Big Data = Big Insights? Operationalising Brooks' Law in a Massive GitHub Data Set

Christoph Gote
cgote@ethz.ch
Chair of Systems Design, ETH Zurich
Zurich, Switzerland

Pavlin Mavrodiev
pmavrodiev@ethz.ch
Chair of Systems Design, ETH Zurich
Zurich, Switzerland

Frank Schweitzer
fschweitzer@ethz.ch
Chair of Systems Design, ETH Zurich
Zurich, Switzerland

Ingo Scholtes*
ingo.scholtes@uni-wuerzburg.de
Chair of Computer Science XV - Machine Learning for Complex Networks, Julius-Maximilians-Universität Würzburg
Würzburg, Germany

ABSTRACT

Massive data from software repositories and collaboration tools are widely used to study social aspects in software development. One question that several recent works have addressed is how a software project's size and structure influence team productivity, a question famously considered in *Brooks' law*. Recent studies using massive repository data suggest that developers in larger teams tend to be less productive than smaller teams. Despite using similar methods and data, other studies argue for a positive linear or even super-linear relationship between team size and productivity, thus contesting the view of software economics that software projects are *diseconomies of scale*.

In our work, we study challenges that can explain the disagreement between recent studies of developer productivity in massive repository data. We further provide, to the best of our knowledge, the largest, curated corpus of *GitHub* projects tailored to investigate the influence of team size and collaboration patterns on individual and collective productivity. Our work contributes to the ongoing discussion on the choice of productivity metrics in the operationalisation of hypotheses about determinants of successful software projects. It further highlights general pitfalls in big data analysis and shows that the use of bigger data sets does not automatically lead to more reliable insights.

ACM Reference Format:
Christoph Gote, Pavlin Mavrodiev, Frank Schweitzer, and Ingo Scholtes. 2022. Big Data = Big Insights? Operationalising Brooks' Law in a Massive GitHub Data Set. In *44th International Conference on Software Engineering (ICSE '22), May 21–29, 2022, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3510003.3510619

*Also with Data Analytics Group, Department of Informatics, University of Zurich.

ICSE '22, May 21–29, 2022, Pittsburgh, PA, USA

ACM ISBN 978-1-4503-9221-1/22/05.
https://doi.org/10.1145/3510003.3510619

1 INTRODUCTION

Empirical research across disciplines is nowadays driven by the availability of big data and methods to process and analyse them efficiently. In empirical software engineering, massive data from software repositories and online collaboration tools are widely used to investigate social and human aspects in software development. This intersects with *computational social science*, which uses big data to test hypotheses about individual and collective human behaviour originally developed in sociology, social psychology, or organisational theory. Data-driven studies of developer productivity in large software projects are an exemplary case of how research in empirical software engineering can advance computational social science. The question of how factors like, e.g., team size, influence the productivity of team members was already addressed by Maximilien Ringelmann [1] in 1913. In social psychology, his finding that individual productivity tends to linearly decrease with team size is known as the *Ringelmann effect*. In software project management, a similar observation is famously paraphrased as *Brooks' law* [2]. Here, the anecdote that "adding manpower to a late project makes it later" captures that the overhead associated with growing team sizes can reduce team efficiency. Studies of collaborative software projects found evidence for a strong Ringelmann effect for different team sizes, programming languages, and development phases [3–6]. Other studies, however, found a positive linear or even super-linear relationship between the size of a team and the productivity of its members [7–9].

The fact that different works studying the same research question yield qualitatively different results, despite applying similar methods to data from similar or even identical sources, should concern us. Referring to the massive number of projects, commits, or developers covered in their studies, authors often corroborate their findings by the size of the data used to obtain them, thus implying that the analysis of bigger data automatically yields more reliable insights. This points to an important general issue relevant for empirical research beyond software engineering: Apart from advantages in terms of coverage, resolution, or statistical confidence, the use of big data also introduces new *threats* for the validity of results. To address this issue, in this work, we explore *four challenges in the analysis of big data*. We study those challenges in a massive GitHub data set and argue that they are likely to explain

conflicting results on the Ringelmann effect that were reported in recent works.

A first challenge is the ***quality of big data*** that, rather than being carefully collected and curated to address a specific research problem, are often incidentally generated as "digital exhaust" of large online platforms. In empirical software engineering, this holds for massive data on software repositories harvested from online platforms like, e.g., *GitHub* or *SourceForge*. While massive repository data promise insights into universals of collaborative software development, they are known to suffer from various quality issues. These originate, e.g., from the inclusion of repositories that do not relate to software projects, projects whose development history is only partially represented in the data, or ambiguities that hinder the reliable identification of developers [10–13]. This poses particular problems for studies addressing the effect of team size on developer productivity, which require reliable data on *collaborative* projects that provide a complete picture of development actions attributable to individual team members.

A second challenge is ***population validity***, which determines whether findings obtained in a given data sample can be extrapolated to a larger population. On the one hand, for reasons of computational efficiency, researchers often base their results on a subset of the observations available in massive data, which can introduce biases that question population validity. On the other hand, we can not necessarily avoid such biases by using all available data since population validity depends on the population for which we want to answer a given research question. In massive repository data, using full information on all *GitHub* projects may be justified if we want to answer a question about the population of *GitHub* projects. However, if we use data on *GitHub* to obtain generalisable findings on how the size of software development teams affects the productivity of team members, we must carefully select projects to avoid biased samples in which either small or large teams are overrepresented.

A third challenge is ***construct validity***, which includes the issue that rich and big data provide various options to operationalise research questions or hypotheses. Whether or not the specific operationalisation chosen by a study is valid to address a research question is an important issue that influences the validity of results. In the context of developer productivity, data on software repositories is an example of high-dimensional and time-resolved data. In such data, productivity can be measured in various ways, and analyses can be applied for different levels of temporal aggregation, which is likely to affect the results.

Finally, ***omitted-variable bias*** is a fourth challenge that limits the reliability of findings if relevant variables are excluded from an analysis. Technically, this is a general challenge that is not *due* to the characteristics of big data. We nevertheless consider it in our work because high-dimensional data are likely to contain variables that can be used to address this issue. In the context of Brooks' law, we can think of multiple explanations for an observed relationship between, e.g., the size of a team and the productivity of its members. One explanation could be a *causal* mechanism by which growing team size influences developer productivity, e.g., by reducing or increasing the motivation of team members. An alternative explanation could be an additional variable related to the size of a team *and* developers' productivity, such as e.g., the collaboration structure of a team. A lack of control for such variables not only introduces biases in the inference of the actual relationship between variables of interest. It can also lead to the identification of spurious cause-effect relationships that negatively influence decision-making.

The four challenges summarised above question both the *internal and external validity* [14] of empirical research in software engineering, which can explain why works studying the same question in the same data arrive at different conclusions. Focusing on the operationalisation of Brooks' law, in this work, we show how to address them in massive *GitHub* data. Our contributions are as follows:

- To address the challenges of ***data quality*** and ***population validity***, we create a large, curated data corpus on Open Source Software (OSS) projects that facilitates the study of the influence of team size on both individual and collective productivity. The projects included in this corpus are *systematically* chosen based on (i) transparent filtering criteria that avoid common perils in GitHub mining [11] and (ii) a stratified sampling that supports unbiased analyses of the impact of team size on developer productivity. We make both our corpus and the pipeline to filter, sample, and process data based on `GHTorrent` [15], a database freely available for researchers.
- To address ***construct validity***, we systematically compare metrics for developer productivity in the data corpus created above. Acknowledging that productivity is a multi-dimensional phenomenon, we select a set of eight code- and commit-based productivity measures. We study their cross-correlation to answer which of the measures are likely to be interchangeable and which capture independent dimensions of productivity.
- Addressing ***omitted-variable bias***, we finally study to what extent changes in productivity can be causally explained by the collaboration structure of projects rather than team size. Building on a recently developed method to construct time-evolving co-editing networks based on `git` repositories [16], we investigate eight *network metrics* whose choice is rooted in social capital theory. We study the cross-correlation of those metrics to identify which of them capture independent dimensions.
- We apply our methods to study the Ringelmann effect in collaborative software development based on the corpus and methods developed above. We find a strong and significant negative relationship between team size and individual productivity that can be explained based on changes in the collaboration structure of software teams. We further show that a failure to account for the challenges outlined above can lead to spurious results that suggest a *positive* relationship.

In summary, we study challenges that can explain the disagreement between recent studies of developer productivity in massive repository data. We further provide, to the best of our knowledge, the largest, curated corpus of *GitHub* projects tailored to investigate the influence of team size and collaboration patterns on individual and collective productivity. Our work contributes to the ongoing

discussion on the choice of productivity metrics in the operationalisation of hypotheses about determinants of successful software projects. It further highlights general pitfalls in big data analysis and shows that the use of bigger data sets does not automatically lead to more reliable insights.

