How Does Core Contributor Disengagement Impact Open Source Project Activity? A Quasi-Experiment

Yunqi Chen
Zhejiang University
Hangzhou, Zhejiang, China
yunqichen@zju.edu.cn

Klaas-Jan Stol
Lero and University College Cork
Cork, Ireland
k.stol@ucc.ie

Fabio Santos
Colorado State University
Fort Collins, CO, USA
fabio.deabreusantos@colostate.edu

Daniel German
University of Victoria
Victoria, BC, Canada
dmg@uvic.ca

Bianca Trinkenreich
Colorado State University
Fort Collins, CO, USA
bianca.trinkenreich@colostate.edu

Abstract

The sustainability of Open Source Software (OSS) projects often depends on a small group of core contributors. When these contributors disengage, projects may face disruption; however, the consequences of such disengagement on OSS contribution workflows remain underexplored. This paper presents a large-scale quasi-experimental study quantifying the impact of core contributor disengagement on pull request (PR) throughput, acceptance rate, and time to merge. We analyzed over 35 million PRs across 50,804 GitHub repositories, identifying 95,958 disengagement bursts (one or more core contributors becoming inactive for >365 days within the same week) involving 174,887 contributors. Using Difference-in-Differences with matched control groups, we found that the impact varies substantially with contributor and project characteristics. Disengagements of contributors with a high share of commits lead to pronounced declines in throughput and acceptance, while long-tenured contributors' disengagements have milder effects on those metrics but increase merge time, suggesting loss of tacit project knowledge. These findings provide empirical evidence of how core contributors' disengagement influences OSS workflows and highlight structural factors associated with project resilience.

CCS Concepts

• **Software and its engineering → Open source model**.

Keywords

Open source software, disengagement, attrition, productivity

ACM Reference Format:
Yunqi Chen, Klaas-Jan Stol, Fabio Santos, Daniel German, and Bianca Trinkenreich. 2026. How Does Core Contributor Disengagement Impact Open Source Project Activity? A Quasi-Experiment. In *2026 IEEE/ACM 48th International Conference on Software Engineering (ICSE '26), April 12–18, 2026, Rio de Janeiro, Brazil.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3744916.3773135

1 Introduction

Open Source Software (OSS) forms the backbone of today's digital infrastructure, with an estimated 60% of websites relying on OSS technologies [40]. Despite their importance, OSS projects are highly vulnerable to the disengagement of key contributors, which can negatively impact project sustainability [14]. The loss of contributors is a major concern among OSS maintainers, project, and community managers, second only to poor code quality [51]. Core developer turnover has been shown to be a common and often unplanned phenomenon, even in mature ecosystems such as Rust, where over 60% of Rust modules (called 'crates') experienced at least one turnover event over two years [24]. OSS project communities tend to follow an 'onion' structure [16], with most contributors participating either sporadically or for short periods [4, 13], or with a limited contribution rate. Only a limited number of core contributors are deeply involved in the daily development process. This dependency on a smaller number of core developers represents a major risk to OSS projects. Nearly two-thirds of 133 popular GitHub projects rely on only one or two developers to survive [3]. In a follow-up study, 16% of the sampled projects were abandoned by maintainers, and 59% of those did not survive [2].

Nearly half of OSS project failures can be attributed to internal team issues, such as contributor turnover and the loss of critical knowledge [14]. When core contributors—sometimes referred to as 'truck factor' contributors due to their indispensable roles [87]—leave a project, the impact can be severe. Over half of projects fail to recover from the departure of such contributors [2]. Core contributor turnover has been associated with declines in team productivity [55] and code quality [29, 72].

Although studies have investigated abandonment [53], early signs of disengagement from OSS projects [59], and the trade-off between productivity and resilience in the aftermath of these exits [66], there is a lack of understanding regarding the prevalence of core disengagement and its consequences for project activities. Despite being a common event, there is limited awareness of the potential long-term effects of disengagement. This highlights the need for systematic investigations into how such disengagements affect project-level collaboration and activity. Thus, we pose the following two research questions:

RQ1 What is the prevalence of core contributor disengagement?

RQ2 What impact does disengagement of core contributors have on remaining contributors' project activity?

Further, any consequences likely vary based on the profile of disengaged contributors, and the projects they disengage from. This paper presents the results of a quasi-experiment, involving a sample of over 50,000 OSS projects, investigating the trends of disengagement among core contributors over the past decade, as well as the impact of core contributors' disengagement on the project activities of remaining contributors. A replication package for this study is available online [11].

In the remainder of this paper, we discuss related work (Sec. 2), our research design (Sec. 3), followed by the results to RQ1 (Sec. 4) and RQ2 (Sec. 5). After discussing threats to validity (Sec. 6), we conclude this paper with a discussion of the results (Sec. 7).

2 Related Work

2.1 Reasons for Contributor Turnover

A wide variety of factors can contribute to disengagement of open source contributors. For some, a reduced level of interest to continue contributing can cause developers to disengage [41, 54]. For others, major life changes such as a change of job can cause contributors to disengage, particularly when the new role is incompatible with contributing to OSS (due to a lack of time or otherwise) [39, 41, 54, 70]. Other reasons include governance issues in an OSS project [41], frustration with uncivil and toxic behaviors [23, 52], a lack of inclusiveness experiences [22, 30, 60], or lack of peer support [54].

Several studies have explored behavioral signals of upcoming disengagement and found that early indicators, such as bursty contributions, limited social interactions, and low integration, can forecast attrition [59, 89, 93].

2.2 Consequences of Contributor Turnover

The consequences of the disengagement of contributors, especially the most experienced, can have a negative impact on the sustainability of the project. One issue that has been explored by the literature is the so-called 'truck factor' [3, 15, 28, 63]. A high truck factor indicates a resilient project with well-distributed knowledge, while a low truck factor signals fragility due to reliance on a few key contributors, especially if no new developers join a project [2].

A high turnover in OSS projects has negative consequences, including knowledge loss [43, 64], decreased software quality [29], decreased productivity [66], and lower survival probability [43, 72]. Turnover within software teams has also been associated with a significant increase in customer-reported defects due to loss of knowledge and experience [55], while also impacting project survival [43, 72], as approximately 80% of open source projects fail due to contributor turnover related issues [72]. Recent research [24] investigated the challenges of core developers' turnover in the Rust package ecosystem, including delayed updates and confusion over responsibilities. Our work complements this line of work by quantitatively estimating the causal impact of core contributor disengagement on project activity across multiple GitHub ecosystems, including metrics such as pull request (PR) throughput, PR acceptance rate, and PR time to merge.

Few studies in the Software Engineering domain have been using causal inference methods to assess the impact of core developer disengagement on project productivity [66]. Although contributor breaks have been noted [4, 7, 41], the phenomenon of a *disengagement burst*, where multiple core contributors leave within a short period, has hitherto not been studied. This study goes beyond estimating an average effect to examine the heterogeneous impacts on project PR workflow activities across different core contributor and repository characteristics, thereby providing a more nuanced and actionable perspective on the problem of core contributor disengagement.

3 Research Design

In this section we lay out our research design.

3.1 Causal Inference Framework

In order to understand how much of the change in project activity is due to developer disengagement, we require a causal inference approach. This allows us to draw conclusions on cause and effect, rather than only establishing associations between variables. Specifically, to empirically estimate the causal impact of core contributor disengagement bursts on GitHub project activities, we use difference-in-differences (DID) models [88] to investigate the effects of disengagement on project activities. DID models are widely used in econometrics and quantitative research in the social sciences [1, 9, 42, 65] and increasingly adopted in software engineering research [10, 25, 50, 66]. DID is a technique to look at the 'effect' of an 'intervention' from observational study data to mimic an experimental research design. It considers a 'treatment' group and a 'control' group.[1] For both, changes in mean values are calculated. In this study, we considered those projects that suffered from core contributors disengaging as the 'treatment' group, and compared changes in activity against projects that did not suffer from disengaged core contributors (the 'control' group). A key requirement for DID to be valid is the *parallel trends assumption*: if the disengagement had not occurred, the treatment group's activity would have followed a similar trend as the control group. Although we cannot directly test this *counterfactual* situation, we carefully selected control groups based on pre-treatment observable trends to ensure they share similar trends with the treatment group before the disengagement burst.

The DID technique is usually applied to a single event (treatment) at one point in time. Since disengagement events happen at different times across repositories in the real world, we adopt a DID design with what is called variations in treatment timing [35], to 'align' these events to a reference point. With such alignment, we can use the regression analysis as a 'traditional' DID and estimate heterogeneous effects of contributor and project characteristics that may moderate the impact of disengagement bursts (one or more core contributors becoming inactive within the same week) [19].

3.2 Data Collection and Preparation

To create the dataset for this study, we used SEART-GHS [18], which is a searchable dataset and tool that indexes GitHub projects based on metadata and metrics. We applied the following filtering criteria. We selected repositories that fulfilled the following criteria:

[1]Because this was not a controlled experiment but rather a quasi-experiment (or *'natural'* experiment), we quote the terms 'treatment,' 'control,' and 'intervention.'

- The project was created before November 2023, ensuring at least one year of development history;
- has at least 100 stars, indicating popularity;
- is not a fork project, ensuring originality;
- has a valid license, ensuring legal clarity for usage and contribution;
- contains open issues, indicating active maintenance and community engagement;
- has pull requests, reflecting collaborative development;
- includes contributions from at least 10 commit authors, demonstrating team involvement;
- has over 100 commits, indicating enough development activity.

