Revisiting Aristotle vs. Ringelmann: The influence of biases on measuring productivity in Open Source software development

Christian Gut
christian.gut@usp.br
University of São Paulo
São Paulo, Brazil

Alfredo Goldman
gold@ime.usp.br
University of São Paulo
São Paulo, Brazil

ABSTRACT

Aristotle vs. Ringelmann was a discussion between two distinct research teams from the ETH Zürich who argued whether the productivity of Open Source software projects scales sublinear or superlinear with regard to its team size. This discussion evolved around two publications, which apparently used similar techniques by sampling projects on GitHub and running regression analyses to answer the question about superlinearity. Despite the similarity in their research methods, one team around Ingo Scholtes reached the conclusion that projects scale sublinear, while the other team around Didier Sornette ascertained a superlinear relationship between team size and productivity. In subsequent publications, the two authors argue that the opposite conclusions may be attributed to differences in project populations, since 81.7% of Sornette's projects have less than 50 contributors. Scholtes, on the other hand, sampled specifically projects with more than 50 contributors.

This publication compares the research from both authors by replicating their findings, thus allowing for an evaluation of how much project sampling actually accounted for the differences between Scholtes' and Sornette's results. Thereby, the discovery was made that sampling bias only partially explains the discrepancies between the two authors. Further analysis led to the detection of instrumentation biases that drove the regression coefficients in opposite directions. These findings were then consolidated into a quantitative analysis, indicating that instrumentation biases contributed more to the differences between Scholtes' and Sornette's work than the selection bias suggested by both authors.

CCS CONCEPTS

• **Software and its engineering → Open source model**; **Programming teams**; *Software version control*; Maintaining software.

KEYWORDS

Mining Software Repositories, Open Source, Empirical Software Engineering, Software Development Productivity, GitHub, Git, Economies of Scale, Diseconomies of Scale, Replication Study, Sampling Bias, Instrumentation Bias

1 INTRODUCTION

With the ever-growing importance of software in modern society, the question of how to develop software as efficiently as possible has been relevant for decades. A subquestion on that topic, whether adding more developers to Open Source projects will generate economies or diseconomies of scale, had been investigated by two different research teams from the ETH Zürich.

First, Sornette et al. deployed a regression analysis on the commit history of GitHub projects and identified a superlinear relationship between team size and team production [21]. Their metric for production was the number of commits, or more precisely the number of edited files in each commit. This measurement enabled the calculation of a linear regression coefficient, denominated β, between the logarithm of the output of the team and the logarithm of the size of the team. The deployment of a 250-day period for each analysis and 5-day windows for the generation of data points allowed the authors to obtain multiple regression analyzes per project. The thereby identified superlinearities were associated with the heavy-tailed nature of the underlying distribution of contributions. Sornette et al. also calculated the mean values for each project, denominated as *average* β, and reported 104 out of 164 projects as being superlinear using this measure.

Almost 2 years later, Scholtes et al. detected diseconomies of scale in GitHub projects using an apparently similar regression analysis [17]. Their approach to measuring production was more fine-grained by calculating the Levenshtein distance for each file edit. This editing distance was then used to calculate the team's productivity, using 7-day windows for the output and a 295-day windows with 7-day increments to determine the team size. Based on these data points, the researchers conducted several regression analyzes, considering Log-Log and Log-Lin models, and accounting for the influence of one-time contributors. The resulting regression coefficients were denominated as α_3. For the case of the Log-Log regression without filtering for one-time contributors, Scholtes et al. discovered solely sublinear relationships across the 58 projects they examined.

The fact that both research teams reached opposite conclusions led to a follow-up publication that provided additional analytical insights [15]. An important argument of that study was that completely different sets of projects were examined, since 134 out of 164 projects analyzed by Sornette et al. had less than 50 contributors and Scholtes et al. deliberately sampled projects with more than 50 contributors. A study by Gote et al., based on Scholtes' original work, also suggested that the difference may be caused by the selection of projects [8].

This work contributes to the Aristotle vs. Ringelmann discussion by addressing the following research question:

> *What factors explain the different conclusions drawn by Sornette et al. and Scholtes et al.?*

Since the two publications not only varied in their choice of projects, but also implemented comparable yet distinct regression techniques, the above research question can be split into these three more specific subquestions:

(1) *How significant was project selection in creating the differences observed between Sornette et al. and Scholtes et al.?*
(2) *What bias in Sornette's regression method could have favored the identification of a superlinear relationship between team size and productivity?*
(3) *What bias in Scholtes' regression method could have favored the identification of a sublinear relationship between team size and productivity?*

The first question was answered by reproducing both regression methods, so that the method of one author can be applied to the other authors' data set. Such an analysis gave quantitative evidence on how much project selection caused the opposite conclusions drawn by Sornette and Scholtes. A response to the second question led to a closer examination of Sornette's regression method, resulting in the discovery of a potential p-value filter that eliminated all regression coefficients close to zero. Similarly, the third question inspired an exploration into how the overall time frame selection affected Scholtes' regression method, revealing a strong influence of the first couple of days on the regression coefficient.

The remainder of this paper is structured as follows. After a short summary of the related work on software development productivity, a comprehensive explanation of the project selection and regression methods used by Sornette and Scholtes is presented. Subsequent sections show how the research methods were reproduced and how replication led to the conclusion that selection bias only partially accounts for the disparities between both authors. Next, quantitative evidence is provided demonstrating how instrumentation biases may have had a significant impact, and a conclusion summarizes how accounting for project selection and instrumentation biases approximates results. The final considerations provide a discussion about the lessons learned, limitations, and future work.

2 RELATED WORK AND STATE-OF-THE-ART

Arguably, the most foundational work on the productivity of software development teams is Brooks' book *The Mythical Man Month* [2], from which the term *Brooks Law* originated. This law states that adding people to a late software project will delay the project even further. Brooks also advocated for surgical teams that consist of a few highly specialized developers making most of the changes to the code, supported by a wide variety of team members who make the work of the few specialists as productive as possible. Another highly referenced classic work on software development productivity is the COCOMO II cost estimation model by Boehm et al. [1]. Although parts of this model might be outdated, the underlying factors like code reuse, the use of tools, or the team size, remain relevant. For the latter, Boehm identified diseconomies of scale.

Recent literature reviews confirm that the impact of team size on productivity in software development continues to be relevant. Chapetta and Travassos conducted a literature review to derive factors that influence developer productivity [3]. At the beginning of their investigation, the authors examined 121 publications related to this topic. Then, they identified relevant factors and mapped their direction and intensity. Chapetta and Travassos also evaluated the belief in these factors by looking at the type and quality of the underlying study. These data points were then combined into a unified *Gain in Belief* metric. The result of this exercise led to the discovery of 16 factors which influence software development productivity, one of which states a slightly adverse effect of team size on developer productivity.

Another review of the literature on software development productivity comes from Duarte, who examined a total of 99 publications [6]. He presented a synopsis of each article, providing valuable information on the potential underlying factors that affect software development productivity. These factors were related to size & structure of the team, knowledge & experience of the developers, development context, process & tools, code structure, and social & emotional factors. The study confirmed a significant negative impact of the size of the project on productivity, while team size exhibited a moderate negative influence.