2 SYSTEMATIC CONSTRUCTION OF DATA CORPUS

We first introduce a framework to select and mine projects from *GitHub* that can be used to address the first two challenges of *data quality* and *population validity*. We use it to systematically sample 201 OSS projects covering the entire range of team sizes on *GitHub*. We further extract time-stamped editing events that we use to investigate whether collaboration structures can explain team productivity.

2.1 Data quality

To select suitable projects from *GitHub*, we propose the project selection and sampling pipeline shown in Figure 1. As a first step, we need to gain access to the information required to apply our selection criteria. Because *GitHub*'s REST API is rate-limited to 5,000 requests per hour[1], retrieving the metadata of more than 100 million repositories hosted on *GitHub* [17] becomes untenable. We, therefore, use the database made available by the `GHTorrent` project [15], which has crawled most of *GitHub*'s REST API using donated API keys. We use the latest[1] available dump from June 2019 that contains data on a total of more than 125 million repositories.

As a second step, we determine which projects in the `GHTorrent` database are suitable to study *collaborative OSS development*. It has already been reported by the authors of [11] that the majority of projects on *GitHub* are either personal, inactive, or very small and should be excluded when analysing collaborative software development. We adopt the filtering criteria proposed by the authors of [11], namely excluding repositories with a single developer, fewer than 50 commits, or a span of fewer than 100 days between the first and last commit in the repository. The resulting data set reduces to around 4.5 million OSS projects, i.e., 3.6% of all `GHTorrent` repositories. In other words, following [11], at least 96.4% of the repositories in the latest *GHTorrent* database are not suitable for studying collaborative software development.

We further improve on these filtering criteria in the following two ways. First, *GitHub* is frequently used for applications not related to software development, such as free file storage or web hosting [11]. Our own analysis revealed a substantial number of popular repositories[2] not representative of collaborative software development, e.g., tutorials on *git*, code snippet repositories, or *git*-based "clocks", which are updated with a new commit every second. Excluding these repositories is crucial to avoid misleading results.

Second, as a result of the functionality to *fork* any public repository, *GitHub* contains a substantial amount of repositories that are in large parts exact copies of other projects. We drop all repositories that are designated as *forks* to avoid biases from analysing the same commit history multiple times.

[1] as of June 2020
[2] popular judged by a high count of forks, commits, developers, or stars

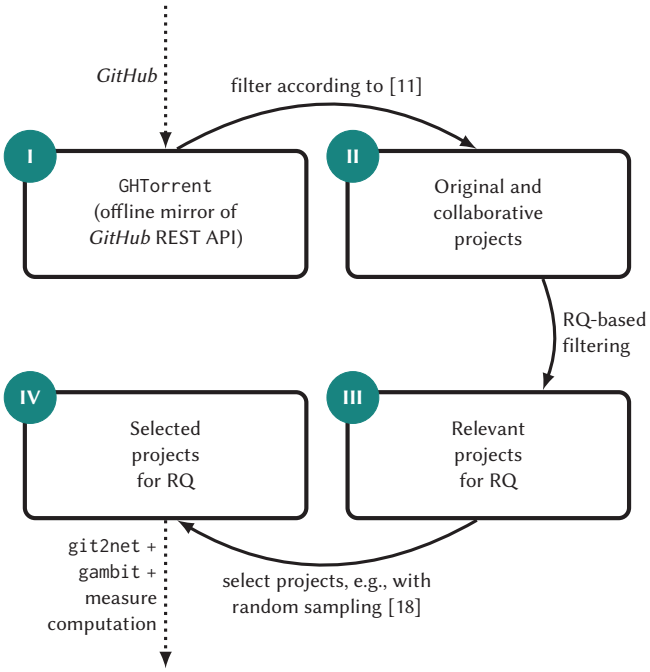

Figure 1: Pipeline to select and sample collaborative software development projects from *GitHub* to address a given research question (RQ).

After applying these two additional filtering steps, we retain a list of around 1.8 million original and collaborative repositories, which is only 1.4% of all repositories in `GHTorrent`. This underlines the importance of proper data selection when analysing collaborative software development on *GitHub*.

2.2 Population validity

Besides projects that are original and collaborative, our study on team productivity requires projects that fulfil additional conditions regarding their (i) activity, (ii) size, and (iii) purpose, i.e., collaborative projects for developing software (step 3 in Figure 1).

Project activity. To avoid issues that could arise from mixing active projects with those where development has ceded long in the past, we focus on projects that are actively developed at the time of our study. We regard a project as active if the last commit was made after May 2020. To ensure this, we take a two-fold approach: we first select projects from `GHTorrent` that have a recorded commit activity after May 2019. In a subsequent step, we then use the *GitHub* REST API to filter those projects that additionally have a recorded commit after May 2020.

Project size. To facilitate an unbiased sampling of projects based on team size, we first need to determine the size of a development team. For OSS projects without formal team memberships, this is a challenging task. The authors of [3] found that the probability of making future contributions to an OSS project drops below 10% after an inactivity of approx. 42 weeks. Based on this finding, we

Table 1: Number of projects with different team size ranges. All given team size ranges include the outer values. Team sizes are computed based on the data available in `GHTorrent`.

Team Size	Projects Tot.	Projects Sel.	Team Size	Projects Tot.	Projects Sel.
2 – 4	95,763	19	59 – 115	1,476	25
5 – 8	33,027	27	116 – 226	665	16
9 – 15	17,221	26	227 – 443	231	13
16 – 30	10,027	27	444 – 871	102	15
31 – 58	3,499	27	872 – 1,711	28	6

compute the size of an OSS development team at time t by counting all developers who committed within a moving time window of 294 days, i.e., between $t - 294$ days and t. Thus, to compute a team size, projects need to have existed for at least 294 days, increasing the requirements beyond the 100 days considered by the authors of [11].

Table 1 shows the number of projects for different team sizes, where team size is computed for the latest available 294-day time window. The project counts are reported for ten $\log_2$-spaced strata, which yields a distribution where the team size roughly doubles for each consecutive stratum. The resulting distribution is right-skewed, where the vast majority of projects have small team sizes. A uniform sample from the complete set of projects would thus primarily select small projects, which would fail to cover a broad spectrum of team sizes. To remedy this, we sample 28 projects from each stratum, where 28 is the size of the stratum with the fewest projects.

While sampling, we ensure that all sampled projects are software development projects and continue to be actively developed at the time of mining. We further remove duplicate projects that originate from manual clones of other repositories. We achieve this by applying the following selection criteria:

Project purpose. To identify a project's purpose, we query the *GitHub* REST API to obtain the most recent information on all considered projects. We first ensure that all sampled projects are software development projects and continue to be actively developed at the time of mining. We consider a project as a software development project if at least 75% of the code in the repository is written in the 17 programming languages supported by the code analysis tool `lizard` [19]. In total, the languages supported by `lizard` account for over 85% of the code submitted to *GitHub* [20].

Deduplication. Finally, our set of projects still contains duplicate repositories that originate from manual clones pushed to a different repository rather than using the fork mechanic recorded in the `GHTorrent` data. Removing these clones is an important challenge when selecting repositories for analysis, and independent data sets listing duplicate repositories have been developed [21]. Unfortunately, these data sets were not yet available at the time of our analysis. Therefore, we manually removed the clones, retaining the original project that was cloned.

The two selection criteria require us to query the *GitHub* REST API or perform manual filtering, respectively. Due to the API's rate limit, this means that neither can be performed at large scale before sampling projects. Instead, they need to be performed during the sampling process. We treat all strata equally and apply the additional selection criteria to the sampled 28 projects from each stratum. As shown in the final column of Table 1, this yields between 6 and 27 projects for each of the ten strata, resulting in a total of 201 projects with a total of more than 100,000 developers and over 3 million commits (step 4 in Figure 1). Overall, we obtain relatively similar project counts for all strata, except for the strata with largest and smallest team sizes.

2.3 Mining co-editing networks from git repositories

We mine all edits and co-edits for the full history of the 201 projects using the Open Source Python tool `git2net` [16]. Besides co-editing relations, we also extract both commit- and code-based productivity measures. To this end, we apply `lizard` [19] and an optimised version of `multimetric` [22] to the source code before and after each change. Obtaining highly granular information on the development process of over 200 OSS projects requires substantial computational resources, in our case, over 1 million CPU-hours. Therefore, we perform all computations on 256 compute cores within a time frame of over six months on the ETH Zurich scientific compute cluster *Euler*.

An additional challenge in the analysis of *git* repositories is the need to disambiguate commit authors. This step is necessary as developers can make contributions using different credentials, e.g., due to spelling errors in usernames or the use of nicknames. Considering different aliases as different users would lead us to overestimate the team size and underestimate the productivity of developers with multiple aliases. We thus use the recently proposed tool `gambit` [13] to disambiguate all developers in all repositories.