From this initial dataset, we selected projects using one of the 10 most popular programming languages [33] as their main language: Python, JavaScript, Java, TypeScript, C++, Golang, C#, C, PHP, and Rust. Following prior work [94], we next excluded repositories potentially used for document storage or course submissions by filtering out projects with keywords such as "homework," "course," and "awesome" in their names.

For each project, we used the GitHub API to collect its history of project activities (including commits and pull requests) until February 2025. Projects whose repositories were deleted or made private during data collection were excluded, as their data could not be fetched. We removed commits from bots as they were identified by filtering login or username containing "bot," followed by manual validation. We then processed author information for commits lacking GitHub logins or with aliases by aggregating similar (name, email) tuples based on established methods [12, 34, 48, 85] to disambiguate author names.

Overall, the dataset used for this study comprised 50,804 projects, containing 35,804,751 pull requests (median per project: 175) and 106,812,303 commits (median per project: 715) from 2011 to February 2025. However, because our disengagement detection requires observing at least 12 months of inactivity, we would not be able to identify disengagements starting after Q1/2024. As a result, all trend plots have a cut-off point to show disengagement data only until the end of 2023.

3.3 Treatment and Control Groups

3.3.1 Treatment group definition. The treatment event is defined as a *disengagement burst*, which we define as one or more core contributors becoming inactive for >365 days within the same week. We tested several other disengagement thresholds (180, 270, and 450 days) to assess the sensitivity of the choice of this specific threshold (see the appendix [11]), and we found consistent results. Figure 1 shows a visual overview of the procedure to detect developer disengagement.

Identifying core contributors. We employed a *commit-based heuristic method* to mark those who made the top 80% commits over the project's entire lifecycle as *core contributors*; this is a much-used definition [7, 17, 27, 56]. We discuss potential threats to validity of this definition later. We excluded contributors who stayed less than 30 days (time from first to last commit) or contributed fewer than 10 commits.

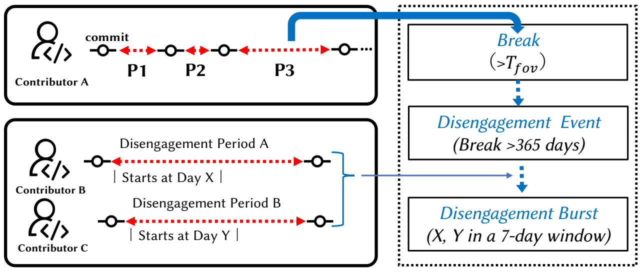

Figure 1: Operationalization of *disengagement burst*

Defining contributor engagement. We define a core contributor as being *engaged* in a project from the moment they have authored a first commit to the project [7].

Detecting inactive periods. To understand unique commit rhythms for each contributor, we first identified the *pauses* (intervals in days between consecutive commit days) for each contributor, creating an array of pauses $P =< p_1, p_2, \ldots, p_n >$. Following previous research [7], we identified *longer-than-usual pauses* as *breaks* by calculating contributor-specific thresholds T_{fov} using the *far out values* approach [81] to detect outliers in a distribution: $T_{fov} = Q_3(P) + 3 \times IQR$, where $Q_1(P)$ is the first quartile of the distribution, $Q_3(P)$ is the third quartile, and $IQR = Q_3(P) - Q_1(P)$ is the interquartile range. While T_{fov} characterizes individual rhythms for each contributor, breaks are not directly used to define the disengagement 'event' for our DID analysis. We defined breaks longer than 365 days as disengagement events and merged any occurring within a 7-day window into a single event.

Compiling disengagement bursts. To isolate the impact of a single disengagement burst and avoid confounding effects from nearby bursts, we focused exclusively on temporally isolated bursts. We chose bursts that were surrounded by a 12-week "buffer zone," which means no other bursts occurred for 12 weeks before or after the eligible burst in the same repository. To validate this approach, we performed a sensitivity analysis with several shorter durations of "buffer zone" (e.g., 6, 8, or 10 weeks) and found our main conclusions remained consistent (see the appendix [11]).

After applying these definitions and filters, our dataset comprised 95,958 eligible disengagement bursts (with 5,298 of them aggregated from multiple disengagements occurring within a week) from 174,887 individual core contributor disengagements (median 3 per project) across 35,229 of the original 50,804 projects.

3.3.2 Control Group Construction. To construct a valid control group for each treatment group at time t, we used *propensity score matching* (PSM) [8] to ensure the parallel trends assumption. PSM helps create comparable groups by matching treated units with control units that had a similar probability (propensity score) of receiving the treatment, based on pre-treatment observational characteristics. Specifically, for each treated repository experiencing a disengagement burst starting in week t, we identify potential control repositories that *did not* experience a burst in week t or within the surrounding 12-week window. Following previous studies [25, 50], we used a logistic regression model to fit the probability of a repository having core contributors disengaged by using the

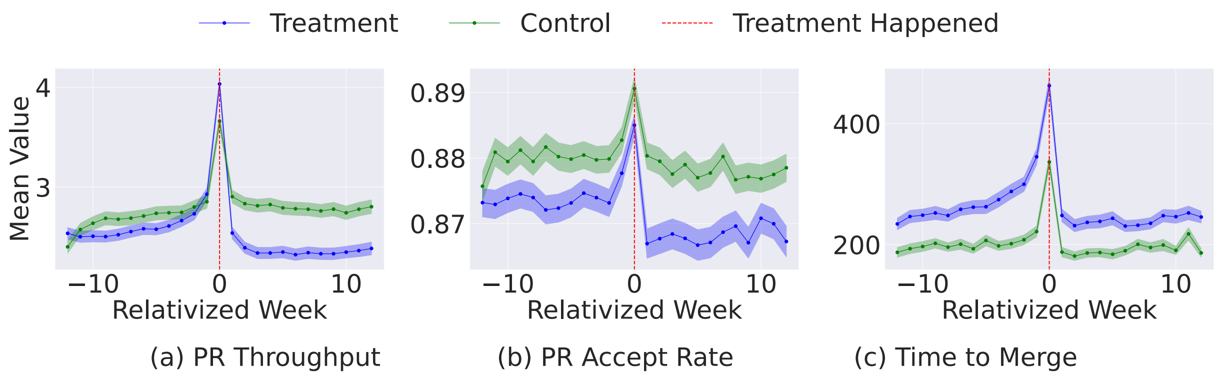

Figure 2: Pre-treatment trends across outcome variables to support the *parallel trend assumption*

log of relative increase (IP_{it}) in outcome variables and the absolute number of outcome variables ($\mathrm{PS}_{t_{ic}}$) for PR throughput, PR acceptance rate, and PR time-to-merge over the 12 weeks prior to week t. We used the log-transform to stabilize variance, handle skewed changes, and make relative increases interpretable, as proportional shifts happen across projects of different scales. The log relative increase is computed as:

$$I_{it} = \log \frac{y_{it} + 1}{y_{i(t-1)} + 1}$$

where y_{it} is the outcome variable for repository i in week t. We add one to both the numerator and the denominator to handle zero counts. The propensity score is calculated as follows:

$$\mathrm{PS}_{t_{ic}} = \sigma\Big(\sum_{j=1}^{12} \mathrm{IP}_{i(t-j)} + \frac{1}{12}\sum_{j=1}^{12} \mathrm{P}_{i(t-j)}\Big)$$

where $PS_{t_{ic}}$ is the probability repository i in cohort c has core contributors disengaged at time t. $IP_{i(t-j)}$ is the relative increase of a given outcome variable of project activity for repository i, and $P_{i(t-j)}$ is the summed number of that outcome variable for repository i, on the j^{th} week before treatment, respectively. We use the σ as a 'sigmoid' function to map the probability to a range between 0 and 1.

For each treated repository (that suffered from disengaged core contributors), we matched up to five control repositories with the closest propensity scores (1:5 matching); this is to make sure that control repositories have, on average, the same pre-treatment trend in outcome variables as the treatment group [8]. Figure 2 displays average weekly trends for (a) PR throughput, (b) PR acceptance rate, and (c) time-to-merge for treatment (blue line) and matched control (green line) groups relative to the disengagement burst week with 95% confidence interval error bands. These graphs show the pre- and post-treatment trend of outcome variables in both treatment and control groups, thus supporting the parallel trends assumption of the DID technique.

3.4 Dependent Variables

Core contributors play a central role in maintaining projects and ensuring code contributions are processed efficiently. When core contributors disengage (the event that serves as independent variable), we expect to see disruptions in the project's ability to review, respond to, and integrate contributions. We focus on three metrics that directly reflect these activities.

Pull Request (PR) Throughput. PR throughput captures the volume of PRs closed (either merged or rejected) within a given period. A drop in throughput may signal that fewer PRs are being processed, suggesting slower overall project activity. Previously, this metric was used as an indicator of project responsiveness and vitality [36, 84]. In our model, *PR Throughput* is defined as the number of PRs successfully merged on a weekly basis [47, 61, 74, 86].

Pull Request Acceptance Rate. The PR acceptance rate reflects the proportion of PRs that are successfully merged. A declining acceptance rate could imply that, in the absence of active core developers, contributions are more frequently left unreviewed or rejected due to lack of oversight, unclear direction, or insufficient integration support. It may also reflect a rise in gatekeeping or stalled decision-making processes that core developers usually facilitate. In our model, *PR Acceptance Rate* is defined as the proportion of PRs merged out of all pull requests that were opened and then either closed or merged on a weekly basis [20, 21, 38, 73, 76, 78, 92].

