

6 Conclusion

We find a significant elasticity of 0.103 (0.0566) between productivity and cluster size for GitHub users conditional on several controls. Regarding the quality of commits, estimated by the number of commits to projects with at least one star it is positive with 0.1245 (0.0642). Projects with more stars likely demand higher quality of their receiving commits and the results, thus, imply an increase in the quality of commits with an increase in cluster size. The heterogeneity analysis showed, that especially low-skilled users, i.e. users with a small number of followers, benefit the most from changes in cluster size.

Selection concerns seem to play a minor role as our dynamic response and event study analysis suggests. Future values of cluster size are only up to one period ahead correlated with current productivity. Larger clusters seem not to systematically attract users with increasing productivity.

The mechanisms underlying the knowledge spillovers, e.g., task specialization or training, could be the focus of future research. With an increase in cluster size, the task distribution might change, e.g., every project member only uses one programming language.

Nevertheless, the results suggest that productivity spillover effects and cluster size play a role in open source software development as well. They are an important input source for firms and, thus, fostering knowledge creation would further increase the benefits from integrating open source software. Especially in times of increasing working from home, cities and urban density remain to play an important role in the diffusion of knowledge.

References

Andersson, M., Klaesson, J., and Larsson, J. P. (2014). The sources of the urban wage premium by worker skills: Spatial sorting or agglomeration economies? *Papers in Regional Science*, 93(4):727–747.

Andersson, R., Quigley, J. M., and Wilhelmsson, M. (2009). Urbanization, productivity, and innovation: Evidence from investment in higher education. *Journal of Urban Economics*, 66(1):2–15.

Au, C.-C. and Henderson, J. V. (2006). Are chinese cities too small? *The Review of Economic Studies*, 73(3):549–576.

Audretsch, D. and Feldmann, M. (1996). R&D spillovers and the geography of innovation and production. *American Economic Review*, 86:630–640.

Autor, D. H., Levy, F., and Murnane, R. J. (2003). The skill content of recent technological change: An empirical exploration. *The Quarterly journal of economics*, 118(4):1279–1333.

Azoulay, P., Graff Zivin, J. S., and Wang, J. (2010). Superstar extinction. *The Quarterly Journal of Economics*, 125(2):549–589.

Bacolod, M., Blum, B. S., and Strange, W. C. (2009). Skills in the city. *Journal of Urban Economics*, 65(2):136–153.

Baum-Snow, N., Gendron-Carrier, N., and Pavan, R. (2020). Local productivity spillovers.

Becker, R. A. and Wilks, A. R. (2018). Package 'maps'. `https://cran.r-project.org/web/packages/maps/maps.pdf`. Accessed: 2021-05-11.

Belenzon, S. and Schankerman, M. (2008). Motivation and sorting in open source software innovation. *CEPR Discussion Papers*, 7012.

Bell, A., Chetty, R., Jaravel, X., Petkova, N., and Reenen, J. V. (2019). Who becomes an inventor in America? The importance of exposure to innovation. *The Quarterly Journal of Economics*, 134(2):647–713.

Borges, H., Hora, A., and Valente, M. T. (2016). Understanding the factors that impact the popularity of github repositories. In *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 334–344. IEEE.

Borjas, G. J. and Doran, K. B. (2015). Which peers matter? The relative impacts of collaborators, colleagues, and competitors. *Review of economics and statistics*, 97(5):1104–1117.

Carlino, G. and Kerr, W. R. (2015). Agglomeration and innovation. *Handbook of Regional and Urban Economics*, 5:349–404.

Carlino, G. A., Chatterjee, S., and Hunt, R. M. (2007). Urban density and the rate of invention. *Journal of Urban Economics*, 61(3):389–419.

Casalnuovo, C., Vasilescu, B., Devanbu, P., and Filkov, V. (2015). Developer onboarding in GitHub: the role of prior social links and language experience. In *Proceedings of the 2015 10th joint meeting on foundations of software engineering*, pages 817–828.

Catalini, C. (2018). Microgeography and the direction of inventive activity. *Management Science*, 64(9):4348–4364.

Charlot, S. and Duranton, G. (2004). Communication externalities in cities. *Journal of Urban Economics*, 56(3):581–613.

Cohen, W., Nelson, R., and Walsh, J. (2000). Protecting their intellectual assets: appopriability conditions and why u.s. manufcaturing firms patent (or not). *National Bureau of Economic Research Working Paper*, (w7552).

Combes, P.-P., Duranton, G., Gobillon, L., Puga, D., and Roux, S. (2012). The productivity advantages of large cities: Distinguishing agglomeration from firm selection. *Econometrica*, 80(6):2543–2594.

Combes, P.-P., Duranton, G., Gobillon, L., and Roux, S. (2010). Estimating agglomeration economies with history, geology, and worker effects. In *Agglomeration economics*, pages 15–66. University of Chicago Press.

Combes, P.-P. and Gobillon, L. (2015). The empirics of agglomeration economies. In *Handbook of regional and urban economics*, volume 5, pages 247–348. Elsevier.

Cornelissen, T., Dustmann, C., and Schönberg, U. (2017). Peer effects in the workplace. *American Economic Review*, 107(2):425–56.

Demsas, J. (2021). Remote work is overrated. America's supercities are coming back. *Vox*.

Duranton, G. and Puga, D. (2001). Nursery cities: Urban diversity, process innovation, and the life cycle of products. *American Economic Review*, 91(5):1454–1477.

Emanuel, N. and Harrington, E. (2020). "working" remotely?

Farhauer, O. and Kröll, A. (2009). Die shift-share-analyse als instrument der regional-und clusterforschung. Technical report, Passauer Diskussionspapiere-Volkswirtschaftliche Reihe.

Foster, N. and Stehrer, R. (2009). Sectoral productivity, density and agglomeration in the wider europe. *Spatial Economic Analysis*, 4(4):427–446.

GIT (2021). `https://git-scm.com/`. Accessed: 2021-05-31.

Glaeser, E. L. (1999). Learning in cities. *Journal of urban Economics*, 46(2):254–277.

Gousios, G. (2013). The ghtorrent dataset and tool suite. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13, pages 233–236, Piscataway, NJ, USA. IEEE Press.

Herbst, D. and Mas, A. (2015). Peer effects on worker output in the laboratory generalize to the field. *Science*, 350(6260):545–549.

Hergueux, J. and Jacquemet, N. (2015). Social preferences in the online laboratory: A randomized experiment. *Experimental Economics*, 18:251–283.

Hindle, A., German, D. M., and Holt, R. (2008). What do large commits tell us? a taxonomical study of large commits. In *Proceedings of the 2008 international working conference on Mining software repositories*, pages 99–108.

Jaffe, A. B., Trajtenberg, M., and Henderson, R. (1993). Geographic localization of knowledge spillovers as evidenced by patent citations. *The Quarterly Journal of Economics*, 108.

Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., German, D., and Damian, D. (2014). The promises and perils of mining GitHub. In *Proceedings of the 11th working conference on mining software repositories*, pages 92–101.

Laurentsyeva, N. (2019). From friends to foes: National identity and collaboration in diverse teams. *CRC TRR 190 Discussion Paper*, 226.

Laurentsyeva, N. and Fackler, T. (2020). Gravity in online collaborations: Evidence from GitHub. *CESifo Forum*, 21(3).

Lee, M. J., Ferwerda, B., Choi, J., Hahn, J., Moon, J. Y., and Kim, J. (2013). Github developers use rockstars to overcome overflow of news. In *CHI'13 Extended Abstracts on Human Factors in Computing Systems*, pages 133–138.

Lima, A., Rossi, L., and Musolesi, M. (2014). Coding together at scale: GitHub as a collaborative social network. *arXiv preprint arXiv:1407.2535*.

Manso, G. (2011). Motivating innovation. *The Journal of Finance*, 66(5):1823–1860.

Marshall, A. (1890). Principles of economics macmillan. *London (8th ed. Published in 1920)*.

McDonald, N. and Goggins, S. (2013). Performance and participation in open source software on GitHub. In *CHI '13 Extended Abstracts on Human Factors in Computing Systems*, pages 139–144. CHI EA '13, Paris, France.