Upon manual inspection, we found that some projects contain very large commits originating from code imports or automated code refactoring tools. Such, mostly automated, commits are not representative for the coordination requirements between developers. However, due to their size, they could lead to a bias our subsequent analysis. Therefore, as a final data cleaning step, we drop outliers by excluding all commits outside the 2.5th and 97.5th percentile regarding their total Levenshtein distance.

A complete list of projects as well as anonymised raw data of all projects considered in our analysis is archived on zenodo.org[3].

3 OPERATIONALISING PRODUCTIVITY AND COLLABORATION STRUCTURE

To study how team size affects the productivity of OSS projects, we need to operationalise (i) the size of OSS teams, and (ii) the productivity of OSS teams. In addition, we aim to understand how coordination between different team members affects this relation. Therefore, we need to also operationalise (iii) the collaboration structure of OSS teams.

We base our operationalisations of all three concepts on the edits and co-edits observed within non-overlapping 42-week time windows. As discussed in Section 2.2, the choice of 42 weeks is

[3] https://doi.org/10.5281/zenodo.5294964

Table 2: Productivity measures considered in this paper. All measures are evaluated over a time window of length Δt and normalised by the team size (TS).

Commit-Based	**Comms**	commits $/\Delta t/$TS
	Events	lines added, modified, or deleted $/\Delta t/$TS
	LevD	characters modified $/\Delta t/$TS
Code-Based	**NLOC**	lines of code changed $/\Delta t/$TS
	Tokens	change in number of tokens $/\Delta t/$TS
	Funcs	change in the number of functions $/\Delta t/$TS
	CycC	change in cyclomatic complexity $/\Delta t/$TS
	HalEff	Halstead effort to make changes $/\Delta t/$TS

motivated by [3] who found that after this time, the probability of a developer making future contributions to a project is less than 10%. Ensuring that the time window is divisible by full weeks is essential to ensure that the weekly productivity patterns present on *GitHub* [23] do not bias our results.

In the next three sections, we will discuss each of the operationalisations in detail.

3.1 Team size

OSS projects utilise the principles of open collaboration to create new software. This means that they rely on contributions of loosely coordinated participants, who differ significantly regarding the size of their contributions. Contributors can further join and leave the team at any time. Due to this method of collaboration, no organised ledgers listing the members of OSS teams exist. This makes operationalising the size of such teams non-trivial.

The consensus of prior literature is that all individuals contributing to an OSS project should be considered as team members [24]. With this work, we study the production of code artefacts. Therefore, we operationalise team size as the count of all individuals who contribute code to a project within a given time window. This includes all developers adding, modifying, or removing code from the project's codebase.

3.2 Productivity measures

Before discussing how we operationalise team productivity, we need to precisely define this term. Productivity captures one aspect of the broader concept of *team effectiveness*. Here, team effectiveness is defined as (i) the productive output of the work group, (ii) the effectiveness of processes to maintain the team's capability in the future, and (iii) the satisfaction of group member's personal needs [25, p. 323]. With our study on team productivity, we focus on the first aspect. Specifically, we assess the input-output relation considering the size of the code changes made by an OSS development team as a function of the team's size.

To operationalise team productivity, we need to define measures that allow us to capture the size of a change in a project's codebase. For this, many different measures have been proposed in the literature. Addressing *construct validity*, we consider eight productivity measures and investigate the extent to which they provide independent information on the construct of productivity. We further carefully investigate and assess how these measures interrelate.

We categorise our eight productivity measures as commit- or code-based measures [26]. Commit-based metrics rely solely on the size of the changes within a repository, e.g., the number of commits, the number of changed lines, or the number of modified characters. Commit-based productivity measures require low computational effort and are independent of the programming language used in a repository. However, by not assessing the content of a repository's code, they do not allow us to distinguish, e.g., between lines of code or comments that are added. Therefore, we also consider code-based measures that take these aspects into account. These include measures based on the number of modified lines of code (NLOC), the number of code tokens or functions, changes in McCabe's cyclomatic complexity [27], or the Halstead effort [28]. We provide a complete overview of the productivity measures considered in our study in Table 2. We compute productivity for each time window and normalise the productivity by the respective team size (TS).

To illustrate how different productivity measures can introduce the challenge of construct validity, consider the exemplary Python code shown in Figure 2. We start with version one of a file that contains three lines of text with a total of 53 characters (whitespaces included). Creating this file from scratch would require three *line modification events*, i.e. three line-additions where lines 1 and 3 contain actual code, and line 2 contains a comment. Therefore, the number of line modification events is three, while the lines of code (NLOC) is two. In the example, we have highlighted all code tokens. To compute the Halstead effort, we need to distinguish between tokens that are operands and operators. These are highlighted in blue and red, respectively, whereas all other tokens are printed in purple. In total, we have 12 tokens. With `a` and `b`, the code contains $\eta_2 = 2$ distinct operands that appear a total of $N_2 = 4$ times. We further have $N_1 = 6$ operators that are all different from each other (i.e. $\eta_1 = 6$). The Halstead effort to create this file is thus defined as $E = (N_1 + N_2) \cdot \log_2(\eta_1 + \eta_2) \cdot \frac{\eta_1}{2} \cdot \frac{N_2}{\eta_2} = 180$. Finally, the file only has one function without any branches resulting in Functions = CycC = 1.

With the first modification, we add a second function implementing the multiplication of two values. The bottom-left green box in Figure 2 shows the productivity of this change. The code-based productivity measures are computed as the productivity difference to create the two consecutive versions of the file. For the commit-based measures, the contents of the two files are compared directly. With the second modification, we merge the two functions into one. Despite the increase in characters, the token and function counts and the Halstead effort of version three are lower than those of version two. If measures such as the number of functions or the cyclomatic complexity were to only increase, this would make the code difficult to maintain and prone to bugs [29, 30]. Therefore, we also consider contributions reducing code complexity, e.g., by consolidating functions or refactoring code, as productive. We achieve this by computing the productivity of a modification as the absolute value of the productivity values to create the versions before and after the observed change.

This simple example shows that our eight productivity measures can yield considerably different results. This prompts the question of which measure we should use as a target variable that we seek to explain through the set of features identified above.

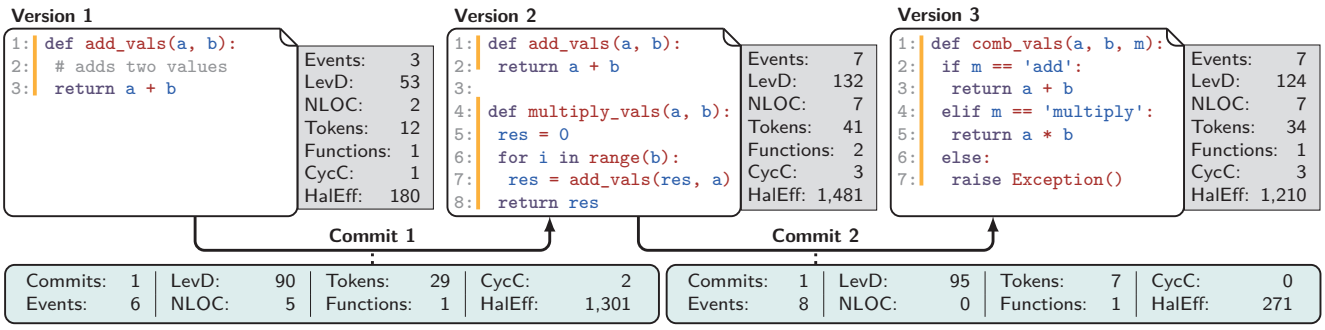

Figure 2: Productivity measures applied to three consecutive versions of an exemplary Python file. In the grey boxes, we report the productivity associated with creating each version of the file from scratch. In the green boxes, we show the productivity related to the changes observed between the versions. We assume that each new version was created in a single commit. All tokens are highlighted. Tokens that are operands and operators are printed in blue and red, respectively. All other tokens are printed in purple. Individual functions are indicated by yellow bars. For the computations in the example, we assume $\Delta t = \mathbf{TS} = 1$.

We address this question in an exploratory study analysing the 201 OSS repositories in our corpus. For this, we split time-series data into non-overlapping 42-week time windows. We then compute our productivity measures and drop all time windows in which a team was inactive, i.e., for which we observe a productivity of zero, yielding a total of 1,188 observations. We find that the distributions of all productivity measures are highly skewed. Therefore, we log-transform all skewed measures such that the resulting distributions resemble a normal distribution.