Time-to-Merge. Time-to-Merge measures how long it takes for a PR to be merged. As core developers play a key role in reviewing and merging contributions, their disengagement leads to less capacity within the project to process PRs, which is likely to result in longer review cycles, increased waiting times, or backlog accumulation. Prior work [79] has linked longer time to merge to reviewer unavailability and bottlenecks in collaborative workflows. In our model, *Time To Merge* is defined as the average time (in minutes) between when a PR is opened and when it is merged during the week [5, 36, 38, 49, 73, 74, 78, 86, 91]

Together, these metrics provide a triangulated view of project responsiveness and collaborative efficiency. They are observable, widely adopted in OSS analytics, and offer a reliable proxy for assessing the operational impact of core developer disengagement. By focusing on these three, we capture volume (throughput), decision tendencies (acceptance rate), and efficiency (merge time), which can be used as signals of shifting project health.

3.5 Moderating Variables

3.5.1 Disengaged Core Contributor Characteristics. We consider three characteristics of disengaged core contributors in the disengagement burst.

- *Tenure*: Average duration (in days) from first commit to last commit of contributors in the burst. Tenure can be interpreted as a proxy for contributor experience and project knowledge, even when "lurking" (i.e. not contributing). Simply a 'presence' within the community and interactions (e.g. mailing lists, IRC or Slack channels) may expose that developer to specific project knowledge that would be lost upon their departure.
- *Commit Percentage*: Summed proportion of commits relative to the total number of project commits by contributors in the burst. Commit percentage indicates how big a role the departed core contributor played in volume of commits that were made. While a rough measure, as commit size tends to vary considerably, we argue that, overall, these variations apply generally to most contributors.
- *Number of Commits*: Total number of commits made by contributors in the burst. Similar to commit percentage, but the number of commits represents absolute activity, whereas commit percentage represents activity in relation to others'.

3.5.2 Repository Characteristics. We consider the following treatment repository characteristics until the time of the disengagement burst.

- *Number of Commits*: Total number of commits in the project up to the point of burst. This is an indicator of overall activity within a project.
- *Number of Contributors*: Total number of contributors in the project up to the point of burst. This is an indicator of the size of the project's community: the effect of core contributors leaving a small project (i.e. with a small developer community) might be larger than when there are many contributors left who could potentially 'step up.' A project's survival rate is positively linked to its number of contributors [67].
- *Project Age*: Age of the project (in days) since the first commit up to the point of disengagement. Projects that have been active for a longer time are more likely to continue to survive [67].
- *Newcomer Count*: Number of new contributors who made their first commit to the project within 12 weeks after the disengagement happened.
- *Main Language*: Primary programming language of the repository (10 categories: C, C#, C++, Go, Java, JavaScript, PHP, Python, Rust, and TypeScript). The popularity of languages changes over time, with some becoming less popular (e.g. PHP), while others are becoming very popular at the time of this study (e.g. Go and Rust).

3.6 Control Variables

Control variables are included in the regression models (Equations (1) and (2), shown in the next section) to account for time-varying factors:

- *Project Commits*: Number of commits made to the repository.
- *Project Contributors*: Number of contributors who made at least one commit to the repository.
- *Project Age*: Number of days since the first commit in the project.

We briefly note the difference between control variables and similarly named repository characteristics. Repository characteristics are fixed snapshot measurements taken prior to the disengagement event of the treatment repository. For each cohort, the characteristics, such as project commits or ages, are assigned based on the treatment repository value at that point, and they remain constant within each pair. In contrast, control variables can differ not only between repositories within the same cohort, but also across different time points for the same repository.

3.7 Model Specification

We refer to all the pre- and post-treatment observations for a given treatment and corresponding control group as a *cohort*. Our analysis is based on weekly observational data.

3.7.1 Average Treatment Effect (ATE) Estimation. We set our first regression model to examine the general average causal effect of core contributor disengagement burst on project activities as below:

$$Y_{itc} = \beta_0 Treat_{ic} \times Post_{tc} + \beta_1 Treat_{ic} + \beta_2 Post_{tc} + \mathbf{X}_{itc}\boldsymbol{\beta}_3 + \gamma_{tc} + \alpha_{ic} \quad (1)$$

Y_{itc} represents the outcome variable for repository i at week t in cohort c. $Treat_{ic}$ is a binary indicator for whether repository i is the treatment repository within cohort c. $Post_{tc}$ is a binary indicator for whether week t is in the post-treatment period for cohort c. $\mathbf{X}_{itc}$ is a vector of time-varying control variables (i.e., project commits, project contributors, and project age) for repository i at week t in cohort c, with $\boldsymbol{\beta}_3$ being the corresponding vector of coefficients. γ_{tc} is "time-cohort" and α_{ic} is "repository-cohort" random effects that capture nested variations specific to the repository between cohorts. The coefficient of interest is β_0, the DID estimator to show the average change in outcome for the treatment repositories post-disengagement, compared to the counterfactual trend estimated from its control repository pairs.

3.7.2 Heterogeneous Effect Estimation. To understand if certain types of projects or contributors are more resilient or vulnerable to disengagement, we investigate how this effect varies based on specific characteristics of the repository or the disengaged core contributor(s). To achieve this, we extend the basic DID model by incorporating interaction terms between the main DID estimator $Treat_{ic} \times Post_{tc}$ and the characteristics of interest $\mathbf{Z}_{ic}$ measured at the time of the disengagement burst, forming a Difference-in-Difference-in-Differences (DDD) analysis [90]. The model is specified as follows:

$$Y_{itc} = \boldsymbol{\beta}_0 Treat_{ic} \times Post_{tc} \times \mathbf{Z}_{ic} + \beta_1 Treat_{ic} + \beta_2 Post_{tc} + \mathbf{X}_{itc}\boldsymbol{\beta}_3 + \gamma_{tc} + \alpha_{ic} \quad (2)$$

where the interaction term $Treat_{ic} \times Post_{tc} \times \mathbf{Z}_{ic}$ ensures that the characteristics of the repository or disengaged core contributor(s) will only affect the value of fitted outcome variables post-treatment. The coefficient vector $\boldsymbol{\beta}_0$ should be interpreted as the moderating effect of those characteristics on the outcome variables.

To analyze the categorical predictor *project_main_language* in our models, we employed **sum contrast coding** (also known as effects coding). The core principle of this method is to compare the effect of each level against the **grand mean** of all levels for that variable [71]. This approach is more suitable than R's default treatment contrast when a study lacks a natural control or baseline group, and the goal is to investigate the performance of each category relative to the overall trend. The coefficients for our specified reference level (`JavaScript`) are not explicitly shown in the model output, as they are implicitly defined by the **sum-to-zero** constraint.

3.7.3 Model Validation. When estimating regressions, we take standard precautions such as log-transforming skewed variables to reduce hetero-scedasticity [32] and checking for multi-collinearity using the variance inflation factor [77]. We report marginal (R^2_m – variance explained by fixed effects) and conditional (R^2_c – variance explained by both fixed and random effects) coefficients of determination as goodness-of-fit measures for generalized mixed-effects models [44, 57].

4 Prevalence of Core Contributor Disengagement in the Past Decade

To answer RQ1 *(What is the prevalence of core contributor disengagement?)*, we report trends of core contributors engaged and disengaged in the data sample from 2011-2023.

4.1 Core Contributor (Dis)Engagement

Figure 3 shows the trends (on a quarterly basis) of new engaged (blue) and disengaged core contributors (red), as well as the distributions across projects as a scatterplot. The number of new core contributors rose from around 1,000 in Q1/2011 to over 2,500 by Q1/2014. It then remained relatively stable between 2,500 and 3,500 from 2014 to Q3/2020 before dropping to 1,163 in Q4/2023, returning to levels seen in 2011. Meanwhile, quarterly disengagements started with just 248 (25% of the engagements), peaked at 3,389 in Q2/2021, and then fell to 1,813 in Q4/2023, about 1.6 times the number of newcomers.

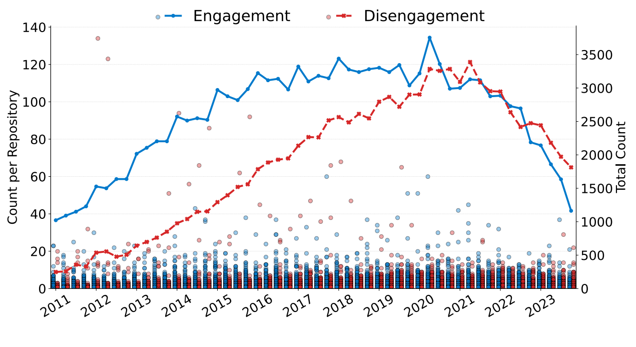

Figure 3: Trend of core contributor disengagement vs. engagement, 2011 to 2023 (right axis). Scatter plots (left axis) show the count of contributors (engaged and disengaged) per repository.

