Evaluating the Impact of Collaboration Modes on Software Delivery Efficiency in Open-Source Projects

Atanas Buntov[1]

Supervisors: Sebastian Proksch[1]**, Shujun Huang**[1]

[1]**EEMCS, Delft University of Technology, The Netherlands**

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 22, 2025

Name of the student: Atanas Buntov
Final project course: CSE3000 Research Project
Thesis committee: Sebastian Proksch, Shujun Huang, Marco Zuñiga Zamalloa

Abstract

The growing complexity of modern software systems, driven by larger codebases and evolving technologies, has amplified the need for effective collaboration in developer teams. Specifically, in open-source software (OSS) projects, where contributors often vary in background and engagement in the process, this complexity may introduce further collaborative challenges. As projects scale, coordination becomes increasingly difficult to maintain, highlighting the importance of understanding the socio-technical dynamics of effective development. While prior research emphasizes the role of collaboration, its influence on software delivery remains underexplored. In order to address the gap, this study examines how team characteristics, such as size and expertise, and communication practices, like interactions on issues and pull requests, relate to delivery efficiency in OSS projects. Based on an empirical analysis of 887 GitHub repositories, we found that team size and project expertise exhibit the strongest relationships with delivery size and frequency. Core contributor activity also shows positive but diminishing effects over time, while communication practices demonstrate no noticeable associations with release efficiency. These findings suggest that optimizing delivery in OSS projects may benefit from considering adaptive team structures and aligning CI/CD practices with the project stage and the evolving dynamics of the developer team.

Keywords: open-source software, team structure, collaboration, communication, delivery efficiency

1 Introduction

Modern-day software systems are becoming increasingly complex, driven by the rapid technological advances and continuous exploration of automation. In an empirical study from 2017, Hatton et al. showed that software systems approximately double in size every 42 months, highlighting the scale of this growth [27]. Meanwhile, the number of new projects has also surged, with GitHub reporting over 100 million repositories being created in 2024 alone, a 25% year-over-year increase [23]. This expansion, along with the iterative nature of present software development, has intensified the demand for effective coordination and communication. In the past, this need has led to the research and adoption of practices and tools for automating software processes [5; 38], and still continues to be a topic of interest [51; 55].

The necessity for effective collaboration is even more pronounced in open-source software (OSS) projects. They might include contributors with varying levels of expertise, familiarity, or engagement in the project, which has resulted in the establishment of different modes of communication [48]. The decentralized and distributed nature of OSS environments means that team structure is also an influential factor for efficiency and project outcome [31; 35].

The complex relationship between team composition and developer productivity has been extensively researched in the past. Brooks famously argued that adding programmers to a late project will only worsen the delays [7], while Putnam's *software equation* suggested that productivity peaks at an optimal team size [41]. Later studies confirmed a non-linear relationship, indicating diminishing returns beyond a certain number of contributors [4; 32]. More recent research, however, shows that a larger number of developers can improve pull request merges, but primarily for PRs created by core, experienced contributors [51]. The results from investigating the impact of developer experience itself remain mixed: while Gill and Kemerer reported a positive correlation with productivity [21], Tubío et al. argued that academic background improves efficiency, but industry experience seems to have no effect [17].

Beyond team structure, communication practices have also been researched and linked to efficiency. In a study from 2021, Forsgren analyzed developer activity during the COVID-19 pandemic and observed that in these times, when teams shifted to more asynchronous communication, indicators such as the volume of PRs, comments on PRs, reviews, and the depth of issue discussions consistently increased together [19]. This suggests that active engagement in PRs and issues might have a positive impact on developer productivity.

Despite these contributions, there is a lack of empirical studies which have systematically connected collaboration modes to concrete measures of delivery efficiency in OSS projects. In OSS development, contributors increasingly rely on efficient CI/CD pipelines to automate processes, making delivery efficiency critical for sustaining contributor engagement, maintaining code quality, and ensuring the timely release of new features and fixes [29]. Understanding how collaboration modes influence delivery efficiency can provide actionable insights for developers – such as strategies to improve productivity, reduce delays, and accelerate delivery cycles – and also offer researchers directions for future research. This study aims to address the present knowledge gap by posing the following research question:

How do collaboration modes affect software delivery efficiency in open-source projects?

Specifically, we examine several structural characteristics of team composition and communication throughout project development, which allow us to holistically describe collaboration modes. To have more granular analysis of the research question, we break it down to the following sub-questions:

- **RQ1:** How do team size and expertise influence delivery frequency and size?
- **RQ2:** Does the fraction of core developer contribution impact delivery frequency and size?
- **RQ3:** Is change lead time affected by the amount and depth of developer communication activities on issues and pull requests?

Our analysis is based on a curated dataset of 887 active GitHub repositories, from which we extract team-related (size, expertise, core contribution) and communication-

related (issue and pull request interactions) metrics over three-month non-overlapping time windows. To measure delivery efficiency, we introduce three proxy metrics: delivery frequency, delivery size, and change lead time. We then apply correlation and regression analyses to examine the strength of the relationships and investigate how changes in the defined independent variables affect the dependent variables.

The results show that team size has the strongest impact on delivery frequency and size, increasing up to about seven developers before leveling off. Developer expertise and core contribution also relate to more frequent releases, with the relationships being nonlinear and time-dependent. In contrast, communication patterns show no pronounced associations with change lead time, suggesting that static measures may not fully capture the dynamic effects of coordination.

The rest of this thesis is organized as follows. Firstly, Section 2 reviews related literature and the current state of research. The subsequent Section 3 defines the studied variables and outlines the data selection, collection, and analysis procedures. The results of the analysis are presented in Section 4 and then discussed in Section 5, including potential directions for future work. Responsible research practices are detailed in Section 6, and Section 7 concludes the study with a summary of the research questions and key findings.

2 Related Work

The relationship between collaboration modes among software development teams and work efficiency has long been a subject of empirical investigation. Foundational studies established core theories on team dynamics, which more recent research has built upon to provide deeper insights.

In one of the first works that acknowledged the non-trivial relationship between team composition and developer productivity, Brooks claimed that adding programmers to a project running late will only exacerbate the delay [7]. This effect, known in software engineering as *Brooks' law*, is studied in social psychology as *Ringelmann effect* and describes how individual productivity drops when group size increases due to coordination and motivation loss [43; 46].