Figure 3a shows the Pearson correlation between all productivity measures. We find values larger than 0.9 between all productivity measures except for the number of commits and Halstead effort. This suggests that the change in both characters and tokens is similar to the change in lines. We further find that the number of functions and cyclomatic complexity are positively correlated, both changing with the number of lines. With values between 0.7 and 0.8, correlations are considerably smaller for the number of commits and Halstead effort. This indicates that commits differ considerably in terms of their size, i.e., with regard to the number of characters, lines, tokens or functions modified with the commit. Halstead effort is unique among the considered productivity measures as, next to the total amount of code, it also considers the size of the vocabulary used. Therefore, slower vocabulary growth compared to the total amount of code could explain the observed smaller correlation with other measures.

In conclusion, all considered productivity measures have different motivations. Some analyse the source code at various levels of detail while others aggregate information at the level of lines or commits. Despite those differences and the strong differences shown in the example in Figure 2, in our corpus of projects, and when computing average developer productivities across teams, all productivity measures are highly correlated.

3.3 Collaboration networks of OSS teams

Addressing the challenges of *omitted-variable bias*, we explore how the collaboration structure of teams might modulate team productivity. In this way, we capture team characteristics beyond mere team size, highlighting additional variables that we need to control for when studying Brooks' law in rich data. The inclusion of additional measures capturing collaboration structure was motivated by [3], which found that a team's network structure affects the slope of the relation in their data. Therefore, for this work, we consider network-based measures that fit this prior work. In addition, we also include software-engineering-specific measures.

Specifically, to capture characteristics of different aspects of the collaboration structures of development teams, we use measures that we compute on the co-editing network constructed for our non-overlapping 42-week time windows. In these co-editing networks, nodes represent different developers and edges $A \rightarrow B$ represent events where developer B modifies a line of code last edited by A. The direction of the edge indicates the change of line ownership from A to B. Multiple co-editing events between two developers are represented as multi-edges between nodes. Developers editing their own code are captured as self-loops. In the following, we present eight measures that can be used as control variables to explain the relation between team productivity and team size. For formal definitions, we refer to [31].

Number of nodes (N). The number of nodes in the co-editing network counts all developers that actively edited code or whose code was edited. The number of nodes is always greater than or equal to the team size.

Number of edges (Edges). The number of edges counts the co-editing events within a time window.

Density (Dens). The density captures the proportion of potential edges present in a network. We compute the density based on the flattened network, in which multi-edges between two nodes are substituted by a single edge.

Diameter (Diam). The network's diameter is given by the length of the longest shortest path between any pair of nodes.

Clustering Coefficient (ClustC). A node's local clustering coefficient is computed as the fraction of pairs of neighbours that are

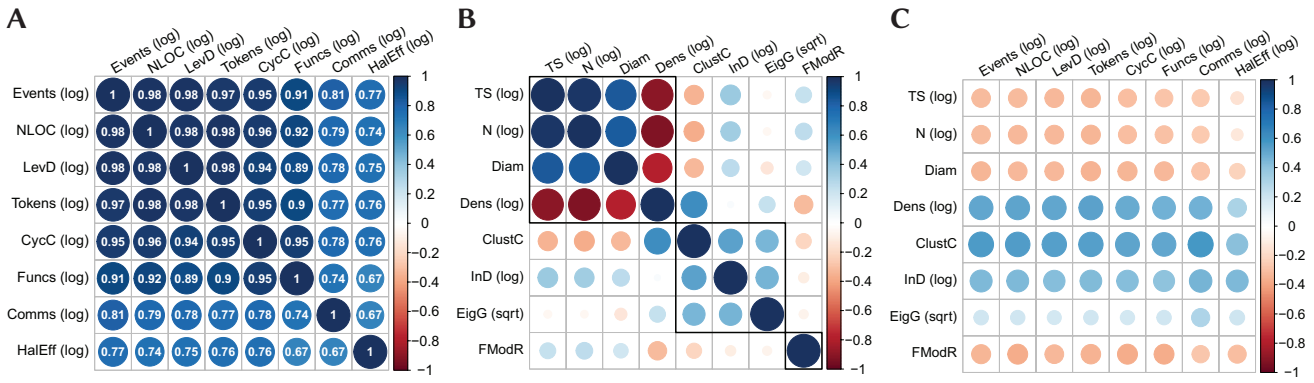

Figure 3: Results of exploratory study on team productivity and collaboration structure. a) Pearson correlation between the transformed productivity measures. b) Pearson correlation between the transformed network measures. Clusters between the measures are marked. c) Cross-correlation (Pearson) between the transformed network and productivity measures.

connected by an edge. The global clustering coefficient is obtained as the average local clustering coefficient across all nodes in the network. Networks with small diameter and large clustering coefficient exhibit the so-called small-world property. Links that connect different clusters in a network lead to low diameters, even for networks with many nodes. The small-world property is directly related to navigability, knowledge transfer and social capital within social networks [32, 33].

Mean Indegree (InD). In the flattened co-editing network, the indegree of a node i indicates the number of developers whose code has been edited by i.

Mean Foreign Modification Ratio (FModR). A recent work shows that the productivity of developers is significantly reduced if they edit code owned by other developers compared to editing their own code [4]. We account for this using the foreign modification ratio, which we compute as the fraction of all co-editing events where the developer edits code owned by another developer. We obtain the number of all co-edits of a developer i as i's indegree, and the number of co-edits where i edit foreign code as the count of all edges to i that are not self-loops. The mean foreign modification ratio of the team is obtained as the mean foreign modification ratio of all team members.

Eigengap (EigG). Finally, the eigengap, also referred to as spectral gap, of a network captures the efficiency of dynamical processes on the network. Networks with larger eigengaps support fast spreading, diffusion and synchronisation, which can be interpreted as a proxy for the efficiency of information exchange and consensus schemes. We compute the eigengap for the largest connected component of the network.

The definitions above enable us to capture the collaboration structure of software development teams in a multi-dimensional feature space. Similar to the productivity measures, we find that the distributions of some network measures are highly skewed. Therefore, we again apply logarithmic or square-root transformations. We report the applied transformation for all measures throughout the remainder of this manuscript.

We next aim to select a minimum set of features that capture independent dimensions of collaboration networks. For this, we study pair-wise correlations between all features, identify clusters of highly correlated features, and select one representative feature per cluster. While we could instead use dimensionality reduction techniques like principal component analysis, our approach provides the advantage that it allows us to analyse interpretable network features rather than principal components.

Figure 3b shows the Pearson correlation between all pairs of network measures. A first visible cluster in the upper-left quadrant contains team size, the number of nodes, and the network diameter, which all show a strong positive correlation. In addition, network density is strongly negatively correlated with all three. Thus, the relative number of co-editing interactions goes down for larger teams, leading to the distance between the two furthest team members in the co-editing network to increase. The second cluster contains clustering coefficient, mean indegree, and eigengap, which are all positively correlated. Thus, in teams where everyone interacts with many different team members we obtain a network structure in which information can spread more quickly throughout the team. The third cluster contains only the mean foreign modification ratio, which quantifies how much other developers' code is edited within the team. For all downstream analyses, we select team size (TS), mean indegree (InD) and the foreign modification ratio (FModR) from the set of network measures, i.e., we use one measure from each cluster.

We finally consider cross-correlations between the network and productivity measures shown in Figure 3c. The results of this analysis confirm that all productivity measures exhibit very similar correlations to the network metrics.

Overall, with the results from our correlation studies, we confirm and extend the prior findings on the relations between social network measures for OSS projects [34, 35] and the relations between classical source code metrics [36–39] for a broader set of measures and in our novel and significantly more extensive corpus of projects designed to study the productivity of OSS development teams.

4 TESTING BROOKS' LAW IN MASSIVE GITHUB DATA

We now address the ongoing scientific discourse on the relationship between team size and productivity [7, 8, 40]. For this, we apply linear and polynomial regression models to different productivity target variables, where we additionally use the network metrics identified in section Section 3.3 as control variables.

Figure 4a shows the productivity per team member as a function of team size, exemplified for a productivity measure based on cyclomatic complexity. We fit a linear (CycC ~ TS) and a polynomial model with maximum degree two (CycC ~ TS + TS^2) to our data. The linear model enables us to infer a possible relationship between team size and productivity. The model CycC ~ TS + TS^2 is the basis to test for the existence of an optimal team size, which is captured by the existence of a global maximum of the quadratic function.

The linear model yields a significant negative relationship between TS and productivity, as reported in Table 3a. Similarly, as shown in Table 3b, we find a significant negative coefficient for TS^2 for the quadratic model. This means that, on average, individual productivity decreases with team size.

