Divergence or Convergence? A Deep Insight into the Crowd Collaboration and its Productivity in Open Source Software based on Entropy

Yang Shen
College of Computer
National University of Defense Technology
Changsha, Hunan, China
State Key Laboratory of Complex & Critical Software Environment
Changsha, Hunan, China
shenyang@nudt.edu.cn

Tao Wang**
College of Computer
National University of Defense Technology
Changsha, Hunan, China
State Key Laboratory of Complex & Critical Software Environment
Changsha, Hunan, China
taowang2005@nudt.edu.cn

Xunhui Zhang
College of Computer
National University of Defense Technology
Changsha, Hunan, China
State Key Laboratory of Complex & Critical Software Environment
Changsha, Hunan, China
zhangxunhui@nudt.edu.cn

Yang Zhang
College of Computer
National University of Defense Technology
Changsha, Hunan, China
State Key Laboratory of Complex & Critical Software Environment
Changsha, Hunan, China
yangzhang15@nudt.edu.cn

Cheng Yang
College of Computer
National University of Defense Technology
Changsha, Hunan, China
State Key Laboratory of Complex & Critical Software Environment
Changsha, Hunan, China
delpiero710@126.com

Bo Ding
College of Computer
National University of Defense Technology
Changsha, Hunan, China
dingbo@nudt.edu.cn

Huaimin Wang
College of Computer
National University of Defense Technology
Changsha, Hunan, China
State Key Laboratory of Complex & Critical Software Environment
Changsha, Hunan, China
whm_w@163.com

Abstract

The Fork and Pull-Request model is widely used in collaborative development of open source software (OSS), fostering innovation through independent repository copies, but it can also lead to inefficiencies and fragmentation. A key underexplored aspect is the integration effectiveness—the degree to which distributed original commits across forks are effectively integrated back into the main repository. It plays a critical role in OSS project productivity but remains poorly understood. In response, we introduce convergence entropy, a novel metric that quantifies the integration effectiveness by measuring the similarity between distributions of original and merged commits across forks, adjusted for integration ratio. This metric highlights not only the volume of contributions but also their diversity and coordination, offering a unique lens to understand forking practices. Moreover, we explore the relationship between convergence entropy and three dimensions of OSS project productivity, showing significant correlations. We also observe that other factors can alter this dynamic.

CCS Concepts

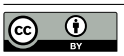
• **Human-centered computing** → **Empirical studies in collaborative and social computing**.

Keywords

Open Source Software, Fork, Convergence Entropy, Productivity

ACM Reference Format:
Yang Shen, Tao Wang, Xunhui Zhang, Yang Zhang, Cheng Yang, Bo Ding, and Huaimin Wang. 2026. Divergence or Convergence? A Deep Insight into the Crowd Collaboration and its Productivity in Open Source Software based

*Corresponding Author(taowang2005@nudt.edu.cn)



CHI '26, Barcelona, Spain

ACM ISBN 979-8-4007-2278-3/26/04
https://doi.org/10.1145/3772318.3791405

on Entropy. In *Proceedings of the 2026 CHI Conference on Human Factors in Computing Systems (CHI '26), April 13–17, 2026, Barcelona, Spain.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3772318.3791405

1 Introduction

Forks play a central role in modern pull-based Open Source Software (OSS) development[26, 61]. In the dynamic ecosystem of OSS, forking enables developers to create independent copies of a repository, fostering innovation, experimentation, bug fixes, and feature additions that can potentially be integrated back into the main project via mechanisms like pull requests [30]. This distributed approach promotes collaboration across global communities, allowing diverse ideas to evolve in parallel and enriching the project's overall development trajectory. Successful OSS projects often exhibit high fork activity, which is widely regarded as a marker of popularity and vitality [11, 12, 59].

However, the outcomes of forking are not always aligned with seamless integration. On one hand, forks intended for contributions to the main repository can suffer from inefficiencies, such as lost, rejected, or redundant efforts, as highlighted in existing research on suboptimal forking practices [34, 68]. These issues can waste developer time and hinder project progress, even in projects with a large number of forks—which is often seen as a sign of successful OSS projects but does not guarantee productivity or health. On the other hand, not all forks are created with the goal of merging back into the main repository; some are established for independent development, leading to divergent paths that may compete with the original project. This competition, while stimulating the broader ecosystem by fostering innovation and community engagement[51], can have a negative impact on the main repository's development [22, 44]. Developer attrition can occur as contributors may prefer to focus on their own forks or alternative forks rather than continuing to contribute to the main repository [16, 50, 69]. As a result, this fragmentation can weaken the cohesion of the main repository community, potentially stalling its progress.

Addressing these challenges requires a systematic approach to measure how effectively a project integrates contributions from its forks. Existing research reveals two key limitations in this regard. First, current metrics for assessing fork integration are often static, coarse-grained, or focused on isolated inefficiency types (e.g., lost contributions, duplicate PRs [34]). They lack a unified, time-sensitive measure that quantifies the alignment between the distribution of original development activity across forks and the distribution of successfully integrated contributions. Second, while prior work has analyzed correlations between fork-related metrics and project characteristics [68], the longitudinal relationship between the integration effectiveness and core project productivity outcomes remains underexplored.

To address these gaps, we propose a novel metric called **Convergence Entropy to quantify the integration effectiveness of fork practice** in OSS repositories. We conceptualize fork practice of OSS as distributed collaboration system that incorporate both decentralized contributions and centralized integration workflows. Our approach uses Jensen-Shannon divergence to measure the discrepancy between two key distributions derived from fork practice: (1) the distribution of original commits across all forks, which reflects decentralized innovation and effort, and (2) the distribution of merged commits across all forks, which indicates successful integration. These two key distributions capture how well a project harnesses fork-based activities to achieve unified progress. Simultaneously, it incorporates the ratio of merged to original commits to account for the overall efficiency of convergence. This dual approach allows Convergence Entropy to encapsulate the multiplex nature of convergence—emphasizing not only the volume but also the diversity and coordination of contributions across forks. As suggested in prior work, diverse and well-coordinated contribution patterns are critical for addressing challenges in OSS development [5]. Convergence Entropy offers a nuanced perspective, project managers can identify patterns of successful integration, understand temporal changes in fork utilization, and devise strategies to encourage productive forking while minimizing risks like community splits.

Furthermore, to investigate the relationship between integration effectiveness and project productivity, we employ Convergence Entropy as a proxy for integration effectiveness. Following Forsgren's framework [15], we select three indicators representing **project productivity dimensions: output performance (new bugs), developer activity quantity (new commits), and team process efficiency (issue resolution time)**. Our empirical analysis draws on commit and fork data from eight prominent GitHub OSS projects, encompassing both corporate-driven and community-driven ones. We incorporate control variables such as the number of forks (NumForks), issues (NumIssues), pull requests (NumPrs), files (NumFiles), core developers (NumCore), external developers (NumExt), and project age (Project Age). The rationale for these choices is detailed in 4.1. This study is guided by the following research questions.

- **RQ1:** *What is the correlation between convergence entropy and the number of new bugs in OSS projects, and what factors could alter this dynamic?*

The number of new bugs is commonly served as a proxy for evaluating software quality [45, 59, 62]. We use this metric to capture project productivity in the Performance dimension, assessing the outcome of the software development process [15]. Our results indicate that convergence entropy is significantly negatively correlated with the number of new bugs. This correlation strengthens with project age and the involvement of core teams in non-company projects but weakens with more issues, files, and external contributors. In contrast, in company projects, the correlation strengthens with external contributors but weakens with more forks, issues, and files.

- **RQ2:** *How is convergence entropy correlated with the number of new commits in OSS projects, and what factors may affect this connection?*

In many studies focusing on OSS development and community, the number of commits is considered as the productivity metric [45, 57, 58]. We adopt this widely used productivity indicator to capture project productivity in the Activity dimension, assessing the output of the software development process [15]. Our findings show that convergence entropy has a distinct correlation with new commits: it is positive in non-company projects but negative in company projects. In non-company projects, the positive correlation strengthens with project age, pull requests, and external

contributors but weakens with more issues, files, and core developers. Conversely, in company projects, the negative correlation strengthens with project age but weakens with more forks, issues, and pull requests.

- **RQ3:** *To what extent does convergence entropy correlate with issue resolution time in OSS projects, and what factors could strengthen or weaken this association?*

The issue resolution time is also regarded as an important metric for productivity [17, 41]. We adopt this metric to capture project productivity in the Efficiency & Flow dimension, assessing the efficiency of the software development process [15]. Regression analysis demonstrates a contrasting correlation with issue resolution time: positive in non-company projects and negative in company projects. In non-company projects, this association strengthens with the number of files but weakens with more forks and issues. In company projects, more forks and external contributors strengthen the reduction in issue resolution time but weakens with more core members.

This article aims to systematically quantify integration effectiveness via convergence entropy and examine their relationships with three dimensions of OSS project productivity. The overall framework of this paper is shown in Figure 1. Through the above studies, we conclude that the integration effectiveness plays a crucial role in understanding the development of OSS projects. The proposed metric serves as an effective and useful indicator for assessing the integration of distributed contributions in fork practices, showing significant correlations with project productivity indicators related to OSS development and maintenance.

In summary, this work makes the following contributions.

- We propose convergence entropy to measure the integration effectiveness in OSS repositories based on fork commit distributions, offering a novel perspective on integration dynamics.
- We employ convergence entropy to study integration dynamics and analyze its correlations with new bugs, new commits, and issue resolution time as proxies for OSS productivity dimensions. Our findings highlight the metric's role in elucidating project productivity.
- We discuss implications of convergence entropy for managing forks and guiding OSS practices.

The rest of the paper is organized as follows. Section 2 presents the related work. In Section 3, we detail the calculation of Convergence Entropy and its properties. Section 4 describes the study design. We report results and findings in Section 5. In Section 6, we discuss implications and threats to validity. Finally, Section 7 concludes and suggests future work.

2 Related Work

This section situates our research within the broader landscape of OSS development. We begin by discussing the fundamental challenges of coordinating distributed development in OSS, highlighting why the effective integration of code contributions is a critical concern. This naturally leads to a focus on the forking mechanism and the concept of convergence, which refers to how well contributions scattered across forks are integrated back. We then synthesize the limitations of prior work and clearly articulate the novel contributions of our study, supported by a structured comparison of key metrics.

2.1 Challenges in OSS Development and the Centrality of Code Integration

Successful OSS projects rely on effectively coordinating contributions from a distributed, often volunteer-based community [13]. Key challenges include maintaining code quality [36], managing a high volume of contributions [21], fostering community cohesion, and ensuring sustainable development pace [52]. Among these, the integration of external code into a stable mainline is a critical bottleneck; inefficient integration can lead to lost contributions, duplicated effort, community fragmentation, and ultimately, reduced project vitality [20, 68]. The widely adopted fork-and-pull model, enabled by platforms like GitHub, formalizes this integration process by allowing developers to create independent copies of a repository. While this mechanism lowers the barrier to contribution and fosters innovation [30], it also decentralizes development. Therefore, understanding and measuring how well a project converges, that is, how effectively it integrates distributed contributions from its forks, is essential for assessing the health and efficiency of OSS development.