Following Brooks' findings and the increase in software complexity, the question of the optimal team size continued to be a topic of research, with one of the first quantitative answers given by Putnam in 1978 [41]. In his paper, he derived the *software equation*, which models the project size as a function of team productivity, effort, and available time. Using this equation, an optimal team size can be obtained – one that maximizes productivity in the given time frame without compromising the project's outcome. Further results on the topic were subsequently presented in the related works of Kandel and Lazear, and Backes-Gellner et al., who analyzed startups and showed a non-linear parabolic relationship between team size and productivity [4; 32]. They focused on behavioral dynamics, such as peer pressure and free-riding, and concluded that there is an optimal team size at which developers can reach maximum efficiency. Empirical analyses of more than 8000 projects by Heričko et al., and Rodríguez et al. concluded that, accounting for the size of the project, this optimal team size is usually between 7 and 10 developers [28; 45]. More recent research, however, presents contrasting evidence of an additive relationship between the number of contributors and productivity [47]. For instance, Vasilescu et al. demonstrate that adding developers to a project leads to a higher number of pull requests being merged, particularly when the pull requests originate from core contributors [51].

Another aspect whose influence on productivity has been investigated is the developer experience. Nevertheless, existing literature does not provide a consistent conclusion on the topic. For example, Gill and Kemerer hypothesize a positive correlation between developer experience and productivity [21]. In contrast, a more recent study by Tubío et al. suggests that while academic experience tends to enhance code quality and developer productivity, industry experience appears to have no significant effect [17]. When examining expertise in the particular project, Espinosa et al. found that task familiarity is beneficial for team productivity in distributed environments, but mainly for less complex tasks [18].

Apart from team composition, communication practices among developers have also been linked to efficiency. Forsgren studied the impact on productivity in OSS environments of the restrictions imposed by the World Health Organization (WHO) during the COVID-19 pandemic [19]. She found that developer productivity, assessed through the number of pull requests, PR comments, PR reviews, and issue comments per user, increased in 2020 compared to 2019, a period during which asynchronous communication intensified. These findings suggest that active engagement in PR and issue discussions can be positively associated with efficiency.

3 Methodology

To understand the impact of collaboration modes on software delivery efficiency, we first defined measurable, representative indicators of both collaboration and release efficiency. We selected a dataset of 887 active OSS projects hosted on GitHub and collected raw information on the commits, issues, pull requests, and releases. From this, we calculated the collaboration and delivery metrics, computed over fixed time windows to capture the temporal evolution of project dynamics. Finally, we conducted a statistical analysis to assess the potential relationships and trends. An overview of the whole data processing pipeline is shown on Figure 1.

3.1 Metrics Definition

In order to study the potential relationships, we define two groups of quantitative metrics. The first group characterizes how contributors interact and coordinate within a project (*collaboration metrics*), while the second reflects the efficiency of code delivery (*delivery indicators*).

In OSS development, collaboration is exhibited in both the composition of the contributors team and the nature of their interactions. Effective collaboration involves not only the presence of active or experienced developers but also meaningful communication through mechanisms such as issue discussions and code reviews. To that end, we define the following collaboration metrics for both team structure and communication activities, which together reflect how contributors interact, coordinate, and share responsibilities.

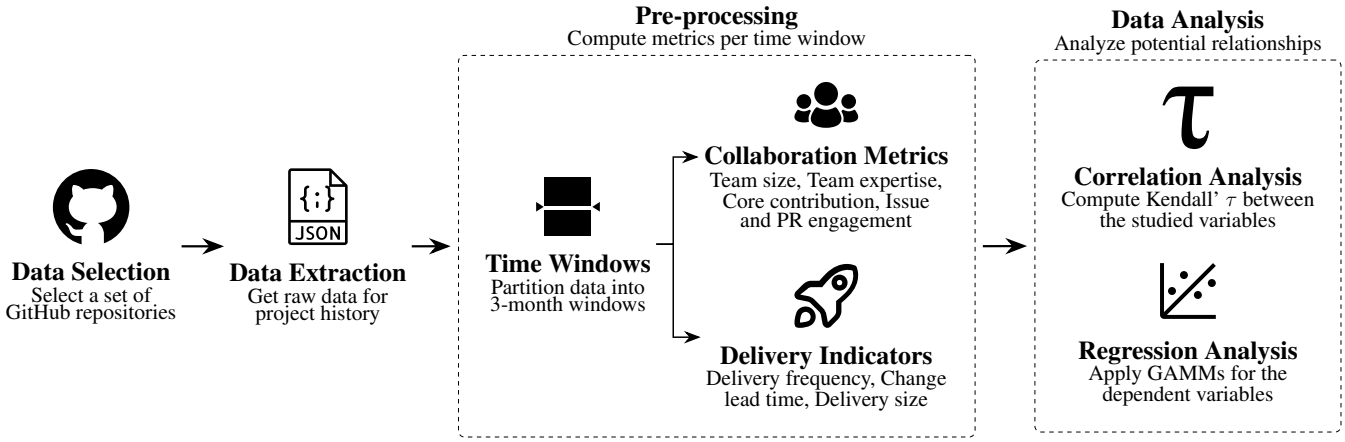


Figure 1: Overview of the data processing pipeline. The methodology involves four sequential phases: selection of suitable repositories, extraction of relevant raw data, pre-processing to compute selected metrics, and analysis of the trends and relationships.

Team size Measured by the number of developers with at least five commits in a time window. This threshold, based on an analysis by Youssef and Capiluppi, ensures that active developers are considered while drive-by users with negligible contributions are excluded [54].

Team expertise Defined as the average prior involvement of team members in the project. For each member in the current time window (i.e., having at least 5 commits, as defined above), we compute the fraction of previous windows in which they were also active. Team expertise is the average of these fractions, reflecting the extent to which the team comprises experienced, previously involved contributors. This approach aligns with prior research highlighting commit count as a key indicator of project expertise [36; 39].

Core contribution Measured by the fraction of commits produced by *core contributors*. As defined in the *onion model*, core developers are the ones contributing most of the code and overseeing the progress of a project [13; 37]. To identify them, we used the *Commit-Based Heuristic*, a popular criterion among studies, grounded in the observation that commit distributions typically exhibit heavy tails [8; 10; 44]. Using this heuristic, core developers in a project are the most active contributors responsible for 80% of the commits.

Issue engagement Captured by the number and average length of comments on issues closed within a time window. This metric reflects the degree and depth of asynchronous, often planning-related communication. It aligns with previous studies that use the number of issue comments and the length of issue discussions as indicators of conversational depth [1; 3].