That said, all coefficients for TS in the quadratic models are positive, and the coefficients for CycC and HalEff are also significant. This means that these models can be represented as inverted parabolas, e.g., as shown by the red curve for CycC in Figure 4a. The maxima of the parabolas for CycC and HalEff provide some evidence for an optimal team size of 7 or 19 team members, respectively, which is roughly in line with the optimal team size of 9 suggested by [41]. However, we argue that the key insight of this result is not the *exact* team size for which the maximum is reached, which is likely an artefact of the simplified model used for our analysis. Instead, the key insight is the possible increase in individual productivity for very small teams—compared to the analysed range from 2 to 1,711 members—with decreases thereafter. Due to the large amount of remaining variance and the small slope of the parabola around the maximum, interpreting a specific number as optimal team size is likely to be misleading. We further note that for the regression analyses using the remaining productivity measures as target variables, the coefficient for team size is insignificant. Hence, despite the coefficients being consistently positive, these models do not provide sufficient statistical evidence to conclude an increase in individual productivity even in small teams, resulting in an overall negative relation between productivity and team size.

Importantly, the models considered so far only partially explain the variance in the relationship between productivity and team size. This can be seen from the low R^2 values of less than 15% in Table 3a and b. Moreover, Figure 4a shows that, especially for small teams, the productivity of team members varies over four decades. This suggests that additional aspects other than team size have a substantial influence on the productivity of developers. We thus consider regression models that additionally control for the network features identified in Section 3.3. Specifically, we add the mean number of interaction partners per team member (InD) and the average amount of edited foreign code (FModR) as additional variables in our regression models.

As shown by the R^2 values in Table 3c, the regression models that include InD and FModR explain close to half of the variance

Table 3: Regression models relating team size (TS) to five productivity measures. The results are based on 1,188 observations. All productivity measures are log-transformed. All coefficients are Bonferroni-corrected for multiple hypotheses testing.

a) Linear relationship

	Comms	Events	LevD	CycC	NLOC	Tokens	Funcs	HalEff
(IC)	3.38^{***}	7.75^{***}	11.24^{***}	5.22^{***}	6.99^{***}	9.02^{***}	4.22^{***}	15.82^{***}
	(0.08)	(0.10)	(0.10)	(0.11)	(0.10)	(0.11)	(0.11)	(0.16)
TS (log)	-0.20^{***}	-0.31^{***}	-0.33^{***}	-0.31^{***}	-0.31^{***}	-0.34^{***}	-0.29^{***}	-0.22^{***}
	(0.02)	(0.03)	(0.03)	(0.03)	(0.03)	(0.03)	(0.03)	(0.04)
R^2	0.07	0.10	0.11	0.09	0.10	0.11	0.08	0.02
Adj. R^2	0.07	0.10	0.11	0.09	0.10	0.11	0.08	0.02

b) Quadratic relationship

	Comms	Events	LevD	CycC	NLOC	Tokens	Funcs	HalEff
(IC)	3.02^{***}	7.06^{***}	10.55^{***}	4.18^{***}	6.23^{***}	8.11^{***}	3.31^{***}	14.10^{***}
	(0.17)	(0.22)	(0.22)	(0.23)	(0.21)	(0.22)	(0.23)	(0.34)
TS (log)	0.02	0.12	0.10	0.35^{*}	0.17	0.24	0.28	0.86^{***}
	(0.10)	(0.12)	(0.13)	(0.13)	(0.12)	(0.13)	(0.13)	(0.19)
TS^2 (log)	-0.03^{*}	-0.06^{***}	-0.06^{***}	-0.09^{***}	-0.06^{***}	-0.08^{***}	-0.08^{***}	-0.15^{***}
	(0.01)	(0.02)	(0.02)	(0.02)	(0.02)	(0.02)	(0.02)	(0.03)
R^2	0.08	0.11	0.12	0.11	0.11	0.13	0.10	0.05
Adj. R^2	0.07	0.11	0.12	0.11	0.11	0.12	0.10	0.05

c) Linear relationship controlling for network properties

	Comms	Events	LevD	CycC	NLOC	Tokens	Funcs	HalEff
(IC)	3.18^{***}	7.62^{***}	11.10^{***}	5.22^{***}	6.95^{***}	8.93^{***}	4.25^{***}	15.78^{***}
	(0.08)	(0.09)	(0.10)	(0.10)	(0.09)	(0.10)	(0.10)	(0.16)
TS (log)	-0.36^{***}	-0.49^{***}	-0.51^{***}	-0.48^{***}	-0.48^{***}	-0.52^{***}	-0.45^{***}	-0.45^{***}
	(0.02)	(0.02)	(0.02)	(0.02)	(0.02)	(0.02)	(0.03)	(0.04)
InD (log)	1.16^{***}	1.45^{***}	1.44^{***}	1.46^{***}	1.42^{***}	1.49^{***}	1.35^{***}	1.90^{***}
	(0.04)	(0.05)	(0.06)	(0.06)	(0.05)	(0.06)	(0.06)	(0.09)
FModR	-1.00^{***}	-1.90^{***}	-1.83^{***}	-2.65^{***}	-2.33^{***}	-2.19^{***}	-2.65^{***}	-3.26^{***}
	(0.22)	(0.27)	(0.28)	(0.30)	(0.27)	(0.28)	(0.30)	(0.46)
R^2	0.45	0.49	0.47	0.46	0.49	0.49	0.42	0.33
Adj. R^2	0.45	0.49	0.46	0.46	0.49	0.49	0.42	0.33

d) Quadratic relationship controlling for network properties

	Comms	Events	LevD	CycC	NLOC	Tokens	Funcs	HalEff
(IC)	3.53^{***}	7.83^{***}	11.30^{***}	5.10^{***}	7.08^{***}	8.95^{***}	4.22^{***}	15.25^{***}
	(0.14)	(0.17)	(0.18)	(0.19)	(0.17)	(0.18)	(0.19)	(0.30)
TS (log)	-0.58^{***}	-0.63^{***}	-0.64^{***}	-0.41^{***}	-0.56^{***}	-0.53^{***}	-0.42^{***}	-0.10
	(0.08)	(0.10)	(0.10)	(0.11)	(0.10)	(0.10)	(0.11)	(0.17)
TS^2 (log)	0.03^{*}	0.02	0.02	-0.01	0.01	0.00	-0.00	-0.05
	(0.01)	(0.01)	(0.01)	(0.01)	(0.01)	(0.01)	(0.01)	(0.02)
InD (log)	1.18^{***}	1.46^{***}	1.45^{***}	1.45^{***}	1.43^{***}	1.49^{***}	1.35^{***}	1.87^{***}
	(0.04)	(0.05)	(0.06)	(0.06)	(0.05)	(0.06)	(0.06)	(0.09)
FModR	-1.04^{***}	-1.93^{***}	-1.85^{***}	-2.64^{***}	-2.34^{***}	-2.19^{***}	-2.65^{***}	-3.20^{***}
	(0.22)	(0.27)	(0.28)	(0.30)	(0.27)	(0.28)	(0.30)	(0.46)
R^2	0.45	0.49	0.47	0.46	0.49	0.49	0.42	0.34
Adj. R^2	0.45	0.49	0.46	0.46	0.49	0.49	0.42	0.33

e) Linear relationship with controls and interaction effects

	Comms	Events	LevD	CycC	NLOC	Tokens	Funcs	HalEff
(IC)	2.72^{***}	7.39^{***}	10.85^{***}	5.01^{***}	6.68^{***}	8.68^{***}	4.06^{***}	15.91^{***}
	(0.11)	(0.14)	(0.14)	(0.15)	(0.13)	(0.14)	(0.15)	(0.23)
TS (log)	-0.22^{***}	-0.43^{***}	-0.44^{***}	-0.42^{***}	-0.40^{***}	-0.45^{***}	-0.39^{***}	-0.48^{***}
	(0.03)	(0.04)	(0.04)	(0.04)	(0.04)	(0.04)	(0.04)	(0.06)
InD (log)	1.83^{***}	1.78^{***}	1.80^{***}	1.76^{***}	1.80^{***}	1.86^{***}	1.62^{***}	1.72^{***}
	(0.12)	(0.15)	(0.16)	(0.17)	(0.15)	(0.16)	(0.17)	(0.26)
TS×InD	-0.18^{***}	-0.09	-0.10	-0.08	-0.10^{*}	-0.10	-0.07	0.05
	(0.03)	(0.04)	(0.04)	(0.04)	(0.04)	(0.04)	(0.04)	(0.07)
FModR	-0.92^{***}	-1.87^{***}	-1.79^{***}	-2.62^{***}	-2.29^{***}	-2.15^{***}	-2.62^{***}	-3.28^{***}
	(0.22)	(0.27)	(0.28)	(0.30)	(0.27)	(0.28)	(0.30)	(0.46)
R^2	0.47	0.49	0.47	0.46	0.50	0.49	0.43	0.34
Adj. R^2	0.46	0.49	0.47	0.46	0.49	0.49	0.42	0.33