2.2 Forking Mechanisms and the Pursuit of Convergence

The forking mechanism is a double-edged sword. On one hand, it enables parallel experimentation, bug fixes, and feature development [30]. On the other, it introduces several documented inefficiencies: lost contributions, rejected pull requests, duplicate development efforts, and community fragmentation [68]. These inefficiencies are influenced by project characteristics such as modularity, coordination style, and the integrators' trade-off between quality assurance and contribution throughput [20]. Furthermore, forks exhibit diverse intents. They can be classified as social (aiming to contribute back), divergent (no intention to merge), or hard forks (pursuing independent development) [8, 22, 44, 69]. This diversity can lead to effort diffusion, where secondary forks contribute to other forks but not the original repository, causing gradual community drift [69].

Substantial research has aimed at mitigating these inefficiencies and improving convergence. Studies have proposed tools to increase transparency by identifying unmerged features in forks [67], frameworks to predict potential PR contributions from forks early [65], and methods to predict the long-term active integrators [27]. These efforts underscore the community's focus on enhancing the integration pipeline. Importantly, prior work also notes that a lower level of direct integration is not always detrimental; it can reflect a project's specific governance structure or a conscious trade-off. For example, some projects exhibit vibrant ecosystems with many independent forks, yet still absorb significant contributions from them [50]. This nuance suggests the need for a nuanced measure of integration effectiveness that goes beyond simple counts of merged PRs.

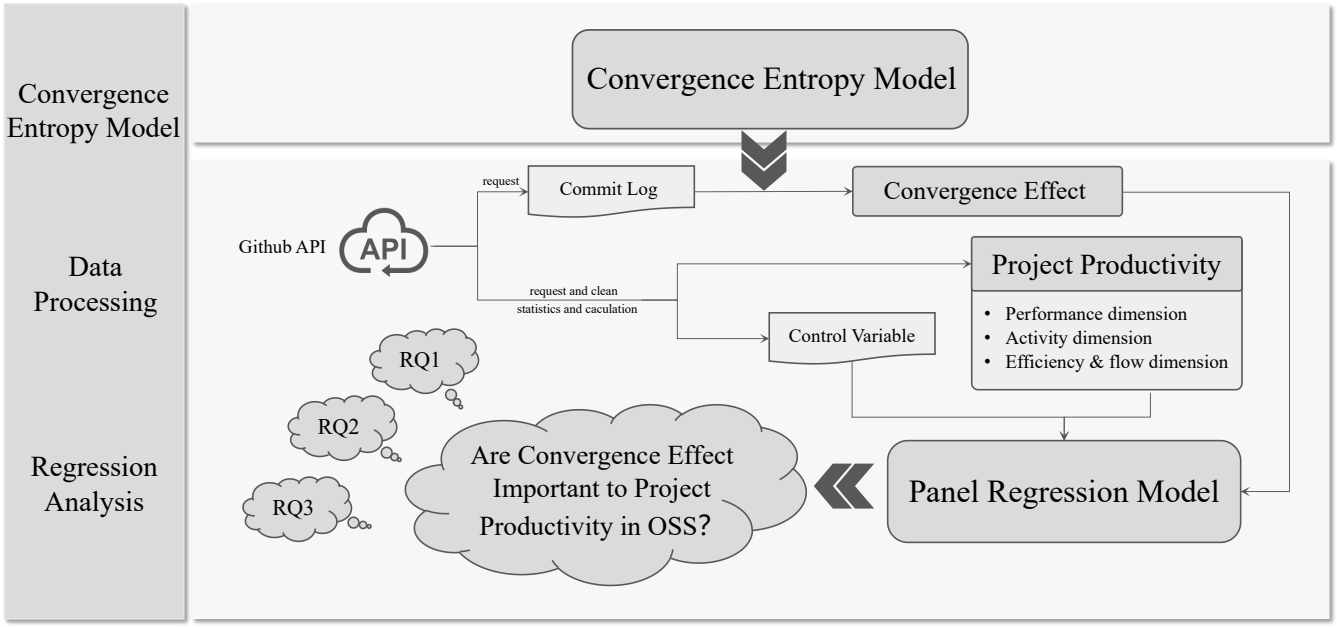


Figure 1: Research framework of this paper

However, a key limitation of much prior work is its static or cross-sectional nature. For instance, Zhou et al. [68] quantified several inefficiency metrics (e.g., ratio of contributing forks, presence of hard forks) at a single point in time or over a whole project history. While their regression analysis related these metrics to project characteristics and practices, the data did not capture how integration dynamics evolve over time. Metrics like "presence of hard forks" are inherently non-time-sensitive; their value over a time window is not meaningful. This limits the understanding of how integration efficiency fluctuates and interacts with project outcomes longitudinally. An exception is the fork entropy proposed by Wang et al. [61], which is calculated as the average pairwise distance between files modification vectors of forks within a time window. While this metric is time-sensitive, it remains unrelated to integration practice because it does not consider whether the modifications are integrated back to the mainline. To clearly differentiate our contribution from key prior works, we present a comparative summary in Table 1.

Our review reveals two interconnected gaps. First, existing metrics for fork integration are often static, coarse-grained, or focused on isolated inefficiency types (e.g., lost contributions, duplicate PRs). They lack a unified, time-sensitive measure that quantifies the alignment between the distribution of original development activity across forks and the distribution of successfully integrated contributions. Second, while prior work has analyzed correlations between fork-related metrics and project characteristics [68] or main repository commit volume [61], the longitudinal relationship between the effectiveness of convergence and core project productivity outcomes remains underexplored.

Our work directly addresses these gaps. We propose Convergence Entropy, a novel, time-window-based metric that quantifies the degree to which the distribution of original commits across forks is effectively mirrored in the distribution of merged commits into the main repository. It combines a measure of distributional alignment (using Jensen-Shannon Divergence) with a scaling factor for merge ratio, producing a single scalar that reflects integration effectiveness within a given time period. Crucially, this design makes it inherently time-sensitive, suitable for longitudinal analysis.

3 Convergence Entropy

In this section, we introduce convergence entropy as a metric to quantify the global integration effectiveness in open-source software repositories based on fork activities. We first present the information-theoretic foundation, then we present the calculation details of convergence entropy and discuss its behavior in various scenarios. Additionally, we describe its application in analyzing the integration of external contributions.

3.1 Information Theory Foundation

Information theory provides tools for quantifying similarity and divergence between distributions [48]. A key measure is the Jensen-Shannon Divergence(JSD), which is a symmetric and smoothed variant of Kullback-Leibler Divergence(KLD), defined for two probability distributions P and Q as:

$$JSD(P||Q) = \frac{1}{2}KLD(P||M) + \frac{1}{2}KLD(Q||M)$$

where $M = \frac{1}{2}(P + Q)$ and KLD is the Kullback-Leibler divergence:

$$KLD(P||Q) = \sum_i P(i) \log_2 \frac{P(i)}{Q(i)}$$

Table 1: Comparison of fork-related metrics in prior work and this study

Metric	Core Formula (Informal)	What it Primarily Measures	Limitations w.r.t. Integrated Practice
Ratio of contributing forks [68]	$\frac{\#\text{contributing forks}}{\#\text{active forks}}$	Inefficiencies regarding lost contributions	Does not consider how much each fork contributes, nor how integrated contributions are distributed across forks.
Ratio of merged PRs [68]	$\frac{\#\text{merged PRs}}{\#\text{closed PRs}}$	Inefficiencies regarding rejected PRs	Treats all PRs equally and independent of their fork of origin, ignores the distribution of integrated contributions across forks.
Ratio of duplicate PRs [68]	$\frac{\#\text{duplicate PRs}}{\#\text{closed PRs}}$	Inefficiencies regarding duplicate development	Treats all PRs equally and independent of their fork of origin, ignores the distribution of integrated contributions across forks.
Presence of hard forks [68]	Indicator variable for hard fork presence	Inefficiencies regarding community fragmentation	Coarse-grained binary signal; does not provide a continuous measure of how integrated contributions are over time.
Fork entropy [61]	$\frac{1}{m^2}\sum_i \sum_j D(c_i, c_j)$ where D is the distance between files modification vectors of two forks	Fork diversity, measured by the average pairwise distance between their files modification vectors within a time window	Unrelated to integration practice; it ignores whether modifications are integrated back to the mainline.
Convergence Entropy (this work)	$(1 - \text{JSD}(P_o \parallel P_m)) \cdot \frac{C_m}{C_o}$ where P_o and P_m are the distributions of original and merged commits across forks	The degree to which distributed original commits across forks are effectively integrated back into the main repository; quantified by the alignment between the distributions of original and merged commits within a time window, scaled by the merge ratio.	Does not distinguish specific inefficiency types; instead provides a unified, time-sensitive scalar indicator that complements the above specialized metrics.

JSD ranges from 0 (identical distributions) to 1 (maximal difference) and is widely used in various fields to measure the distance between two distributions[19, 31].

3.2 Definition and Calculation of Convergence Entropy

We model the fork practice of OSS as a distributed collaboration system that encompasses decentralized contributions and centralized integration workflows. In this framework, forks produce original commits (decentralized innovation), with some of these commits being merged back into the main repository (effective integration). Drawing from information theory, convergence entropy quantifies this integration effectiveness by measuring the similarity between two key distributions: the distribution of original commits across all forks and the distribution of merged commits from those forks. This metric captures the extent to which decentralized contributions are successfully integrated into the main repository.

A specific example illustrates the calculation. Consider a project with multiple forks. For a given time window (e.g., a month), we extract:

- **The original commit distribution:** For each fork f_k, count the number of original commits, normalized to form probability distribution $P_o = \{p_o(f_1), p_o(f_2), \ldots, p_o(f_k), \ldots\}$, where $\sum p_o(f_k) = 1$.
- **The merged commit distribution:** For each fork f_k, count the number of commits merged into the main repository, normalized to form $P_m = \{p_m(f_1), p_m(f_2), \ldots, p_m(f_k), \ldots\}$, where $\sum p_m(f_k) = 1$.

These distributions reflect the decentralized nature of fork activities, where original commits represent the breadth of innovation across contributors, and merged commits indicate which innovations are successfully integrated. The distribution of original commits highlights how effort is spread among forks, revealing patterns of independent development or focused contributions from specific forks. Meanwhile, the merged commit distribution shows the selective integration process, emphasizing which forks' efforts are valued and adopted by the main project. The alignment (or misalignment) between these distributions is significant because it reveals the efficiency of knowledge transfer in OSS ecosystems. For example, if original commits are evenly distributed but merges

favor only a few forks, it may indicate barriers to entry for new contributors or biases in review processes. Conversely, close alignment suggests a meritocratic system where diverse innovations are proportionally recognized and incorporated, fostering inclusivity and project vitality. These distributions provide insights into collaborative dynamics, such as competition among forks, the role of core maintainers in curation, and the overall health of the fork ecosystem.