Pull request activity Measured by the average number of reviews and comments per pull request in a time window. This metric reflects the level and thoroughness of collaborative code review during development. It is consistent with prior work that considers reviews and comments as measures for PR merge success [42].

The delivery efficiency indicators we study are derived from DORA's *Four Keys* [16] for evaluating deployment efficiency but are adapted to the OSS context, where deployments often occur outside the public repositories and are therefore not observable. Instead, we can look at *software releases*, defined as deployable software iterations [22]. Hence, the following indicators serve as practical proxies for the efficiency of code delivery, allowing for an estimation of the pace and granularity of completed software iterations.

Delivery frequency Measured by the total number of releases in a time window. Analogous to DORA's *deployment frequency* metric, this measure indicates how often complete software versions are released.

Change lead time Reflects the average time it takes for a commit to appear in a release – i.e., the duration between the commit creation and the first release that includes it. This metric, equivalent to DORA's *change lead time*, indicates how quickly changes flow through the development pipeline into a released version of the software.

Delivery size Defined as the number of changed lines of code between two successive releases within a time window. This measure captures the volume of changes per release and is a proxy for the amount of delivered code.

Together, these metrics provide a structured view of how developers collaborate in OSS environments and how efficiently they deliver software. By analyzing their interplay, we aim to uncover patterns that link team characteristics and communication patterns with software releases, and provide insights into the factors that shape efficient code delivery.

3.2 Data Selection and Data Extraction

To analyze the considered metrics in real-world projects, we curated a dataset of active popular open-source GitHub repositories, applying inclusion criteria previously established as filtering parameters in other studies. The repository selection criteria were: (i) active maintenance with at least one year of development history [26], (ii) a minimum of 20 releases to ensure sufficient delivery data, (iii) sufficient amount of development activity (at least 200 commits on the main branch

Table 1: Aggregate statistics of the selected repositories.

Metric	Mean	Median	Min	Max	Total
Commits	679.43	628	210	1,713	602,658
Contributors	34.20	24	1	234	30,334
Pull Requests	268.26	184	1	1,381	237,951
Issues	584.76	465	3	4,465	518,686
Releases	62.47	47	20	520	55,413
Stars	4,369.75	2,657	1,005	44,639	N/A
Age (months)	37.58	34	12	189	N/A
Main Language: TypeScript (23.4%), Python (17.0%), Go (10.4%), Rust (9.1%), JavaScript (5.6%), Others (34.5%)					

Table 2: Attributes extracted for each repository.

Category	Attributes
Commits	id (SHA-1 value), author, timestamp
Pull Requests	merge status, merge date, number of reviews, number of comments
Issues	comments, creation date, closing date
Releases	associated tag name, timestamp

[2]), and (iv) substantial popularity with at least 1000 stars [30]. Additionally, repositories that were forks, archived, or mainly for documentation purposes were not included in the dataset. In total, 887 repositories were selected, with various aggregate statistics for the dataset provided in Table 1.

After the dataset of repositories was established, we used the *GitHub REST API* to collect data about the commits, pull requests, issues, and releases of each project. The extracted metadata about collaboration and release activity of each examined repository is summarized in Table 2.

3.3 Pre-processing

Following the data extraction phase, we performed several pre-processing steps to transform the raw data into measurable indicators of collaboration and delivery performance.

Time windows Firstly, the data was partitioned into non-overlapping windows of three months so that the evolution of collaboration practices and delivery efficiency over time can be observed per project. The selection of a three-month window is based on the research on the Rapid Release (RR) methodology by da Silva et al. [9], and on a large-scale study of over 178,000 PyPI packages, which reported a median release interval of 16.37 days [6]. The chosen window length thus accommodates both frequent and infrequent release patterns, allowing for sufficient data points per project.

Collaboration metrics For each time window, we went through the set of commits to determine the team size, considering only contributors with at least five commits, as defined in Section 3.1. Using this information across all windows, we calculated the team expertise per window. Additionally, based on the entire repository history, we identified the core contributors, and computed the core contribution per window as the fraction of commits made by these developers.

To quantify communication activities, we analyzed the pull requests and issues within each time window. We only included pull requests that were both merged and later part of a release, using the merge date to assign them to windows. Similarly, only closed issues were considered, with the closing date used for window assignment. For each time slice, we computed the number of comments and code reviews on pull requests. For issues, we calculated the number of comments and their average length (in characters), covering all forms of commentary, including inline comments and discussion threads. Comments from both human users and automated bots were included, assuming that all commentaries can influence release processes by providing input, such as code checks, feedback, or suggestions.

Delivery efficiency metrics To compute the delivery efficiency metrics, we analyzed the code releases, which were assigned to a time window based on their publishing date. Using this assignment, we first computed the total number of releases per window, which represents delivery frequency.

To measure delivery size, we calculated the changed lines of code (LOC) per release. For each release, we computed the changed LOC between the tags pointing to this release and the one before it. The number of changes for the first release was defined as the changes since the first (initial) commit, assuming that it contains template code and should not be counted in the first release. Having the changes per release, we then calculated the total number of changed LOC for each window, which represents delivery size.

For measuring the average change lead time (CLT), we computed, for each commit, the time elapsed between its commit date and the date of the first release in which it was included. These commit-level CLTs were first averaged per release and subsequently averaged across all releases in a window to obtain a representative CLT value for each time slice.

Data filtering To ensure robust analysis of communication patterns, we excluded repositories that lacked the corresponding mode of collaboration. For instance, when studying pull request reviews, we removed projects that did not contain any reviews across their entire history. The same approach was applied for issue comments and pull request comments. In total, one repository was excluded from the issue comment analysis, 30 – from the PR comment analysis, and 91 – from the PR review analysis. This filtering step helped avoid skewed or misleading results from inactive or atypical repositories and ensured that only projects providing empirical evidence for the studied communication mode were included.

3.4 Data Analysis

After we transformed the raw repository data into numerical collaborative and delivery measures for non-overlapping time windows, we performed statistical analysis to study the potential relationships between them. The analysis consists of two stages — a correlation analysis to quantify the strength and direction of the associations, and a regression analysis to

provide quantitative insights into how collaborative characteristics affect delivery efficiency.

Correlation analysis The first step in our analysis was to quantify the strength and direction of the relationships between the independent and dependent variables using correlation analysis. Although correlation does not imply causation [12], this step allowed us to detect consistent patterns of association that might indicate potential influence between collaboration practices and delivery efficiency.