$^{***}p < 0.001$; $^{**}p < 0.01$; $^{*}p < 0.05$

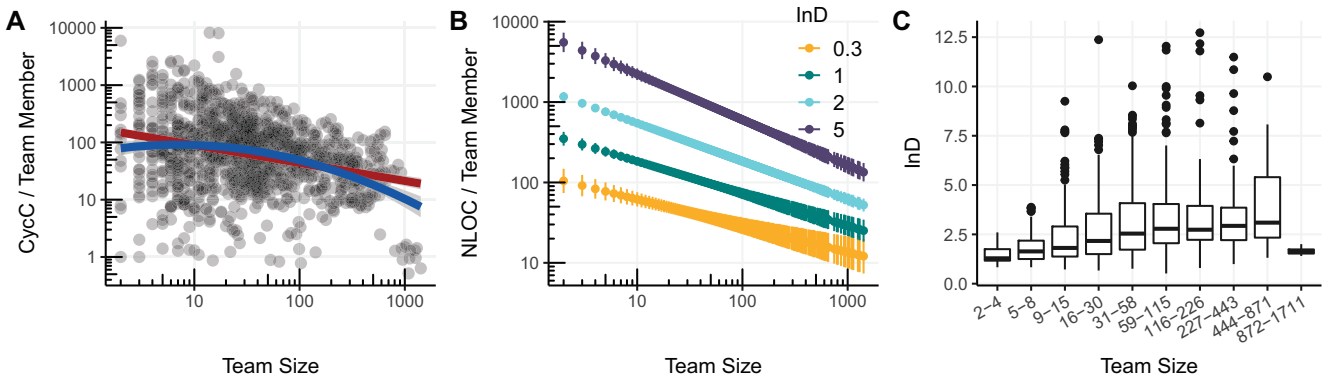

Figure 4: a) Productivity (CycC) per team member as a function of team size. A linear ■ and quadratic model ■ have been fitted to the data. b) Marginal effect of the mean indegree on the relationship between team size and productivity (NLOC). c) Increase of mean indegree (InD) with team size.

in the productivity observations. We further again find a negative relationship between team size and productivity for all five operationalisations. Assuming constant InD and FModR, the regression results suggest that by doubling the size of a development team, we reduce the average productivity of team members by between 22% and 30%. Notably, a higher mean indegree is accompanied by higher productivity. Moreover, the foreign modification ratio has a strong negative relationship with productivity.

Adding a quadratic term (TS^2) to the model, i.e.,

$$\text{PROD} \sim \text{TS} + \text{TS}^2 + \text{InD} + \text{FModR}, \tag{1}$$

we find that contrary to before, the coefficient of team size remains negative and significant while team size squared is insignificant (see Table 3d). Thus, the non-linear relationship between productivity and team size found in Table 3b does not persist when accounting for the mean indegree and foreign modification ratio.

In conclusion, in a large-scale study using 201 collaborative *GitHub* projects sampled in a systematic and unbiased fashion across different strata of team sizes, we confirm the negative relationship between team size and productivity found by prior studies. This negative relationship is robust against the choice of productivity measure and persists when controlling for the team's collaboration structure. Only considering team size as a predictor, our data provide some evidence for an optimal team size of 7 or 19 members for two of the eight operationalisations of productivity. However, for six out of eight productivity measures, the optimal productivity per team member is reached for a team size of one. This finding is further supported by the fact that the squared team size has no significant relationship when controlling for other network measures.

As mentioned above, our results suggest that the mean indegree of developers is positively related to productivity, while team size is negatively related to productivity. Importantly, the regression models considered so far assume that the effect of the team size on productivity is independent of the mean indegree and vice-versa, i.e., their effect is purely additive. However, it is reasonable to assume that the productivity of developers in a team with dense collaboration structures is more strongly affected by team size, compared to a team where each developer collaborates, on average, with few other team members. This motivates a last experiment, where we include an interaction term that captures the combined effect of team size and mean indegree as an additional variable:

$$\text{PROD} \sim \text{TS} + \text{InD} + \text{TS} \times \text{InD} + \text{FModR} \tag{2}$$

The results in Table 3e suggest a negative coefficient for this interaction term. However, the effect is only significant for Comms and NLOC. The analysis of the marginal effect of the mean indegree on the relationship between team size and productivity is shown in Figure 4b. In line with the positive coefficient of the mean indegree, we find that developers in teams with larger mean indegree tend to be more productive on average, i.e. lines corresponding to larger mean indegrees tend to have larger intercepts (but negative slopes). Moreover, as shown by the negative coefficient of the interaction term, the negative effect of team size on productivity grows with the mean indegree, i.e. lines corresponding to larger mean indegrees tend to have steeper negative slopes. The whisker plot in Figure 4c further reveals a positive relationship between the size of a team (x-axis) and the mean indegree of developers (y-axis), i.e. developers in larger teams tend to edit code of a larger number of other developers. This positive relationship specifically holds for smaller team sizes, while the mean indegree in larger teams with more than approximately 50 developers is similar. Importantly, the fact that (i) developers in larger teams tend to have a higher indegree and (ii) developers in teams with higher indegree tend to be more productive does *not* imply that developers in larger teams are, on average, more productive. This is confirmed by the negative coefficients of team size in all our regression models as well as the clear negative marginal effects shown in Figure 4b.

We note that the positive relationship between team size and the mean indegree could explain the positive coefficient for TS in Table 3b that suggests the existence of an optimal team size. We further conjecture that this non-trivial finding could be a reason why empirical studies that do not account for the distribution of team sizes in *GitHub* repositories, and thus inadvertently focus on projects with small team size, erroneously find a positive relationship between team size and productivity. Specifically, projects

with small team sizes tend to have a small mean indegree, which corresponds to a line with smaller intercept (and negative slope) in Table 3b. Conversely, teams with larger team sizes tend to have a larger mean indegree, which corresponds to a line with larger intercept (and negative slope) in Table 3b. The failure to control for this effect can lead to a reversal of the relationship between productivity and team size, i.e., a wrong *positive* slope for the effect is erroneously found. This can be viewed as a specific instance of Simpson's paradox, where the aggregate effect in a sample that combines data from different "groups" of projects—i.e., teams with different mean indegrees—can be positive, even though a negative relationship holds for each group separately.

5 LIMITATIONS AND THREATS TO VALIDITY

A first threat to the validity of our results could be the operationalisation of productivity. To guard against this, we have studied eight different measures that capture different notions of productivity. A common issue of productivity measures that are based on commit log data is that they do not account for the structure of contributed code. To guard against this issue and appropriately value commits that decrease the complexity of code and thus make it more maintainable, we consider measures that account for tokens, functions, and control structures.

A second aspect that could potentially influence our results is the method to assess the size of a software team. We compute the team size at a given time t by counting all developers who have made a commit up to 42 weeks before t. This approach to infer the team size is necessary since there is no formalised notion of team size in *GitHub*. The specific choice of this time window is based on the inter-commit time distribution for *GitHub* projects found in [3]. We have tested the robustness of the results by choosing a different window size of two years. Due to the computational effort that is due to the recalculation of all network metrics, a comprehensive study of different window sizes was beyond the scope of our study but could be an interesting question for future work.

In Section 2.1, we identified project selection and data preparation as a major threat to the validity. We thus spent considerable effort to develop a general project selection pipeline as well as Open Source software tools to infer collaboration networks from commit data. Despite these efforts, there may be remaining issues, such as the possibility to manually modify the history of a *git* repository, which we can neither detect nor account for. Due to data issues related to some merge commits, we were further not able to process all commits of the `Linux Kernel` project[4]. We therefore excluded this project from our analysis.

Addressing the issue of omitted variables, our regression models explain roughly 50% of the variance in the relationship between team size and productivity, which considerably improves the variance explained by prior studies. Nevertheless, there is additional variance in the relationship that we cannot explain. This could either be due to the stochastic nature of the underlying process or the existence of additional variables that are not included in our models. To address this issue, future studies could additionally study data from issue trackers and mailing lists [42–44].

[4]https://github.com/torvalds/linux

In our study, we considered 201 OSS projects sampled to represent the entire spectrum of team sizes present on *GitHub*. While our findings are representative for collaborative software development on *GitHub*, it is unclear whether they can be generalised to proprietary software projects or other Open Source collaboration platforms. Platforms like SourceForge or Bitbucket have different characteristics [45–47] that will require modifications to our selection pipeline, which is an interesting issue for future work.

6 RELATED WORK

Our work touches on issues that have been studied in empirical software engineering, computational social science, network science, and organisational theory. Namely, how we can use repository log data to quantify productivity in software development, how the size of teams influences the productivity of its members, and how network models can be used to study social aspects in collaborative software projects. At a meta-level, our study further addresses common challenges and pitfalls in the analysis of big repository data, some of which have been previously highlighted in [10–12, 45].