Convergence entropy is then defined as:

$$H_{conv} = (1 - \text{JSD}(P_o||P_m)) \times \left(\frac{C_m}{C_o}\right)$$

Where $\text{JSD}(P_o||P_m)$ is the Jensen-Shannon divergence between P_o and P_m, measuring dissimilarity (JSD ranges from 0 to 1, with 0 indicating identical distributions). C_o is the total number of original commits across all forks in the time window. C_m is the total number of merged commits from all forks in the time window. $(1 - \text{JSD})$ captures the alignment between decentralized innovation and effectively integration, while $\frac{C_m}{C_o}$ is efficiency factor and scales it by the overall merge ratio. This is computed periodically (e.g., monthly) to track temporal changes in repository dynamics.

Convergence entropy provides insights into repository dynamics, particularly at the extremes of the entropy spectrum. When entropy is minimal, it implies fragmented or inefficient integration, where original commits are dispersed across many forks but only a small fraction or a skewed subset are merged. For instance, if numerous forks produce innovative commits but merges are dominated by a single fork (low 1-JSD and low merge ratio), it might suggests challenges like poor pull request quality, maintainer bottlenecks, or competitive silos. This low entropy may indicate a less collaborative environment, where decentralized efforts fail to converge, potentially leading to contributor attrition, duplicated work, community splits, or stalled project growth. It signals to project managers the need for interventions, such as improved documentation, mentorship for fork owners, or streamlined review processes to enhance integration.

Conversely, when entropy is high, it reflects a balanced and effective convergence, where the distribution of original commits closely mirrors that of merged commits, with a high merge ratio. In this scenario, diverse forks contribute innovations proportionally, and these are broadly integrated, indicating a vibrant, inclusive ecosystem. High entropy suggests a complex, multifaceted development process where various stakeholders participate, bringing unique expertise that enriches the main repository. This diversity of fork involvement often correlates with deeper project evolution, as integrated contributions from multiple sources drive innovation and resilience. Such a process adds greater value to the project, benefiting from a comprehensive, collaborative approach [5].

3.3 Application Practice in Forks from External Contributors

Although convergence entropy captures overall fork dynamics, this study specifically focuses on forks from external contributors. Since external contributors do not have write access to projects, fork and pull-request is their primary, if not the only, means of contribution [22], [24], making our study more focused on the features and performance related to the integration effectiveness. In contrast, core team members can commit directly without forking. Moreover, external contributors tend to participate sporadically, adding a layer of unpredictability to the contribution integration process, which is not typically observed with the more consistent and structured involvement of core team members [4]. By examining fork activities, we aim to understand how these distributed contributions converge or diverge globally and how they relate to project productivity.

OSS development teams consist of a small, well-organized core team and a larger, loosely-coupled group of external contributors [32, 66]. Previous studies used GitHub's repository permission mechanism to distinguish core team members from external contributors[23]. However, GitHub does not allow anyone except the repository owners to access the list of members with management privileges. Wang et al. [62] proposed a more reliable method by identifying core team members through their performance of events that require write permission. However, this approach may miss some core team members whose roles do not require write permissions. For example, the role of "Triage" on GitHub allows core team members to manage issues and pull requests without write permissions.

Building on Wang et al.'s method, this study uses a permissions-check mechanism to distinguish core team members from external contributors, but it considers all privileged events, not just those requiring write permissions.

Specifically, based on GitHub's permissions documentation [18], permissions for 53 types of issue-level events were manually annotated to determine which actions require management privileges. It was found that 16 of these events could be performed by external contributors without needing management privileges. Users who have performed events requiring management privileges are labeled as "core team member", while all others are classified as "external contributor". Notably, the required permissions for some actions depend on the target object. For instance, users can close issues they created themselves but need management privileges to close issues created by others, even though both scenarios are categorized as "Closed Event" on GitHub. See the replication package for details[1].

4 STUDY DESIGN

In this section, we firstly present the selection of key productivity metrics based on the SPACE (Satisfaction & well-being, Performance, Activity, Communication & collaboration, Efficiency & flow) framework [15], and relevant control variables. We then outline the data collection and cleaning process from eight open source projects on GitHub. Finally, we explain the use of panel regression analysis to evaluate the relationship between convergence entropy and project productivity.

4.1 Variables

Forsgren et al. [15] introduced the SPACE framework, which presents a comprehensive view of productivity by assessing five dimensions: Satisfaction & well-being, Performance, Activity, Communication & collaboration, and Efficiency & flow. To capture the project productivity, we have selected three metrics closely tied to the fork and

[1] https://github.com/anonymous2026CHI/2026CHI-replication-package

commit activities according to the SPACE framework guidelines, spanning three dimensions: Performance, Activity, and Efficiency & Flow. These metrics are new bugs, new commits, and issue resolution time, and all of them are significant within the context of fork convergence, where bugs represent critical outcomes of development processes, commits serve as tangible outputs, and issue resolution time reflects team efficiency.

We did not include the Satisfaction & well-being dimension which typically captured through surveys, due to the impracticality of tracking these changes monthly. The Communication & collaboration dimension is also excluded, as convergence entropy implicitly encapsulates information related to convergence and collaboration within fork activities. This approach aligns with the SPACE framework's guidelines, allowing for such adaptability.

In addition, we also introduce some relevant control variables to ensure a comprehensive evaluation of the project's productivity.

New bugs. The number of new bugs is commonly used as a proxy for evaluating software quality in projects [45, 59, 62]. We use this metric to capture project productivity in the Performance dimension, assessing the outcome of the software development process [15]. Specifically, we calculate the number of new bugs found during a project-month. Note that we do not distinguish bug reports from core members or external contributors, because we measure the productivity of the whole project. On GitHub, the issue can be of various types, e.g., discussion, new feature request, improvement request, and so on. To categorize these issues, software developers often employ some keywords to tag them. However, because tagging is often project specific, we adopt Vasilescu et al.'s [59] method to distinguish bug issues from other issue types in this study. We set up a list of bug-related keywords, i.e., defect, error, bug, issue, mistake, incorrect, fault, and flaw, and then search for these words in both the issue tags and issue titles. If any tags or title of an issue contains at least one keyword, we identify it as a bug issue.

New commits. In many studies focusing on OSS development and community, the number of commits is considered as the productivity metric [45, 57, 58]. We adopt this widely used productivity indicator to capture project productivity in the Activity dimension, assessing the output of the software development process [15]. Note that we count the commits from all contributors rather than from external developers only, because we measure the impact on the productivity of the whole team. Similarly, as the productivity data, we compute the number of new commits in every project-month.

Issue resolution time. The issue resolution time is also regarded as an important metric for productivity [17, 41]. We adopt this metric to capture project productivity in the Efficiency & Flow dimension, assessing the efficiency of the software development process [15]. To calculate the average issue resolution time each project-month, we consider all issues (including bugs) closed within the respective time window and compute the duration from creation to closure for each. The average of these durations is then taken as the issue resolution time for that project-month.

Control variables. Based on prior software engineering literature [20, 59, 61] and our experience, we include the following factors as control variables.

- **Project Age:** The time span from project creation to the current month, measured in months. Growth dynamics may be different in younger vs. older projects [59].
- **NumForks:** The number of forks of the project until the current month. We use this as a proxy measure for the popularity of the project. A higher number of forks typically correlates with an increased volume of bug reports[59] and augmented external contributions[61].
- **NumIssues:** The number of new issues opened within the current month.
- **NumPrs:** The number of pull-requests opened within the current month. More opened pull-requests typically correlates with more people discussing concerns, and a longer issue resolution time [33].
- **NumFiles:** The number of files of the project until the current month. We use this as a proxy measure for the size of the project. Larger projects naturally receive more contributions [20, 59].
- **NumCore** and **NumExt:** The number of active core developers and active external developers for the month. Active developers are defined as those who engage with the repository through at least one recorded activity (e.g., commit, fork, PR) within a given month.

4.2 Data Collection and Cleaning

To answer the three research questions presented in Section 1, we conduct an empirical study based on eight well-known open source projects. The study follows a systematic project selection process designed and the steps are as follows:

We began by obtaining an initial project pool comprising the top 1,000 projects from GitHub, ranked by star count. This pool was refined through the following filtering criteria to arrive at qualified candidate pool:

- To ensure robust analysis, we required projects to possess sufficient development history and collaborative records. Specifically, we applied the following quantitative thresholds:
  (1) Project age over 5 years: Ensures complete release cycles and potential evolution of maintenance patterns.
  (2) Mainline commits over 3,000: Ensures adequate activity for analyzing code integration patterns.
  (3) Contributors Over 300: Indicates a sizable community, facilitating the examination of collaboration.
- The selected projects should be engaged in software development, rather than serving as repositories for document storage or course teaching. We use Zhou's method [68] to remove non-development repositories. We set up a list of keywords, i.e., "homework", "assignments", "course", "learn", "free", "awesome", and then search for these words in both project names and descriptions. matched project were then manually inspected and only development project were retained.

Among the 261 candidate projects selected with the above criteria, we ultimately sample 8 projects for study. To avoid domain-specific bias [61], the final sample spans various application domains such as documentation tools, testing frameworks, and service

discovery components. Acknowledging the trend of increasing corporate involvement in open source, the sample comprises both enterprise-driven and community-driven projects. Following the heuristic proposed by Spinellis et al. [49] to identify enterprise-driven projects, we classify facebook/docusaurus, microsoft/playwright, alibaba/nacos, and google/googletest as enterprise-driven. Conversely, fastapi/fastapi, vuejs/vue, rustdesk/rustdesk, and ohmyzsh/ohmyzsh are categorized as community-driven projects.

We collected data on forks, commits, and all related events for each project through the GitHub REST API and GraphQL API. We excluded forks with no new commits, as they represent the majority of forks, which are often created for backup purposes and are irrelevant to the integration effectiveness we discuss [28]. This dataset also excludes any data that has been removed from GitHub, as it is typically not considered meaningful by project managers. Additionally, we excluded any actions performed by bots (i.e., automated processes presented as GitHub users who perform event-driven actions), as our focus is solely on human activities.

In total, we collected 12,720 forks and 39,300,748 commits across these eight repositories. See the replication package for details [2].

4.3 Regression Analysis

We calculate monthly variables for each project spanning from its creation to May 2025. For convergence entropy and each control variable, we standardized them to a mean of zero and a standard deviation of one. This ensures that all estimated coefficients in the model are on the same scale so that we can perform comparative analysis. Additionally, we manually remove about 1% of values as outliers to ensure the models are robust against outliers [42].

To construct our linear models, we employ panel regression analysis, a robust econometric technique well-suited for our dataset, which is inherently unbalanced panel data. The use of simple OLS multivariate linear regression is not appropriate in this context; it fails to account for the intrinsic differences across the various projects and over the distinct time periods, as noted by Wooldridge [64].

Moreover, panel regressions provide another benefit, their intrinsic ability to handle commercial, social, and technical confounding factors without explicit enumeration. For example, project-specific factors (e.g., project domain, etc.) are treated as unobserved time invariant within the regression models. This is particularly useful when facing numerous confounding factors that are challenging to list exhaustively in statistical analyses—the selection of a subset for inclusion could introduce selective biases. While, the independent variables' effects could be precisely estimated without worrying about potential interactions and selective biases with many confounding factors when we do not aim for causality[64]. Intuitively, each project has its own characteristics, so we use the random effects model, which assumes that individual project characteristics are randomly distributed and uncorrelated with the explanatory variables.