Contemporary research also conducted quantitative analysis of the productivity of software development. Lavazza et al. is a publication based on the ISBG[1] data set [14]. This set consists of industry data that quantifies the output of the software development process in function points, which includes additional information, such as the programming language used, the industry involved, and whether a data point pertains to a new project or the maintenance of an existing one. The level of data quality in the ISBG data set varies widely, leading to the removal of more than 50% of the data points from the analysis. On the basis of the remaining data points, Lavazza et al. worked on two research questions: *What factors influence the productivity of software development? And what factors influence economies or diseconomies of scale?* In most cases, they were unable to identify economies or diseconomies of scale with statistical confidence. Exceptions were projects written in Java and Visual Basic, as well as certain types of enhancement projects, for which the authors detected economies of scale. Projects written in PL/I were the only exception where diseconomies of scale could be measured.

A different analysis of software development productivity was conducted by Muríc et al., who specifically investigated projects on GitHub with fewer than 20 contributors [16]. Within these limitations, the authors found that Open Source projects scale superlinear. However, this effect started to decrease once the team size exceeded 10 team member. An interesting aspect of their study is the notion of an effective team size n. This measurement is defined by $n = 2^H$, where $H = -\sum_{i=1}^{N} f_i \cdot log_2(f_i)$ and $f_i = w_i/W$, with N being the number of team members, w_i being the work done by the team member i, and W being the sum of the work performed by the entire team. The effective team size reaches its maximum, $n = N$, if all members of the team contribute the same amount of work. Muríc et al. identified a negative correlation between effective team size and productivity, meaning that productivity increases if the work load is primarily handled by a few developers, who incorporate sporadic contributions from the rest of their team members. Interestingly, such a team structure has resemblance to the surgical teams described by Brooks.

Last but not least, there are two articles that can be directly linked to the Aristotle vs. Ringelmann discussion. First, there is a direct comparison of the two publications by Maillart and Sornette [15]. They argue that the opposite conclusions drawn by Sornette et al. and Scholtes et al. may be attributed to variations in the project

[1] International Software Benchmarking Group

population, because the superlinearity found in Sornette et al. holds for no more than 30 to 50 developers, and Scholtes et al. sampled specifically projects with more than 50 developers. Maillart and Sornette also emphasized that their work linked the phenomenon of superlinearity to the statistical distribution of contributions, which showed a heavy-tailed power-law behavior. This behavior can be explained by two possible generating mechanisms, one based on cascading interactions and the other involving large deviations of the underlying process. They further asserted that the link between these generating mechanisms and superlinear production was validated in their work by looking at bursts of activity per contributor. Since the frequency of these bursts of activity decreases with larger projects, and given that Scholtes' data set mainly consists of large projects, the authors suggested that these factors could explain the divergent conclusions drawn by the two research groups. In the end, Maillart and Sornette highlighted that the primary focus of their work was different from Scholtes' research, since they examined bursts of production while Scholtes considered averages. Nevertheless, it should be mentioned that, despite focusing on bursts of production, even the averages reported by Sornette et al. showed superlinearity, which justifies the investigations conducted in this study.

The other publication related to the Aristotle vs. Ringelmann discussion was conducted by Gote et al., which refined Scholtes' initial work by introducing team size stratification [8]. This work also provided new insights into the correlation of output metrics, economies of scale, and the influence of foreign code edits on productivity. Initially, the authors identified a high correlation between multiple output metrics, such as the number of commits, the Levenshtein distance, changes in LOC, or changes in cyclomatic complexity. This implies that the different output metrics may only have a marginal influence on the results when analyzing the productivity of software development teams. Gote's approach to stratify their analysis by team size led to the identification of slight economies of scale for teams with less than 20 contributors. For larger projects, diseconomies of scale prevailed. Based on this finding, they argued that any study favoring small projects in its sampling may mistakenly infer the presence of overall economies of scale. Finally, they identified the ratio of foreign code edits, which is the percentage of modifications to source code that has not been written by the author itself, as a factor that negatively correlates with productivity. This conclusion not only seems logical, but could also provides a rationale for the occurrence of diseconomies of scale, as having a larger team working on a project may lead to an increased likelihood of foreign code edits.

This study adds to the latest research by demonstrating that selection bias alone is not a sufficient explanation for the Aristotle vs. Ringelmann debate. It further illustrates how instrumentation biases appear to be a relevant factor. Identifying these instrumentation biases not only contributes to the specific debate between Sornette and Scholtes but also hints at general factors that must be considered when examining the productivity of Open Source software development.

3 DIFFERENCES IN SORNETTE'S AND SCHOLTES' WORK

In general terms, Sornette et al. and Scholtes et al. used the same regression approach to determine superlinearity:

(1) Select a set of projects
(2) Divide an overall time frame for each project into smaller time windows
(3) For each time window
   (a) Determine the team size
   (b) Determine the output produced by the team
(4) Plot output and team size on a graph
(5) Run a linear regression analysis

Figure 1 is a visualization of this general approach. However, these approaches differ significantly in terms of their implementation specifics.

3.1 Project selection

Both articles seemed to have made a similar choice in terms of data collection: Mine all commits from a set of GitHub projects starting with the very first commit and ending with some cut-off date in the middle of the 2010's.

Nevertheless, a closer examination reveals some fundamental differences in the characteristics of the selected projects. Sornette et al. looked at 164 projects while Scholtes et al. examined only 58 projects. Intriguingly, only 3 projects were analyzed by both authors: Rails, Django, and jQuery. This represents less than 1.4% of all 219 projects. Furthermore, Sornette et al. opted for a random sampling approach, while Scholtes et al. made the deliberate choice to select only projects with more than 50 contributors.

3.2 Regression method

The authors differed not only in the selection of projects, but also in the details of their regression analysis. These differences are illustrated in Figure 2 and will be explained in more detail below.

3.2.1 Definition of time windows. Sornette et al. divided the project first into 250-day periods and then subdivided each 250-day period into 5-day windows. The team output and team size were determined for each 5-day window.

Scholtes et al. divided all data for a project into 7-day windows. These 7-day windows were used to calculate the team output. Remarkably, the team size itself had been determined by using 295-day windows which were incremented by 7-day intervals that match the end date of the team output window for each iteration. This implies that the team size windows overlapped, so that a commit was included multiple times when determining team sizes, but was only counted once when calculating team output.

3.2.2 Measurement of output. Sornette et al. reported commit count as an output measure, but an evaluation of Sornette's Numpy files on PLOS suggested that the authors actually worked with the number of modified files in every commit. This assumption could be confirmed during a subsequent email exchange with Thomas Maillart, one of the coauthors.