Melo, M. T., Nickel, S., and Saldanha-Da-Gama, F. (2009). Facility location and supply chain management–a review. *European journal of operational research*, 196(2):401–412.

Microsoft (2021). Microsoft to acquire github for $7.5 billion. `https://news.microsoft.com/2018/06/04/microsoft-to-acquire-github-for-7-5-billion/`. Accessed: 2021-06-01.

Moretti, E. (2021). The effect of high-tech clusters on the productivity of top inventors. *American Economic Review*, 111(10):3328–75.

Nagle, F. (2019). Open source software and firm productivity. *Management Science*, 65(3):1191–1215.

Pischke, S. (2007). Lecture notes on measurement error.

Rosenthal, S. S. and Strange, W. C. (2012). Female entrepreneurship, agglomeration, and a new spatial mismatch. *Review of Economics and Statistics*, 94(3):764–788.

Rosenthal, S. S. and Strange, W. C. (2020). How close is close? The spatial reach of agglomeration economies. *Journal of Economic Perspectives*, 34(3):27–49.

Schumpeter, J. A. (1939). *Business cycles*, volume 1. McGraw-Hill New York.

Simplemaps (2021). United states cities database. `https://simplemaps.com/data/us-cities`. Accessed: 2021-05-10.

Tsay, J., Dabbish, L., and Herbsleb, J. (2014). Influence of social and technical factors for evaluating contribution in GitHub. In *Proceedings of the 36th international conference on Software engineering*, pages 356–366.

Vasilescu, B., Yu, Y., Wang, H., Devanbu, P., and Filkov, V. (2015). Quality and productivity outcomes relating to continuous integration in GitHub. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, pages 805–816.

Wright, N., Nagle, F., and Greenstein, S. M. (2020). Open source software and global entrepreneurship. *Harvard Business School Technology & Operations Mgmt. Unit Working Paper*, (20-139):20–139.

Wuchty, S., Jones, B. F., and Uzzi, B. (2007). The increasing dominance of teams in production of knowledge. *Science*, 316(5827):1036–1039.

A Appendix

A.1 Tables

Table 11: Summary Statistics Commits by Programming Languages - Full Data

Language	Min.	Median	Mean	Max.	Projects	N	Commits	Share
C	0	8	88.81	80,344	183,934	64,696	5,745,719	4.37%
C#	0	10	69.76	20,296	181,239	56,909	3,970,176	3.02%
C++	0	9	94.41	185,564	229,197	80,712	7,620,004	5.8%
CSS	0	17	70.60	225,948	871,683	262,552	18,535,077	14.1%
Go	0	9	105.12	25,959	149,806	38,830	4,081,920	3.11%
Java	0	11	75.72	221,308	460,135	131,598	9,965,244	7.58%
JavaScript	0	22	112.73	172,039	1,631,447	262,312	29,570,296	22.5%
Objective-C	0	7	47.94	9,353	70,519	24,030	1,151,925	0.88%
PHP	0	9	87.48	218,318	217,492	61,774	5,403,837	4.11%
Python	0	15	96.08	94,083	858,595	204,723	19,669,615	14.96%
R	0	9	88.41	406,159	72,650	21,977	1,942,997	1.48%
Ruby	0	16	148.15	73,674	710,861	88,111	13,053,992	9.93%
Rust	0	11	96.01	20,612	45,683	12,590	1,208,731	0.92%
Shell	0	8	62.43	281,997	182,177	78,089	4,875,133	3.71%
Swift	0	9	49.41	30,077	83,915	24,370	1,204,187	0.92%
TypeScript	0	8	62.73	20,600	151,790	54,851	3,441,061	2.62%

Table 15: Largest Clusters for Top 5 Most Used Programming Languages in 202103 Snapshot

	Size
JavaScript	
San Francisco-Oakland-San Jose, CA	0.11493
Washington-Baltimore, DC-MD-VA-WV-PA	0.06326
Los Angeles-Riverside-Orange County, CA-AZ	0.06191
Seattle-Tacoma-Bremerton, WA	0.05353
Toronto	0.04361
Boston-Worcester-Lawrence-Lowell-Brockton, MA-NH-RI-VT	0.03602
Chicago-Gary-Kenosha, IL-IN-WI	0.03415
Denver-Boulder-Greeley, CO-KS-NE	0.03089
Portland-Salem, OR-WA	0.02707
Austin-San Marcos, TX	0.02623
Python	
San Francisco-Oakland-San Jose, CA	0.13063

Washington-Baltimore, DC-MD-VA-WV-PA	0.07390
Los Angeles-Riverside-Orange County, CA-AZ	0.05705
Seattle-Tacoma-Bremerton, WA	0.05366
Boston-Worcester-Lawrence-Lowell-Brockton, MA-NH-RI-VT	0.05051
Toronto	0.03602
Chicago-Gary-Kenosha, IL-IN-WI	0.03519
New York-No. New Jer.-Long Island, NY-NJ-CT-PA-MA-VT	0.02950
Denver-Boulder-Greeley, CO-KS-NE	0.02640
Ottawa	0.02385
CSS	
San Francisco-Oakland-San Jose, CA	0.10656
Washington-Baltimore, DC-MD-VA-WV-PA	0.06821
Los Angeles-Riverside-Orange County, CA-AZ	0.05817
Seattle-Tacoma-Bremerton, WA	0.04889
Boston-Worcester-Lawrence-Lowell-Brockton, MA-NH-RI-VT	0.03936
Toronto	0.03879
Chicago-Gary-Kenosha, IL-IN-WI	0.03647
New York-No. New Jer.-Long Island, NY-NJ-CT-PA-MA-VT	0.02959
Denver-Boulder-Greeley, CO-KS-NE	0.02781
Portland-Salem, OR-WA	0.02446
Ruby	
San Francisco-Oakland-San Jose, CA	0.13852
Washington-Baltimore, DC-MD-VA-WV-PA	0.06121
Los Angeles-Riverside-Orange County, CA-AZ	0.05210
Chicago-Gary-Kenosha, IL-IN-WI	0.04751
Seattle-Tacoma-Bremerton, WA	0.04751
Denver-Boulder-Greeley, CO-KS-NE	0.04368
Boston-Worcester-Lawrence-Lowell-Brockton, MA-NH-RI-VT	0.04160
Portland-Salem, OR-WA	0.03248
Toronto	0.03189
New York-No. New Jer.-Long Island, NY-NJ-CT-PA-MA-VT	0.02855
Java	
San Francisco-Oakland-San Jose, CA	0.12131
Washington-Baltimore, DC-MD-VA-WV-PA	0.07615
Seattle-Tacoma-Bremerton, WA	0.05430

Los Angeles-Riverside-Orange County, CA-AZ	0.05351
Boston-Worcester-Lawrence-Lowell-Brockton, MA-NH-RI-VT	0.04063
Toronto	0.03786
Chicago-Gary-Kenosha, IL-IN-WI	0.03324
Ottawa	0.02907
New York-No. New Jer.-Long Island, NY-NJ-CT-PA-MA-VT	0.02800
Dallas-Fort Worth, TX-AR-OK	0.02225

Table 12: Summary Statistics Commits by Programming Languages - Regression Data

Language	Min.	Median	Mean	Max.	Projects	N	Commits	Share
C	1	19	358.93	60,823	62,831	7,090	2,544,785	6.59%
C#	1	21	342.12	20,296	31,758	3,407	1,165,603	3.02%
C++	1	24	412.13	185,564	56,356	7,195	2,965,287	7.68%
CSS	1	78	280.82	225,948	113,043	14,838	4,166,808	10.79%
Go	1	21	299.00	25,176	51,227	5,176	1,547,626	4.01%
Java	1	26	413.50	221,308	83,344	7,899	3,266,237	8.46%
JavaScript	1	124	519.24	134,514	295,915	14,550	7,555,004	19.56%
Objective-C	1	10	118.92	8,202	14,998	2,870	341,295	0.88%
PHP	1	23	357.98	218,318	55,949	5,703	2,041,574	5.29%
Python	1	67	478.21	35,085	165,340	12,240	5,853,278	15.16%
R	1	29	660.30	406,159	15,691	1,415	934,331	2.42%
Ruby	1	35	363.57	48,738	117,255	8,674	3,153,640	8.17%
Rust	1	20	201.07	20,612	14,826	2,117	425,658	1.1%
Shell	1	27	158.92	35,587	54,173	10,299	1,636,682	4.24%
Swift	1	15	147.47	30,077	10,112	1,692	249,513	0.65%
TypeScript	1	15	159.05	20,600	25,945	4,861	773,163	2%