Before fitting the model, we conducted a Skewness Test on the control variables. For control variables with skewed distributions, we performed a log transformation to stabilize the variance and reduce heteroscedasticity [37]. For each dependent variable, we use Amemiya method to estimate the random effects model parameters, which is robust to potential heteroskedasticity and autocorrelation in the data. After we finish the model estimation, we perform a series of regression diagnostics for examining the time-specific effects and empirically justifying the use of random effects models. These regression diagnostics include time-fixed effects test, F-test (pFtest), Hausman Test (pHtest), and Heteroskedasticity Test, Skewness Test. Given that our sampled projects consist of 4 company projects and 4 non-company ones, it is natural to investigate if convergence entropy's impacts on project's productivity are sensitive to these project characteristics. Therefore, we perform the same regression analyses to the two subsamples. The results are reported accordingly. All the panel regressions, if not otherwise stated, are performed with R's plm package [10]. We follow the ASA's principles to present and interpret statistical significance [63].

[2]https://github.com/anonymous2026CHI/2026CHI-replication-package

5 RESULTS

In this section, we report the results of regression analyses for research questions, and all data and code have been made publicly available for download.

Moreover, it's essential to understand that our regression models are intended to pinpoint correlation rather than causation. In interpreting the findings, we give some conjectures that suggest possible causal relationships. This approach is in line with standard practices in empirical research examining human and social behaviors [9, 43]. However, these conjectured causal links should be regarded as speculative and not definitive, which require further empirical investigation to be substantiated [47].

5.1 RQ1: What is the correlation between convergence entropy and the number of new bugs in OSS projects, and what factors could alter this dynamic?

To answer RQ1, we model the number of new bugs as a function of convergence entropy. Our model accounts for a variety of control factors, including project age, the number of forks, the number of issues, the number of pull requests, the number of files, the number of active core developers, and the number of active external contributors. Additionally, we have included interaction terms between convergence entropy and each of the control factors.

Table 2 presents the regression analyses across three distinct samples. Model B1 examine non-company projects, while Models B2 focus on company projects, enabling systematic comparison between these project types. Model B3 provides the whole-sample analysis for overall effect estimation. All models demonstrate strong explanatory power, with both R^2 and adjusted R^2 indicating substantial variance explanation. The highly significant Wald χ^2 statistics confirm the joint significance of all predictors.

Whole Sample Regression Results. In Model B3, convergence entropy ($\mathcal{H}_{conv}$) shows a significant negative correlation with the number of new bugs. This suggests that higher convergence entropy, indicating stronger integration effectiveness from forks, corresponds to a significant decrease in the number of new bugs within

OSS projects. This finding aligns with the view that effective aggregation of contributions reduces fragmentation and bug propagation. [25, 39]

The interaction effects reveal nuanced dynamics. The negative effect of convergence entropy on new bugs strengthens as projects mature, but weakens when the number of issues increases. Conversely, more pull requests enhance the bug-reducing impact of convergence entropy. The presence of more external contributors and forks may slightly weaken the beneficial effects in some contexts.

Subsample Regression Results. In Models B1 and B2, the negative correlation between convergence entropy and new bugs remains robust, with the effect appearing slightly stronger in company projects.

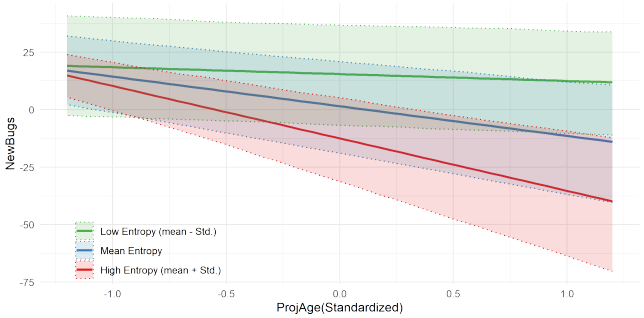
For the model's interaction terms, Figure 2 illustrates the strongest interaction effects in Models B1 and B2 to showcase their details. Figure 2a illustrates the trends of new bugs with the increasing of project age at low, middle, and high levels of convergence entropy in Model B1, respectively. We define the low, middle, and high levels of convergence entropy respectively to correspond to its values with mean minus one standard deviation (Mean - Std.), mean, and mean plus one standard deviation (Mean + Std.) after consulting the literature [47]. The transparent parts represent the 90% confidence intervals. In general, new bugs decrease with the increase in project age. And the interaction suggests a higher level of convergence entropy is associated with a steeper decrease rate of new bugs with project age. With a fixed project age, an increase in a project's convergence entropy is related to a decrease in new bugs. This may indicate that non-company projects benefit from the integration effectiveness, with the advantages increasing over time. A possible explanation for this trend is that the progress of non-company projects is more dependent on effectively integrating the efforts of external contributors, a process that requires substantial time to accumulate and cultivate a dedicated contributor community [1].

Figure 2b illustrates the interaction between convergence entropy and the number of forks on new bugs in Model B2. We observe that a decrease in new bugs is correlated to the increase of the number of forks, especially in the case of low levels of convergence entropy. When the number of forks is low, new bugs decrease as convergence entropy increases, but when the number of forks is high, the opposite occurs. This interaction may suggest that in company projects, a high number of forks combined with strong convergence entropy could lead to increased complexity in integration, potentially introducing more bugs due to the structured but resource-intensive processes typical in corporate environments, where rapid scaling of forks might overwhelm centralized review mechanisms [56].

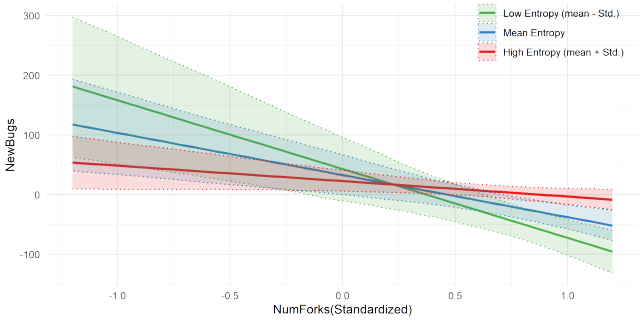
Interactions with NumIssues and NumFiles are positive in both subsamples, but $\mathcal{H}_{conv}$:NumExternals is positive in non-company and negative in company projects, highlighting how company support might better integrate external contributions.

In summary, we answer RQ1 as below.

(1) Convergence entropy is significantly negatively correlated with the number of new bugs.
(2) In non-company projects, the correlation between convergence entropy and new bugs strengthens with project age and core teams, but weakens with more issues, files and external contributors.
(3) In company projects, this correlation strengthens with external contributors, but weakens with more forks, issues and files.



(a) Interaction between Convergence Entropy and Project Age on New Bugs in Non-Company Projects



(b) Interaction between Convergence Entropy and Number of Forks on New Bugs in Company Projects

Figure 2: Visualization of Strongest Interaction Effects in Models B1 and B2

5.2 RQ2: How is convergence entropy correlated with the number of new commits in OSS projects, and what factors may affect this connection?

To answer RQ2, we model the number of new commits as a function of convergence entropy. Our model accounts for a variety of control factors, including project age, the number of forks, the number of issues, the number of pull requests, the number of files, the number of active core developers, and the number of active external contributors. Additionally, we have included interaction terms between convergence entropy and each of the control factors.

Table 3 presents the regression analyses across three distinct samples. Model C1 examine non-company projects, while Models C2 focus on company projects, enabling systematic comparison between these project types. Model C3 provides the whole-sample analysis for overall effect estimation. All models demonstrate strong

Table 2: Regression Models for New Bugs

	Noncompany Sample Model B1		Company Sample Model B2		Whole Sample Model B3	
	Coeffs	Std. Error	Coeffs	Std. Error	Coeffs	Std. Error
(Intercept)	10.936	11.327	60.344*	25.187	26.517*	10.970
$\mathcal{H}_{conv}$	−12.357**	4.175	−14.507**	5.618	−13.220***	3.264
Project Age	−12.898***	3.201	37.489***	5.589	2.545	1.917
NumForks	−6.487**	2.438	−70.600***	7.707	−4.114	2.182
NumIssues	8.982***	1.857	26.180***	3.228	16.339***	1.608
NumPrs	4.856***	0.995	3.185	1.920	4.377***	1.020
NumFiles	20.986***	2.813	−21.736***	3.740	−0.262	1.724
NumCores	−2.267*	1.133	−1.588	1.470	−2.275**	0.868
NumExternals	4.599**	1.399	3.518	3.220	4.095**	1.395
$\mathcal{H}_{conv}$:Project Age	−9.900***	2.361	−7.856	4.732	−4.285**	1.581
$\mathcal{H}_{conv}$:NumForks	−0.334	0.930	44.705***	4.186	2.586*	1.116
$\mathcal{H}_{conv}$:NumIssues	5.197**	1.892	5.044*	2.645	8.334***	1.440
$\mathcal{H}_{conv}$:NumPrs	−0.403	0.970	−2.719	1.747	−7.839***	0.910
$\mathcal{H}_{conv}$:NumFiles	6.868**	2.469	7.650**	2.545	1.067	1.617
$\mathcal{H}_{conv}$:NumCores	−2.076*	1.057	−2.692	1.777	−0.995	0.960
$\mathcal{H}_{conv}$:NumExternals	5.496***	1.379	−5.685*	2.413	2.533*	1.165
R^2	0.742		0.758		0.602	
Adj. R^2	0.731		0.747		0.594	
Wald $\chi^2(15)$	1073.450***		1020.390***		1081.660***	

***p<0.001, **p<0.01, *p<0.05

explanatory power, with both R^2 and adjusted R^2 indicating substantial variance explanation. The highly significant Wald χ^2 statistics confirm the joint significance of all predictors. Notably, the Models C2 exhibit superior model performance, suggesting distinct underlying mechanisms between project types.

Whole Sample Regression Results. In Model C3, convergence entropy ($\mathcal{H}_{conv}$) lacks a significant main effect, but interactions reveal nuanced dynamics. This suggests that the overall correlation with new commits depends heavily on contextual factors. For the model's interaction terms, $\mathcal{H}conv$:NumForks and $\mathcal{H}conv$:NumIssues are positive, suggesting that more forks and issues amplify the positive impact of convergence on commits. Similarly, $\mathcal{H}conv$:NumPrs is positive, indicating enhanced commit activity with pull requests. However, $\mathcal{H}conv$:NumFiles is negative, implying that larger codebase may hinder convergence-driven commits, possibly due to the complexity of code review [38, 56].

Subsample Regression Results. The correlation between convergence entropy and new commits differs markedly: positive and significant in non-company projects but negative in company projects. This suggests that non-company projects may rely on effective integration of external contributions, while company projects face potential overload.

For the model's interaction terms, Figure 3 illustrates the strongest interaction effects in Models C1 and C2 to showcase their details. Figure 3a illustrates the interaction between convergence entropy and the number of issues on new commits in Model C1. We observe that the correlation between the number of issues and new commits differs with different levels of convergence entropy. New commits decrease with the number of issues when convergence entropy is at the median or high level, while they remain basically unchanged when convergence entropy is low. This phenomenon is opposite in company projects. A possible explanation is that in non-company projects, high convergence entropy may lead to overload when issues accumulate, diverting efforts from new commits to resolution, whereas low convergence allows for stable commit activity regardless of issue volume, possibly due to decentralized community handling [54].