Scholtes et al. used the Levenshtein editing distance to determine the output of the team. This distance measure was applied over all

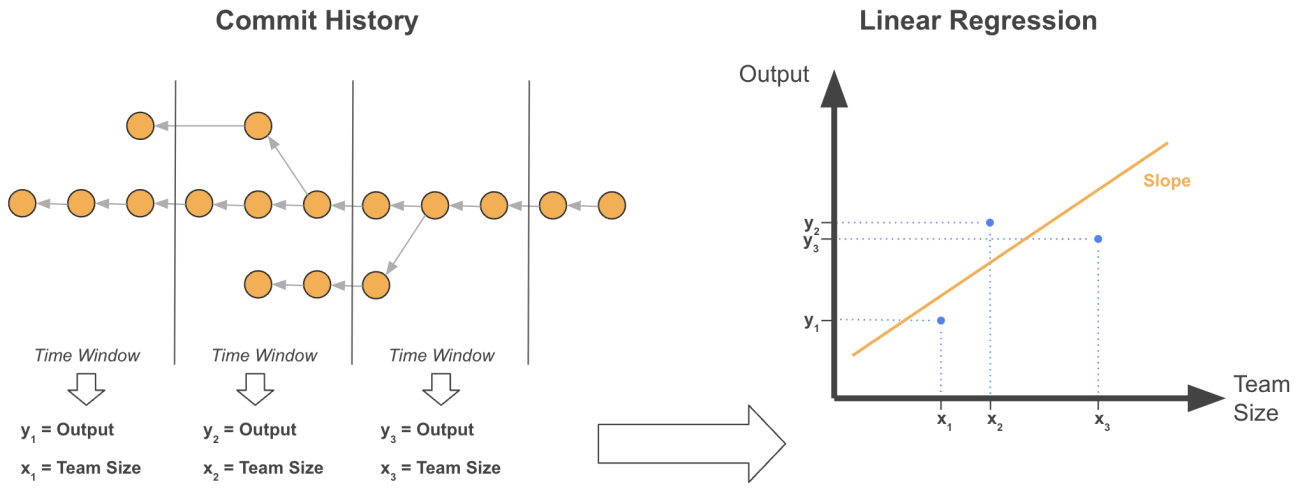
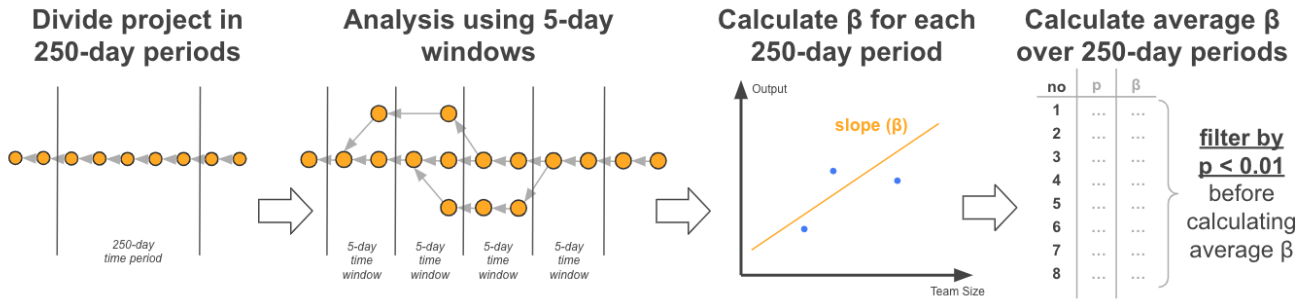
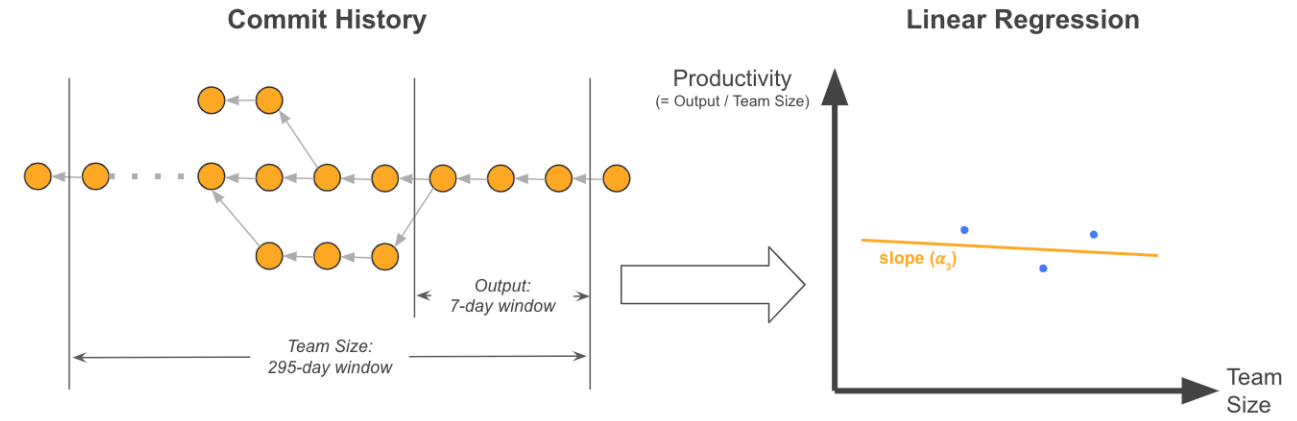


Figure 1: General regression approach used by both authors



(a) Illustration of Sornette's regression method



(b) Illustration of Scholtes' regression method

Figure 2: Illustration regression method details

non-binary files by comparing the file's content before the commit against the file's content after the commit. Depending on the particular case, it can make a difference if the distance is calculated on a line-by-line, block-by-block, or file-by-file basis. Since inferring the exact method was not possible, an email exchange with Ingo Scholtes was initiated, who provided assistance by recommending the use of the line-by-line calculation method for the replication.

3.2.3 Measurement of team size. To determine the team size, each team member must be uniquely identified. For this purpose, a Git commit provides four possible values: *Author Email*, *Author Name*, *Committer Email*, or *Committer Name*. In the majority of instances, the author and committer fields are the same. However, if modifications are made to a commit (e.g. via the `git rebase` command), the committer fields may differ from the author fields. This study used *Author Email* as the unique identifier of a team member. Scholtes et al. clearly stated the same choice in their publication, and an examination of the data on PLOS indicated that Sornette et al. also opted for this approach.

3.2.4 Regression analysis. Both authors determined if a project scales superlinearly by running a linear regression. However, variations in the setup of the regression analyzes have been observed:

- Sornette et al. defined the final metric for a project as an aggregation of multiple regression coefficients, called *average* β. Scholtes et al. calculated the coefficient of a simple linear regression, which they denominated α_3.
- Sornette et al. plotted the team's output against the team size while Scholtes et al. compared the team's productivity with the team size. Since Scholtes' productivity metric divides the team's output by team size, this difference is negligible in practical terms. It only means that Scholtes' method identifies superlinear projects for a slope > 0, while the same applies to Sornette's method for a slope > 1.
- Sornette et al. divided each project into non-overlapping 250-day periods before running a regression analysis on each of them, while Scholtes et al. used all available data points for one simple regression analysis per project.
- Scholtes et al. provided an additional analysis for Log-Lin relationships in which the logarithm of the productivity measure was plotted against the plain unmodified team size.
- Scholtes et al. also controlled for one-time contributors by providing an analysis that excluded contributions from anyone who submitted only one commit.
- Sornette et al. took statistical significance into account by calculating the *average* β only over those 250-day periods where the p-value was $p < 0.01$.

It should also be noted that the $p < 0.01$ filter identified in Sornette's method had only been detected while attempting to replicate the results. Without using this filter, reproduction of the *average* β values was not possible. Even minor alterations, such as a $p < 0.05$ filter, resulted in significantly higher p-values for the statistical tests, corroborating with the hypothesis that the $p < 0.01$ filter was actually part of the original study. This filter also implies that *average* β cannot be determined if the filter causes an exclusion of all β values.