Comparing engagement and disengagement of core contributors overall, we find that from early 2015 to mid-2020, more core contributors disengaged, while engagement remained stable from Q1/2014 to Q3/2020. Although disengagement increased during this period, the number of core contributors joining still outpaced those leaving until Q3/2020. This means the overall pool of core contributors grew from 2015 to Q3/2020 with a slowing pace. After Q3/2020, both engagement and disengagement numbers declined. However, in this period, the number of disengagement consistently exceeded engagement, indicating that the overall number of core contributors started to reduce after Q3/2020. The timing of this turning point coincides with the global aftermath of the first wave of the COVID-19 pandemic. This period introduced sustained stressors: remote work fatigue, increased caregiving responsibilities, job insecurity, and mental health challenges [58]. While some contributors may have initially increased OSS activity during early lockdowns (e.g., engagement peaks in Q2/2020), the longer-term pressures of the pandemic may have led to burnout and withdrawal from volunteer-driven communities by the end of 2020.

Beyond these general trends, we also examined the distribution of engagement and disengaged core contributors on a per-project basis, as shown by the blue and red scatter plot in Figure 3. The vast majority of projects see one or two core contributors joining or leaving per quarter. However, some outlier projects have more than 10 (even over 100) core contributors gained or lost within a single quarter, potentially significantly impacting project activities.

4.2 Trends across GitHub Repositories Based on Project Characteristics

We segmented the sample by three project characteristics. First, *main project language*: different programming languages have different contributor pool sizes, are associated with different levels of software code quality [62], and project popularity [6]. Second, *project age*: the code growth curve of OSS projects that typically starts rapidly, then slows and levels off as projects mature and require less effort to add features [45, 46, 80, 82]. As a project ages, it may enter into different patterns of engagement and disengagement. Due to space constraints, we only present representative figures for these dimensions. The full set of plots is available in the appendix [11].

4.2.1 Trends by main project language. Examination of the plots by main language reveals three trends of core contributor engagement and disengagement across different programming languages in our dataset: *growth*, *stability*, and *attrition*.

First, a trend of *growth*, i.e., a net growth of core contributor pool for Go, Rust, and PHP projects. Rust (see Figure 4(c)) displayed a sustained high level of engagement relative to disengagement, with engagement peaking around Q2/2022 before declining, though it remained higher than disengagement.

Second, a trend of *stability* can be identified when the numbers of engaged and disengaged are roughly commensurate, resulting in a relatively stable contributor pool size over time. This trend is evident in Python, C++, and TypeScript projects. In Python projects (see Figure 4(b)), this shift from net inflow to a balance between engagement and disengagement is noted around Q4/2020.

Third, a trend of *attrition* can be observed when the number of disengaging core contributors surpasses new engagements from some time points, implying a net decrease in core contributors. This is observable in projects with C, C#, Java, and JavaScript. For example, for C projects (see Figure 4(a)) the shift occurred after 2016 when disengagement consistently exceeded engagement thereafter.

4.2.2 Trends by project age. Project age is a critical factor in OSS disengagement [2]. We separate project age into 4 groups: younger than 1 year, 1–2 years, 2–4 years, and older than 4 years.

Projects in the '<1 year' and '1-2 years' age groups are usually at the initial stage of the project 'lifecycle,' and generally experience a consistent net inflow of core contributors. Projects in the '<1-year' group, as displayed in Fig. 4(d), show a net inflow with engagement consistently higher than disengagement. Peak engagement for this group was observed around Q2/2020. This inflow period extends to the '1-2 years' age group, as shown in Fig. 4(e), although the gap between engagement and disengagement is narrower than the '<1 year' group, and engagement started to decline after Q4/2018. This indicates that while both younger categories attract core contributors, the number of net core contributor inflow is greater in the earliest stages of a project.

A transition appears for '2-4 years' projects (Fig. 4(f)); the dynamics shift towards a more precarious balance. While there are periods of net inflow, particularly before 2017, the periods after 2018 show that disengagement approaches or even exceeds engagement. Overall, projects in this age group begin to show signs of potential net attrition.

Finally, projects older than 4 years (Fig. 4(g)), which we may call long-established projects, exhibit an apparent challenge of sustainability. The number of disengaged core contributors markedly and consistently exceeds new engagements. This sustained loss, particularly after 2017, reveals the challenge long-running projects may face in maintaining or attracting core contributors.

Observation 1: OSS projects have been following different trajectories in core contributor engagement. Ecosystems like Rust, Go, and PHP consistently attracted more core contributors than they lost, while others, such as JavaScript, Java, C, and C#, faced sustained losses. Younger projects (<1 year) have been attracting more core contributors than they lose. As projects age (1–4 years), gains and losses become more balanced. In projects older than 4 years, disengagement more often exceeds engagement, indicating greater retention challenges in older projects.

Given this widespread phenomenon of core contributor disengagement, it is important to understand its potential effects. Therefore, we now shift our focus to understanding the effects of such disengagements on project activities of those contributors who stayed.

5 Impact of Core Contributor Disengagement on Project Activity

To answer RQ2 *(What impact does disengagement of core contributors have on remaining contributors' project activity?)*, we use difference-in-differences (DID) models (see Sec. 3) to investigate the effects of disengagement on project activities. We started with the general effects due to the disengagement of core developers, as detailed in Table 1. Our primary focus is the two-way interaction term (**Is treated group: Is post-treatment**), which represents the estimated effect on the treatment group (i.e., those projects that suffered from disengaged core contributors) during the post-treatment period. Note that the dependent variable is log-scaled, coefficients can be interpreted as an approximate percentage of increase. On average projects exhibited approximately 10% drop in the number of PR accepted (Table 1, Model 1, *pr_throughput*, $\beta = -0.099$, $p < .001$) and a slight but significant, about 0.3% decrease in the rate at which PRs are accepted (Table 1, Model 2, *pr_accept_rate*, $\beta = -0.003$, $p < .001$) over the following 12 weeks. However, the story does not end there. For those PRs that are accepted, the process appears to speed up, with about 4% reduction in the average time taken to merge them (Table 1, Model 3, *time_to_merge*, $\beta = -0.043$, $p < .001$). This suggests that while losing core contributors slows down overall PR activities' flow and reduces PR acceptance rate, remaining team members appear to be processing the PRs more rapidly.

While the general effects of disengagement revealed significant disruptions to PR activity following the disengagement of core contributors, this general analysis does not distinguish between different project and contributor contexts. To develop a better understanding of what factors might moderate these effects, we investigated the influence of core contributor characteristics and repository characteristics.

5.1 Moderating Disengagement Effects

We segmented the analysis of the consequences of disengagement according to two groups of moderating variables (described in Sec. 3.5): *core contributor characteristics* (i.e., tenure, commit percentage, number of commits) and *repository characteristics.* (i.e., number of commits, number of contributors, age, main language) act as interacting *moderators*, potentially altering the strength of the previously observed disengagement effects.

We analyzed 3-way interaction terms with DDD models [90] from Equation (2), which allows us to check if the impact of disengagement (the difference in outcomes between the 'treated' and 'control' groups after disengagement) changes significantly when a specific contributor or repository characteristic is concurrently considered. For example, does the drop in PR throughput vary depending on the tenure of the departed core contributor? Table 2 presents the results of this segmented analysis (see Models 4-6).

5.1.1 Moderating Effects of Disengaged Core Contributor Characteristics. We examined the moderating influence of three characteristics of disengaged core contributors: PROJECT TENURE, COMMIT PERCENTAGE OF THE PROJECT'S TOTAL COMMITS, and NUMBER OF COMMITS (see 'Core Contributor Characteristics' in Table 2).

Longer tenure lessens drop in PR throughput and acceptance rates, but increases the time to merge. As Table 2 shows, when a core contributor with longer tenure leaves, projects show resilience. The usual drop in PR throughput is significantly smaller ($\beta = .041$, $p < .001$, Model 4), and the PR acceptance rate loss is also slightly mitigated ($\beta = .001$, $p < 0.01$, Model 5). However, the longer core contributors have stayed, their disengagement lengthens the time required to merge PRs ($\beta = .025$, $p < .001$, Model 6). This could

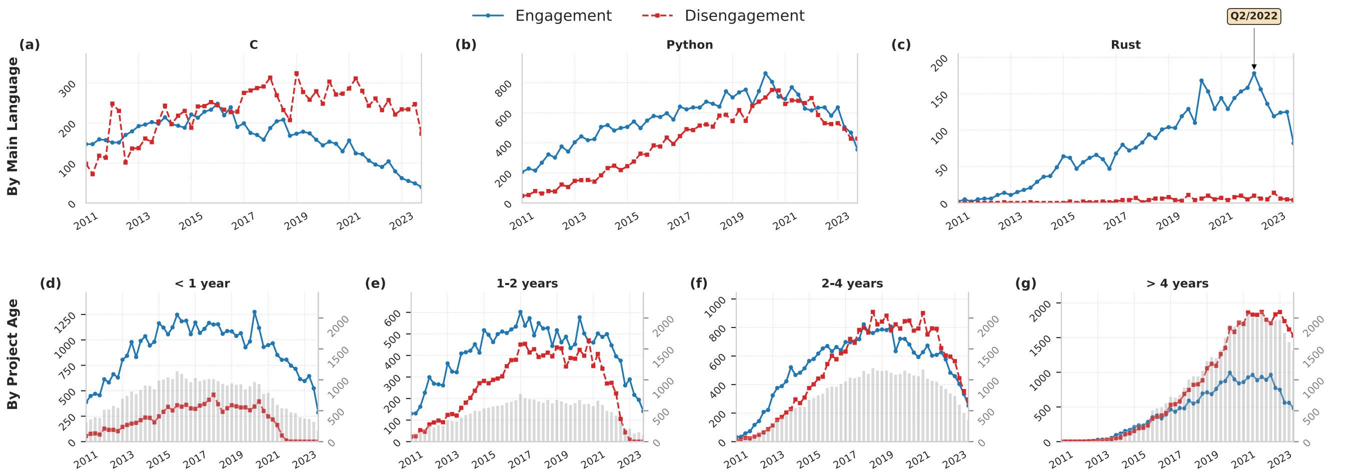

Figure 4: Core contributor (dis)engagement trends across C, Python, and Rust, as well as per age category. The histograms show the number of repositories for which the total number of (dis)engaged core contributors were calculated at each point in time.