We employed *Kendall's rank correlation coefficient* (also called Kendall's τ), a non-parametric measure of ordinal association that evaluates the degree of monotonic relationships between two variables [33]. It was better suited for our data than other correlation measures, like Pearson's and Spearman's, due to its robustness to non-normality, tied ranks, and outliers [15; 34]. To interpret the strength of the correlations, we followed Cohen's criteria, which define correlations of magnitudes from 0.1 to 0.3 as *small*, 0.3 to 0.5 – as *medium*, and above 0.5 – as *large* [11; 20].

For each metric pair, we computed Kendall's τ per project over the time windows to assess the within-project variations via aggregate statistics. Additionally, we calculated the collective correlation for all repositories to capture the overall trend. The correlations and summary statistics were computed with *Python*'s *SciPy*, *NumPy*, and *pandas* libraries.

Regression analysis After examining the associations between collaboration and delivery using correlation analysis, we proceeded with regression analysis, which enables predictive modeling and more nuanced quantification of the relationships. In choosing a suitable statistical procedure, we acknowledged the non-trivial dependencies in our dataset due to the repeated measurements per project. Specifically, data points may exhibit project-level dependencies (e.g., teams with inherently higher release frequency), and the evolution of projects over time may lead to temporal trends (e.g., increased release activity in the early development stages).

To address these interdependencies and avoid assumptions about the normality of variable distributions or the linearity of potential relationships, we employed a *Generalized Additive Mixed Model* (GAMM), which combines the flexibility of *Generalized Additive Models* (GAMs) with the hierarchical modeling capabilities of *Mixed-Effects Models*. In particular, GAMMs support various distributional assumptions, nonlinear relationships (e.g., polynomials or splines), accommodate heteroscedasticity [49; 53], and are preferred over other statistical methods, like the Anova family, for longitudinal data with inherent dependencies [24; 25].

When defining the model, we differentiated between the fixed and random effects that would influence the dependent variables. Fixed effects capture the overall relationships between the collaboration characteristics and delivery indicators, assuming they hold consistently among all repositories. In contrast, random effects account for within-project and temporal variability, explaining unobserved heterogeneity and potential autocorrelation across time [14; 50].

Following procedures from prior studies, we defined the collaboration metrics and their interaction with time as fixed effects [24; 40]. We also included two random effects – a project-level intercept (to account for baseline differences between repositories) and a random slope for time (to model within-project temporal dynamics). Non-linear relationships were captured using smooth functions, and interactions between collaboration metrics and time were modeled as tensor product terms. The complete model specification is presented as a formula in *R* in Equation 1:

$$\mathrm{Y} \sim s(\mathrm{X}) + s(\mathrm{time}) + ti(\mathrm{X}, \mathrm{time}) + \big(1 + \mathrm{time} \mid \mathrm{repo}\big) \quad (1)$$

Here, Y is the delivery indicator, X – the collaboration measure, $time$ – the index of the time window per project, and $repo$ represents the grouping of repositories. Additionally, $s(\cdot)$ denotes a smooth function capturing non-linear effects, and $ti(X, time)$ is a tensor product interaction term. Upon examination of the dataset, we selected suitable distribution families for the model – delivery frequency was modeled with a quasi-Poisson distribution (due to overdispersion), while delivery size and change lead time were modeled with Gaussian distributions. All models were implemented in R using the *gamm()* function from the *mgcv* package.

4 Results

This section presents the results of the quantitative analysis, as outlined in Section 3.4. For each research question, we first report the results of the Kendall correlation analysis, which quantifies the association of collaboration with delivery efficiency both across and within projects. Subsequently, we present the results from the generalized additive mixed regression model, used to estimate the effect of collaboration metrics on delivery outcomes over time, accounting for project-level variability and temporal dependencies.

4.1 Team Size and Expertise (RQ1)

The first research question we address pertains to the effect of team size and expertise on delivery frequency and size. For each pair of a team metric and a delivery metric, descriptive statistics of the correlation analysis, both at a project level and collectively, are presented in Table 3.

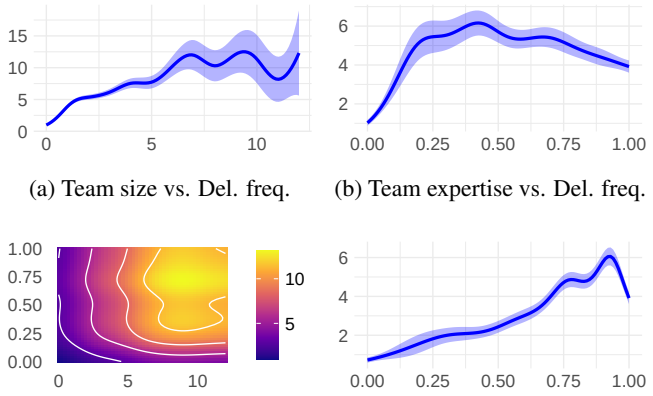
From the collective results, it can be observed that team size exhibits positive correlations with both delivery frequency ($\tau = 0.385,\ p < 0.001$) and size ($\tau = 0.386,\ p < 0.001$), which are classified as medium by Cohen's scale. Looking at the project-level correlations, most repositories show a pronounced positive association, with the median correlations exceeding 0.43. Team experience shows weaker but still noticeable correlations with delivery frequency ($\tau = 0.308,\ p < 0.001$) and size ($\tau = 0.263,\ p < 0.001$).

The results of applying the GAMM, presented in Figure 2 and Table 4, show that team size has a complex nonlinear effect on delivery frequency ($edf = 8.401,\ p < 0.001$). As can be seen in Figure 2a, the positive impact is more noticeable for small teams and plateaus at approximately seven to eight developers. For delivery size, the effect of team size is modeled as mostly linear ($edf = 1.000,\ p < 0.001$). Across both models, the positive $Xs(Time)$ values suggest that, when controlling for the team size, teams tend to deliver more frequently and in larger batches over time.

Table 3: Kendall correlations between team size (TS) / team expertise (TE) and delivery frequency (DF) / delivery size (DS).

Group	Mean	Std	Median	Combined Corr.
TS/DF	0.407	0.347	0.462	0.385^{M} (p<2e-16 ***)
TS/DS	0.385	0.326	0.436	0.386^{M} (p<2e-16 ***)
TE/DF	0.289	0.365	0.327	0.308^{M} (p<2e-16 ***)
TE/DS	0.229	0.367	0.258	0.263^{S} (p<2e-16 ***)

*Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' '*
Cohen's scale: L 'large', M 'medium', S 'small', - 'no correlation'



(a) Team size vs. Del. freq. (b) Team expertise vs. Del. freq.