Table 1: Comparison of projects analyzed and regression method used by Sornette et al. and Scholtes et al.

	Sornette et al.	Scholtes et al.
Projects	164 projects (3 overlapping)	58 projects (3 overlapping)
	5-3,074 contributors	55-4,107 contributors
Regression method	Output: 5-day windows	Output: 7-day windows
	Team size: 5-day windows	Team size: 295-day windows
	No. of file edits	Levenshtein distance
	Over 250-day periods	Over whole project
	Average of regression slopes	One regression slope only
	Filtered by $p < 0.01$	No filtering
	Log-Log regression only	Log-Log and Log-Lin
	No control for any variable	Control for one-time contributors

3.3 Summary

The two authors made quite distinct design choices, which are summarized in Table 1. Sornette et al. opted for a random sampling of projects and a simpler approach in measuring the team output, but adopted a more elaborated method in deriving the time windows for measuring the regression coefficients. Scholtes et al. chose a more straightforward method to calculate the regression coefficients, but selected projects more deliberately and studied more variants of their regression method by looking into Log-Lin regressions and controlling for one-time contributors. They also provided a more fine-grained measurement of the team's output. However, the influence of this fine-grained measure may not be as relevant due to the high degree of correlation between different output metrics identified by Gote et al. [8].

4 REPLICATION OF SORNETTE'S AND SCHOLTES' WORK

To ensure that any extrapolation of the original study is free from errors, it is essential to confirm that the original findings of Sornette et al. and Scholtes et al. can be replicated with statistical confidence. Shull et al. refer to this practice as performing a dependent replication, with the aim of closely reproducing the original findings, and recommend that it should come before any conceptual replications, which involve altering parameters to gain new insights [20]. Therefore, this section provides an in-depth review of the replication process before exploring possible selection and instrumentation biases.

4.1 Technical implementation

The technical implementation of this study is based on a set of Python scripts, using the PyDriller library [22]. These scripts ran on Google's Colab notebooks, using BigQuery as a storage engine. This choice had been made for reasons of convenience, with the objective of reducing the amount of time required to configure systems and maintain an IT infrastructure.

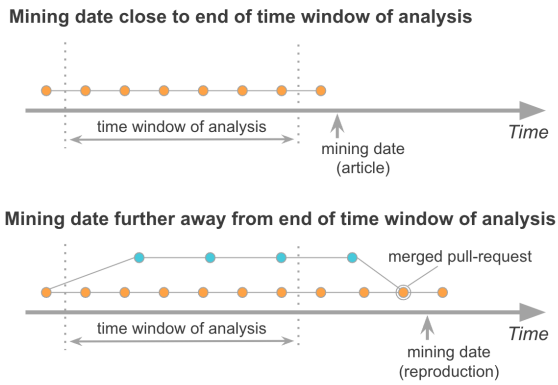


Figure 3: Potential exclusion of relevant commits if time window end date and mining date lie too close together

Another approach would have been the use of `git2net`, a tool developed by Scholtes' team [9]. This tool provides additional network analysis capabilities that are not needed for this research. Since these additional capabilities imply significantly longer execution times, the creation of a simpler and more performant mining script appeared to be the better option for data collection.

The creation of own tools comes with the risk of introducing errors. For example, Levenshtein distance calculations might have been performed differently than in the original tools. To mitigate these risks, the output of the scripts had been compared with the data provided by `git2net`, confirming that both tools compute exactly the same distances for all relevant, non-merge commits.

GitHub itself bears some pitfalls when it comes to the reproducibility of results. In particular, the `git rebase` command allows for a retroactive modification of commits. These modifications can increase or decrease the number of commits, which is a known problem with little remediation when it comes to the reproduction of studies. Furthermore, pull requests may cause distortions if the mining date is close to the end date of the time window of analysis, as demonstrated in Figure 3. In these cases, merging a pull request could add relevant commits to a repository that were not previously visible to the researcher. Finally, some very special cases of manual amendments might put the commit dates out-of-order, thus tripping off date filters, which assume all commits to be ordered by date.

The reproduction of results requires the exact GitHub URLs, which were not provided by neither author. So, a manual mapping of project names to GitHub URLs had to be performed. For Sornette et al., it was possible to identify the URL for 159 out of 164 projects. In the case of Scholtes et al. all 58 projects could be mapped to an URL.

Reproducing results also necessitates identifying the correct time windows of analysis for each GitHub project. For Sornette et al. the dates could be inferred by extracting the number of days of analysis from the Numpy data files published on PLOS, and by working with the assumption that the first commit on GitHub equals the start date of the analysis. In Scholtes' case the exact start and end dates of the analysis were part of the article's appendix.

4.2 Statistical tests

All of the factors in the above section are reasons why the replication of precise figures can be challenging, as was the case in this study. Therefore, a pairwise comparison was carried out to assess whether the measured and reported values for each project were drawn from the same distribution. Thereby, the Wilcoxon test was performed since the Shapiro-Wilk test indicated that none of the regression coefficients adhered to a normal distribution [24][18]. In both cases, the SciPy Python package had been leveraged [23].

Both authors not only reported the regression coefficients in their *Article*, but also provided *Commit Data* via PLOS or Zenodo. These data sets in addition to the data mined on *GitHub* allow for the following pairwise comparisons:

(1) *Article vs. GitHub*: The ultimate test if the results can be reproduced.
(2) *Article vs. Commit Data*: A check if the regression calculations are being performed correctly.
(3) *Commit Data vs. GitHub*: An additional check of consistency.

All these comparisons used Scholtes' Log-Log regression model without a filter of one-time contributors, since this variant is closest in resemblance to Sornette's method.

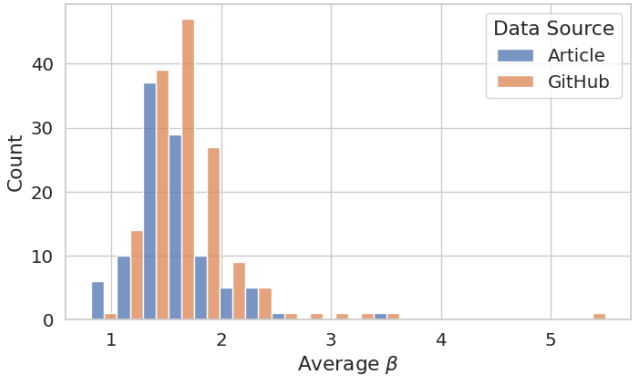
The histogram in Figure 4 compares the data reported in the *Article* and the reproduced results using data from *GitHub*. The apparent similarity of the distributions could be confirmed by Wilcoxon tests, whose values are depicted in Figure 5. In all cases, the null hypothesis that the data came from different distributions could not be rejected, implying that the results were similar and that the replicated data could be used for further analysis.