Table 1: General effect of core contributor disengagement on project activity

	Model 1	Model 2	Model 3
Outcome (log)	*pr_throughput*	*pr_accept_rate*	*time_to_merge*
Main Treatment Effects			
Is post-treatment	0.721***	0.000	−0.063***
Is treated group	−0.067***	−0.002**	0.153***
Is treated group : Is post-treatment	−0.099***	−0.003***	−0.043***
Controls (log)			
project commits	0.268***	0.006***	0.003
project contributors	0.326***	−0.025***	0.470***
project age	−0.213***	0.005***	0.037***
Number of observations	3,482,133	1,372,097	1,372,097
R^2_m (R^2_c)	0.26 (0.66)	0.02 (0.27)	0.08 (0.37)

Note: * *p < 0.05;* ** *p < 0.01;* *** *p < 0.001*

suggest that, while overall throughput and acceptance remain relatively stable, replacing a long-tenured core contributor's specialized knowledge of the merge process takes time.

Higher commit percentage worsens the loss in PR throughput and PR acceptance rate, but reduces the time to merge. Losing a core contributor with a larger portion of the project's commits significantly reduces PR throughput (β = −0.024, p < .001, Model 4), and reduces the PR acceptance rate, making approvals more difficult (β = −0.003, p < .001, Model 5). Nevertheless, a higher commit percentage from the disengaged contributor is linked to a shorter time to merge for PRs (β = −0.020, p < .001, Model 6).

Departure of a core contributor with a higher commit count exacerbates PR throughput loss but has no detected impact on other activities. Furthermore, a higher number of commits by the disengaged core contributor significantly causes a greater reduction in PR throughput (β = −0.020, p < .001, Model 4). However, the number of commits does not appear to significantly moderate the PR acceptance rate or the time to merge, as these effects are not statistically significant.

Observation 2: Contributor characteristics significantly shape how disengagement impacts project activity. The loss of a contributor with a high share of commits leads to a sharper decline in PR throughput and acceptance rate, alongside faster merge times. In contrast, losing a long-tenured contributor results in milder declines in throughput and acceptance rate but still accelerates merging. A higher total number of commits is associated only with reduced throughput, indicating that volume of past contributions alone is a weaker predictor of disruption.

5.1.2 Moderating Effects of Repository Characteristics. We examined the moderating influence of different repository characteristics

Table 2: Repository and core contributor characteristics as moderators of the effect of core contributor disengagement

	Model 4	Model 5	Model 6
Outcome (log)	*pr_throughput*	*pr_accept_rate*	*time_to_merge*
Core Contributor Characteristics (Sec. 5.1.1)[1]			
Tenure (log)	0.041***	0.001**	0.025***
Commit Percentage (log)	−0.024***	−0.003***	−0.020***
#Commits (log)	−0.020***	0.000	−0.001
Repository Characteristics (Sec. 5.1.2)[1]			
#Commits before treatment (log)	0.020***	0.000	0.063***
#Contributors before treatment (log)	−0.003	−0.001	−0.050***
Project Age before treatment (log)	−0.015***	0.000	−0.037***
#Newcomers after treatment (log)	0.011***	−0.001*	0.013***
MainLanguage-JavaScript (Reference)			
MainLanguage-C++	−0.058***	−0.004***	−0.095***
MainLanguage-C	0.019***	0.001	0.015
MainLanguage-C#	0.051***	0.006***	0.039
MainLanguage-Go	−0.010**	0.002*	0.046***
MainLanguage-Java	−0.006*	−0.002**	0.037***
MainLanguage-PHP	−0.019***	−0.002	−0.006
MainLanguage-Python	0.017***	0.001	−0.036***
MainLanguage-Rust	−0.062***	0.000	−0.024
MainLanguage-TypeScript	0.078***	0.001	0.045
Controls (log)			
Project commits	0.265***	0.005***	−0.011**
Project contributors	0.322***	−0.024***	0.478***
Project age	−0.207***	0.005***	0.042***
Number of observations	3,482,133	1,372,097	1,372,097
R^2_m (R^2_c)	0.26 (0.66)	0.02 (0.27)	0.08 (0.37)

[1] All are 3-way interaction terms with Is treated group? : Is post-treatment?, omitted for clarity.
*Note: * p < 0.05; ** p < 0.01; *** p < 0.001*

(i.e., number of commits, contributors, age, newcomers after disengagement, and main programming language) on PR throughput, acceptance rate, and time to merge (see Table 2).

Projects Commit Count. Projects with more commits experienced a small but significant increase in PR throughput after core contributor disengagement (β = .020, p < .001, Model 4) but a slightly longer time to merge pull requests (β = .063, p < .001, Model 6). The total number of project commits did not have a significant effect on the PR acceptance rate.

Projects Contributor Count. Projects with more contributors before core contributor disengagement had a small but significantly decline in PR throughput (β = −0.003, p < .001, Model 4) and a faster time to merge pull requests (β = −0.050, p < .001, Model 6). The number of contributors did not significantly affect the PR acceptance rate.

Projects Age. Older projects experienced a significantly smaller reduction in PR throughput after core contributor disengagement (β = −0.015, p < .001, Model 4), but faster time to merge pull requests (β = −0.037, p < .001, Model 6). Project age does not significantly affect PR acceptance rate.

Newcomers after Disengagement. Receiving a higher influx of new contributors after disengagement is associated with a smaller increase in PR throughput (β = .011, p < .001, Model 4), a slight reduction in PR acceptance (β = .001, p < .05, Model 5), and also slight longer time to merge pull requests (β = .061, p < .001, Model 6).

Programming Languages Moderated Disengagement Effects in Distinct Ways. For PR THROUGHPUT (Model 4), projects using C++ (β = −0.058, p < .001), PHP (β = −0.019, p < .001), and Rust (β = −0.062, p < .001) exhibited significantly greater reductions. Go (β = −0.010, p < .01) and Java (β = −0.006, p < .05) also showed small but significant negative moderation. In contrast, TypeScript (β = 0.078, p < .001), C# (β = 0.051, p < .001), C (β = 0.019, p < .001), and Python (β = 0.017, p < .001) were associated with higher throughput after disengagement.

For PR ACCEPTANCE RATE (Model 5), C++ (β = −0.004, p < .001) and Java (β = −0.002, p < .01) showed significant reduction. Conversely, C# (β = 0.006, p < .001) and Go (β = 0.002, p < .05) were associated with increased acceptance rates.

For TIME TO MERGE (Model 6), shorter merge times were observed in C++ (β = −0.095, p < .001) and Python (β = −0.036, p < .001). In contrast, Go (β = 0.046, p < .001) and Java (β = 0.037, p < .001) were associated with longer merge times, suggesting slower integration workflows after disengagement.

Observation 3: Repository characteristics shaped how projects absorbed the impact of core contributor disengagement. Projects with more commits or older age saw smaller declines in PR throughput but experienced longer merge times. In contrast, projects with more contributors experienced sharper PR throughput drops but merged more quickly. An influx of newcomers helps maintain throughput, yet also delays merges.

6 Threats to Validity

Internal Validity. The difference-in-differences technique is a quasi-experimental design. This approach assumes treated and control projects would have followed parallel trends in the absence of core contributor disengagement. While we tested and validated the parallel trends assumption in our pre-treatment data, unobserved confounding factors (e.g., concurrent organizational changes, dependency updates, or policy shifts) may still influence project activity in ways not attributable solely to contributor disengagement. To mitigate this, we used a large-scale sample with matched controls and ran robustness checks across alternative disengagement thresholds and effect length.

External Validity. This study offers robust insights into the impact of core contributors within PR-based workflows, which are widely adopted across OSS projects [36, 37, 78, 83, 95]. We focus on three key metrics (PR throughput, acceptance rate, and time to merge) enabled by the structured, fine-grained data available in PR-based systems. This focus ensures methodological consistency and supports scalable analysis across a large number of projects. While the findings apply to projects using PR workflows, future work can extend this analysis to include commit-based activities, broadening the scope to other collaboration models.

Construct Validity. We defined disengagement as the absence of activity (e.g., commits, PRs, issue comments) from previously active core contributors over a defined window. While this threshold-based approach aligns with prior OSS research [3, 7, 14], it may miss nuanced forms of disengagement such as diminished influence or temporary breaks. Our activity-based outcome metrics (PR throughput, acceptance rate, and time to merge) capture responsiveness but do not reflect review quality or team dynamics. Core contributor status is inferred from commit share across the project lifecycle, which may misclassify contributors whose roles evolve. We also do not account for contributors' compensation status, which could moderate disengagement behavior. Finally, our burst model, defined as multiple disengagements within the same week and buffered by 12 weeks, limits detection of staggered or smaller-scale events. We also do not distinguish burst size or role composition, which future work should explore to capture heterogeneous disengagement dynamics.

Additionally, we did not differentiate between paid and volunteer contributors. Compensation status could influence contributors' motivations, time commitment, and susceptibility to disengagement, which may moderate the observed effects and can be targeted by future research.