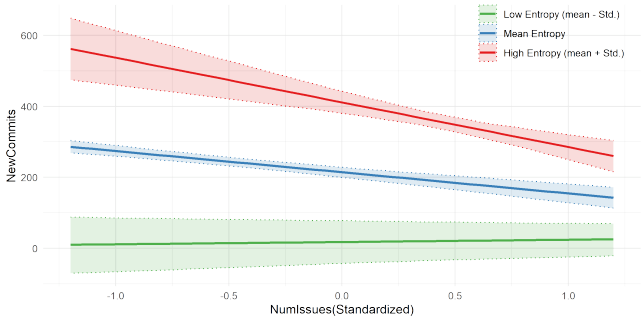
Figure 3b illustrates the interaction between convergence entropy and project age on new commits in Model C2. We observe that the correlation between project age and new commits exhibits opposite patterns in high and low entropy scenarios. New commits decrease with project age when convergence entropy is at the high level, while they increase when convergence entropy is low. Interestingly, this phenomenon is opposite in non-company projects. A possible explanation is that in company projects, core team often gradually establish a well-defined project roadmap and development priorities. They aim to adhere to this roadmap, may become more selective and tend to reject contributions that are perceived as diverging from the project's core objectives, even if those contributions are valuable in isolation [21]. In contrast, non-company project may adopt a more inclusive approach which helps sustain contributor motivation and encourages ongoing participation [66].

The remaining interactions show varied patterns: $\mathcal{H}conv$:NumPrs is positive in both C1 and C2; $\mathcal{H}conv$:NumExternals is positive in non-company but negative in company projects; $\mathcal{H}conv$:NumCores is negative in non-company but positive in company projects; and

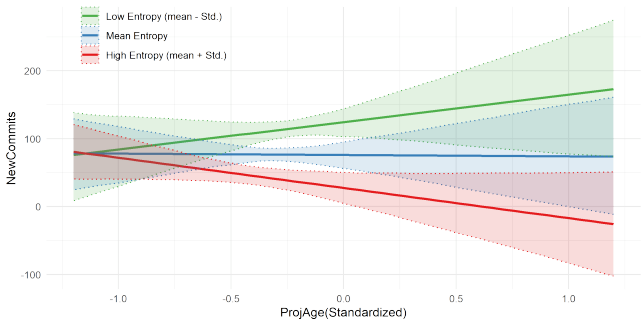
$\mathcal{H}_{conv}$:NumFiles is positive in non-company but negative in company projects. These indicate how contextual factors differentially moderate integration effectiveness across project types.

In summary, we answer RQ2 as below.

(1) Convergence entropy has a distinct correlation with new commits: positive in non-company projects but negative in company projects.
(2) In non-company projects, the positive correlation strengthens with project age, pull requests, and external contributors but weakens with more issues, files, and core developers.
(3) In company projects, the negative correlation strengthens with project age but weakens with more forks, issues, and pull requests.



(a) Interaction between Convergence Entropy and Number of Issues on New Commits in Non-Company Projects



(b) Interaction between Convergence Entropy and Project Age on New Commits in Company Projects

Figure 3: Visualization of Strongest Interaction Effects in Models C1 and C2

5.3 RQ3: To what extent does convergence entropy correlate with issue resolution time in OSS projects, and what factors could strengthen or weaken this association?

To answer RQ3, we model the project's average issue resolution time as a function of convergence entropy. Our model accounts for a variety of control factors, including project age, the number of forks, the number of issues, the number of pull requests, the number of files, the number of active core developers, and the number of active external contributors. Additionally, we have included interaction terms between convergence entropy and each of the control factors.

Table 4 presents the regression analyses across three distinct samples. Model T1 examines non-company projects, while Models T2 focus on company projects, enabling systematic comparison between these project types. Model T3 provides the whole-sample analysis for overall effect estimation. The highly significant Wald χ^2 statistics confirm the joint significance of all predictors. The explanatory power of the regression models is moderate, with R^2 values of 0.265 for T1, 0.183 for T2. These values are acceptable for modeling issue resolution time, which depends on many complex factors in OSS development. The low R^2 in Model T3 likely results from heterogeneity between company and non-company projects, as evident from the divergent patterns in T1 and T2. Therefore, we focus our interpretation on the subsample analyses while still reporting the whole-sample results for completeness.

Subsample Regression Results. The correlation differs: positive in non-company projects, indicating higher convergence entropy may prolong fix times, but negative in company projects, suggesting efficiency gains.

For the model's interaction terms, Figure 4 illustrates the strongest interaction effects in Models T1 and T2 to showcase their details. Figure 4a illustrates the interaction between convergence entropy and the number of files on issue resolution time in Model T1. We observe that the correlation between the number of files and issue resolution time differs with different levels of convergence entropy. Issue resolution time decreases with the number of files when convergence entropy is at the median or low level, while it increases when convergence entropy is high. A possible explanation is that in non-company projects, low to moderate convergence allows larger codebases to benefit from distributed community efforts that efficiently address issue, but high convergence introduces coordination overhead in integrating diverse contributions, prolonging resolution times as the codebase grows.

Figure 4b illustrates the interaction between convergence entropy and the number of forks on issue resolution time in Model T2. We observe that an increase in issue resolution time is correlated with the increase in the number of forks, especially in the case of low levels of convergence entropy. Combining with the confidence intervals, when the number of forks is low, the influence of convergence entropy on resolution time is minimal, but when the number of forks is high, higher convergence entropy is associated with shorter resolution times. This interaction may suggest that in company projects, a high number of forks can introduce more complexity, leading to increased issue resolution time, but high levels of convergence entropy, which indicate stronger aggregation effects, can mitigate this by enabling better integration and resource allocation through corporate oversight, effectively reducing fix times in scaled fork environments; this differs from non-company projects, where forks may amplify decentralized challenges without such structure [55].

The remaining interactions show varied patterns: $\mathcal{H}conv$:NumIssues is negative in non-company projects; $\mathcal{H}conv$:NumFiles is positive in non-company projects but negative in company projects; $\mathcal{H}conv$:NumExternals is positive (though non-significant) in non-company projects but negative in company projects; and $\mathcal{H}_{conv}$:NumCores is positive in company projects. These indicate how factors

Table 3: Regression Models for New Commits

	Noncompany Sample Model C1		Company Sample Model C2		Whole Sample Model C3	
	Coeffs	Std. Error	Coeffs	Std. Error	Coeffs	Std. Error
(Intercept)	183.111***	29.567	72.924***	6.214	93.087***	12.621
$\mathcal{H}_{conv}$	198.257***	38.763	−39.967***	11.267	13.597	13.097
Project Age	89.283**	27.941	−2.021	12.303	14.512	7.510
NumForks	20.978	23.925	−35.193*	16.644	−19.814*	9.810
NumIssues	−59.701***	15.811	1.860	7.292	0.721	6.084
NumPrs	104.771***	8.298	56.971***	4.403	62.584***	4.141
NumFiles	−66.317**	25.634	1.111	5.182	−14.846*	6.336
NumCores	−19.650*	8.939	−4.731	3.598	−6.984*	3.464
NumExternals	24.501*	11.799	4.830	7.903	7.577	5.416
$\mathcal{H}_{conv}$:Project Age	50.727*	19.941	−42.432***	8.360	1.032	6.116
$\mathcal{H}_{conv}$:NumForks	−7.201	8.464	23.131*	10.877	14.562**	4.838
$\mathcal{H}_{conv}$:NumIssues	−66.114***	15.738	16.212**	6.152	16.647**	5.433
$\mathcal{H}_{conv}$:NumPrs	52.763***	8.083	8.518*	4.033	12.442***	3.648
$\mathcal{H}_{conv}$:NumFiles	12.651	21.214	−6.669	6.017	−21.592***	6.500
$\mathcal{H}_{conv}$:NumCores	−17.235*	8.666	3.006	4.219	0.610	3.873
$\mathcal{H}_{conv}$:NumExternals	23.331*	11.751	−12.484*	6.026	−7.954	4.487
R^2	0.483		0.729		0.432	
Adj. R^2	0.462		0.717		0.420	
Wald $\chi^2(15)$	338.044***		835.509***		531.084***	

***p<0.001, **p<0.01, *p<0.05

like forks and issues can enhance efficiency in decentralized settings, while company structures may introduce overhead with core teams but better handle externals.

In summary, we answer RQ3 as below.

(1) Convergence entropy shows contrasting associations with issue resolution time: positive in non-company projects but negative in company projects.
(2) In non-company projects, this association strengthens with the number of files but weakens with more forks and issues.
(3) In company projects, more forks and external contributors strengthen the reduction in issue resolution time but weakens with more core members.

6 DISCUSSION

6.1 Discussion of the Findings

The empirical findings from our regression analyses provide valuable insights into the role of convergence entropy in OSS project dynamics, highlighting its correlations with key productivity metrics such as new bugs, new commits, and issue resolution time. By distinguishing between non-company and company-backed projects, our results underscore the contextual nuances that shape these relationships. Below, we discuss the implications of each RQ. Note that, in interpreting the findings, we give some conjectures that suggest possible causal relationships. This approach is in line with standard practices in empirical research examining human and social behaviors [9, 43]. However, these conjectured causal links should be regarded as speculative and not definitive, which require further empirical investigation to be substantiated [47].

For **RQ1**, our results indicate that a stronger integration effectiveness is associated with fewer new bugs. This relationship may be explained by enhanced code unification and improved collaboration efficiency, as increased convergence reduces fragmentation risks. Timely integration of contributions from forks allows the mainline to benefit from diverse external contribution while minimizing the potential for bug propagation [25, 39] or inefficient collaboration [20, 68] that often result from fork practice. For instance, when innovations or fixes from forks are efficiently integrated, typically through rigorous review and testing, they can preemptively address potential defects, contributing to a more stable and consistent codebase. This process creates a positive feedback loop that transforms distributed development efforts into collective quality improvements, rather than isolated changes that introduce instability. In non-company projects, the strengthening of this correlation with project age and core team size might suggest that mature, community-driven ecosystems develop more effective mechanisms for filtering and integrating converged contributions, thereby reducing bug-inducing divergences [1]. Conversely, the weakening effects associated with more issues, files, and external contributors may reflect coordination challenges in decentralized environments, where a high volume of external contributions under strong convergence could lead to integration errors without effective management. In company projects, the distinct dynamics, such as the weakened association with forks, point to possible scalability limits.