5 SELECTION BIAS

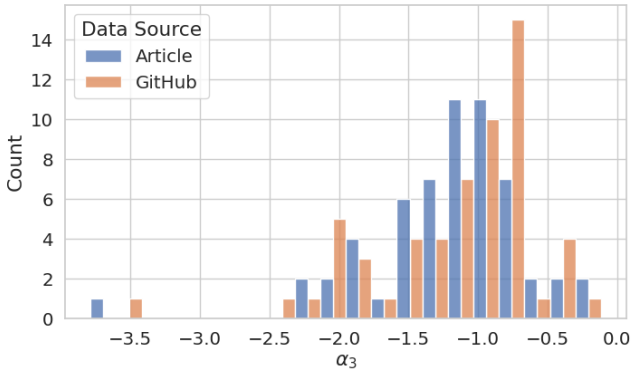
Subsequent research indicated that the differing conclusions reached by Sornette et al. and Scholtes et al. can be attributed to the selection of projects [15][8]. This section is an attempt to effectively quantify the impact of project selection by calculating Sornette's *average* β and Scholtes' α_3 for all projects. If selection bias was significant, many of the projects chosen by Sornette et al. would produce values of $\alpha_3 > 0$, while the projects selected by Scholtes et al. would generate values of *average* $\beta < 1$.

The result of this exercise is shown in Figure 6. For both regression methods, the distributions generated by each data set seem to differ, which could be confirmed by the p-values of two-sample Kolmogorov-Smirnov tests using the SciPy package [11][23]: For *average* β a p-value of $p = 0.00000000099$ was obtained and for α_3 a value of $p = 0.001383$ was measured, indicating that the results from both data sets differed significantly for either method.

However, the information presented in Figure 6 and Table 2 also suggests that the selection of projects only partially accounts for the differences between Sornette and Scholtes. For most of Scholtes' projects, a superlinear relationship had been detected, even when Sornette's method was applied. Likewise, Scholtes' method measured sublinear relationships for the majority of projects in Sornette's data set. Both facts clearly indicate that selection bias alone is not a sufficient explanation for the differences between Sornette et al. and Scholtes et al., necessitating further investigation of biases within the regression methods.



(a) Results for Sornette's *average* β



(b) Results for Scholtes' α_3

Figure 4: Comparison between regression coefficients as reported in the article and the measured values using data from GitHub

Data Source	Article	GitHub	PLOS
Article	1.0000	0.5535	0.9475
GitHub	0.5535	1.0000	0.8795
PLOS	0.9475	0.8795	1.0000

(a) Results for Sornette et al.

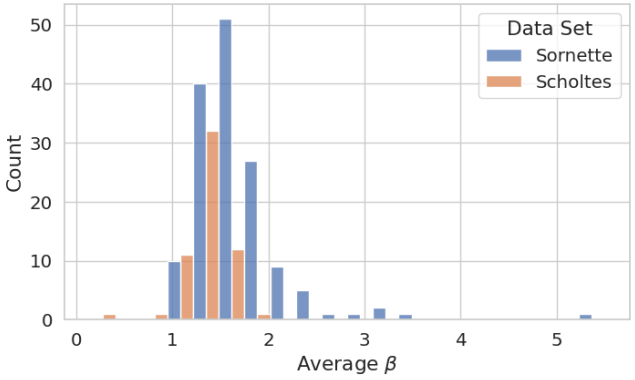
Data Source	Article	GitHub	Zenodo
Article	1.0000	0.1048	0.5331
GitHub	0.1048	1.0000	0.2198
Zenodo	0.5331	0.2198	1.0000

(b) Results for Scholtes et al.

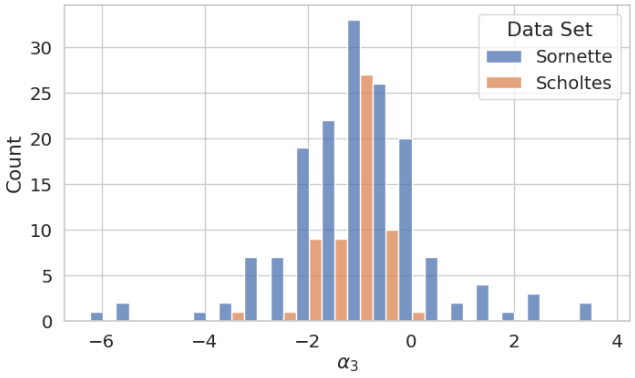
Figure 5: Results of the Wilcoxon tests

Table 2: Number of projects with sub- and superlinear relationship between team size and productivity

Regression Method	Data Set	Sublinear Projects	Superlinear Projects	Projects Total
Sornette	Sornette	1	147	148
	Scholtes	3	55	58
Scholtes	Sornette	130	29	159
	Scholtes	58	0	58



(a) Comparison for Sornette's average β values



(b) Comparison for Scholtes' α_3 values

Figure 6: Cross-comparison of data sets and regression methods

6 INSTRUMENTATION BIASES

The preceding section has already shown that project selection only partially accounts for the discrepancies between Sornette and Scholtes. This section eludes to potential biases found in either regression method, thus providing an additional explanation for the differences between Sornette's and Scholtes' findings.

6.1 Sornette's p-value filter

Section 3 mentions that reproduction efforts for the work of Sornette et al. were only successful after applying a p-value filter with $p < 0.01$ when calculating the *average* β values for a project. However, these p-values tend to get high for slopes close to zero, because they represent results *"for a hypothesis test whose null hypothesis is that the slope is zero, using Wald Test with t-distribution of the test statistic"* [5]. Hence, the filter can introduce bias by excluding small values of β from the computation of the *average* β.

Figure 7 compares the distribution of *average* β values with a $p < 0.01$ filter to values without any filter. It can be clearly seen that the removal of the filter causes a reduction of *average* β, which was 14.82% on average. A Wilcoxon test with $p = 0.0000000$ also confirms that the reduction was of statistical significance.

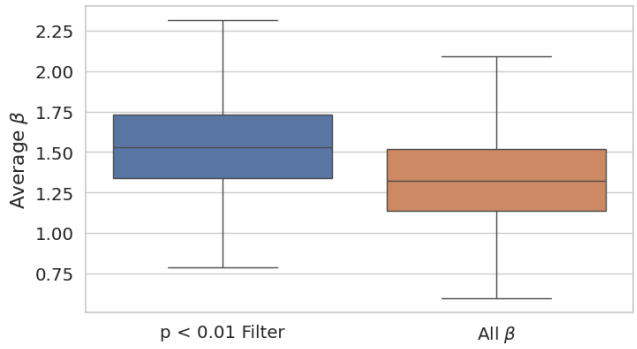


Figure 7: Distribution of *average* β, comparing values using $p < 0.01$ to values which use all β

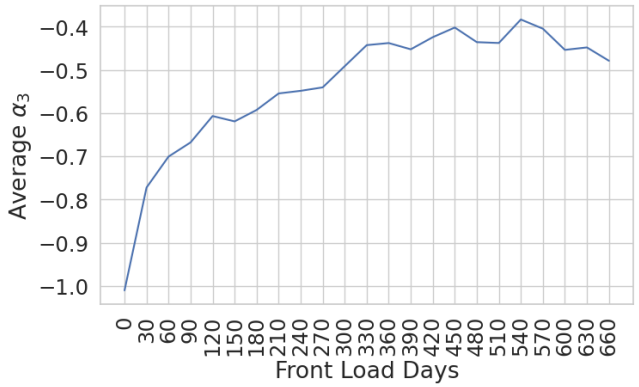


Figure 8: The influence of *Front Load Days* on the average of α_3 over all projects

Table 3: Effect of p-value filter on number of projects with sublinear relationship between team size and productivity

Regression Method	Data Set	Sublinear Projects	Superlinear Projects	Projects Total
Sornette ($p < 0.01$)	Sornette	1	147	148
	Scholtes	3	55	58
Sornette (all β)	Sornette	14	134	148
	Scholtes	16	42	58

The effect of the filter also influences the sub- and superlinearity, as shown in Table 3. For Sornette's data set the number of projects with diseconomies of scale increases from 1 to 14, and for Scholtes' data set the increase is from 3 to 16 projects. These numbers indicate that the p-value filter seemed to have a real influence on Sornette's conclusions.