(c) Size(x)/Exp.(y) vs. Del. freq. (d) Core contr. vs. Del. freq.

Figure 2: Estimated effects as modeled by GAMM. Figures (a), (b), and (d) show the fitted curves for delivery frequency as a function of a team metric. Figure (c) shows the curve for delivery frequency as a bivariate function of team size (x-axis) and expertise (y-axis).

Team expertise also influences delivery frequency nonlinearly ($edf = 7.632,\ p < 0.001$), with Figure 2b showing that the positive relationship is mostly present until an expertise level of 0.5 (i.e., active contributions in half of the time windows), after which it levels off. The interaction effect has a complex and slightly positive impact ($Xti \approx 0.891,\ edf = 10.456,\ p < 0.001$), suggesting that expertise becomes more influential in the later stages of a project. This higher importance may reflect the increasing value of deeper project knowledge and architectural familiarity, which become critical as codebases grow in complexity. For delivery size, the effects of team expertise are less pronounced, suggesting that it might be mostly driven by other project-specific characteristics, like complexity, tools, or user requirements.

When considering the joint effect of team size and expertise on delivery frequency, the GAMM results show that both predictors influence release behavior, each with distinct nonlinear patterns ($edf = 4.375,\ p < 0.001$ for size and $edf = 5.203,\ p < 0.001$ for expertise). From the contours of the plot in Figure 2c, we can observe that, as expected, to accelerate release cycles, teams can either include a larger number of less experienced developers or a smaller number of contributors with extensive expertise in the project.

Table 4: Summary of the GAMM results for the effects of team size (TS) and expertise (TE) on delivery frequency (DF) and size (DS).

Group	Term	edf	p-value	Xs/Xti
TS/DF	s(TS)	8.401	<2e-16 ***	1.437
	s(Time)	7.219	<2e-16 ***	1.127
	ti(TS,Time)	2.353	0.0928 .	0.007
TS/DS	s(TS)	1.000	<2e-16 ***	8465.96
	s(Time)	3.073	3.28e-07 ***	10160.35
	ti(TS,Time)	1.000	0.567	-49628.28
TE/DF	s(TE)	7.632	<2e-16 ***	-1.585
	s(Time)	7.281	<2e-16 ***	-1.214
	ti(TE,Time)	10.456	<2e-16 ***	0.891
TE/DS	s(TE)	2.565	1.67e-04 ***	618.61
	s(Time)	3.497	<2e-16 ***	-14505.57
	ti(TE,Time)	1.000	0.2137	22211.96
TS+TE/DF	s(TS)	4.375	<2e-16 ***	0.195
	s(TE)	5.203	<2e-16 ***	-0.748
	s(Time)	6.723	<2e-16 ***	0.803
	ti(TS,Time)	1.000	0.0066 **	2.095
	ti(TE,Time)	9.685	<2e-16 ***	0.583

*Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' '*

RQ1: *Both team size and expertise appear to positively affect delivery frequency and size, with the most pronounced effects observed for team size. The findings highlight the necessity of considering their joint effect, which reveals trade-offs and synergies that remain hidden when factors are analyzed in isolation.*

4.2 Core Contribution (RQ2)

The second research question explores the impact of core contribution on delivery frequency and size. Analogously to Section 4.1, we present descriptive statistics for the correlation analysis collectively and at a project level in Table 5.

The aggregated data reveal a positive but minor correlation between core contribution and delivery frequency ($\tau = 0.148,\ p < 0.001$) as well as delivery size ($\tau = 0.143,\ p < 0.001$). The median project-level correlations are around 0.2 for both delivery metrics, revealing that in most projects, greater core contribution is slightly associated with more frequent and larger deliveries. The variability of correlations across projects, as indicated by the standard deviations, suggests heterogeneity in the relationship across the repositories.

The results of the GAMM analysis, presented in Table 6, show that delivery frequency exhibits a complex nonlinear dependence on core contribution ($edf = 8.430,\ p < 0.001$), which decreases over time ($Xti = -0.512$). As illustrated

Table 5: Kendall correlations between core contribution (CC) and delivery frequency (DF) / delivery size (DS).

Group	Mean	Std	Median	Combined Corr.
CC/DF	0.167	0.385	0.200	0.148^S (p<2e-16 ***)
CC/DS	0.156	0.374	0.195	0.143^S (p<2e-16 ***)

*Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' '*
Cohen's scale: L 'large', M 'medium', S 'small', - 'no correlation'

Table 6: Summary of the GAMM results for the effects of core contribution (CC) on delivery frequency (DF) and size (DS).

Group	Term	edf	p-value	Xs/Xti
CC/DF	s(CC)	8.430	<2e-16 ***	-0.603
	s(Time)	7.543	<2e-16 ***	-1.398
	ti(CC,Time)	4.254	0.031 *	-0.512
CC/DS	s(CC)	1.000	5.89e-04 ***	4797.3
	s(Time)	3.276	<2e-16 ***	-11917.2
	ti(CC,Time)	1.000	0.263	22705.1

*Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' '*

in Figure 2d, the initially positive relationship diminishes beyond 75% core contribution, likely due to many single-developer projects, in which core contribution is 100% by definition, and which typically release less frequently due to limited development capacity. For delivery size, the overall effect of core contribution is positive ($Xs(CC) = 4797.3$), indicating that higher core contribution leads to more delivered code over time. The interaction term in the model is statistically insignificant, suggesting that the release size might be influenced more by other factors such as feature complexity, release policies, or architectural constraints, which are less dependent on core developer contribution.

RQ2: *Core contribution exhibits a modest positive association with delivery frequency, suggesting that core developers help sustain a steady release pace, whereas delivery size appears to be less affected.*

4.3 Communication Activities (RQ3)

The third research question examines the impact of issue and pull request interactions on change lead time. Table 7 presents the Kendall correlation coefficients for each relationship, both collectively and at a project level.

The combined Kendall correlations are statistically significant and suggest no associations between communication metrics and change lead time. Among the communication indicators, PR comments ($\tau = 0.070,\ p < 0.001$) and reviews ($\tau = 0.090,\ p < 0.001$) exhibit the highest τ-values,

Table 7: Kendall correlations between #issue comments (IC) / issue comments length (ICL) / #PR comments (PRC) / #PR reviews (PRR) and change lead time (CLT).