Lastly, our analysis defines a disengagement *"burst"* as multiple core contributors becoming inactive within the same week, with a 12-week buffer before and after to isolate the effects. While this design helps prevent confounding from overlapping disengagements, it limits our ability to examine smaller or staggered disengagement events. For instance, if two disengagements occur within a short interval but outside a one-week window, they are treated as separate, and both may be excluded due to proximity. Furthermore, we do not currently distinguish the size or composition of bursts, such as whether the departure of three contributors has disproportionately greater impact than that of two. Future work should explore heterogeneous burst structures, including single-person disengagements and variations in contributor roles or tenure, to better understand how the scale and context of disengagement events affect downstream project outcomes.

Conclusion Validity. Our statistical models yield consistent and significant results across multiple outcomes and robustness specifications. However, the practical effect sizes, particularly for acceptance rate, are small and should be interpreted accordingly. We use conservative significance thresholds and include interaction effects to explore moderators, but correlation does not imply causation, especially in observational data. While the Difference-in-Differences approach strengthens our causal inference, we cannot rule out the influence of unmeasured variables. Future work incorporating qualitative insights or mixed-methods validation could help triangulate and extend our conclusions.

7 Discussion and Conclusion

We now discuss the key findings and directions for further research.

7.1 Disrupted Throughput, Accelerated Merges: Signs of Compensatory Pressure?

The results for RQ2 show a reduction in PR activity when core contributors have disengaged: PR throughput drops by 10%, and PR acceptance rate reduces by .3% (Table 1, Models 1–2). Yet, for accepted PRs, merges happen 4% faster on average (Table 1, Model 3). This combination suggests a possible compensatory shift in team behavior. While fewer in number, those PRs that are accepted may be processed more quickly, potentially reflecting an effort to maintain momentum despite reduced capacity. Whether this acceleration reflects improved efficiency or pressure-driven shortcuts (e.g. faster but less thorough reviews) remains unclear, and a direction for future work.

Our findings show that a higher total number of commits by the disengaged contributor was associated with a steeper decline in throughput, yet this disengagement had no detectable impact on pull request acceptance or merge time. This aligns with existing research indicating that the loss of developers responsible for a large share of commits, reflecting a low bus factor, can sharply reduce project activity levels [15]. Contributors with substantial commit histories play a critical role in sustaining throughput, and their disengagement leads to significant drops in productivity even when the acceptance rates of incoming contributions remain stable.

7.2 Contributor Characteristics as Moderators

Contributor characteristics moderate disengagement effects in distinct and sometimes unexpected ways. We found that the longer the tenure of core contributors who leave a project, the smaller the reduction in PR throughput and acceptance rate; further, we found that the time-to-merge was reduced. This finding indicates some resilience in processing volume and decision-making. This pattern may reflect a temporary increase in urgency, or a 'recalibration' of the workflow (redistribution of responsibilities) among remaining developers that removes bottlenecks, or the presence of stronger underlying team structures in projects resilient to the loss of long-tenured contributors. In contrast, losing contributors with a higher commit percentage *intensified* negative effects on throughput and acceptance, though merges became faster—possibly due to a similar redistribution mechanism.

7.3 Trading efficiency and capacity

Older projects exhibit a trade-off between efficiency and capacity following core contributor disengagement. Our results show that older projects experience some reduction in PR throughput after core contributors' disengagement (Table 2, $\beta = -0.015$, $p < .001$, Model 4), but the PRs tend to be merged more quickly (Table 2, $\beta = -0.037$, $p < .001$, Model 6). Mature projects may enter a triage mode: with reduced capacity, remaining core contributors may need to focus on a smaller subset of PRs, often those that are more critical, and handle them more efficiently. Such behavior may be enabled by streamlined workflows, established norms, or technical maturity. However, as shown in RQ1 (Sec. 4.2.2), older projects have faced a persistent net loss of core contributors over time [29], indicating that, despite their short-term procedural resilience, older projects face mounting sustainability challenges [2]. Their ability to merge PRs quickly may mask a gradual erosion in review capacity and throughput, potentially leading to accumulated backlog or contributor frustration [24]. The combination of recurrent loss of core talent (RQ1) and reduced PR throughput (RQ2) points to a structural tension between operational efficiency and long-term viability. Future research could explore how these projects adapt over time, whether through tooling, modularization, or shifting governance, to maintain resilience in the face of ongoing contributor turnover.

7.4 New Contributors Help Sustain Project Activity

An influx of new contributors can help sustain project activity, as shown by the smaller decline in PR throughput. However, this also coincides with longer merge times, which may reflect onboarding delays, less familiar code contributions, or increased coordination needs. The presence of newcomers does not significantly impact PR acceptance rates. Future work could examine targeted onboarding strategies to reduce merge delays associated with newcomer contributions [26, 75], including mentorship programs, structured review pairings, or automated onboarding tools that guide new contributors through project conventions and workflows. Additionally, a more granular analysis of the types of newcomer contributions (e.g. bug fixes, feature additions, or documentation updates) could help identify specific activities aligned to newcomer skills [68, 69] that minimize coordination overhead while effectively sustaining project throughput.

7.5 New Languages, New Lifelines

RQ1 revealed that projects in newer languages like Rust continue to attract more core contributors than they lose, while older-language projects such as Java and C show sustained net attrition (Figure 4(c)). This broader trend provides important context for RQ2, where we found that different programming languages moderated the effects of core contributor disengagement on project activity (Table 2, Models 4–6). While Rust continues to grow in core contributor engagement (RQ1), its projects experienced significant PR throughput reductions post-disengagement (Table 2 $\beta = -0.062$, $p < .001$), indicating that language popularity alone does not guarantee resilience. This contrast may reflect a tension between technical promise and social integration within Rust ecosystems. As seen in recent Linux kernel debates [31], the adoption of Rust has sparked "almost religious" divisions between C and Rust communities, with contributors reporting burnout from non-technical conflicts around infrastructure and cultural expectations. These sociotechnical frictions may limit Rust projects' ability to quickly redistribute responsibilities after core turnover, despite the language's reputation for safety and modern tooling. Thus, Rust's ecosystem illustrates that technical advantages alone may not guarantee procedural resilience without aligned community norms and tooling maturity.

The results presented in this paper are not always intuitive or expected (such as simultaneously having a reduction in PR throughput and decrease in time to merge). This seems to imply that more research is necessary to fully understand the dynamics and consequences of core contributor's disengagements in open source projects.

Acknowledgments

This work is supported by Research Ireland (formerly Science Foundation Ireland) grant 13/RC/2094-P2 to Lero.

References

[1] Joshua D Angrist and Jörn-Steffen Pischke. 2009. *Mostly harmless econometrics: An empiricist's companion.* Princeton university press.
[2] Guilherme Avelino, Eleni Constantinou, Marco Tulio Valente, and Alexander Serebrenik. 2019. On the abandonment and survival of open source projects: An empirical investigation. In *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM).* IEEE, 1–12.
[3] Guilherme Avelino, Leonardo Passos, Andre Hora, and Marco Tulio Valente. 2016. A novel approach for estimating truck factors. In *2016 IEEE 24th International Conference on Program Comprehension (ICPC).* IEEE, 1–10.
[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* (2020).
[5] João Helis Bernardo, Daniel Alencar da Costa, Uirá Kulesza, and Christoph Treude. 2023. The impact of a continuous integration service on the delivery time of merged pull requests. *Empirical Software Engineering* 28, 4 (2023), 97.
[6] Hudson Borges, Andre Hora, and Marco Tulio Valente. 2016. Understanding the factors that impact the popularity of GitHub repositories. In *2016 IEEE international conference on software maintenance and evolution (ICSME).* IEEE, 334–344.
[7] Fabio Calefato, Marco Aurelio Gerosa, Giuseppe Iaffaldano, Filippo Lanubile, and Igor Steinmacher. 2022. Will you come back to contribute? Investigating the inactivity of OSS core developers in GitHub. *Empirical Software Engineering* 27, 3 (2022), 76.
[8] Marco Caliendo and Sabine Kopeinig. 2008. Some practical guidance for the implementation of propensity score matching. *Journal of Economic Surveys* 22, 1 (2008), 31–72.

[9] Brantly Callaway and Pedro HC Sant'Anna. 2021. Difference-in-differences with multiple time periods. *Journal of econometrics* 225, 2 (2021), 200–230.

[10] Annalí Casanueva, Davide Rossi, Stefano Zacchiroli, and Théo Zimmermann. 2025. The impact of the COVID-19 pandemic on women's contribution to public code. *Empirical Software Engineering* 30, 1 (2025), 1–35.

[11] Yunqi Chen, Klaas-Jan Stol, Fabio Santos, Daniel German, and Bianca Trinkenreich. 2025. Appendix to "How Does Core Contributor Disengagement Impact Open Source Project Activity?". https://figshare.com/s/dddc84c102241c665be6

[12] Yunqi Chen, Zhiyuan Wan, Yifei Zhuang, Ning Liu, David Lo, and Xiaohu Yang. 2025. Understanding the OSS Communities of Deep Learning Frameworks: A Comparative Case Study of PyTorch and TensorFlow. *ACM Transactions on Software Engineering and Methodology* 34, 3 (2025), 1–30.