6.2 Scholtes' time window selection

The regression analysis performed by Scholtes et al. took into account all contributions, from the very first commit until a certain cut-off date in 2014. An exploratory analysis indicated that removing the first few days from a project significantly influences the results of Scholtes' method, leading to the suspicion that the choice of time window may bear an instrumentation bias. A more detailed investigation led to Figure 8, showing the effect on α_3 if the first days of a project, denoted as *Front Load Days*, are removed from the regression analysis. The steady increase until a peak at 540 days is evidence that Scholtes' method may indeed be affected by instrumentation bias.

One possible explanation for this bias could be code imports during the first days of publication on GitHub. A developer or a team of developers may already have engaged in significant, non-measurable work before publishing the first commit. This work remains invisible until its inclusion via code imports after the project's launch. Since the early stages of a project tend to involve fewer developers, imports potentially generate artificial spikes of team output, which correlate with small team sizes, and thus lead to a negative bias for the regression analysis. Figure 9 illustrates this effect.

The research of Gote et al. also addresses the impact of code imports by excluding the top and bottom 2.5% of contributions, in terms of Levenshtein distance [8]. However, the effect of *Front Load Days* prevails even when filtering by these outliers, as shown in Figure 10. A potential reason could be that code imports can happen in small chunks (i.e. module-by-module) and therefore would not entirely be filtered out by accounting for outliers.

The influence of *Front Load Days* is negligible in the case of Sornette's regression method because the use of 250-day periods inhibits propagation of that bias throughout the entire regression analysis. A comparison of the *average* β values where the first 250-day period had been removed with the values of the original method, as shown in Figure 11, demonstrates that the exclusion of the first couple of days had little influence on the overall outcomes. A Wilcoxon test with a p-value of $p = 0.9841$ supports this finding.

Eliminating *Front Load Days* comes with the drawback of having to exclude certain projects from the analysis due to insufficient data points for the calculation of the regression coefficient. Consequently, the decision was made to choose 330 *Front Load Days* to examine the impact on superlinearity. This choice eliminated less than 5% of all projects from the analysis. At the same time, 330 days seems to be a point where the average α_3 appears to stabilize. For 330 *Front Load Days* the number of projects with a superlinear relationship increases from 29 to 65 projects in the case of Sornette's data set, and from 0 to 11 projects in the case of Scholtes' data set, as shown in Table 4. This increase suggests that the choice of *Front Load Days* has a relevant impact on the results of Scholtes' regression method.

7 CONCLUSION

The discussion of Aristotle vs. Ringelmann evolved around the question whether the productivity of Open Source projects scales sublinear or superlinear with regard to its team size. The conflicting findings reported by Sornette et al. and Scholtes et al. prompted both research groups to publish follow-up studies, suggesting project selection as a possible explanation. The answer to the first research question of this study – *"How significant was project selection in*

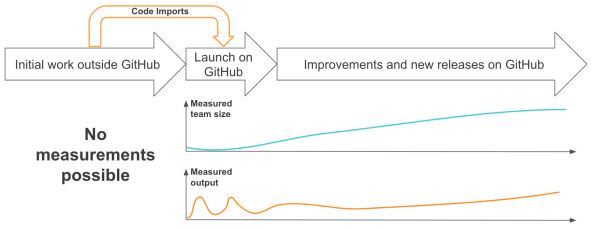


Figure 9: Illustration of *Front Load Days* effect: Code imports during the launch phase on GitHub may distort results

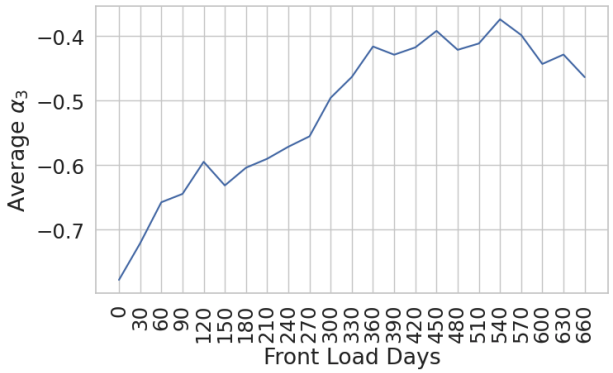


Figure 10: The influence of *Front Load Days* on the average of α_3 over all projects, when excluding the top and bottom 2.5% of contributions based on Levenhstein distance

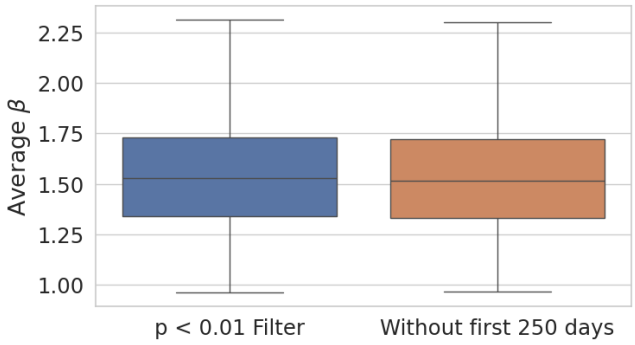


Figure 11: Comparison of original *average* β distribution with the values where the first 250-days were removed

creating the differences observed between Sornette et al. and Scholtes et al.?" – probes into the significance of the suggested selection bias by replicating both regression methods on both data sets. This analysis revealed that 130 of 159 projects from Sornette's data set scaled sublinearly using Scholtes' method. Likewise, 55 of 58 projects from Scholtes' data set showed superlinear scaling if subjected to Sornette's method. Thus, indicating that selection bias had only a minor influence.

Table 4: Effect of *Front Load Days* on number of projects with sublinear relationship between team size and productivity

Regression Method	Data Set	Sublinear Projects	Superlinear Projects	Projects Total
Scholtes (0 days)	Sornette	130	29	159
	Scholtes	58	0	58
Scholtes (330 days)	Sornette	85	65	150
	Scholtes	46	11	57

This finding led to the second research question – *"What bias in Sornette's regression method could have favored the identification of a superlinear relationship between team size and productivity?"* – which identified a p-value filter as a source of bias, whose removal increased the total number of projects that scale sublinearly from 4 to 30. Similarly, an inquiry into the third research question – *"What bias in Scholtes' regression method could have favored the identification of a sublinear relationship between team size and productivity?"* – led to the discovery of *Front Load Days*, whose removal increased the total number of projects that scale superlinearly from 29 to 76.