Group	Mean	Std	Median	Combined Corr.
IC/CLT	-0.046	0.353	-0.013	0.033^- (p=2.31e-06 ***)
ICL/CLT	-0.004	0.341	0.000	0.027^- (p=1.2e-04 ***)
PRC/CLT	-0.009	0.352	0.000	0.070^- (p<2e-16 ***)
PRR/CLT	-0.003	0.347	0.027	0.090^- (p<2e-16 ***)

*Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' '*
Cohen's scale: L 'large', M 'medium', S 'small', - 'no correlation'

Table 8: Summary of the GAMM results for the effects of #issue comments (IC), their average length (ICL), #PR comments (PRC), and #PR reviews (PRR) on change lead time (CLT).

Group	Term	edf	p-value	Xs/Xti
IC/CLT	s(IC)	2.413	1.6e-06 ***	-2324.3
	s(Time)	6.296	<2e-16 ***	-325.4
	ti(IC, Time)	1.000	0.661	18919.4
ICL/CLT	s(ICL)	1.000	0.171	-556.2
	s(Time)	6.240	<2e-16 ***	-953.5
	ti(ICL, Time)	1.854	0.783	13038.3
PRC/CLT	s(PRC)	1.000	0.095 .	-860.1
	s(Time)	6.243	<2e-16 ***	-1499.9
	ti(PRC, Time)	5.591	0.203	91210.3
PRR/CLT	s(PRR)	1.000	0.115	-967.7
	s(Time)	6.038	<2e-16 ***	-2981.2
	ti(PRR, Time)	1.000	0.301	60187.6

*Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' '*

but coefficients remain under 0.1 and indicate no correlation according to Cohen's criteria.

Table 8 reports the results of the GAMM analysis. The modeled relationships are weak and inconsistent, with most terms statistically insignificant at the 0.01 level. This suggests that, in some projects, more extensive communication may reflect underlying complexity or unresolved disagreements that hinder rather than accelerate change processing.

RQ3: *The analyzed communication metrics exhibit no noticeable associations with change lead time. The findings point out that the investigated static indicators of communication practices do not meaningfully explain variations in code release timelines and delivery cycles.*

5 Discussion

The results from the previous section highlighted how team composition and communication practices relate differently to software delivery efficiency in open-source projects. We now delve deeper into these findings, examining their dynamics and broader implications for collaboration in OSS development, alongside directions for future research.

Team size trade-off The empirical results underscored the significant role of team size in shaping delivery efficiency. The identification of an optimal team size around seven to eight developers aligns with past research suggesting that expanding teams may not always yield proportional gains. Instead, it calls for adaptive structuring to balance coordination costs, productivity, and agility, especially considering the nonlinear and diminishing returns observed as teams grow beyond this threshold. The findings suggest that while larger teams bring capacity, they might also require mechanisms – such as modularization or shared leadership – to mitigate the coordination overhead. Future research could build on these results by examining how task allocation, modularity, or intra-team dynamics moderate these trade-offs.

Temporal dynamics of expertise Team expertise exhibited a complex relationship with delivery performance, characterized by notable temporal effects. While moderate levels of expertise positively influenced delivery frequency, the effect became negative beyond 0.75, likely due to the presence of single-developer projects with a 100% level of expertise but limited release activity. The interaction with time revealed that expertise becomes more influential in later project phases. This may reflect the increasing need for deeper design and architectural knowledge as projects grow in complexity and maintenance becomes harder. The importance of long-term contributors also suggests that designing onboarding practices, which gradually transition contributors into expert roles, may help sustain release efficiency over time.

Flaws of communication volume The examined measures of communication activities showed no consistent associations with change lead time. This indicates that volume alone may not fully reflect productive coordination and could instead signal unresolved complexity or conflict, making it an insufficient proxy for the dynamics that influence delivery efficiency. These results highlight the limitations of static quantitative metrics of communication and point to the need for more nuanced analyses. Future studies could incorporate aspects such as the quality, tone, content, or sentiment of interactions, e.g., by leveraging natural language processing techniques, to better understand how coordination and conflict resolution influence performance.

Implications and future work The observed patterns indicate that team composition and expertise could inform not only human resource strategies but also technical aspects such as CI/CD pipeline design, potentially enabling more tailored automation that adapts to team characteristics. Contextual factors such as programming language, project domain and ownership, or platform-specific policies might further confound the studied associations, underscoring that multiple environmental and technological variables should be accounted for in future empirical research and practical applications. These findings and directions for future research collectively suggest that optimizing software delivery requires not only structuring teams effectively but also refining onboarding procedures and adapting CI/CD processes to the dynamic composition of development teams.

Threats to validity While this study offers empirical insights about OSS projects, several limitations must be considered. Internally, the study remains observational in nature, and although statistical techniques such as Kendall correlation and generalized additive mixed models were used to capture both linear and nonlinear trends, causal inferences cannot be drawn. Potential confounding factors, such as project governance models, contributor incentives, or organizational sponsorship, may also influence collaboration patterns and delivery performance. Moreover, the observed temporal effects, like the increasing influence of expertise, may reflect unmeasured changes in project tooling or coordination norms rather than inherent shifts in team dynamics.

Externally, the dataset consists of diverse active GitHub projects, but it may not fully represent the broader landscape of open-source or proprietary software development. Projects on alternative platforms, e.g., GitLab or Bitbucket, or with different collaboration structures – such as those governed through formal foundations or developed in private repositories – may exhibit other release patterns. Therefore, while the findings provide useful indications, they should be interpreted with caution when generalizing to different contexts.

6 Responsible Research

In conducting this study, we followed established practices for responsible research to ensure transparency, validity, and reproducibility of our methodology and findings. Specifically, we adhered to the principles outlined in the *Netherlands Code of Conduct for Research Integrity*: honesty, scrupulousness, transparency, independence, and responsibility [52]. The remainder of this section elaborates on specific measures taken to uphold these core values.

Data collection We constructed our dataset by identifying and extracting information from publicly available GitHub repositories, accessed through the official public GitHub REST API. All analyzed projects are open-source and meet our inclusion criteria, as described in Section 3.2. No sensitive data or one with restricted access was used at any stage of this research. The full repository dataset, along with the computed metrics and the exact dates of data retrieval for each project, is available in a publicly archived dataset on Zenodo[1].

Reproducibility The complete data analysis process, including the computation of all metrics and statistical procedures, is documented in Section 3. To support independent verification and enable further building on our findings, we publicly share all code related to data collection and metrics computation at an archived repository on Zenodo[2].