A large body of works has investigated methods to measure the *productivity* of developers based on repository data [48, 49]. *Commit-based productivity measures* calculate productivity based on the number of commits [7, 50] or pushes [8]. While this approach does not require a detailed analysis of the committed source code, it has the problem that the amount of code changed with each commit can vary significantly both within and between projects [23]. [26] found that productivity rankings based on commit-based measures only show low correlations with rankings obtained from team leaders. Therefore, more fine-granular measures such as the number of modified lines or the number of modified characters [3] have been proposed. *Code-based productivity measures* aim to overcome this limitation by analysing the code contained in a commit. Commonly used metric include the number of code lines [5, 51–53], function points [54, 55], or tokens [56] changed per time interval.

A large body of works in organisational theory, computational social science and empirical software engineering have studied the question of how the size of a team is related to its performance. In the context of software development, [2] argues that the increased coordination requirements in larger teams lead to a reduction of developer productivity. While this proposed mechanism has been corroborated by quantitative studies [3, 57], other works suggest synergistic effects that lead to an increase in developer productivity as teams grow in size [7, 8]. The combination of these findings indicates that an optimal size for a software development team may exist, which is also discussed in the literature [41, 58]. A report cited by [41] suggests that for proprietary software development projects, an optimal team size with respect to productivity is achieved for nine team members.

Utilising a method to construct a network representation of co-editing relations from the commit-log history of a project, our work finally addresses questions that received attention from the network science and computational social science community. A number of works have investigated how the topology of communication, collaboration, or coordination networks is related to the performance [59–61], resilience [62, 63], or productivity [3] of teams in various

contexts. In the context of research on developer productivity, recent studies found that a densification of co-editing networks due to shared code ownership can explain the decrease in productivity observed for larger teams [3, 4].

7 CONCLUSION

Massive data from software repositories and collaboration tools provide compelling new opportunities to study social aspects in software development. Within this context, the question of how the size and collaboration patterns of software development teams influence the productivity of developers has emerged as an important research question at the intersection of computational social science and empirical software engineering. Recent empirical studies using big data from software repositories have come to contradictory answers to this important research question, even though those studies used similar data sets and empirical methods.

Addressing common challenges and pitfalls in the analysis of big repository data, our work offers a possible explanation for this disagreement between recent works in empirical software engineering. To this end, we provide the, to the best of our knowledge, largest, curated corpus of *GitHub* projects that is specifically tailored to investigate the influence of team size and collaboration patterns on individual and collective productivity. The projects included in this corpus are *systematically* chosen such that we avoid common perils in *GitHub* mining. We use a stratified sampling that supports unbiased analyses of the impact of team size on developer productivity. We systematically compare a set of eight code- and commit-based productivity measures and study which of the measures are likely to be interchangeable and which capture independent dimensions of productivity. Building on a method to construct time-evolving co-editing networks from *git* repositories, we consider eight network metrics that capture different dimensions of the social organisation of software teams. We finally use those methods to study the Ringelmann effect in collaborative software development. Our results highlight a robust negative relationship between team size and developer productivity that can be explained based on the team's collaboration structure. We argue that neglecting the highly skewed distribution of team sizes on *GitHub* can lead to a reversal of the relationship between team size and productivity, thus offering a possible explanation for recent contradictory results.

Apart from this, our work provides several insights that are relevant for the management of software projects: In particular, we find (i) an overall negative relation between individual productivity and team size, and (ii) a non-linear relationship that gives rise to an optimal team size for small teams. These findings can be useful to define advanced cost estimation models that incorporate the found non-linear relationship between team size and productivity, thus providing better estimates for the work force required to develop projects with a known (estimated) size of the code base. Our analysis further highlights additional factors that influence team productivity, such as the amount of foreign code that is edited and the number of interaction partners of developers. This insight not only allows us to further improve cost estimation models, it also points to factors that can possibly be optimised by project maintainers, e.g., by carefully decomposing the code base into modules addressed by different (sub-)teams or by optimising organisational structures of the development team.

In summary, our work contributes to the ongoing discussion on how the size and structure of teams influence productivity. Investigating the cross-correlations of productivity and network metrics in a systematically constructed corpus of software projects, we further contribute a valuable new resource for researchers in empirical software engineering and computational social science. By focusing on generic network measures, we further provide the perspective that our results can generalise beyond empirical software engineering. Highlighting pitfalls in the analysis of big data, our work finally demonstrates that the use of bigger data sets does not automatically lead to more reliable insights.

DATA AVAILABILITY AND REPRODUCIBILITY

To facilitate the reproduction of our results and enable future research based on the extensive data set mined for our study, we have archived both a reproducibility package and our full data sets on zenodo.org[5].

ACKNOWLEDGMENTS

We thank Christian Zingg for contributing to the development of the infrastructure to mine edits and co-edits from software projects on the ETH Zurich scientific compute cluster *Euler*. Ingo Scholtes acknowledges financial support from the Swiss National Science Foundation through grant no. 176938. Christoph Gote and Ingo Scholtes wrote parts of this manuscript on a joint research retreat at *Niederzerfermühle* that was financially supported by the Department of Informatics at University of Zurich and the Chair of Systems Design at ETH Zurich.

REFERENCES

[1] M. Ringelmann, "Recherches sur les moteurs animes: Travail de l'homme," *Annales de l'Institut National Agronomique*, vol. 12, no. 1, pp. 1–40, 1913.
[2] F. P. Brooks, "The mythical man-month," 1975.
[3] I. Scholtes, P. Mavrodiev, and F. Schweitzer, "From Aristotle to Ringelmann: A large-scale analysis of team productivity and coordination in Open Source software projects," *Empirical Software Engineering*, vol. 21, no. 2, pp. 642–683, 2016.
[4] C. Gote, I. Scholtes, and F. Schweitzer, "Analysing time-stamped co-editing networks in software development teams using git2net," *Empirical Software Engineering*, vol. 26, no. 4, pp. 1–41, 2021.
[5] J. D. Blackburn, G. D. Scudder, and L. N. Van Wassenhove, "Improving speed and productivity of software development: A global survey of software developers," *IEEE Transactions on Software Engineering*, vol. 22, no. 12, pp. 875–885, 1996.
[6] K. D. Maxwell, L. Van Wassenhove, and S. Dutta, "Software development productivity of european space, military, and industrial applications," *IEEE Transactions on Software Engineering*, vol. 22, no. 10, pp. 706–718, 1996.
[7] D. Sornette, T. Maillart, and G. Ghezzi, "How much is the whole really more than the sum of its parts? 1 + 1 = 2.5: Superlinear productivity in collective group actions," *Plos one*, vol. 9, no. 8, p. e103023, 2014.
[8] G. Muric, A. Abeliuk, K. Lerman, and E. Ferrara, "Collaboration drives individual productivity," *PACMHCI*, vol. 3, no. CSCW, pp. 74:1–74:24, 2019.
[9] T. Maillart and D. Sornette, "Aristotle vs. Ringelmann: On superlinear production in open source software," *Physica A: Statistical Mechanics and its Applications*, vol. 523, pp. 964–972, 2019.
[10] C. Bird, P. C. Rigby, E. T. Barr, D. J. Hamilton, D. M. German, and P. Devanbu, "The promises and perils of mining git," in *2009 6th IEEE International Working Conference on Mining Software Repositories*. IEEE, 2009, pp. 1–10.
[11] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian, "The promises and perils of mining GitHub," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, 2014, pp. 92–101.

[5]Reproducibility package: https://doi.org/10.5281/zenodo.5294015
Data sets: https://doi.org/10.5281/zenodo.5294964

[12] ——, "An in-depth study of the promises and perils of mining GitHub," *Empirical Software Engineering*, vol. 21, no. 5, pp. 2035–2071, 2016.

[13] C. Gote and C. Zingg, "gambit – An Open Source name disambiguation tool for version control systems," in *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*, 2021, pp. 80–84.

[14] D. T. Campbell and J. C. Stanley, *Experimental and quasi-experimental designs for research*. Ravenio Books, 2015.

[15] G. Gousios, "The GHTorrent dataset and tool suite," in *Proceedings of the 10th Working Conference on Mining Software Repositories*, ser. MSR '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 233–236.

[16] C. Gote, I. Scholtes, and F. Schweitzer, "git2net – Mining time-stamped co-editing networks from large git repositories," in *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. IEEE, 2019, pp. 433–444.

[17] J. Warner, "Thank you for 100 million repositories," https://github.blog/2018-11-08-100m-repos/, 2021. [Online]. Available: https://github.blog/2018-11-08-100m-repos/

[18] J. Seawright and J. Gerring, "Case selection techniques in case study research: A menu of qualitative and quantitative options," *Political Research Quarterly*, vol. 61, no. 2, pp. 294–308, 2008.

[19] T. Yin, "Lizard," https://github.com/terryyin/lizard, 2020. [Online]. Available: https://github.com/terryyin/lizard

[20] F. Beuke, "GitHut 2.0," https://madnight.github.io/githut/#/pull_requests/2020/2, 2020. [Online]. Available: https://madnight.github.io/githut/#/pull_requests/2020/2

[21] D. Spinellis, Z. Kotti, and A. Mockus, "A dataset for github repository deduplication," in *Proceedings of the 17th international conference on mining software repositories*, 2020, pp. 523–527.