Table 13: Summary Statistics of Commits, Projects and Users - Full Data

Variable	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
Length User Observed	1	5	8	7.31	10	10
Commits per User	0	22	70	295.22	216	411,048
Commit per Project per Snapshot	1	1	3	14.36	9	184,681
Stars per Project	0	0	0	12.71	0	259,118
Stars per Project - Star > 0 and non-forked Projects	1	1	2	71.22	7	259,118
Forks per Project	0	0	0	3.59	0	145,997
Forks per Project - Forks > 0 and non-forked Projects	1	1	2	25.44	6	145,997
Programming Language per City	3	16	16	15.20	16	16
Programming Language per City per Snapshot	1	10	14	12.20	15	16
Programming Language per User	1	2	3	3.30	4	16
Programming Language per User per Snapshot	1	1	2	2.09	3	16

Table 14: Summary Statistics of Commits, Projects and Users - Regression Data

Variable	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
Length User Observed	10	10	10	10.00	10	10
Commits per User	36	535	1,092	2,232.14	2,369	411,048
Commit per Project per Snapshot	1	1	3	18.43	10	125,722
Stars per Project	0	0	0	45.61	1	259,118
Stars per Project - Star > 0 and non-forked Projects	1	1	3	152.57	16	259,118
Forks per Project	0	0	0	11.88	1	145,997
Forks per Project - Forks > 0 and non-forked Projects	1	1	3	47.08	11	145,997
Programming Language per City	1	10	15	12.78	16	16
Programming Language per City per Snapshot	1	5	10	9.31	13	16
Programming Language per User	1	5	6	6.36	8	16
Programming Language per User per Snapshot	1	2	3	3.21	4	15

Table 16: Summary Statistics - Clusters

Language	10th Perc.	Median	90th Perc.	Max.
C	0.00000	0.00077	0.01007	0.12342
C#	0.00004	0.00102	0.01038	0.10728
C++	0.00005	0.00079	0.00975	0.12567
CSS	0.00004	0.00068	0.01000	0.10656
Go	0.00000	0.00074	0.01020	0.17377
Java	0.00006	0.00067	0.01058	0.12131
JavaScript	0.00004	0.00066	0.00851	0.11493
Objective-C	0.00000	0.00092	0.01173	0.22056
PHP	0.00015	0.00114	0.01073	0.06843
Python	0.00004	0.00068	0.00837	0.13063
R	0.00000	0.00111	0.01166	0.08887
Ruby	0.00003	0.00073	0.01031	0.13852
Rust	0.00000	0.00105	0.01333	0.13907
Shell	0.00007	0.00088	0.01092	0.12119
Swift	0.00000	0.00069	0.01323	0.17912
TypeScript	0.00003	0.00069	0.01015	0.09802

Table 17: Cross-Field Spillover

	Log(Commit)						
	C	C#	C++	CSS	Go	Java	JavaScript
	(1)	(2)	(3)	(4)	(5)	(6)	(7)
Log(C)	0.8655**	0.2901	0.4497**	0.1411	-0.2797	0.0746	-0.0644
	(0.3332)	(0.3365)	(0.1947)	(0.1368)	(0.2952)	(0.3553)	(0.1674)
Log(C#)	-0.0279	-0.2074	0.1780	0.0452	-0.0228	0.0443	0.1249
	(0.2040)	(0.3545)	(0.1912)	(0.1278)	(0.2791)	(0.3332)	(0.1551)
Log(C++)	-0.0300	0.2589	-0.4548	-0.1040	-0.0689	-0.3129	-0.0423
	(0.3547)	(0.3183)	(0.3426)	(0.1464)	(0.3707)	(0.3164)	(0.1590)
Log(CSS)	-0.0146	-0.9773	0.4211	-0.2065	-0.4309	0.0251	0.0305
	(0.4670)	(0.8858)	(0.5231)	(0.2751)	(0.7287)	(0.6789)	(0.3692)
Log(Go)	0.1995	0.0291	0.0514	0.0217	-0.0651	-0.0367	0.0062
	(0.1414)	(0.2795)	(0.1320)	(0.0752)	(0.2245)	(0.1724)	(0.0843)
Log(Java)	-0.7931**	-0.1336	0.1148	0.2075	0.1145	0.3061	0.1517
	(0.3481)	(0.4175)	(0.3876)	(0.1790)	(0.3991)	(0.3804)	(0.2427)
Log(JavaScript)	-0.2220	1.5520**	0.2985	0.5192**	0.2729	-0.1749	0.1405
	(0.5383)	(0.7799)	(0.4458)	(0.2617)	(0.6480)	(0.7887)	(0.3359)
Log(Objective-C)	0.0587	-0.0389	-0.1717	-0.0469	-0.0235	-0.0817	-0.0352
	(0.1484)	(0.1376)	(0.1389)	(0.0640)	(0.1161)	(0.1126)	(0.0857)
Log(PHP)	-0.0724	-0.4316	-0.1688	-0.0310	0.3552	-0.4035	0.0201
	(0.2953)	(0.2938)	(0.2113)	(0.1335)	(0.2578)	(0.2697)	(0.1652)
Log(Python)	0.0380	0.1091	-0.4587	-0.1648	0.2758	0.2834	-0.2391
	(0.2976)	(0.5085)	(0.5010)	(0.2440)	(0.5446)	(0.4346)	(0.3770)
Log(R)	-0.0226	0.0447	-0.0789	-0.0332	0.0308	0.0595	-0.0792
	(0.1694)	(0.1724)	(0.1213)	(0.0820)	(0.2114)	(0.1214)	(0.0910)
Log(Ruby)	0.1136	-0.6016	-0.0657	-0.1786	0.1822	0.0442	-0.0647
	(0.2167)	(0.4531)	(0.2270)	(0.1399)	(0.2854)	(0.4874)	(0.2240)
Log(Rust)	-0.0216	0.1114	-0.0093	-0.0126	-0.0861	-0.0917	-0.0056
	(0.0682)	(0.0870)	(0.0634)	(0.0407)	(0.0729)	(0.0691)	(0.0440)
Log(Shell)	0.1102	-0.0155	0.1487	0.0153	0.1238	0.1643	-0.0223
	(0.2729)	(0.3880)	(0.3289)	(0.1652)	(0.2930)	(0.2884)	(0.1815)
Log(Swift)	-0.0290	0.0683	-0.0054	-0.0100	0.0287	0.1844	-0.0858
	(0.1163)	(0.1762)	(0.1230)	(0.0745)	(0.1989)	(0.1227)	(0.0848)
Log(TypeScript)	-0.1007	-0.1568	-0.1448*	-0.0929	-0.2873*	0.0273	0.0543
	(0.0887)	(0.1543)	(0.0870)	(0.0569)	(0.1629)	(0.1085)	(0.0723)
R^2	0.640	0.674	0.709	0.729	0.696	0.717	0.751
Observations	162,253	56,426	112,488	198,846	89,564	153,443	448,731