Table 4: Regression Models for Resolution Average Time

	Noncompany Sample Model T1		Company Sample Model T2		Whole Sample Model T3	
	Coeffs	Std. Error	Coeffs	Std. Error	Coeffs	Std. Error
(Intercept)	91.359***	11.409	38.933***	11.054	56.217***	7.224
$\mathcal{H}_{conv}$	86.725***	19.333	−30.049**	11.175	−18.141	14.278
Project Age	59.077***	13.809	−78.358***	18.425	−4.207	9.572
NumForks	27.152	17.584	114.638***	28.193	−1.183	11.301
NumIssues	−60.967***	13.389	−23.370	12.599	−25.618***	7.442
NumPrs	8.851**	2.840	27.739***	5.721	11.070	6.186
NumFiles	−35.781**	12.219	5.875	5.890	4.505	6.082
NumCores	16.871***	4.762	−10.029	7.621	12.879	6.914
NumExternals	38.586***	10.320	18.026***	1.529	15.526*	7.002
$\mathcal{H}_{conv}$:Project Age	12.815	11.620	2.642	9.822	−5.926	7.186
$\mathcal{H}_{conv}$:NumForks	−14.724*	6.537	−43.547*	19.002	6.315	8.686
$\mathcal{H}_{conv}$:NumIssues	−51.561***	10.294	3.563	2.812	−5.956	4.220
$\mathcal{H}_{conv}$:NumPrs	6.372	4.902	−0.379	4.936	4.793	2.618
$\mathcal{H}_{conv}$:NumFiles	53.889***	5.652	−7.458	6.886	−9.708	9.900
$\mathcal{H}_{conv}$:NumCores	−6.371	8.726	8.962*	3.969	0.677	6.296
$\mathcal{H}_{conv}$:NumExternals	11.369	7.126	−10.779***	1.987	0.443	3.416
R^2	0.265		0.183		0.077	
Adj. R^2	0.235		0.145		0.058	
Wald $\chi^2(15)$	131.048***		72.386***		61.934***	

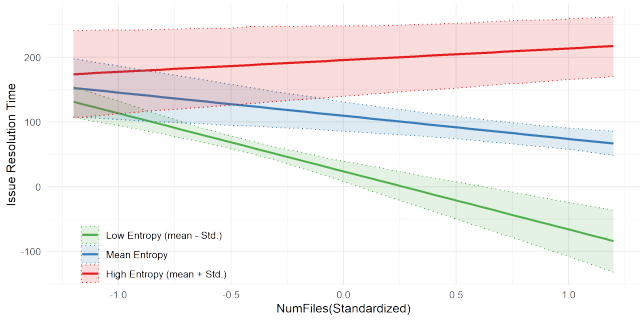
***p<0.001, **p<0.01, *p<0.05

For example, a large number of forks combined with strong convergence may overburden centralized review processes, resulting in rushed integrations and ultimately more bugs [9].

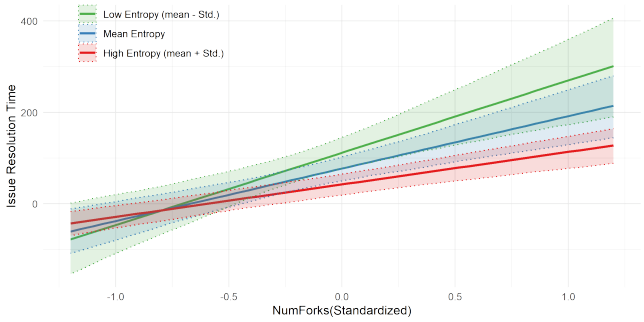
For **RQ2**, the findings reveal a divergent correlation between convergence entropy and the number of new commits across project types, positive in non-company projects but negative in company ones. This might suggest that in decentralized, community-driven settings, stronger convergence may facilitate greater commit by effectively integrate distributed contributions into mainline, potentially fostering a virtuous cycle of innovation and participation. This is consistent with previous research, high convergence could streamline the incorporation of fork-derived changes, encouraging more frequent commits as contributors see their work integrated swiftly, thereby sustaining motivation and accelerating development velocity [3, 53]. The strengthening of this positive association with project age, pull requests, and external contributors in non-company projects may indicate that over time, these ecosystems cultivate robust collaborative norms and tools that amplify the benefits of convergence, transforming a growing pool of external inputs and structured pull requests into heightened commit output [1]. However, the weakening effects from increased issues, files could stem from resource dilution; for instance, a larger codebase or issue backlog under strong convergence might overload volunteer-based core teams, shifting focus from new commits to maintenance and integration tasks, leading to inefficiencies in handling complexity [54, 62]. In contrast, the negative correlation in company projects, which intensifies with project age, might reflect bureaucratic hurdles or selective integration practices that prioritize quality over quantity [21], where high convergence in mature projects could result in more rigorous vetting processes that inadvertently suppress commit volume to maintain alignment with corporate roadmaps. The weakening of this negative effect with more forks, issues, and pull requests may imply that in scaled corporate environments, these factors introduce sufficient external pressure or diversity to counterbalance convergence-driven selectivity, prompting more commits to address emerging needs despite structured oversight.

For **RQ3**, our analysis reveals contrasting correlations between convergence entropy and average issue resolution time: positive in non-company projects (prolonging resolution) and negative in company projects (accelerating it). This divergence likely stems from organizational differences. In non-company projects, which often rely on decentralized, volunteer-driven structures, high convergence may create coordination bottlenecks, as integrating diverse fork contributions demands extensive community consensus and ad-hoc efforts, leading to delays [55]. In company projects, characterized by centralized governance and dedicated resources [60], the negative correlation might highlight efficiency gains, as structured organizational processes facilitate rapid incorporation of converged fork innovations into fixes. The visibility and attention that come with more forks, issues and external contributors mean that bugs are more likely to be spotted and addressed promptly [46]. This may be attributed to the wide array of ideas and skills that contributors bring to the table, enhancing the detection and addressing of bugs [2].

Across RQs, a key theme emerges: convergence entropy is generally associated with fewer bugs; however, its relationships with



(a) Interaction between Convergence Entropy and Number of Files on Issue Resolution Time in Non-Company Projects



(b) Interaction between Convergence Entropy and Number of Forks on Issue Resolution Time in Company Projects

Figure 4: Visualization of Strongest Interaction Effects in Models T1 and T2

commit activity and issue resolution time vary by project type. Non-company projects show positive correlations with quality and output metrics but tend to exhibit negative correlations with efficiency measures, while company projects demonstrate positive associations with both quality and efficiency, though sometimes with a negative correlation to commit volume. This interplay suggests that convergence entropy captures not just technical aggregation but also socio-organizational dynamics, extending theories on OSS sustainability [35]. Future research could explore causal pathways, such as through longitudinal studies or interventions, to validate these observed correlations.

6.2 Recommendations

The observations and analyses derived from our study provoke a number of actionable recommendations for practitioners and maintainers of open-source software (OSS) projects. Accordingly, we provide a set of guidelines which aims to unlock the potential of fork convergence, thereby enhancing the sustainability of OSS projects.

- **Enhancing Governance in Non-Company Projects:** Our finding highlight that high convergence entropy may prolong issue resolution in non-company projects due to coordination challenges in decentralized structures. To address this, maintainers should prioritize lightweight governance tools, such as modular review workflows or automated testing pipelines, to streamline fork integration without imposing heavy hierarchies. This approach can reduce delays while leveraging the bug-reduction benefits that strengthen with project age, fostering a more efficient volunteer-driven ecosystem.
- **Streamlining Resources in Company Projects:** In company projects, convergence entropy is negatively correlated with commit volume, potentially due to selective integration practices that prioritize internal roadmaps over external contributions, leading to a more exclusive development process. To counteract this, leaders should consider decentralizing certain governance aspects, such as empowering external contributors through delegated decision-making rights or open review committees, to encourage broader integration of fork innovations. This can mitigate commit suppression, amplify efficiency gains in issue resolution, particularly with increasing forks and contributors, while addressing scalability challenges in larger teams and fostering a more inclusive alignment with corporate goals.
- **Promoting Convergence Across Project Types:** The consistent negative correlation between convergence entropy and new bugs underscores the value of unified contributions for quality improvement. Practitioners should foster convergence through incentives, such as contributor recognition programs or automated fork-syncing bots, to encourage timely integrations. This not only lowers bug introductions but also enhances the moderating effects of pull requests and external inputs, supporting long-term sustainability in both decentralized and centralized environments.
- **Auditing Complexity in Mature Projects:** As projects age, factors like codebase size and issue backlogs can weaken convergence benefits, varying by project type. Maintainers should conduct periodic audits, refactoring large codebase into modules or using community voting for issue triage, to counteract these effects. This helps sustain positive dynamics, such as increased commits in non-company projects or faster resolutions in company ones, preventing overload from accumulated complexity.

6.3 Implications

Previous studies have explored various aspects of fork practices, such as commit distribution characteristics within forks [7] and techniques for automatically generating overviews of active forks [67]. By integrating these efforts with our work, we can develop a more comprehensive understanding of fork dynamics in OSS ecosystems. Specifically, by capturing integration effectiveness through convergence entropy, we can finely observe the divergence or convergence of collective intelligence in fork practices, thereby enabling deeper exploration of project sustainability.

Further research into the temporal patterns of convergence entropy could significantly advance OSS development. Employing pattern mining and time series analysis techniques would enable the extraction of evolutionary trends and future projections for convergence entropy, while identifying key factors and events that

drive its fluctuations. Analyzing these influences on project sustainability and vitality would, in turn, inform the creation of monitoring tools and best-practice guidelines to foster efficient collaboration throughout OSS development and maintenance.

Beyond its analytical contribution, the Convergence Entropy metric holds significant potential from a Human-Computer Interaction (HCI) perspective, particularly in enhancing transparency, coordination, and trust in distributed software work—themes well-established in prior research [11, 14, 24]. By providing a succinct, temporal indicator of integration health, it can serve as a key element in project dashboards. This offers contributors an at-a-glance understanding of whether contributions from the broader fork ecosystem are being absorbed effectively, thereby alleviating, to some degree, the inherent transparency challenges in distributed software development. This newfound transparency enables proactive coordination. A sustained low Convergence Entropy is not merely a number; it is a diagnostic signal of underlying coordination failures, such as duplicated efforts [34], bureaucratic hurdles [21], or emerging community drift [68]. For maintainers, this shifts their role from reactive integration managers to proactive ecosystem stewards. They can initiate targeted interventions, such as reaching out to isolated but active forks or adjusting contribution guidelines to better channel community efforts. For contributors, visibility into a project's historical and current convergence trends can inform their decision to invest effort, fostering trust by signaling a fair and efficient integration process. For platforms, integrating such a metric could enable novel features—for instance, triggering alerts when convergence entropy drops significantly, which may suggest potential bottlenecks or community drift. Thus, by bridging measurement with actionable insight, our work aims not only to advance empirical understanding but also to provide a practical tool for improving the quality and sustainability of open-source collaboration.

Importantly, the notion of contribution distribution and convergence, applied here to fork activities, lends itself to broader collaborative contexts where workflows involve distributed contributions and centralized integration, all quantifiable via convergence metrics. For example, in distributed teams or version control systems outside OSS, convergence entropy could illuminate integration efficiency and reveal bottlenecks in crowd collaboration.

6.4 Threats to Validity

Construct validity. This study introduces convergence entropy as a metric to quantify the extent to which distributed original commits across forks are effectively integrated back into the main repository. Jensen-Shannon Divergence (JSD) is a well-established indicator, widely applied across various fields to measure the distance between two distributions [19, 31]. In convergence entropy, we employ JSD to assess the distance between the distributions of original commits and merged commits, further adjusted by an efficiency factor. Consequently, we are confident that this approach poses no significant threat to construct validity.