[13] Jinghui Cheng and Jin LC Guo. 2019. Activity-based analysis of open source software contributors: Roles and dynamics. In *IEEE/ACM 12th International Workshop on Cooperative and Human Aspects of Software Engineering*. IEEE, 11–18.

[14] Jailton Coelho and Marco Tulio Valente. 2017. Why modern open source projects fail. In *2017 11th Joint meeting on foundations of software engineering*. 186–196.

[15] Valerio Cosentino, Javier Luis Cánovas Izquierdo, and Jordi Cabot. 2015. Assessing the bus factor of git repositories. In *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, 499–503.

[16] Kevin Crowston and James Howison. 2005. The social structure of free and open source software development. *First Monday* 10, 2 (2005).

[17] Kevin Crowston, Kangning Wei, Qing Li, and James Howison. 2006. Core and periphery in free/libre and open source software team communications. In *39th annual Hawaii international conference on system sciences (HICSS'06)*, Vol. 6. IEEE, 118a–118a.

[18] Ozren Dabic, Emad Aghajani, and Gabriele Bavota. 2021. Sampling Projects in GitHub for MSR Studies. In *18th IEEE/ACM International Conference on Mining Software Repositories, MSR 2021*. IEEE, 560–564.

[19] Clément De Chaisemartin and Xavier d'Haultfoeuille. 2023. Two-way fixed effects and differences-in-differences with heterogeneous treatment effects: A survey. *The econometrics journal* 26, 3 (2023), C1–C30.

[20] Tapajit Dey, Andrey Karnauch, and Audris Mockus. 2021. Representation of developer expertise in open source software. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 995–1007.

[21] Tapajit Dey and Audris Mockus. 2020. Effect of technical and social factors on pull request quality for the npm ecosystem. In *14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. 1–11.

[22] Christina Dunbar-Hester. 2020. *Hacking diversity: The politics of inclusion in open technology cultures*. Princeton University Press.

[23] Ramtin Ehsani, Rezvaneh Rezapour, and Preetha Chatterjee. 2023. Exploring moral principles exhibited in oss: A case study on github heated issues. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 2092–2096.

[24] Meng Fan, Yuxia Zhang, Klaas-Jan Stol, and Hui Liu. 2025. Core Developer Turnover in the Rust Package Ecosystem: Prevalence, Impact, and Awareness. *ACM on Software Engineering* 2, FSE (2025), 2759–2781.

[25] Hongbo Fang, Hemank Lamba, James Herbsleb, and Bogdan Vasilescu. 2022. "This is damn slick!" estimating the impact of tweets on open source project popularity and new contributors. In *44th International Conference on Software Engineering*. 2116–2129.

[26] Zixuan Feng, Katie Kimura, Bianca Trinkenreich, Anita Sarma, and Igor Steinmacher. 2024. Guiding the way: A systematic literature review on mentoring practices in open source software projects. *Information and Software Technology* (2024), 107470.

[27] Fabio Ferreira, Luciana Lourdes Silva, and Marco Tulio Valente. 2020. Turnover in open-source projects: The case of core developers. In *XXXIV Brazilian Symposium on Software Engineering*. 447–456.

[28] Mívian Ferreira, Marco Tulio Valente, and Kecia Ferreira. 2017. A comparison of three algorithms for computing truck factors. In *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*. IEEE, 207–217.

[29] Matthieu Foucault, Marc Palyart, Xavier Blanc, Gail C Murphy, and Jean-Rémy Falleri. 2015. Impact of developer turnover on quality in open-source software. In *2015 10th joint meeting on foundations of software engineering*. 829–841.

[30] Hana Frluckaj, Laura Dabbish, David Gray Widder, Huilian Sophie Qiu, and James D Herbsleb. 2022. Gender and participation in open source software development. *ACM on Human-Computer Interaction* 6, CSCW2 (2022), 1–31.

[31] B. Cameron Gain. 2024. The New Stack: Linus Torvalds: C vs. Rust Debate Has 'Religious Undertones'. https://thenewstack.io/linus-torvalds-c-vs-rust-debate-has-religious-undertones/ Accessed: June 2025.

[32] Andrew Gelman and Jennifer Hill. 2007. *Data analysis using regression and multilevel/hierarchical models*. Cambridge university press.

[33] GitHub. 2024. Octoverse 2024: The State of Open Source and Software Development. https://github.blog/news-insights/octoverse/octoverse-2024/ Accessed: November 2023.

[34] Mathieu Goeminne and Tom Mens. 2013. A comparison of identity merge algorithms for software repositories. *Science of Computer Programming* 78, 8 (2013), 971–986.

[35] Andrew Goodman-Bacon. 2021. Difference-in-differences with variation in treatment timing. *Journal of econometrics* 225, 2 (2021), 254–277.

[36] Georgios Gousios, Martin Pinzger, and Arie van Deursen. 2014. An exploratory study of the pull-based software development model. In *36th international conference on software engineering*. 345–355.

[37] Georgios Gousios, Margaret-Anne Storey, and Alberto Bacchelli. 2016. Work practices and challenges in pull-based development: The contributor's perspective. In *38th international conference on software engineering*. 285–296.

[38] Georgios Gousios, Andy Zaidman, Margaret-Anne Storey, and Arie Van Deursen. 2015. Work practices and challenges in pull-based development: The integrator's perspective. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 1. IEEE, 358–368.

[39] Mariam Guizani, Amreeta Chatterjee, Bianca Trinkenreich, Mary Evelyn May, Geraldine J Noa-Guevara, Liam James Russell, Griselda G Cuevas Zambrano, Daniel Izquierdo-Cortazar, Igor Steinmacher, Marco A Gerosa, et al. 2021. The long road ahead: Ongoing challenges in contributing to large oss organizations and what to do. *Proceedings of the ACM on Human-Computer Interaction* 5, CSCW2 (2021), 1–30.

[40] Morela Hernandez, David A. Hyman, Charles W. Ibser, and Sonali K. Shah. 2021. The Digital Economy Runs on Open Source. Here's How to Protect It. *Harvard Business Review* (21 September 2021). https://hbr.org/2021/09/the-digital-economy-runs-on-open-source-heres-how-to-protect-it

[41] Giuseppe Iaffaldano, Igor Steinmacher, Fabio Calefato, Marco Aurélio Gerosa, and Filippo Lanubile. 2019. Why do developers take breaks from contributing to OSS projects. *A preliminary analysis. CoRR abs/1903.09528 (2019)* (2019), 1–8.

[42] Guido W Imbens. 2024. Causal inference in the social sciences. *Annual Review of Statistics and Its Application* 11 (2024).

[43] Daniel Izquierdo-Cortazar, Gregorio Robles, Felipe Ortega, and Jesus M Gonzalez-Barahona. 2009. Using software archaeology to measure knowledge loss in software projects due to developer turnover. In *2009 42nd Hawaii International Conference on System Sciences*. IEEE, 1–10.

[44] Paul CD Johnson. 2014. Extension of Nakagawa & Schielzeth's R2GLMM to random slopes models. *Methods in ecology and evolution* 5, 9 (2014), 944–946.

[45] Nicolas Jullien, Robert Viseur, and Jean-Benoît Zimmermann. 2025. A theory of FLOSS projects and Open Source business models dynamics. *Journal of Systems and Software* 224 (June 2025), 112383. https://doi.org/10.1016/j.jss.2025.112383

[46] Stefan Koch and Georg Schneider. 2002. Effort, co-operation and co-ordination in an open source software project: GNOME. *Information Systems Journal* 12, 1 (2002), 27–42.

[47] Oleksii Kononenko, Tresa Rose, Olga Baysal, Michael Godfrey, Dennis Theisen, and Bart De Water. 2018. Studying pull request merges: A case study of shopify's active merchant. In *40th international conference on software engineering: software engineering in practice*. 124–133.

[48] Erik Kouters, Bogdan Vasilescu, Alexander Serebrenik, and Mark GJ Van Den Brand. 2012. Who's who in Gnome: Using LSA to merge software repository identities. In *2012 28th IEEE International Conference on Software Maintenance (ICSM)*. IEEE, 592–595.

[49] Chandra Maddila, Chetan Bansal, and Nachiappan Nagappan. 2019. Predicting pull request completion time: a case study on large scale cloud services. In *2019 27th acm joint meeting on european software engineering conference and symposium on the foundations of software engineering*. 874–882.

[50] Danaja Maldeniya, Ceren Budak, Lionel P Robert Jr, and Daniel M Romero. 2020. Herding a deluge of good samaritans: How github projects respond to increased attention. In *Web Conference 2020*. 2055–2065.

[51] Josianne Marsan, Mathieu Templier, Patrick Marois, Bram Adams, Kevin Carillo, and Georgia Leida Mopenza. 2018. Toward solving social and technical problems in open source software ecosystems: using cause-and-effect analysis to disentangle the causes of complex problems. *IEEE Software* 36, 1 (2018), 34–41.

[52] Courtney Miller, Sophie Cohen, Daniel Klug, Bogdan Vasilescu, and Christian KaUstner. 2022. " Did you miss my comment or what?" understanding toxicity in open source discussions. In *Proceedings of the 44th international conference on software engineering*. 710–722.

[53] Courtney Miller, Mahmoud Jahanshahi, Audris Mockus, Bogdan Vasilescu, and Christian Kästner. 2025. Understanding the response to open-source dependency abandonment in the npm ecosystem. In *47th IEEE/ACM International Conference on Software Engineering*.