Table 18: Cross-Field Spillover Continued

	Log(Commit)						
	Objective-C	PHP	Python	R	Ruby	Rust	Shell
	(1)	(2)	(3)	(4)	(5)	(6)	(7)
Log(C)	0.3160	0.1128	0.2080	0.2009	0.2794	0.5632	0.0548
	(0.7642)	(0.2200)	(0.1512)	(0.2907)	(0.3043)	(1.1114)	(0.2443)
Log(C#)	0.2690	0.1363	-0.0909	0.2570	0.1300	-0.0869	0.2081
	(0.7871)	(0.2498)	(0.1627)	(0.2639)	(0.2227)	(0.8366)	(0.1979)
Log(C++)	-0.2014	0.0017	-0.1645	0.0914	-0.2489	-0.0100	-0.1387
	(1.1614)	(0.2217)	(0.1780)	(0.3179)	(0.2494)	(0.7325)	(0.2019)
Log(CSS)	0.6279	-0.3176	0.2354	-0.3206	0.4017	1.2765	0.0353
	(1.7177)	(0.5741)	(0.2721)	(0.7451)	(0.5820)	(1.6245)	(0.3719)
Log(Go)	0.2302	-0.0478	0.0733	-0.0638	-0.1011	0.2853	0.0402
	(0.5352)	(0.1074)	(0.0750)	(0.1463)	(0.1298)	(0.4897)	(0.0933)
Log(Java)	-0.2738	-0.0808	-0.0480	-0.0963	0.0444	0.0309	-0.1180
	(0.8253)	(0.3517)	(0.1855)	(0.4182)	(0.3263)	(1.1695)	(0.2584)
Log(JavaScript)	-0.3331	0.2101	-0.3208	-0.5057	-0.6619	-1.1046	-0.9413
	(1.5349)	(0.5464)	(0.3312)	(0.6451)	(0.5563)	(1.5346)	(0.7255)
Log(Objective-C)	-0.1773	0.0649	-0.0794	0.0024	0.0613	-0.5177	-0.0228
	(0.5051)	(0.1030)	(0.0683)	(0.1477)	(0.0973)	(0.3600)	(0.0668)
Log(PHP)	-0.2394	0.0682	0.0786	-0.0826	-0.0352	-0.5932	0.2489
	(0.9273)	(0.2392)	(0.1650)	(0.3658)	(0.2093)	(0.6814)	(0.1864)
Log(Python)	0.3390	0.2136	0.0194	0.5702	-0.5542	-0.4456	0.3353
	(1.5354)	(0.4083)	(0.2081)	(0.5519)	(0.3752)	(1.6123)	(0.3153)
Log(R)	0.0175	-0.0321	0.0425	-0.0304	-0.0112	-0.0129	-0.0672
	(0.5257)	(0.1425)	(0.0882)	(0.2111)	(0.1335)	(0.5184)	(0.0998)
Log(Ruby)	-0.0135	0.0547	-0.1285	-0.0665	0.4610	0.2027	0.1822
	(1.1629)	(0.2624)	(0.1635)	(0.3536)	(0.3967)	(0.8713)	(0.3179)
Log(Rust)	0.0343	0.0483	0.0013	-0.0444	-0.0554	0.1808	-0.0079
	(0.2500)	(0.0615)	(0.0399)	(0.1228)	(0.0570)	(0.2651)	(0.0617)
Log(Shell)	-0.8111	-0.1583	0.0930	-0.1031	0.2760	-0.1042	0.1134
	(0.9899)	(0.2576)	(0.1661)	(0.3276)	(0.2902)	(1.2556)	(0.2348)
Log(Swift)	0.0316	-0.1299	0.0518	0.1060	-0.0051	0.1972	0.0656
	(0.4291)	(0.1059)	(0.0693)	(0.2005)	(0.1300)	(0.5403)	(0.1603)
Log(TypeScript)	0.1997	-0.1334	0.0105	0.0074	0.0578	0.5280	0.0629
	(0.2920)	(0.1294)	(0.0568)	(0.1570)	(0.0944)	(0.4244)	(0.0712)
R^2	0.824	0.749	0.713	0.730	0.714	0.762	0.751
Observations	20,957	89,524	284,392	27,674	192,096	23,883	98,851

Table 19: Cross-Field Spillover Continued

	Log(Commit)	
	Swift	TypeScript
	(1)	(2)
Log(C)	-0.4674	0.1157
	(0.9546)	(0.6595)
Log(C#)	0.2043	0.1016
	(0.7530)	(0.5695)
Log(C++)	-0.2038	-0.9832
	(0.9233)	(0.9948)
Log(CSS)	-1.4997	0.5388
	(1.7098)	(1.3725)
Log(Go)	-0.0211	-0.3426
	(0.4454)	(0.5318)
Log(Java)	-0.3429	0.0536
	(1.3184)	(0.9812)
Log(JavaScript)	0.4877	-0.7892
	(1.9530)	(1.3602)
Log(Objective-C)	-0.0682	0.2281
	(0.5924)	(0.3085)
Log(PHP)	-0.1259	0.1259
	(1.0235)	(0.7434)
Log(Python)	0.6160	0.3713
	(1.4145)	(0.8875)
Log(R)	0.2533	0.2713
	(0.4941)	(0.3220)
Log(Ruby)	0.1079	0.5164
	(1.0733)	(0.7433)
Log(Rust)	0.0845	0.1838
	(0.2863)	(0.1797)
Log(Shell)	-0.2328	-0.0192
	(0.9791)	(0.8486)
Log(Swift)	0.5604	-0.0981
	(0.5205)	(0.3742)
Log(TypeScript)	0.3886	0.0135
	(0.4569)	(0.3932)
R^2	0.789	0.819
Observations	14,941	39,685

*Signif. Codes: ***: 0.01, **: 0.05, *: 0.1.*

Notes: Standard Errors are clustered by city × (programming) language. Every column presents a regression. In column C, only C-users are included in the sample. In all regressions, fixed effects for city, time, programming language, city × programming language, programming language × time, user and project are included.

Table 20: Absolute Cluster Size

	Log(Commit)					
	(1)	(2)	(3)	(4)	(5)	(6)
Log(Abs. Cluster Size)	0.0102	0.0128	0.0114	0.0366***	0.0641	0.1030*
	(0.0157)	(0.0097)	(0.0115)	(0.0125)	(0.0420)	(0.0566)
Fixed-effects						
City	Yes	Yes	Yes	Yes	Yes	Yes
Time	Yes	Yes	Yes	Yes	Yes	Yes
Language	Yes	Yes	Yes	Yes	Yes	Yes
User		Yes	Yes	Yes	Yes	Yes
City x Time			Yes	Yes	Yes	Yes
Language x Time				Yes	Yes	Yes
Project					Yes	Yes
City x Language						Yes
R^2	0.015	0.105	0.106	0.107	0.700	0.702
Observations	2,095,978	2,095,978	2,095,978	2,095,978	2,095,978	2,095,978

*Signif. Codes: ***: 0.01, **: 0.05, *: 0.1.*

Notes: Standard Errors are clustered by city x (programming) language. Every column presents a regression.

Table 21: Baseline Estimates - Excluding Projects large Commits and large Projects

	Log(Commit)					
	(1)	(2)	(3)	(4)	(5)	(6)
Log(Size)	0.0350**	0.0219***	0.0238**	0.0225**	0.0583	0.1114**
	(0.0141)	(0.0084)	(0.0106)	(0.0104)	(0.0357)	(0.0559)
Fixed-effects						
City	Yes	Yes	Yes	Yes	Yes	Yes
Time	Yes	Yes	Yes	Yes	Yes	Yes
Language	Yes	Yes	Yes	Yes	Yes	Yes
User		Yes	Yes	Yes	Yes	Yes
City x Time			Yes	Yes	Yes	Yes
Language x Time				Yes	Yes	Yes
Project					Yes	Yes
City x Language						Yes
R^2	0.017	0.097	0.099	0.100	0.706	0.708
Observations	1,946,910	1,946,910	1,946,910	1,946,910	1,946,910	1,946,910