[1]https://doi.org/10.5281/zenodo.15681547

[2]https://doi.org/10.5281/zenodo.15711350

Integrity and independence This research was conducted without external influence from commercial or institutional stakeholders. The analyses and interpretations were performed in an unbiased and independent manner, with all findings and conclusions being based on evidence presented in the data, without distortion, manipulation, or selective interpretation. We maintained full integrity throughout the research process to ensure that the results accurately reflect the observations and conducted analyses.

7 Conclusions

This empirical study examined 887 active GitHub repositories to explore how team structure and communication practices relate to delivery efficiency in open-source software development. The results indicate that team size is the most influential factor, with delivery frequency increasing up to an optimal threshold of around seven contributors, beyond which returns diminish. Developer expertise and core contributor activity are also linked to shorter release cycles; however, while the influence of core contribution weakens as projects mature, the effect of developer expertise becomes more pronounced over time. In contrast, static measures of communication activity showed no noticeable association with change lead time, suggesting that volume alone may not capture coordination effectiveness. These findings underscore the importance of aligning collaboration strategies with project phase and delivery goals, and point to the need for more nuanced representations of communication activities. Future research could expand the dataset across different collaboration platforms, incorporate additional contextual variables, and apply causal analysis for deeper insights. Overall, the study provides evidence that collaboration modes shape delivery outcomes in complex and temporally evolving ways, with implications for how teams are structured and coordinated in OSS environments.

References

[1] Kamel Alrashedy and Ahmed Binjahlan. How do software engineering researchers use github? an empirical study of artifacts & impact, 2024.

[2] Idan Amit and Dror G. Feitelson. The corrective commit probability code quality metric, 2020.

[3] Deeksha Arya, Wenting Wang, Jin L. C. Guo, and Jinghui Cheng. Analysis and detection of information types of open source software issue discussions, 2019.

[4] Uschi Backes-Gellner, Arndt Werner, and Alwine Mohnen. Team size and effort in start-up-teams – another consequence of free-riding and peer pressure in partnerships. *University of Zurich, Institute for Strategy and Business Economics (ISU), Working Papers*, 01 2004.

[5] Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Brian Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas. Manifesto for agile software development. https://agilemanifesto.org/, 2001. Accessed: 2025-05-01.

[6] Ethan Bommarito and Michael Bommarito. An empirical analysis of the python package index (pypi), 2019.

[7] Frederick Brooks, Jr. *The Mythical Man-Month: Essays on Software Engineering*. Reading, Mass.:Addison-Wesley Pub. Co., 01 1975.

[8] Fabio Calefato, Marco Aurélio Gerosa, Giuseppe Iaffaldano, Filippo Lanubile, and Igor Steinmacher. Will you come back to contribute? investigating the inactivity of oss core developers in github. *Empirical Software Engineering*, 27(3), March 2022.

[9] Antonio Cesar Brandão Gomes da Silva, Glauco De Figueiredo Carneiro, Fernando Brito e Abreu, and Miguel Pessoa Monteiro. Frequent releases in open source software: A systematic review. *Information*, 8(3), 2017.

[10] Jailton Coelho, Marco Tulio Valente, Luciana L. Silva, and André Hora. Why we engage in floss: Answers from core developers. In *Proceedings - International Conference on Software Engineering*, page 114 – 121, 2018.

[11] Jacob Cohen. *Statistical Power Analysis for the Behavioral Sciences*. Routledge, rev. ed. edition, 1988.

[12] Jacob Cohen, Patricia Cohen, Stephen G. West, and Leona S. Aiken. *Applied Multiple Regression/Correlation Analysis for the Behavioral Sciences, Third Edition*. Applied Multiple Regression/Correlation Analysis for the Behavioral Sciences, Third Edition, 2013.

[13] Kevin Crowston and James Howison. The social structure of free and open source software development. *First Monday*, 2005.

[14] Reza Dastranj and Martin Kolář. Age-gender-country-specific death rates modelling and forecasting: a linear mixed-effects model. *Scandinavian Actuarial Journal*, page 1–16, April 2025.

[15] Giuseppe Destefanis, Giulio Concas, Michele Marchesi, and Roberto Tonelli. An empirical study of software metrics for assessing the phases of an agile project. *International Journal of Software Engineering and Knowledge Engineering*, 22, 06 2012.

[16] DevOps Research and Assessment (DORA). Accelerate state of devops report 2024. https://dora.dev/research/2024/dora-report/, 2024. Accessed: 2025-04-25.

[17] Óscar Dieste Tubío, Alejandrina Aranda, Fernando Uyaguari, Burak Turhan, Ayse Tosun, Davide Fucci, Markku Oivo, and Natalia Juristo. Empirical evaluation of the effects of experience on code quality and programmer productivity: an exploratory study. In *ICSSP '18: Proceedings of the 2018 International Conference on Software and System Process*, pages 111–112, 05 2018.

[18] J. Alberto Espinosa, Sandra A. Slaughter, Robert E. Kraut, and James D. Herbsleb. Familiarity, complexity, and team performance in geographically distributed software development. *Organization Science*, 18(4):613–630, 2007.

[19] Nicole Forsgren. Octoverse spotlight: An analysis of developer productivity, work cadence, and collaboration in the early days of covid-19. Link, May 2020. Accessed: 2025-04-25.

[20] Gilles E. Gignac and Eva T. Szodorai. Effect size guidelines for individual differences researchers. *Personality and Individual Differences*, 102:74–78, 2016.

[21] Geoffrey K. Gill and Chris F. Kemerer. Productivity impacts of software complexity and developer experience. *Sloan Working Papers*, 1990.

[22] GitHub. About releases. https://docs.github.com/en/repositories/releasing-projects-on-github/about-releases, 2022. Accessed: 2025-05-01.

[23] GitHub Staff. Octoverse: Ai leads python to top language as the number of global developers surges. https://github.blog/news-insights/octoverse/octoverse-2024, October 2024. Accessed: 2025-04-27.

[24] Daniel Graziotin, Xiaofeng Wang, and Pekka Abrahamsson. Are happy developers more productive? the correlation of affective states of software developers and their self-assessed productivity. *CoRR*, abs/1306.1772, 2013.

[25] Ralitza Gueorguieva and John H. Krystal. Move over anova: Progress in analyzing repeated-measures data and its reflection in papers published in the archives of general psychiatry. *Archives of General Psychiatry*, 61(3):310 – 317, 2004.