Internal validity. We implemented several safeguards to ensure that our data collection process mitigated most of the common pitfalls outlined in [6, 29]. All analyzed projects were substantial in scale, featuring mature governance structures and established practices that leverage forks and commits for contribution management. We relied on objective activity records sourced from GitHub, which facilitated an unbiased analytical process. Established and widely recognized techniques were utilized throughout, with the adoption of random effects models in panel regressions justified through empirical validation. In this study, we investigated the correlation between convergence entropy and OSS project productivity, adhering to the guidelines in [15] by using new bugs, new commits, and issue resolution time as metrics to encompass diverse aspects of project productivity. A potential threat is that the capture of the efficiency and flow dimension by issue resolution time may be influenced by issue complexity. However, we believe it is an appropriate indicator of efficiency and flow, as it directly reflects the ability to make progress with minimal interruptions or delays—a core component of this dimension. This operationalization is also consistent with prior literature, which explicitly recommends time-based measures as proxies for efficiency and flow. Furthermore, many empirical studies have adopted metrics such as issue resolution time or bug fixing time as indicators of efficiency[40, 41]. From an experimental design perspective, our dataset is sufficiently large and temporally granular. We calculate the monthly average issue resolution time for each project, which helps smooth individual-level variations in complexity. As a result, issue complexity functions more like statistical noise than systematic bias, since its variance is averaged out under large numbers in our longitudinal analysis. Additionally, our panel regression model implicitly absorbs time-invariant project characteristics that may influence inherent issue complexity, such as project domain or governance structure.Importantly, issue complexity tends to naturally increase as projects evolve. This time-varying component is already controlled for via our project age variable. After accounting for project age, the key variables in our model remain significant, demonstrating the robustness of our findings even under potential complexity-related confounding.Nonetheless, we acknowledge that our model cannot fully eliminate all effects of issue complexity.

External validity. Regarding external validity, we believe the design of convergence entropy introduces no major concerns. Although our study centers on fork activities, it represents a generalizable model for quantifying convergence in distributed collaboration scenarios. This stems from the fact that our definition and computation make no assumptions about workflows beyond distributed contributions and centralized integration. However, we acknowledge that the empirical findings may not generalize to all OSS projects. A key limitation is that our dataset comprises only eight large-scale projects, selected to ensure sufficient traceable records of fork practices. We advise caution when extrapolating these results to small-scale or inactive OSS projects. Expanding the study to include a broader range of projects is planned for future work.

7 Conclusion

In the realm of open-source software (OSS) development, forking serves as a cornerstone mechanism that promotes innovation and collaboration through independent repository copies, yet it often introduces challenges such as inefficiencies and fragmentation. While the importance of integrating fork contributions is acknowledged,

the integration effectiveness remains underexplored, with its implications for project productivity unclear.

In this work, we introduce convergence entropy, a novel metric grounded in information theory that quantifies this integration effectiveness by assessing the similarity between distributions of original and merged commits across forks. We apply convergence entropy to evaluate the convergence dynamics in fork practices and investigate its associations with key productivity proxies: new bugs, new commits, and issue resolution time. Our findings reveal a consistent negative correlation with new bugs, indicating quality enhancements; divergent correlations with new commits (positive in non-company projects, boosting output, but negative in company projects, potentially suppressing volume); and contrasting links with issue resolution time (positive in non-company projects, extending durations, but negative in company projects, accelerating them). Additionally, we identify significant interactions with factors such as project age, number of forks, and codebase size, which modulate these relationships based on project type.

In light of these insights, we discuss the underlying mechanisms, propose practical recommendations to harness convergence benefits, and explore broader implications for OSS ecosystem health and collaborative workflows. Furthermore, we reflect on the potential of convergence entropy to inform future research and tools, offering fresh perspectives on the sustainability and vitality of OSS projects. Future work will expand our analysis to a wider array of projects, including smaller-scale ones, and delve into causal investigations through longitudinal studies or experimental interventions to further validate and refine these conjectures.

Acknowledgments

This work is supported by NSFC No.62332005

References

[1] Mark Aberdour. 2007. Achieving Quality in Open Source Software. *IEEE Softw.* 24, 1 (jan 2007), 58–64. doi:10.1109/MS.2007.2

[2] Ritu Agarwal. 2010. The Effects of Diversity in Global , Distributed Collectives : A Study of User Participation in Open Source Projects. https://api.semanticscholar.org/CorpusID:15201970

[3] Amirhosein "Emerson" Azarbakht. 2017. *Longitudinal Analysis of Collaboration in Forked Open Source Software Development Projects.* Ph. D. Dissertation. Oregon State University.

[4] Ann Barcomb, Andreas Kaufmann, Dirk Riehle, Klaas-Jan Stol, and Brian Fitzgerald. 2020. Uncovering the Periphery: A Qualitative Survey of Episodic Volunteering in Free/Libre and Open Source Software Communities. *IEEE Transactions on Software Engineering* 46, 9 (2020), 962–980. doi:10.1109/TSE.2018.2872713

[5] Anne Bartel-Radic and Nicolas Lesca. 2011. Do intercultural teams need "requisite variety" to be effective? *Management International* 15 (10 2011), 89–104. doi:10.7202/1005435ar

[6] Christian Bird, Peter C. Rigby, Earl T. Barr, David J. Hamilton, Daniel M. German, and Prem Devanbu. 2009. The promises and perils of mining git. In *2009 6th IEEE International Working Conference on Mining Software Repositories.* 1–10. doi:10.1109/MSR.2009.5069475

[7] Scott Brisson, Ehsan Noei, and Kelly Lyons. 2020. We Are Family: Analyzing Communication in GitHub Software Repositories and Their Forks. In *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER).* 59–69. doi:10.1109/SANER48275.2020.9054834

[8] John Businge, Moses Openja, Sarah Nadi, and Thorsten Berger. 2022. Reuse and maintenance practices among divergent forks in three software ecosystems. *Empirical Software Engineering* 27 (03 2022). doi:10.1007/s10664-021-10078-2

[9] Josh Cowls and Ralph Schroeder. 2015. Causation, Correlation, and Big Data in Social Science Research. *Policy & Internet* 7 (08 2015), n/a–n/a. doi:10.1002/poi3.100

[10] Yves Croissant and Giovanni Millo. 2008. Panel Data Econometrics in R: The plm Package. *Journal of Statistical Software* 27, 2 (2008), 1–43. doi:10.18637/jss.v027.i02

[11] Laura Dabbish, H. Colleen Stuart, Jason Tsay, and James Herbsleb. 2012. Social Coding in GitHub: Transparency and Collaboration in an Open Software Repository. *Proceedings of the ACM Conference on Computer Supported Cooperative Work, CSCW*, 1277–1286. doi:10.1145/2145204.2145396

[12] Gregor Eibl and Lőrinc Thurnay. 2023. The promises and perils of open source software release and usage by government – evidence from GitHub and literature. 180–190. doi:10.1145/3598469.3598489

[13] Fabian Fagerholm and Jürgen Münch. 2012. Developer Experience: Concept and Definition. doi:10.1109/ICSSP.2012.6225984

[14] Xinxin Fan, Ling Liu, Rui Zhang, Quanliang Jing, and Jingping Bi. 2020. Decentralized Trust Management: Risk Analysis and Trust Aggregation. 53, 1, Article 2 (Feb. 2020), 33 pages. doi:10.1145/3362168

[15] 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 (02 2021), 20–48. doi:10.1145/3454122.3454124

[16] Kam Hay Fung, Aybüke Aurum, and David Tang. 2012. Social Forking in Open Source Software: An Empirical Study. In *th International Conference on Advanced Information Systems Engineering (CAiSE), Gdansk, Poland, June 28, 2012 (CEUR Workshop Proceedings, Vol. 855)*, Marite Kirikova and Janis Stirna (Eds.). CEUR-WS.org, 50–57. http://ceur-ws.org/Vol-855/paper6.pdf

[17] Emanuel Giger, Martin Pinzger, and Harald Gall. 2010. Predicting the fix time of bugs. In *Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering* (Cape Town, South Africa) *(RSSE '10)*. Association for Computing Machinery, New York, NY, USA, 52–56. doi:10.1145/1808920.1808933

[18] Github. 2024. *Permissions for each role.* https://docs.github.com/en/organizations/managing-user-access-to-your-organizations-repositories/managing-repository-roles/repository-roles-for-an-organization [Online; accessed 02-January-2024].

[19] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative Adversarial Networks. arXiv:1406.2661 [stat.ML] https://arxiv.org/abs/1406.2661

[20] Georgios Gousios, Martin Pinzger, and Arie Deursen. 2014. An exploratory study of the pull-based software development model. doi:10.1145/2568225.2568260

[21] Georgios Gousios, Margaret-Anne Storey, and Alberto Bacchelli. 2016. Work practices and challenges in pull-based development: the contributor's perspective. In *Proceedings of the 38th International Conference on Software Engineering* (Austin, Texas) *(ICSE '16)*. Association for Computing Machinery, New York, NY, USA, 285–296. doi:10.1145/2884781.2884826

[22] Mahsa Hadian, Scott Brisson, Bram Adams, Soude Ghari, Ehsan Noei, Marios Fokaefs, Kelly A. Lyons, and Shurui Zhou. 2022. Exploring trends and practices of forks in open-source software repositories. In *Proceedings of the 32nd Annual International Conference on Computer Science and Software Engineering, CASCON 2022, Toronto, Ontario, Canada, November 15-17, 2022*, Paria Shirani, Iosif-Viorel Onut, and Tinny Ng (Eds.). ACM, 120–129. doi:10.5555/3566055.3566069

[23] Marvin Hanisch, Carolin Häussler, Stefan Berreiter, and Sven Apel. 2018. Developers' Progression from Periphery to Core in the Linux Kernel Development Project. *Academy of Management Proceedings* 2018 (04 2018), 14263. doi:10.5465/AMBPP.2018.117

[24] J.D. Herbsleb and A. Mockus. 2003. An empirical study of speed and communication in globally distributed software development. *IEEE Transactions on Software Engineering* 29, 6 (2003), 481–494. doi:10.1109/TSE.2003.1205177

[25] Md Hasibul Islam and Manishankar Mondal. 2024. Study and Analysis of Bug Propagation in Evolving Software through Micro-clones. 1–7. doi:10.1109/ICCCNT61001.2024.10726019

[26] He Jiang, Yulong Li, Shikai Guo, Xiaochen Li, Tao Zhang, Hui Li, and Rong Chen. 2023. DupHunter: Detecting Duplicate Pull Requests in Fork-Based Development. *IEEE Transactions on Software Engineering* 49, 4 (2023), 2920–2940. doi:10.1109/TSE.2023.3235942

[27] Jing Jiang, Fuli Feng, Xiaoli Lian, and Li Zhang. 2016. Long-Term Active Integrator Prediction in the Evaluation of Code Contributions. 177–182. doi:10.18293/SEKE2016-030

[28] Jing Jiang, David Lo, Jiahuan He, Xin Xia, Pavneet Singh Kochhar, and Li Zhang. 2017. Why and how developers fork what from whom in GitHub. *Empirical Software Engineering* 22 (02 2017). doi:10.1007/s10664-016-9436-6