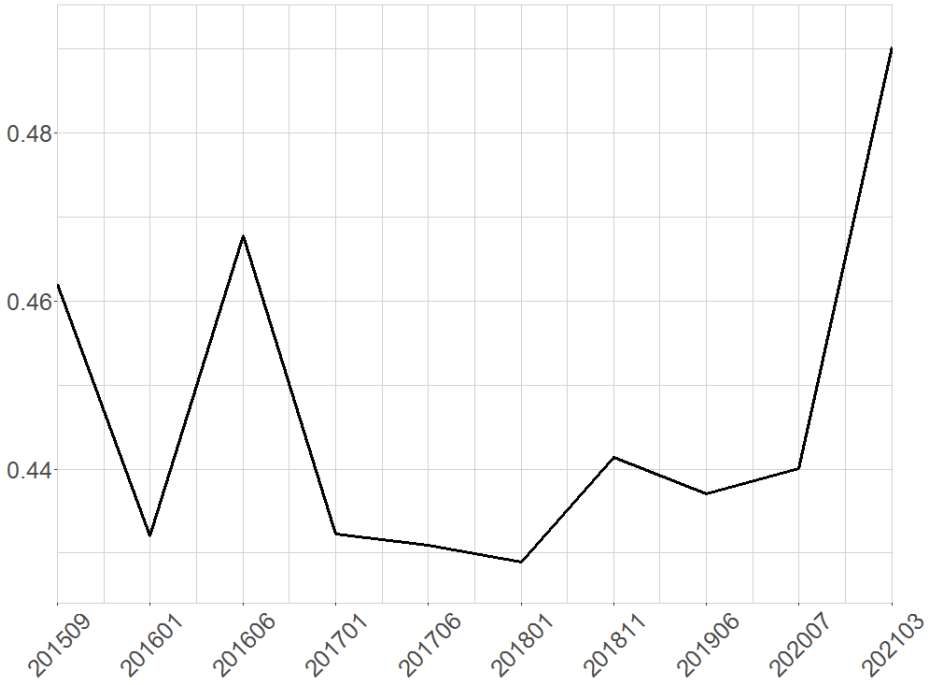
*Signif. Codes: ***: 0.01, **: 0.05, *: 0.1.*

Notes: Standard Errors are clustered by city x (programming) language. Every column presents a regression. Sample includes only projects with less than 40 users committing to and commits to projects less than 100.