[22] K. Weihmann, "Multimetric," https://github.com/priv-kweihmann/multimetric, 2020. [Online]. Available: https://github.com/priv-kweihmann/multimetric

[23] GitHub Inc., "The 2020 state of the Octoverse – Finding balance between work and play," https://octoverse.github.com/static/github-octoverse-2020-productivity-report.pdf, 2020. [Online]. Available: https://octoverse.github.com/static/github-octoverse-2020-productivity-report.pdf

[24] B. Vasilescu, D. Posnett, B. Ray, M. G. van den Brand, A. Serebrenik, P. Devanbu, and V. Filkov, "Gender and tenure diversity in github teams," in *Proceedings of the 33rd annual ACM conference on human factors in computing systems*, 2015, pp. 3789–3798.

[25] J. R. Hackman, "The design of work teams," in *Handbook of organizational behaviour*, J. W. Lorsch, Ed. Englewood Cliffs, N.J.: Prentice-Hall, 1987, ch. 20, pp. 315–342.

[26] E. Oliveira, E. Fernandes, I. Steinmacher, M. Cristo, T. Conte, and A. Garcia, "Code and commit metrics of developer productivity: A study on team leaders perceptions," *Empirical Software Engineering*, vol. 25, no. 4, pp. 2519–2549, 2020.

[27] T. J. McCabe, "A complexity measure," *IEEE Transactions on Software Engineering*, vol. 4, pp. 308–320, 1976.

[28] M. H. Halstead *et al.*, *Elements of software science*. Elsevier New York, 1977, vol. 7.

[29] I. Stamelos, L. Angelis, A. Oikonomou, and G. L. Bleris, "Code quality analysis in Open Source software development," *Information Systems Journal*, vol. 12, no. 1, pp. 43–60, 2002.

[30] R. Baggen, J. P. Correia, K. Schill, and J. Visser, "Standardized code quality benchmarking for improving software maintainability," *Software Quality Journal*, vol. 20, no. 2, pp. 287–307, 2012.

[31] M. Newman, *Networks*. Oxford University Press, 2018.

[32] S. Milgram, "The small world problem," *Psychology today*, vol. 2, no. 1, pp. 60–67, 1967.

[33] M. S. Granovetter, "The strength of weak ties," *American Journal of Sociology*, vol. 78, no. 6, pp. 1360–1380, 1973.

[34] M. Y. Allaho and W.-C. Lee, "Analyzing the social networks of contributors in open source software community," in *Applications of Social Media and Social Network Analysis*. Springer, 2015, pp. 57–75.

[35] J. Teixeira, G. Robles, and J. M. González-Barahona, "Lessons learned from applying social network analysis on an industrial free/libre/open source software ecosystem," *Journal of Internet Services and Applications*, vol. 6, no. 1, pp. 1–27, 2015.

[36] S. Henry, D. Kafura, and K. Harris, "On the relationships among three software metrics," *ACM SIGMETRICS Performance Evaluation Review*, vol. 10, no. 1, pp. 81–88, 1981.

[37] S. N. Woodfield, V. Y. Shen, and H. E. Dunsmore, "A study of several metrics for programming effort," *Journal of Systems and Software*, vol. 2, no. 2, pp. 97–103, 1981.

[38] M. A. A. Mamun, C. Berger, and J. Hansson, "Correlations of software code metrics: an empirical study," in *Proceedings of the 27th international workshop on software measurement and 12th international conference on software process and product measurement*, 2017, pp. 255–266.

[39] D. Landman, A. Serebrenik, and J. Vinju, "Empirical analysis of the relationship between cc and sloc in a large corpus of java methods," in *2014 IEEE International Conference on Software Maintenance and Evolution*. IEEE, 2014, pp. 221–230.

[40] I. Scholtes, N. Wider, R. Pfitzner, A. Garas, C. J. Tessone, and F. Schweitzer, "Causality-driven slow-down and speed-up of diffusion in non-Markovian temporal networks," *Nature Communications*, vol. 5, p. 5024, 2014.

[41] D. Rodríguez, M. Sicilia, E. García, and R. Harrison, "Empirical findings on team size and productivity in software development," *Journal of Systems and Software*, vol. 85, no. 3, pp. 562–570, 2012.

[42] A. Bacchelli, M. Lanza, and M. D'Ambros, "Miler: A toolset for exploring email data," in *Proceedings of the 33rd International Conference on Software Engineering*. ACM, 2011, pp. 1025–1027.

[43] C. Bird, A. Gourley, P. Devanbu, M. Gertz, and A. Swaminathan, "Mining email social networks," in *Proceedings of the 2006 International Workshop on Mining Software Repositories*. ACM, 2006, pp. 137–143.

[44] A. Guzzi, A. Bacchelli, M. Lanza, M. Pinzger, and A. v. Deursen, "Communication in Open Source software development mailing lists," in *Proceedings of the 10th Working Conference on Mining Software Repositories*. IEEE Press, 2013, pp. 277–286.

[45] J. Howison and K. Crowston, "The perils and pitfalls of mining SourceForge," in *MSR*. IET, 2004, pp. 7–11.

[46] Y. Ma, C. Bogart, S. Amreen, R. Zaretzki, and A. Mockus, "World of code: An infrastructure for mining the universe of open source VCS data," in *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. IEEE, 2019, pp. 143–154.

[47] T. Xie, S. Thummalapenta, D. Lo, and C. Liu, "Data mining for software engineering," *Computer*, vol. 42, no. 8, 2009.

[48] A. N. Meyer, T. Fritz, G. C. Murphy, and T. Zimmermann, "Software developers' perceptions of productivity," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 2014, pp. 19–29.

[49] G. Sudhakar, A. Farooq, and S. Patnaik, "Measuring productivity of software development teams," *Serbian Journal of Management*, vol. 7, no. 1, pp. 65–75, 2012.

[50] A. Mockus, R. T. Fielding, and J. D. Herbsleb, "Two case studies of Open Source software development: Apache and Mozilla," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 11, no. 3, pp. 309–346, 2002.

[51] P. Devanbu, S. Karstu, W. Melo, and W. Thomas, "Analytical and empirical evaluation of software reuse metrics," in *Proceedings of the 18th International Conference on Software Engineering*. IEEE Computer Society, 1996, pp. 189–199.

[52] H. Hulkko and P. Abrahamsson, "A multiple case study on the impact of pair programming on product quality," in *Proceedings of the 27th International Conference on Software Engineering*. ACM, 2005, pp. 495–504.

[53] V. Nguyen, L. Huang, and B. Boehm, "An analysis of trends in productivity and cost drivers over years," in *Proceedings of the 7th International Conference on Predictive Models in Software Engineering*. ACM, 2011, p. 3.

[54] C. Jones, "Software metrics: Good, bad and missing," *Computer*, vol. 27, no. 9, pp. 98–100, 1994.

[55] S. Wagner and M. Ruhe, "A systematic review of productivity factors in software development," *arXiv preprint arXiv:1801.06475*, 2018.

[56] J. A. Lane and D. Zubrow, "Intergrating measurement with improvement: An action-oriented approach: Experience report," in *Proceedings of the 19th International Conference on Software Engineering*. ACM, 1997, pp. 380–389.

[57] Z. Jiang, P. Naudé, and C. Comstock, "An investigation on the variation of software development productivity," *International Journal of Computer, Information, and Systems Sciences, and Engineering*, vol. 1, no. 2, pp. 72–81, 2007.

[58] M. Heričko, A. Živkovič, and I. Rozman, "An approach to optimizing software development team size," *Information Processing Letters*, vol. 108, no. 3, pp. 101–106, 2008.

[59] L. Wu, "Social network effects on productivity and job security: Evidence from the adoption of a social networking tool," *Information systems research*, vol. 24, no. 1, pp. 30–51, 2013.

[60] H.-L. Yang and J.-H. Tang, "Team structure and team performance in is development: A social network perspective," *Information & management*, vol. 41, no. 3, pp. 335–349, 2004.

[61] R. Reagans and E. W. Zuckerman, "Networks, diversity, and productivity: The social capital of corporate R&D teams," *Organization science*, vol. 12, no. 4, pp. 502–517, 2001.

[62] G. F. Massari, I. Giannoccaro, and G. Carbone, "Team social network structure and resilience: A complex system approach," *IEEE Transactions on Engineering Management*, 2021.

[63] M. S. Zanetti, I. Scholtes, C. J. Tessone, and F. Schweitzer, "The rise and fall of a central contributor: Dynamics of social organization and performance in the Gentoo community," in *CHASE/ICSE '13 Proceedings of the 6th International Workshop on Cooperative and Human Aspects of Software Engineering*, 2013, pp. 49–56.