Figure 12 summarizes the influences of selection and instrumentation biases. Since the population size for each analysis vary, all results are reported in percent to have a relative comparison. For example, using the original method on the original data set identified 0% superlinearity for the 58 projects from Scholtes. In the case of Sornette et al., the original method yielded results for 148 projects, with 99.32% showing superlinearity. Consequently, the difference is 99.32%. Applying both regression methods to all projects reveals that selection bias alone does not fully account for the discrepancies, leaving a difference of 84.69%. A sole adjustment of the regression methods approximates results better, resulting in a difference of 70.64%. And, using the adjusted methods on all projects leaves a gap of 48.54%. Thus, the conclusion can be drawn that selection bias as well as instrumentation biases contributed to the different conclusions drawn by Sornette et al. and Scholtes et al.

8 FINAL CONSIDERATIONS

8.1 Discussion

The results of this research not only add to the Aristotle vs. Ringelmann debate, but also offer general insights into measuring productivity using Mining Software Repositories techniques. Initially, the study confirms that selection bias has an influence. Secondly, it eludes to the fact that the choice of time windows is critical, as demonstrated by the influence of the *Front Load Days*. Finally, the effect of the p-value filter illustrated that the design of the data pipeline requires careful consideration to avoid the introduction of instrumentation biases.

This study also addresses the challenge of reproducibility in Empirical Software Engineering. In the related field of effort estimation for software development, Shepperd et al. identified only 28 replication studies [19]. Based on their assessment, there are

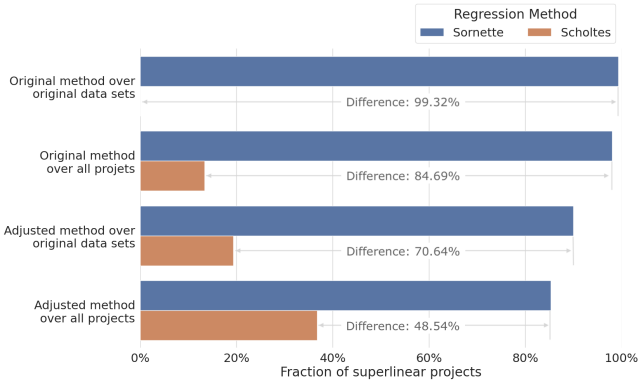


Figure 12: Summary on how selection and instrumentation biases influence the fraction of projects with a superlinear relationship for each regression method

probably hundreds or thousands of publications in this area, leading to the conclusion that only a very small percentage are actually being replicated. Cockburn et al. alert that publication bias leads to practices such as project selection, p hacking, and HARKing[2], which might put computer science research at risk [4]. They argue in favor of better statistical practices, experiment preregistration, openness of data, and encouragement of replications. By successfully reproducing the findings of Sornette et al. and Scholtes et al., this research contributes to the existing collection of published replication studies. Furthermore, this publication made efforts to adhere to the recommendations from Cockburn et al. by including a thorough reporting p-values and by providing public access to all data and calculations used in this study.

8.2 Lessons Learned

The work carried out in this research also provided insight into factors that are of importance when replicating studies and offered a deeper understanding of how to reduce biases.

First, taking small steps proved to be beneficial. For example, instead of jumping right to the calculation of correlation coefficients, the reproduction of intermediate steps, such as the calculation of Levenshtein distances for single commits, already raised important questions. Second, cross-checks between different data sources, i.e. reproduced data, the results from the article and published data sets, triangulated problems, and increased confidence in the replication. Third, statistical tests also revealed to be important, leading to the detection of the $p < 0.01$ filter in Sornette's method. And last but not least, reaching out to the authors of the original papers helped to clarify implementation details.

As this study shows, avoiding bias is not a simple undertaking. Accounting for as many factors as possible, e.g., team size, definitely addresses this issue; however, this approach has its limitations if the number of available variables gets too big or if relevant variables are either unknown or difficult to measure. Another effective method to mitigate biases is to encourage reproductions, as independent parties' reviews are an established approach to guarantee quality and reliability.

[2]Hypothesizing After the Results are Known

8.3 Limitations and Future Work

This study has some limitations, which may serve as a starting point for future work.

The influence of time window selection has been addressed only partially. Apart from the *Front Load Days*, there is also the question of whether the number of days for team size and team output windows influenced the results, whether the variation in the total number of years analyzed for each project was relevant, or whether data from a more recent period would have had an impact.

Calculating the correct team size might be another factor to consider. In that regard, Gote and Zing addressed the problem of author disambiguation and provided a tool for its solution [10]. However, disambiguation has been excluded from this study to maintain consistency with the original research methodologies. Another emerging and significant question revolves around the influence of commits made by bots and AI agents.

This research has not examined confounders or other influencing variables, except for team size. Lavazza et al. identified additional factors that impact productivity, such as programming language or project type [14], and Gote et al. discovered the importance of foreign code edits [8]. So, factors related to dimensions like team structure, source code structure, or software engineering practices may provide additional insights.

Furthermore, the existing method of measuring productivity, which divides team output by team size, can be complemented by additional factors such as quality, maintainability, or sustainability. Recent recommendations to evaluate software development productivity in a more multidimensional manner, like the SPACE metrics, already provide a suitable framework [7]. The complexity of software development may also require complementary qualitative evaluations to improve the understanding of the underlying creative process [12][13]. Such extensions could serve as an approach to producing rigorous and quantitative research, which also offers practical insights for software development practitioners.

ACKNOWLEDGMENTS

We would like to thank Sornette et al. and Scholtes et al. for providing a great inspiration on how to conduct rigorous quantitative analysis of productivity in the realm of Open Source software development. In particular, we thank Thomas Maillart and Ingo Scholtes for their clarifications and feedback.

We are also grateful for the invaluable insights and discussions provided by the Software Engineering Research Group at the University of São Paulo.

ARTEFACT AVAILABILITY

All data sets, Colab notebooks, and Python code are available on Zenodo (see https://zenodo.org/records/12755951).

REFERENCES

[1] Barry Boehm, Bradford Clark, Ellis Horowitz, Chris Westland, Ray Madachy, and Richard Selby. 1995. Cost models for future software life cycle processes: COCOMO 2.0. *Annals of Software Engineering* 1, 1 (Dec. 1995), 57–94. https://doi.org/10.1007/BF02249046

[2] Frederick P. Brooks. 1995. *The mythical man-month: essays on software engineering* (anniversary ed ed.). Addison-Wesley Pub. Co, Reading, Mass.