[54] Courtney Miller, David Gray Widder, Christian Kästner, and Bogdan Vasilescu. 2019. Why do people give up flossing? a study of contributor disengagement in open source. In *Open Source Systems: 15th IFIP WG 2.13 International Conference, OSS 2019, Montreal, QC, Canada, May 26–27, 2019, Proceedings 15*. Springer, 116–129.

[55] Audris Mockus. 2010. Organizational volatility and its effects on software defects. In *eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*. 117–126.

[56] Audris Mockus, Roy T Fielding, and James D Herbsleb. 2002. Two case studies of open source software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 11, 3 (2002), 309–346.

[57] Shinichi Nakagawa and Holger Schielzeth. 2013. A general and simple method for obtaining R2 from generalized linear mixed-effects models. *Methods in ecology and evolution* 4, 2 (2013), 133–142.

[58] Paulo Anselmo da Mota Silveira Neto, Umme Ayda Mannan, Eduardo Santana De Almeida, Nachiappan Nagappan, David Lo, Pavneet Singh Kochhar, Cuiyun Gao, and Iftekhar Ahmed. 2021. A deep dive into the impact of COVID-19 on software development. *IEEE Transactions on Software Engineering* 48, 9 (2021), 3342–3360.

[59] Mian Qin, Yuxia Zhang, Klaas-Jan Stol, and Hui Liu. 2025. Who Will Stop Contributing to OSS Projects? Predicting Company Turnover Based on Initial Behavior. *ACM on Software Engineering* 2, FSE (2025), 2782–2805.

[60] Huilian Sophie Qiu. 2022. *Understanding and Designing Mechanisms for Attracting and Retaining Open-Source Software Contributors*. Ph. D. Dissertation. Carnegie Mellon University.

[61] Mohammad Masudur Rahman and Chanchal K Roy. 2014. An insight into the pull requests of github. In *11th working conference on mining software repositories*. 364–367.

[62] Baishakhi Ray, Daryl Posnett, Vladimir Filkov, and Premkumar Devanbu. 2014. A large scale study of programming languages and code quality in github. In *22nd ACM SIGSOFT international symposium on foundations of software engineering*. 155–165.

[63] Filippo Ricca and Alessandro Marchetto. 2010. Are heroes common in FLOSS projects?. In *2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*. 1–4.

[64] Martin P Robillard. 2021. Turnover-induced knowledge loss in practice. In *29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1292–1302.

[65] Jonathan Roth, Pedro HC Sant'Anna, Alyssa Bilinski, and John Poe. 2023. What's trending in difference-in-differences? A synthesis of the recent econometrics literature. *Journal of Econometrics* 235, 2 (2023), 2218–2244.

[66] Giuseppe Russo Latona, Christoph Gote, Christian Zingg, Giona Casiraghi, Luca Verginer, and Frank Schweitzer. 2024. Shock! Quantifying the Impact of Core Developers' Dropout on the Productivity of OSS Projects. In *Companion ACM Web Conference 2024*. 706–709.

[67] Ioannis Samoladas, Lefteris Angelis, and Ioannis Stamelos. 2010. Survival analysis on the duration of open source projects. *Information and Software Technology* 52, 9 (2010), 902–922.

[68] Fabio Santos, Jacob Penney, João Felipe Pimentel, Igor Wiese, Igor Steinmacher, and Marco A Gerosa. 2023. Tell me who are you talking to and i will tell you what issues need your skills. In *2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR)*. IEEE, 611–623.

[69] Fabio Santos, Bianca Trinkenreich, João Felipe Pimentel, Igor Wiese, Igor Steinmacher, Anita Sarma, and Marco A Gerosa. 2022. How to choose a task? mismatches in perspectives of newcomers and existing contributors. In *16th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. 114–124.

[70] Jaydeb Sarker, Asif Kamal Turzo, and Amiangshu Bosu. 2025. The Landscape of Toxicity: An Empirical Investigation of Toxicity on GitHub. *Proceedings of the ACM on Software Engineering* 2, FSE (2025), 623–646.

[71] Daniel J Schad, Shravan Vasishth, Sven Hohenstein, and Reinhold Kliegl. 2020. How to capitalize on a priori contrasts in linear (mixed) models: A tutorial. *Journal of memory and language* 110 (2020), 104038.

[72] Andreas Schilling. 2014. What do we know about FLOSS developers' attraction, retention, and commitment? A literature review. In *2014 47th Hawaii International Conference on System Sciences*. IEEE, 4003–4012.

[73] Daricélio Moreira Soares, Manoel Limeira de Lima Júnior, Leonardo Murta, and Alexandre Plastino. 2015. Acceptance factors of pull requests in open-source projects. In *30th annual ACM symposium on applied computing*. 1541–1546.

[74] Fangchen Song, Ashish Agarwal, and Wen Wen. 2024. The impact of generative AI on collaborative open-source software development: Evidence from GitHub Copilot. *arXiv preprint arXiv:2410.02091* (2024).

[75] Igor Steinmacher, Sogol Balali, Bianca Trinkenreich, Mariam Guizani, Daniel Izquierdo-Cortazar, Griselda G Cuevas Zambrano, Marco Aurelio Gerosa, and Anita Sarma. 2021. Being a mentor in open source projects. *Journal of Internet Services and Applications* 12 (2021), 1–33.

[76] Josh Terrell, Andrew Kofink, Justin Middleton, Clarissa Rainear, Emerson Murphy-Hill, Chris Parnin, and Jon Stallings. 2017. Gender differences and bias in open source: Pull request acceptance of women versus men. *PeerJ Computer Science* 3 (2017), e111.

[77] Christopher Glen Thompson, Rae Seon Kim, Ariel M Aloe, and Betsy Jane Becker. 2017. Extracting the variance inflation factor and other multicollinearity diagnostics from typical regression results. *Basic and applied social psychology* 39, 2 (2017), 81–90.

[78] Jason Tsay, Laura Dabbish, and James Herbsleb. 2014. Influence of social and technical factors for evaluating contribution in GitHub. In *36th international conference on Software engineering*. 356–366.

[79] Jason Tsay, Laura Dabbish, and James Herbsleb. 2014. Let's talk about it: evaluating contributions through discussion in GitHub. In *22nd ACM SIGSOFT international symposium on foundations of software engineering*. 144–154.

[80] Qiang Tu et al. 2000. Evolution in open source software: A case study. In *Proceedings 2000 International Conference on Software Maintenance*. IEEE, 131–142.

[81] John Wilder Tukey et al. 1977. *Exploratory data analysis*. Vol. 2. Springer.

[82] Marat Valiev, Bogdan Vasilescu, and James Herbsleb. 2018. Ecosystem-level determinants of sustained activity in open-source projects: A case study of the PyPI ecosystem. In *2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 644–655.

[83] Bogdan Vasilescu, Stef Van Schuylenburg, Jules Wulms, Alexander Serebrenik, and Mark GJ van den Brand. 2014. Continuous integration in a social-coding world: Empirical evidence from GitHub. In *2014 IEEE international conference on software maintenance and evolution*. IEEE, 401–405.

[84] Bogdan Vasilescu, Yue Yu, Huaimin Wang, Premkumar Devanbu, and Vladimir Filkov. 2015. Quality and productivity outcomes relating to continuous integration in GitHub. In *2015 10th joint meeting on foundations of software engineering*. 805–816.

[85] Chengcheng Wan, Shicheng Liu, Henry Hoffmann, Michael Maire, and Shan Lu. 2021. Are machine learning cloud apis used correctly?. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 125–137.

[86] Mairieli Wessel, Joseph Vargovich, Marco A Gerosa, and Christoph Treude. 2023. Github actions: the impact on the pull request process. *Empirical Software Engineering* 28, 6 (2023), 131.

[87] Laurie Williams and Robert R Kessler. 2003. *Pair programming illuminated*. Addison-Wesley Professional.

[88] Jeffrey M Wooldridge. 2016. *Introductory econometrics a modern approach*. South-Western cengage learning.

[89] Wenxin Xiao, Hao He, Weiwei Xu, Yuxia Zhang, and Minghui Zhou. 2023. How early participation determines long-term sustained activity in github projects?. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 29–41.

[90] Yaxuan Yin and Jacob Thebault-Spieker. 2025. The Effect of Population Density on Remote Humanitarian Mapping Activities: A Triple-Difference Analysis. *Proceedings of the ACM on Human-Computer Interaction* 9, 2 (2025), 1–21.

[91] Yue Yu, Gang Yin, Tao Wang, Cheng Yang, and Huaimin Wang. 2016. Determinants of pull-based development in the context of continuous integration. *Science China Information Sciences* 59 (2016), 1–14.

[92] Xunhui Zhang, Yue Yu, Georgios Gousios, and Ayushi Rastogi. 2022. Pull request decisions explained: An empirical overview. *IEEE Transactions on Software Engineering* 49, 2 (2022), 849–871.

[93] Minghui Zhou and Audris Mockus. 2015. Who will stay in the floss community? modeling participant's initial behavior. *IEEE Transactions on Software Engineering* 41, 1 (2015), 82–99.

[94] Shurui Zhou, Bogdan Vasilescu, and Christian Kästner. 2019. What the fork: a study of inefficient and efficient forking practices in social coding. In *2019 27th ACM joint meeting on european software engineering conference and symposium on the foundations of software engineering*. 350–361.

[95] Jiaxin Zhu, Minghui Zhou, and Audris Mockus. 2016. Effectiveness of code contribution: From patch-based to pull-request-based tools. In *2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. 871–882.