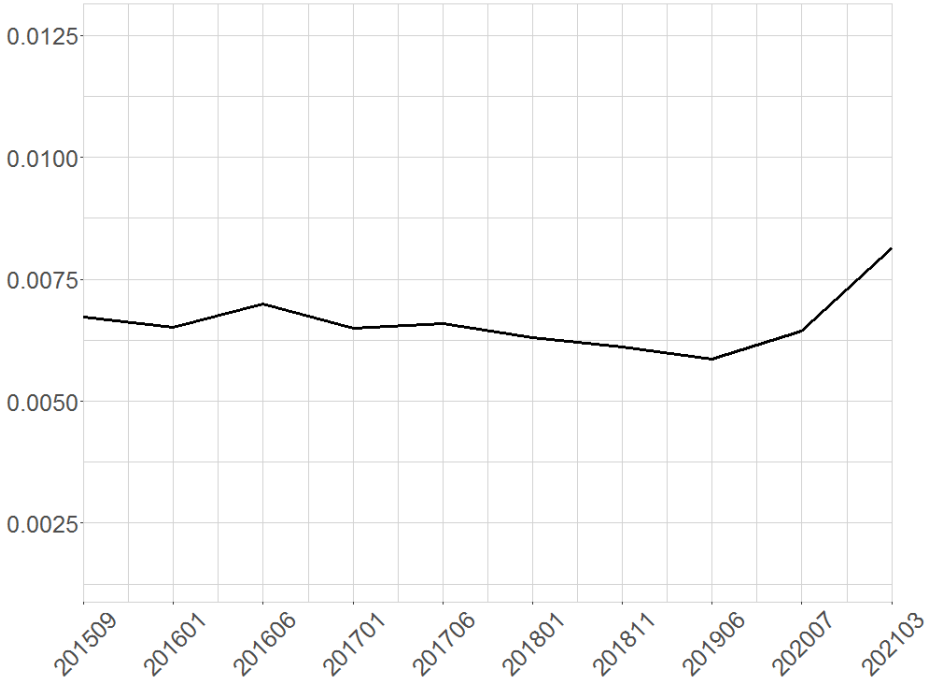
A.2 Figures

Figure 6: Share of Top 10 Cities for Each Programming Languages

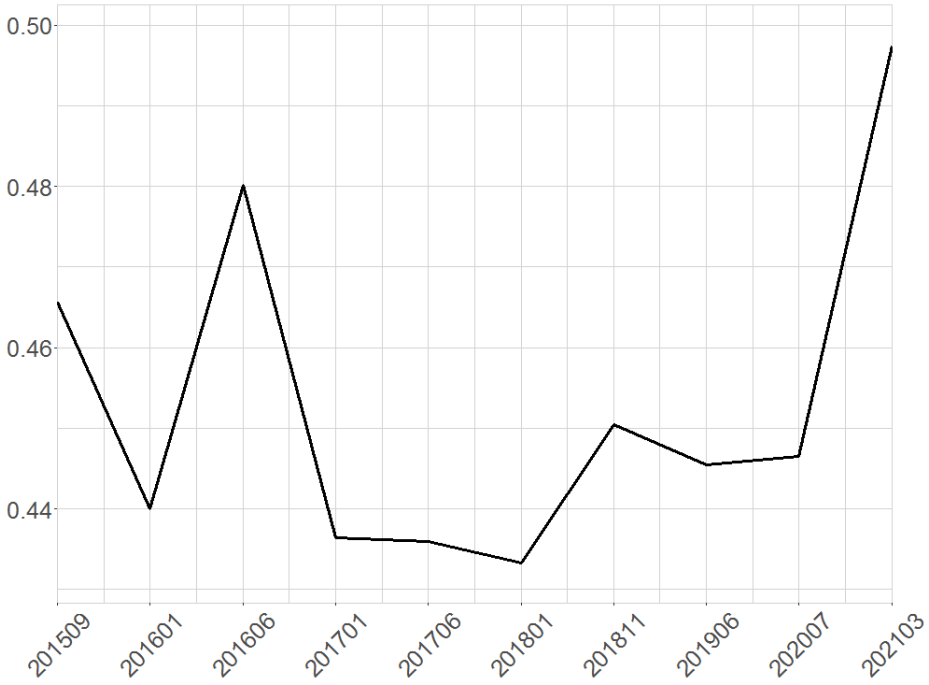
(a) C



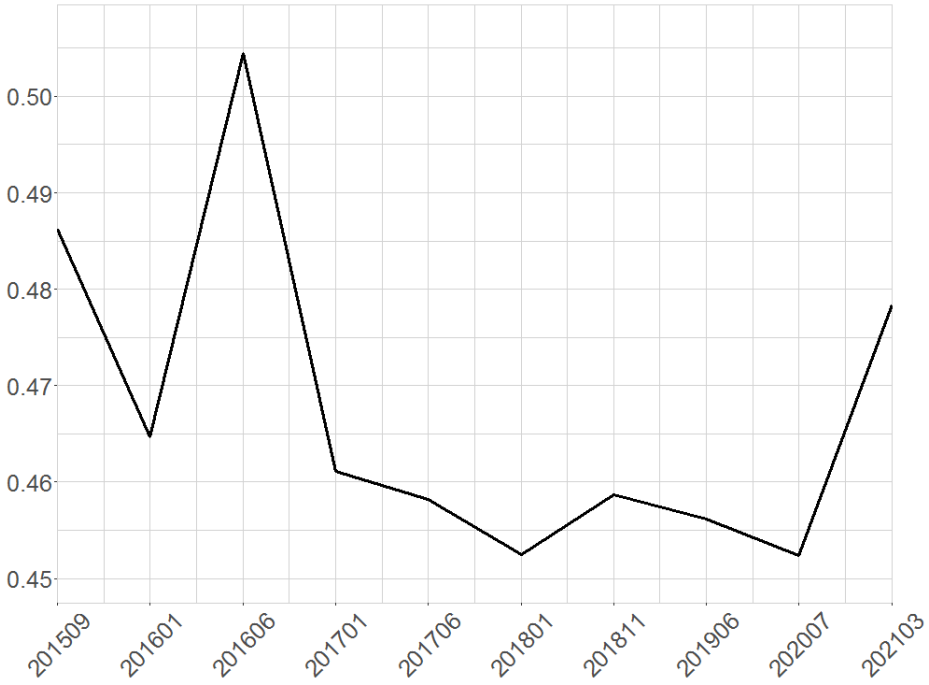
(b) C#



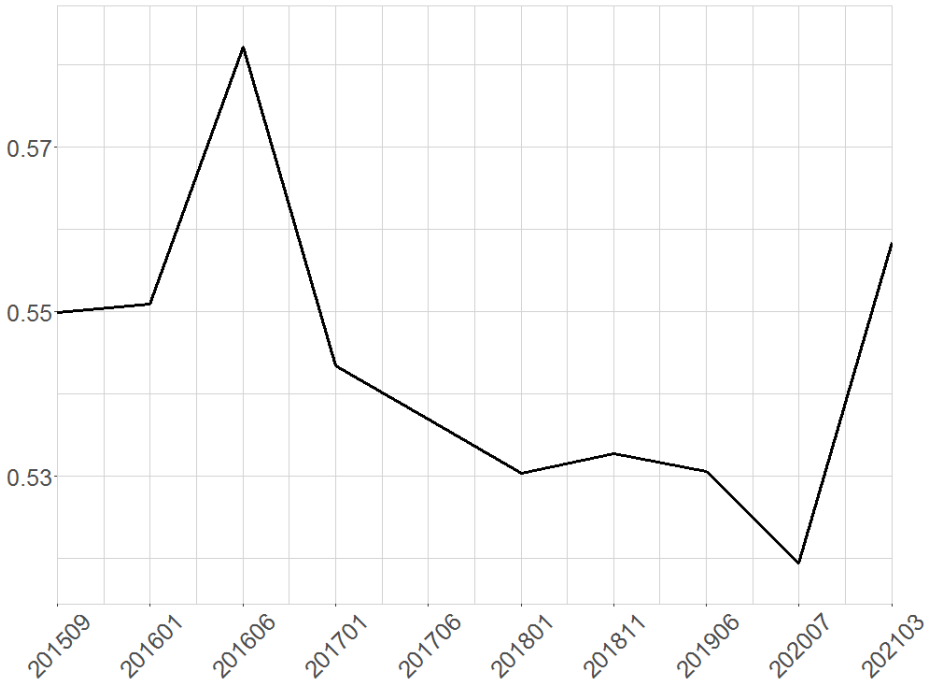
(c) C++


0.50
0.48
0.46
0.44
201509
201601
201606
201701
201706
201801
201811
201906
202007
202103

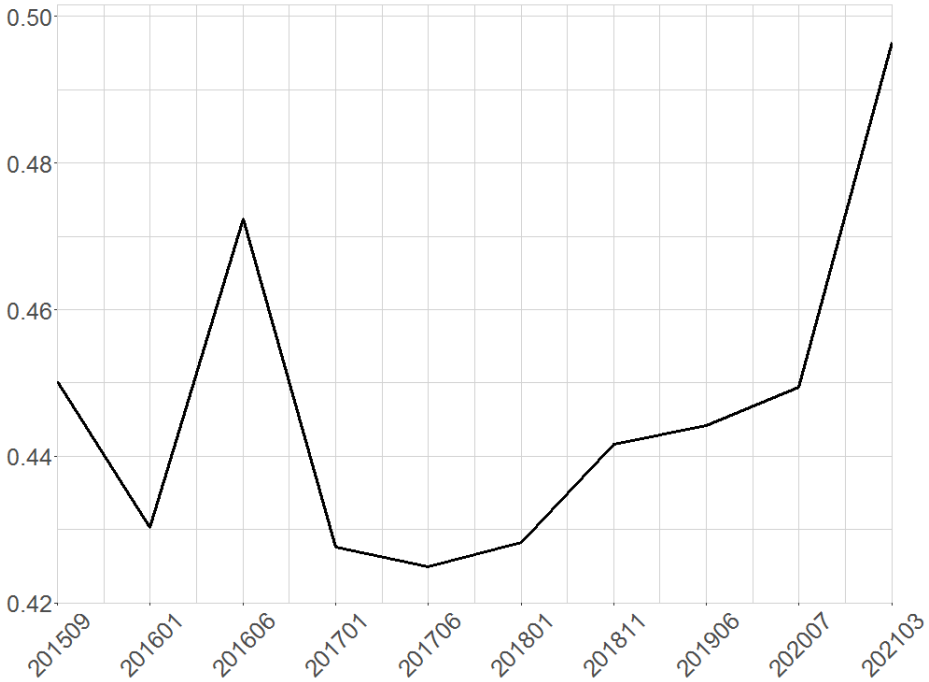