[3] Wladmir Araujo Chapetta and Guilherme Horta Travassos. 2020. Towards an evidence-based theoretical framework on factors influencing the software development productivity. *Empirical Software Engineering* 25, 5 (Sept. 2020), 3501–3543. https://doi.org/10.1007/s10664-020-09844-5

[4] Andy Cockburn, Pierre Dragicevic, Lonni Besançon, and Carl Gutwin. 2020. Threats of a Replication Crisis in Empirical Computer Science – Communications of the ACM. https://cacm.acm.org/research/threats-of-a-replication-crisis-in-empirical-computer-science/

[5] The SciPy community. 2008. linregress — SciPy v1.14.0 Manual. https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.linregress.html

[6] Carlos Henrique C. Duarte. 2022. Software Productivity in Practice: A Systematic Mapping Study. *Software* 1, 2 (May 2022), 164–214. https://doi.org/10.3390/software1020008

[7] Nicole Forsgren, Margaret-Anne Storey, Chandra Maddila, Thomas Zimmermann, Brian Houck, and Jenna Butler. 2021. The SPACE of Developer Productivity: There's more to it than you think. *Queue* 19, 1 (Feb. 2021), 20–48. https://doi.org/10.1145/3454122.3454124

[8] Christoph Gote, Pavlin Mavrodiev, Frank Schweitzer, and Ingo Scholtes. 2022. Big data = big insights?: operationalising brooks' law in a massive GitHub data set. In *Proceedings of the 44th International Conference on Software Engineering*. ACM, Pittsburgh Pennsylvania, 262–273. https://doi.org/10.1145/3510003.3510619

[9] Christoph Gote, Ingo Scholtes, and Frank Schweitzer. 2019. git2net - Mining Time-Stamped Co-Editing Networks from Large git Repositories. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. IEEE, Montreal, QC, Canada, 433–444. https://doi.org/10.1109/MSR.2019.00070

[10] Christoph Gote and Christian Zingg. 2021. gambit – An Open Source Name Disambiguation Tool for Version Control Systems. http://arxiv.org/abs/2103.05666 arXiv:2103.05666 [physics].

[11] J. L. Hodges. 1958. The significance probability of the smirnov two-sample test. *Arkiv för Matematik* 3, 5 (Jan. 1958), 469–486. https://doi.org/10.1007/BF02589501

[12] Ciera Jaspan and Caitlin Sadowski. 2019. No Single Metric Captures Productivity. In *Rethinking Productivity in Software Engineering*, Caitlin Sadowski and Thomas Zimmermann (Eds.). Apress, Berkeley, CA, 13–20. https://doi.org/10.1007/978-1-4842-4221-6_2

[13] Amy J. Ko. 2019. Why We Should Not Measure Productivity. In *Rethinking Productivity in Software Engineering*, Caitlin Sadowski and Thomas Zimmermann (Eds.). Apress, Berkeley, CA, 21–26. https://doi.org/10.1007/978-1-4842-4221-6_3

[14] Luigi Lavazza, Sandro Morasca, and Davide Tosi. 2018. An Empirical Study on the Factors Affecting Software Development Productivity. *e-Informatica Software Engineering Journal* 12 (2018), 27–49. https://doi.org/10.5277/E-INF180102 Medium: PDF Publisher: Institute of Applied Informatics, Wrocław University of Technology, Wrocław.

[15] Thomas Maillart and Didier Sornette. 2019. Aristotle vs. Ringelmann: On superlinear production in open source software. *Physica A: Statistical Mechanics and its Applications* 523 (June 2019), 964–972. https://doi.org/10.1016/j.physa.2019.04.130

[16] Goran Murić, Andres Abeliuk, Kristina Lerman, and Emilio Ferrara. 2019. Collaboration Drives Individual Productivity. *Proceedings of the ACM on Human-Computer Interaction* 3, CSCW (Nov. 2019), 1–24. https://doi.org/10.1145/3359176

[17] Ingo Scholtes, Pavlin Mavrodiev, and Frank Schweitzer. 2016. From Aristotle to Ringelmann: a large-scale analysis of team productivity and coordination in Open Source Software projects. *Empirical Software Engineering* 21, 2 (April 2016), 642–683. https://doi.org/10.1007/s10664-015-9406-4

[18] S. S. SHAPIRO and M. B. WILK. 1965. An analysis of variance test for normality (complete samples)†. *Biometrika* 52, 3-4 (Dec. 1965), 591–611. https://doi.org/10.1093/biomet/52.3-4.591 _eprint: https://academic.oup.com/biomet/article-pdf/52/3-4/591/962907/52-3-4-591.pdf.

[19] Martin Shepperd, Nemitari Ajienka, and Steve Counsell. 2018. The role and value of replication in empirical software engineering results. *Information and Software Technology* 99 (July 2018), 120–132. https://doi.org/10.1016/j.infsof.2018.01.006

[20] Forrest J. Shull, Jeffrey C. Carver, Sira Vegas, and Natalia Juristo. 2008. The role of replications in Empirical Software Engineering. *Empirical Software Engineering* 13, 2 (April 2008), 211–218. https://doi.org/10.1007/s10664-008-9060-1

[21] Didier Sornette, Thomas Maillart, and Giacomo Ghezzi. 2014. 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* 9, 8 (Aug. 2014), e103023. https://doi.org/10.1371/journal.pone.0103023

[22] Davide Spadini, Maurício Aniche, and Alberto Bacchelli. 2018. PyDriller: Python framework for mining software repositories. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2018)*. Association for Computing Machinery, New York, NY, USA, 908–911. https://doi.org/10.1145/3236024.3264598

[23] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. Van Der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul Van Mulbregt, SciPy 1.0 Contributors, Aditya Vijaykumar, Alessandro Pietro Bardelli, Alex Rothberg, Andreas Hilboll, Andreas Kloeckner, Anthony Scopatz, Antony Lee, Ariel Rokem, C. Nathan Woods, Chad Fulton, Charles Masson, Christian Häggström, Clark Fitzgerald, David A. Nicholson, David R. Hagen, Dmitrii V. Pasechnik, Emanuele Olivetti, Eric Martin, Eric Wieser, Fabrice Silva, Felix Lenders, Florian Wilhelm, G. Young, Gavin A. Price, Gert-Ludwig Ingold, Gregory E. Allen, Gregory R. Lee, Hervé Audren, Irvin Probst, Jörg P. Dietrich, Jacob Silterra, James T Webber, Janko Slavič, Joel Nothman, Johannes Buchner, Johannes Kulick, Johannes L. Schönberger, José Vinícius De Miranda Cardoso, Joscha Reimer, Joseph Harrington, Juan Luis Cano Rodríguez, Juan Nunez-Iglesias, Justin Kuczynski, Kevin Tritz, Martin Thoma, Matthew Newville, Matthias Kümmerer, Maximilian Bolingbroke, Michael Tartre, Mikhail Pak, Nathaniel J. Smith, Nikolai Nowaczyk, Nikolay Shebanov, Oleksandr Pavlyk, Per A. Brodtkorb, Perry Lee, Robert T. McGibbon, Roman Feldbauer, Sam Lewis, Sam Tygier, Scott Sievert, Sebastiano Vigna, Stefan Peterson, Surhud More, Tadeusz Pudlik, Takuya Oshima, Thomas J. Pingel, Thomas P. Robitaille, Thomas Spura, Thouis R. Jones, Tim Cera, Tim Leslie, Tiziano Zito, Tom Krauss, Utkarsh Upadhyay, Yaroslav O. Halchenko, and Yoshiki Vázquez-Baeza. 2020. SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature Methods* 17, 3 (March 2020), 261–272. https://doi.org/10.1038/s41592-019-0686-2

[24] Frank Wilcoxon. 1945. Individual Comparisons by Ranking Methods. *Biometrics Bulletin* 1, 6 (1945), 80–83. https://doi.org/10.2307/3001968 Publisher: [International Biometric Society, Wiley].