[29] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M. German, and Daniela Damian. 2014. The Promises and Perils of Mining GitHub. In *Proceedings of the 11th Working Conference on Mining Software Repositories* (Hyderabad, India) *(MSR 2014)*. Association for Computing Machinery, New York, NY, USA, 92–101. doi:10.1145/2597073.2597074

[30] Rajiv Krishnamurthy, Varghese Jacob, Suresh Radhakrishnan, and Kutsal Dogan. 2016. Peripheral Developer Participation in Open Source Projects. *ACM Transactions on Management Information Systems* 6 (01 2016), 1–31. doi:10.1145/2820618

[31] Pedro Lamberti, A. Majtey, Marcos Madrid, and Mari´a Pereyra. 2007. Jensen-Shannon Divergence: A Multipurpose Distance for Statistical and Quantum Mechanics. *AIP Conference Proceedings* 913 (05 2007). doi:10.1063/1.2746720

[32] Wei Li, Wenjun Wu, Huai-min Wang, Xue-qi Cheng, Hua-jun Chen, Zhi-hua Zhou, and Rong Ding. 2017. Crowd intelligence in AI 2.0 era. *Frontiers of*

Information Technology & Electronic Engineering 18 (02 2017), 15–43. doi:10.1631/FITEE.1601859

[33] Zhixing Li, Yue Yu, Tao Wang, Yan Lei, Ying Wang, and Huaimin Wang. 2023. To Follow or Not to Follow: Understanding Issue/Pull-Request Templates on GitHub . *IEEE Transactions on Software Engineering* 49, 04 (April 2023), 2530–2544. doi:10.1109/TSE.2022.3224053

[34] Zhixing Li, Yue Yu, Minghui Zhou, Tao Wang, Gang Yin, Long Lan, and Huaimin Wang. 2020. Redundancy, Context, and Preference: An Empirical Study of Duplicate Pull Requests in OSS Projects. *IEEE Transactions on Software Engineering* PP (08 2020), 1–1. doi:10.1109/TSE.2020.3018726

[35] Zhifang Liao, Libing Deng, Xiaoping Fan, Yan Zhang, Hui Liu, Xiaofei Qi, and Yun Zhou. 2018. Empirical Research on the Evaluation Model and Method of Sustainability of the Open Source Ecosystem. *Symmetry* 10 (12 2018), 747. doi:10.3390/sym10120747

[36] Shane McIntosh, Yasutaka Kamei, Bram Adams, and Ahmed E. Hassan. 2015. An empirical study of the impact of modern code review practices on software quality. *Empirical Software Engineering* 21 (04 2015). doi:10.1007/s10664-015-9381-9

[37] Charles E. Metz. 1978. Basic principles of ROC analysis. *Seminars in Nuclear Medicine* 8, 4 (1978), 283–298. doi:10.1016/S0001-2998(78)80014-2

[38] Rahul Mishra and Ashish Sureka. 2014. Mining Peer Code Review System for Computing Effort and Contribution Metrics for Patch Reviewers. In *2014 IEEE 4th Workshop on Mining Unstructured Data*. 11–15. doi:10.1109/MUD.2014.11

[39] Manishankar Mondal, Banani Roy, Chanchal Roy, and Kevin Schneider. 2019. An Empirical Study on Bug Propagation through Code Cloning. *Journal of Systems and Software* 158 (09 2019), 110407. doi:10.1016/j.jss.2019.110407

[40] Marco Ortu, Bram Adams, Giuseppe Destefanis, Parastou Tourani, Michele Marchesi, and Roberto Tonelli. 2015. Are Bullies More Productive? Empirical Study of Affectiveness vs. Issue Fixing Time. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. 303–313. doi:10.1109/MSR.2015.35

[41] Marco Ortu, Giuseppe Destefanis, Mohamad Kassab, Steve Counsell, Michele Marchesi, and Roberto Tonelli. 2015. Would you mind fixing this issue? An Empirical Analysis of Politeness and Attractiveness in Software Developed Using Agile Boards, Vol. 212. doi:10.1007/978-3-319-18612-2_11

[42] Jason W. Osborne and Amy Overbay. 2004. The power of outliers (and why researchers should ALWAYS check for them). *Practical Assessment, Research, and Evaluation* 9 (1 2004). Issue 1. doi:10.7275/qf69-7k43

[43] Robert L. Brennan;Dale J. Prediger;. 1981. Coefficient Kappa: Some Uses, Misuses, and Alternatives. *Educational and Psychological Measurement* 41, 3 (1981), 687–699. doi:10.1177/001316448104100307

[44] Ayushi Rastogi and Nachiappan Nagappan. 2016. Forking and the Sustainability of the Developer Community Participation – An Empirical Investigation on Outcomes and Reasons. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, Vol. 1. 102–111. doi:10.1109/SANER.2016.27

[45] Baishakhi Ray, Daryl Posnett, Premkumar Devanbu, and Vladimir Filkov. 2017. A large-scale study of programming languages and code quality in GitHub. *Commun. ACM* 60, 10 (Sept. 2017), 91–100. doi:10.1145/3126905

[46] Eric S. Raymond and Bob Young. 2001. *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. O'Reilly & Associates, Inc., USA.

[47] Mike Savage and Roger Burrows. 2007. The Coming Crisis of Empirical Sociology. *Sociology* 41 (10 2007). doi:10.1177/0038038507080443

[48] C. E. Shannon. 1948. A mathematical theory of communication. *The Bell System Technical Journal* 27, 3 (1948), 379–423. doi:10.1002/j.1538-7305.1948.tb01338.x

[49] Diomidis Spinellis, Zoe Kotti, Konstantinos Kravvaritis, Georgios Theodorou, and Panos Louridas. 2020. A Dataset of Enterprise-Driven Open Source Software. In *Proceedings of the 17th International Conference on Mining Software Repositories* (Seoul, Republic of Korea) *(MSR '20)*. Association for Computing Machinery, New York, NY, USA, 533–537. doi:10.1145/3379597.3387495

[50] Stefan Stanciulescu, Thorsten Berger, Eric Walkingshaw, and Andrzej Wasowski. 2016. Concepts, Operations, and Feasibility of a Projection-Based Variation Control System. 323–333. doi:10.1109/ICSME.2016.88

[51] Stefan Stanciulescu, Sandro Schulze, and Andrzej Wasowski. 2015. Forked and integrated variants in an open-source firmware project. 151–160. doi:10.1109/ICSM.2015.7332461

[52] Igor Steinmacher, Tayana Conte, David Redmiles, and Marco Aurelio Gerosa. 2015. Social Barriers Faced by Newcomers Placing Their First Contribution in Open Source Software Projects. doi:10.1145/2675133.2675215

[53] Igor Steinmacher, Gustavo Pinto, Igor Wiese, and Marco Aurelio Gerosa. 2018. Almost there: a study on quasi-contributors in open source software projects. 256–266. doi:10.1145/3180155.3180208

[54] Xin Tan, Yiran Chen, Haohua Wu, Minghui Zhou, and Li Zhang. 2023. Is It Enough to Recommend Tasks to Newcomers? Understanding Mentoring on Good First Issues. 653–664. doi:10.1109/ICSE48619.2023.00064

[55] Xin Tan and Minghui Zhou. 2020. Scaling Open Source Software Communities: Challenges and Practices of Decentralization. *IEEE Software* PP (09 2020). doi:10.1109/MS.2020.3025959

[56] Xin Tan, Minghui Zhou, and Brian Fitzgerald. 2020. Scaling open source communities: an empirical study of the Linux kernel. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering* (Seoul, South Korea) *(ICSE '20)*. Association for Computing Machinery, New York, NY, USA, 1222–1234. doi:10.1145/3377811.3380920

[57] Bogdan Vasilescu, Vladimir Filkov, and Alexander Serebrenik. 2013. StackOverflow and GitHub: Associations between Software Development and Crowdsourced Knowledge. In *2013 International Conference on Social Computing*. 188–195. doi:10.1109/SocialCom.2013.35

[58] Bogdan Vasilescu, Daryl Posnett, Baishakhi Ray, Mark G.J. van den Brand, Alexander Serebrenik, Premkumar Devanbu, and Vladimir Filkov. 2015. Gender and Tenure Diversity in GitHub Teams. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (Seoul, Republic of Korea) *(CHI '15)*. Association for Computing Machinery, New York, NY, USA, 3789–3798. doi:10.1145/2702123.2702549

[59] Bogdan Vasilescu, Yue Yu, Huaimin Wang, Premkumar Devanbu, and Vladimir Filkov. 2015. Quality and Productivity Outcomes Relating to Continuous Integration in GitHub. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering* (Bergamo, Italy) *(ESEC/FSE 2015)*. Association for Computing Machinery, New York, NY, USA, 805–816. doi:10.1145/2786805.2786850

[60] Patrick Adam Wagstrom. 2009. *Vertical interaction in open software engineering communities*. Ph. D. Dissertation. Carnegie Mellon University.

[61] Liang Wang, Zhiwen Zheng, Xiangchen Wu, Baihui Sang, Jierui Zhang, and Xianping Tao. 2023. Fork Entropy: Assessing the Diversity of Open Source Software Projects' Forks. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 204–216. doi:10.1109/ASE56229.2023.00168

[62] Zhendong Wang, Yang Feng, Yi Wang, James A. Jones, and David Redmiles. 2020. Unveiling Elite Developers' Activities in Open Source Projects. 29, 3, Article 16 (jun 2020), 35 pages. doi:10.1145/3387111

[63] Ronald Wasserstein and Nicole Lazar. 2016. The ASA's Statement on p-Values: Context, Process, and Purpose. *The American Statistician* 70 (03 2016), 00–00. doi:10.1080/00031305.2016.1154108

[64] Jeffrey M. Wooldridge. 1999. Introductory Econometrics: A Modern Approach. https://api.semanticscholar.org/CorpusID:56503746

[65] Xiangchen Wu, Liang Wang, and Xianping Tao. 2025. EarlyPR: Early Prediction of Potential Pull-Requests from Forks. In *2025 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 575–586. doi:10.1109/SANER64311.2025.00060

[66] Yunwen Ye and K. Kishida. 2003. Toward an understanding of the motivation of open source software developers. In *25th International Conference on Software Engineering, 2003. Proceedings.* 419–429. doi:10.1109/ICSE.2003.1201220

[67] Shurui Zhou, Stefan Stanciulescu, Olaf Leßenich, Yingfei Xiong, Andrzej Wasowski, and Christian Kästner. 2018. Identifying Features in Forks. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*. 105–116. doi:10.1145/3180155.3180205

[68] Shurui Zhou, Bogdan Vasilescu, and Christian Kästner. 2019. What the fork: a study of inefficient and efficient forking practices in social coding. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Tallinn, Estonia) *(ESEC/FSE 2019)*. Association for Computing Machinery, New York, NY, USA, 350–361. doi:10.1145/3338906.3338918

[69] Shurui Zhou, Bogdan Vasilescu, and Christian Kästner. 2020. How has forking changed in the last 20 years? a study of hard forks on GitHub. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Companion Proceedings* (Seoul, South Korea) *(ICSE '20)*. Association for Computing Machinery, New York, NY, USA, 268–269. doi:10.1145/3377812.3390911