[26] Junxiao Han, Shuiguang Deng, Xin Xia, Dongjing Wang, and Jianwei Yin. Characterization and prediction of popular projects on github. In *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, volume 1, pages 21–26, 2019.

[27] Les Hatton, Diomidis Spinellis, and Michiel van Genuchten. The long-term growth rate of evolving software: Empirical results and implications. *Journal of Software: Evolution and Process*, 29(5), 2017.

[28] Marjan Hericko, Aleš Živkovič, and Ivan Rozman. An approach to optimizing software development team size. *Information Processing Letters*, 108:101–106, 10 2008.

[29] Michael Hilton, Timothy Tunnell, Kai Huang, Darko Marinov, and Danny Dig. Usage, costs, and benefits of continuous integration in open-source projects. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, ASE '16, page 426–437, New York, NY, USA, 2016. Association for Computing Machinery.

[30] Hadhemi Jebnoun, Houssem Ben Braiek, Mohammad Masudur Rahman, and Foutse Khomh. The scent of deep learning code: An empirical study. In *Proceedings of the 17th International Conference on Mining Software Repositories*, MSR '20, page 420–430, New York, NY, USA, 2020. Association for Computing Machinery.

[31] Mitchell Joblin, Barbara Eckl, Thomas Bock, Angelika Schmid, Janet Siegmund, and Sven Apel. Hierarchical and hybrid organizational structures in open-source software projects: A longitudinal study. *ACM Trans. Softw. Eng. Methodol.*, 32(4), May 2023.

[32] Eugene Kandel and Edward P. Lazear. Peer pressure and partnerships. *Journal of Political Economy*, 100(4):801–817, 1992.

[33] M. G. Kendall. A new measure of rank correlation. *Biometrika*, 30(1/2):81–93, 1938.

[34] Md Abdullah Mamun, Christian Berger, and Jörgen Hansson. Effects of measurements on correlations of software code metrics. *Empirical Software Engineering*, 24, 08 2019.

[35] Mariam Mezouar, Feng Zhang, and Ying Zou. An empirical study on the teams structures in social coding using github projects. *Empirical Software Engineering*, 24, 12 2019.

[36] Joao Eduardo Montandon, Luciana Lourdes Silva, and Marco Tulio Valente. Identifying experts in software libraries and frameworks among github users. In *IEEE International Working Conference on Mining Software Repositories*, volume 2019-May, page 276 – 287, 2019.

[37] Kumiyo Nakakoji, Yasuhiro Yamamoto, Yoshiyuki Nishinaka, Kouichi Kishida, and Yunwen Ye. Evolution patterns of open-source software systems and communities. In *International Workshop on Principles of Software Evolution (IWPSE)*, page 76 – 85, 2002.

[38] L. Osterweil. Software processes are software too. In *Proceedings of the 9th International Conference on Software Engineering*, ICSE '87, page 2–13, Washington, DC, USA, 1987. IEEE Computer Society Press.

[39] Quentin Perez, Christelle Urtado, and Sylvain Vauttier. Dataset of open-source software developers labeled by their experience level in the project and their associated software metrics. *Data in Brief*, 46, 2023.

[40] Leo Polansky and Martha Robbins. Generalized additive mixed models for disentangling long-term trends, local anomalies, and seasonality in fruit tree phenology. *Ecology and Evolution*, 3:3141–3151, 09 2013.

[41] Lawrence H. Putnam. A general empirical solution to the macro software sizing and estimating problem. *IEEE Transactions on Software Engineering*, SE-4(4):345 – 361, 1978.

[42] Mohammad Masudur Rahman and Chanchal K. Roy. An insight into the pull requests of github. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, ICSE '14, page 364–367. ACM, May 2014.

[43] Max Ringelmann. Recherches sur les moteurs animés: Travail de l'homme. *Annales de l'Insitut National Agronomique*, 12(1):1–40, 1913.

[44] Gregorio Robles, Jesus M. Gonzalez-Barahona, and Israel Herraiz. Evolution of the core team of developers in libre software projects. In *Proceedings of the 2009 6th IEEE International Working Conference on Mining Software Repositories, MSR 2009*, page 167 – 170, 2009.

[45] D. Rodríguez, M.A. Sicilia, E. García, and R. Harrison. Empirical findings on team size and productivity in software development. *Journal of Systems and Software*, 85(3):562–570, 2012. Novel approaches in the design and implementation of systems/software architecture.

[46] Ingo Scholtes, Pavlin Mavrodiev, and Frank Schweitzer. From aristotle to ringelmann: a large-scale analysis of team productivity and coordination in open source software projects. *Empirical Softw. Engg.*, 21(2):642–683, April 2016.

[47] Didier Sornette, Thomas Maillart, and Giacomo 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*, 9(8), 2014.

[48] Margaret-Anne Storey, Alexey Zagalsky, Fernando Figueira Filho, Leif Singer, and Daniel M. German. How social and communication channels shape and challenge a participatory culture in software development. *IEEE Transactions on Software Engineering*, 43(2):185–204, 2017.

[49] Feiyang Sun, Peng Chen, and Junfeng Jiao. Promoting public bike-sharing: A lesson from the unsuccessful pronto system. *Transportation Research Part D: Transport and Environment*, 63:533 – 547, 2018.

[50] Zachary Townsend, Sean P Jack Buckley, Masataka Harada, and Marc A. Scott. *The choice between fixed and random effects*, pages 73–88. SAGE Publications Inc., January 2013.

[51] Bogdan Vasilescu, Yue Yu, Huaimin Wang, Premkumar Devanbu, and Vladimir Filkov. Quality and productivity outcomes relating to continuous integration in github. In *2015 10th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, ESEC/FSE 2015 - Proceedings*, page 805 – 816, 2015.

[52] Netherlands code of conduct for research integrity. https://doi.org/10.17026/dans-2cj-nvwu, 2018. Accessed: 2025-05-27.

[53] Simon N Wood. *Generalized additive models: an introduction with R*. chapman and hall/CRC, 2017.

[54] Ahmmad Youssef and Andrea Capiluppi. The impact of developer team sizes on the structural attributes of software. In *International Workshop on Principles of Software Evolution (IWPSE)*, volume 30-Aug-2015, page 38 – 45, 2015.

[55] Yangyang Zhao, Alexander Serebrenik, Yuming Zhou, Vladimir Filkov, and Bogdan Vasilescu. The impact of continuous integration on other software development practices: A large-scale empirical study. In *ASE 2017 - Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*, page 60 – 71, 2017.