(d) CSS


0.50
0.49
0.48
0.47
0.46
0.45
201509
201601
201606
201701
201706
201801
201811
201906
202007
202103

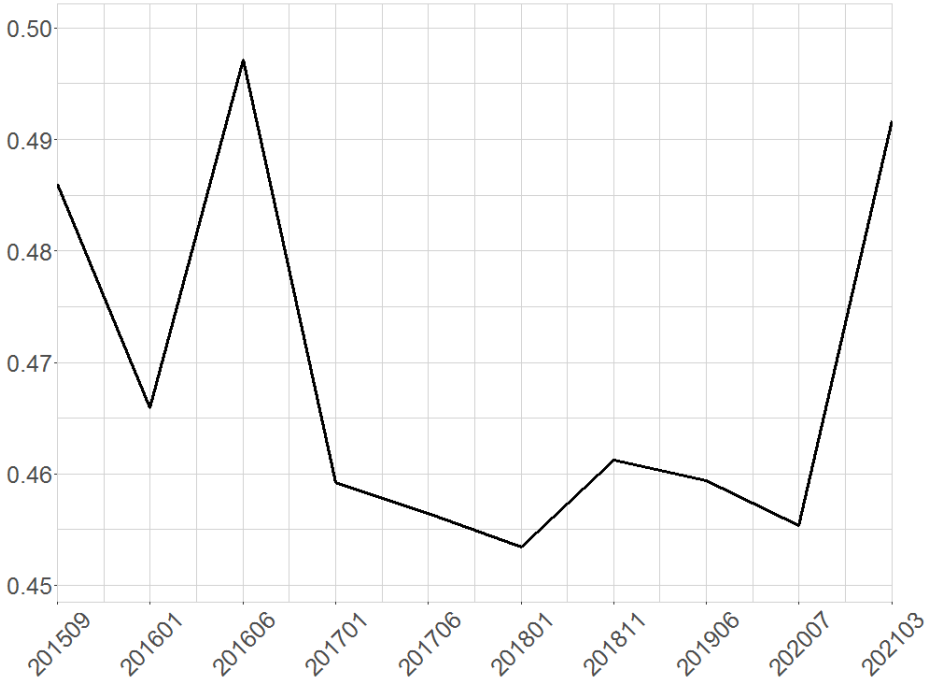
(e) Go


0.57
0.55
0.53
201509
201601
201606
201701
201706
201801
201811
201906
202007
202103

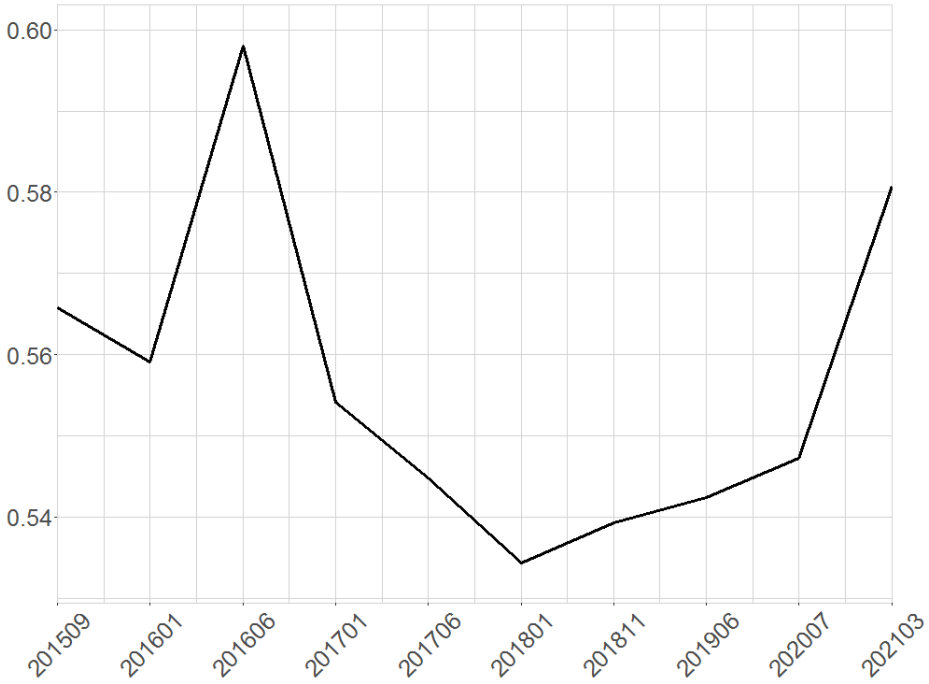
(f) Java


0.50
0.48
0.46
0.44
0.42
201509
201601
201606
201701
201706
201801
201811
201906
202007
202103

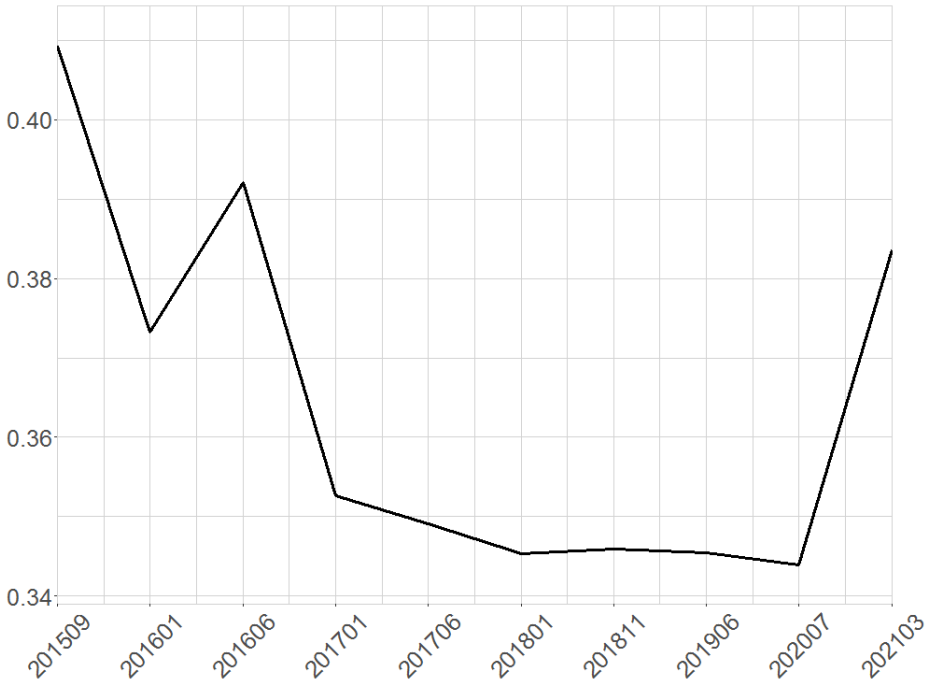
(g) JavaScript



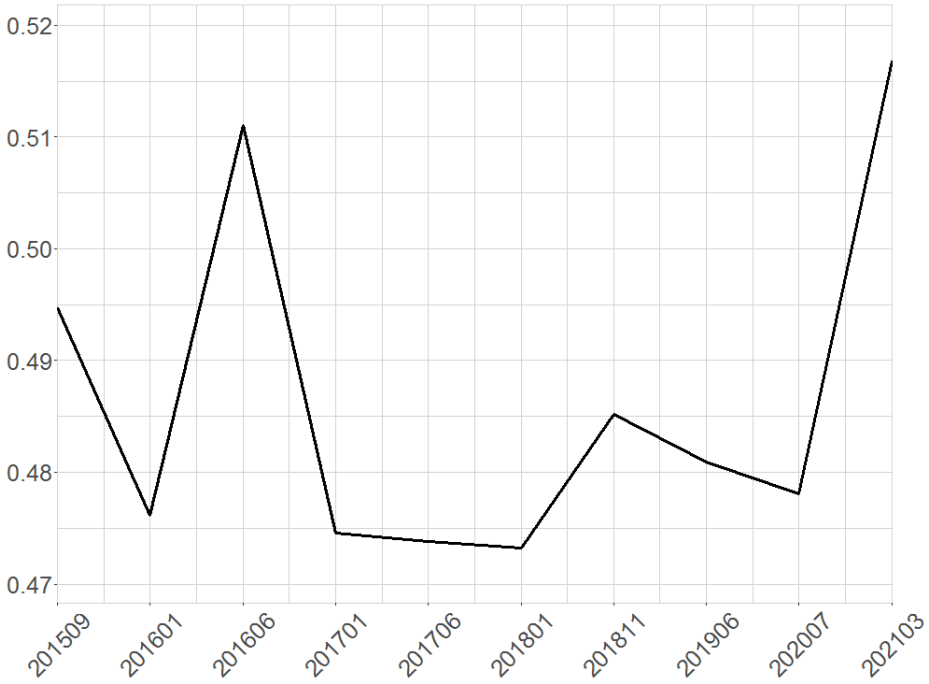
(h) Objective-C



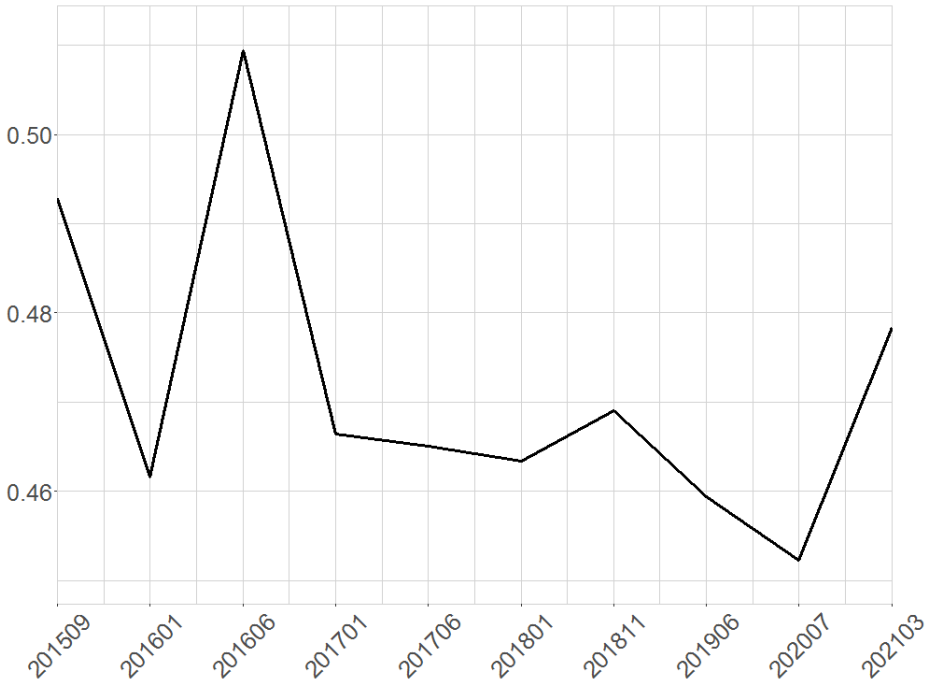
(i) PHP



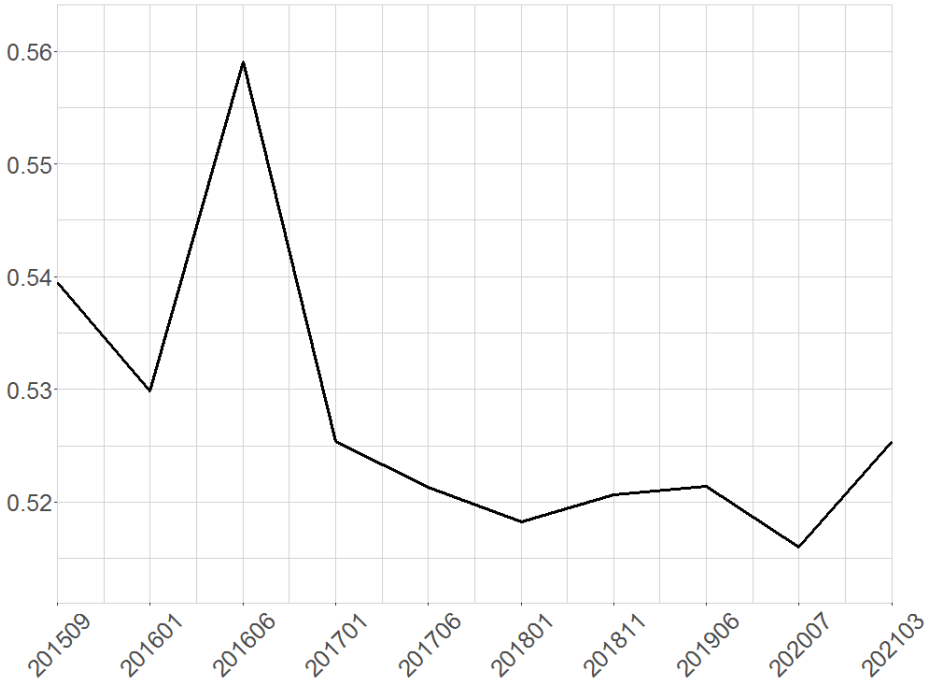
(j) Python



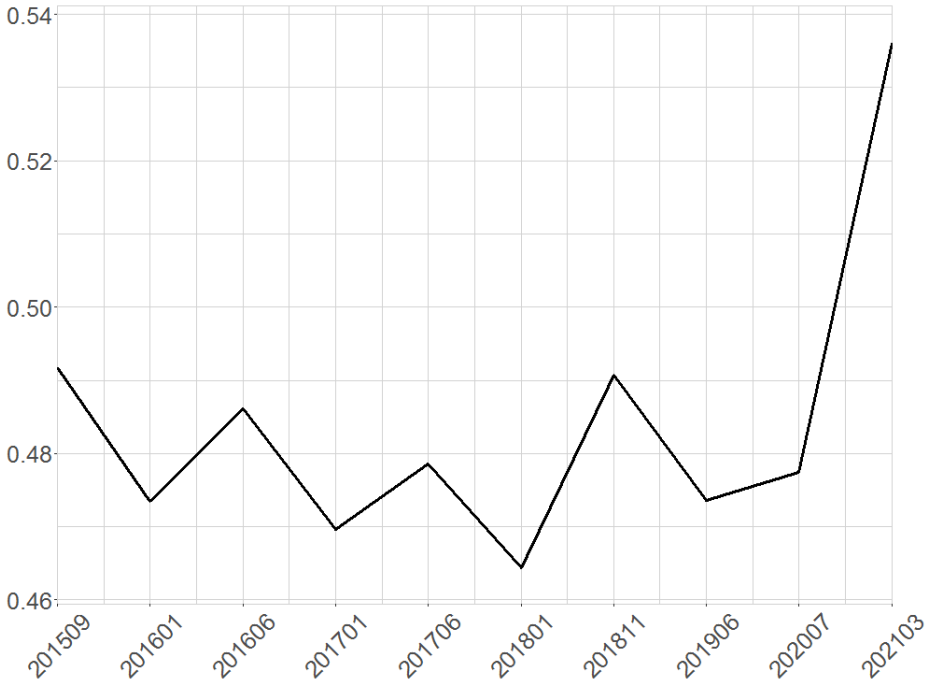
(k) R


0.50
0.48
0.46
201509
201601
201606
201701
201706
201801
201811
201906
202007
202103

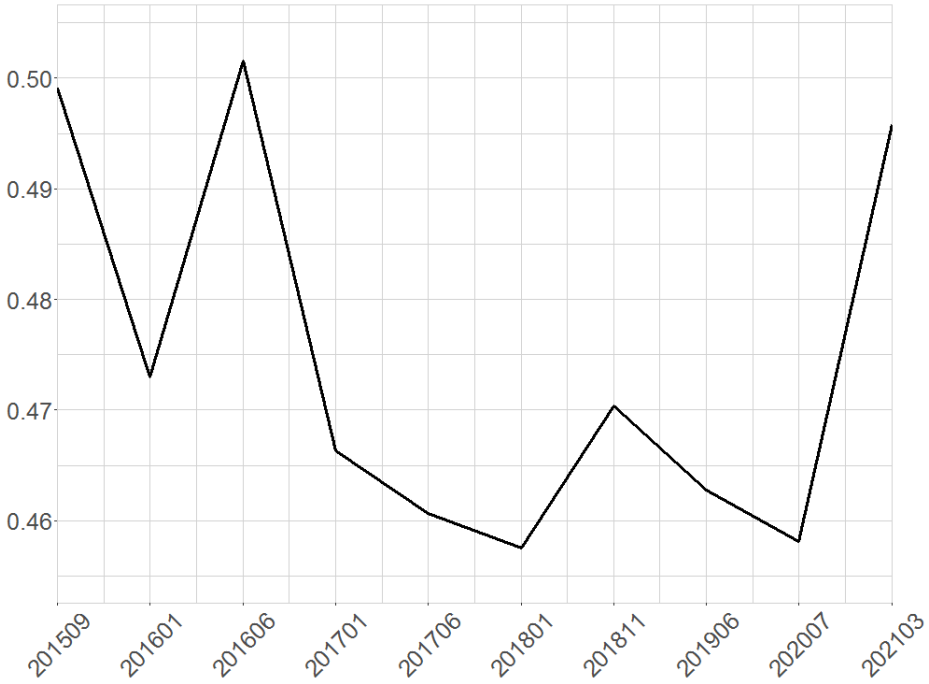
(l) Ruby


0.56
0.55
0.54
0.53
0.52
201509
201601
201606
201701
201706
201801
201811
201906
202007
202103

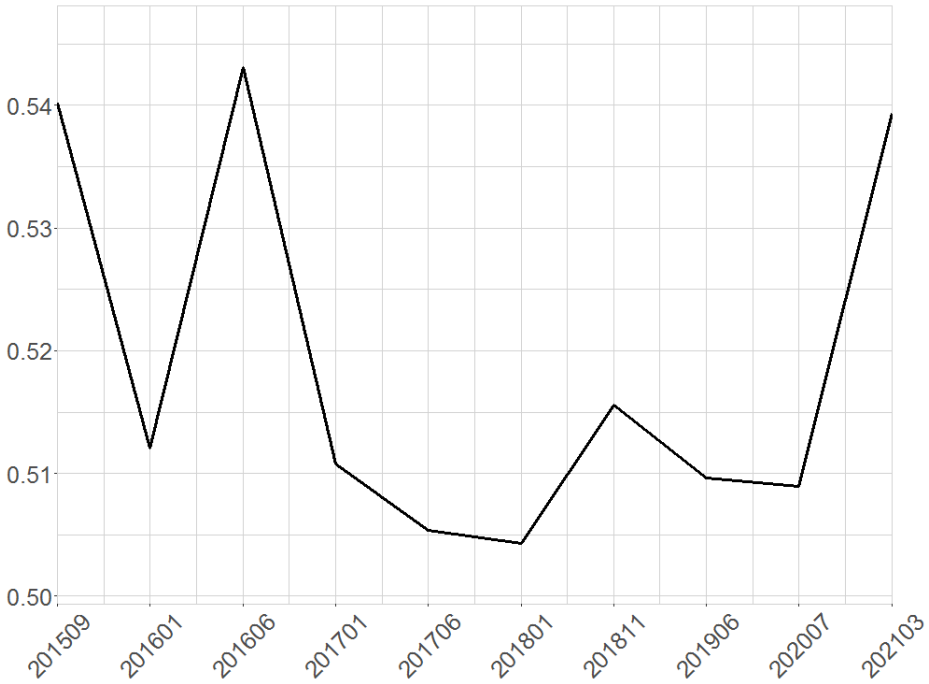
(m) Rust



(n) Shell



(o) Swift



(p) TypeScript