Shock! Quantifying the Impact of Core Developers' Dropout on the Productivity of OSS Projects

Giuseppe Russo Latona
EPFL
giuseppe.russo@epfl.ch

Christoph Gote
ETH Zurich
cgote@ethz.ch

Christian Zingg
ETH Zurich
czingg@ethz.ch

Giona Casiraghi
ETH Zurich
gcasiraghi@ethz.ch

Luca Verginer
ETH Zurich
lverginer@ethz.ch

Frank Schweitzer
ETH Zurich
fschweitzer@ethz.ch

ABSTRACT

Open Source Software (OSS) projects play a critical role in the digital infrastructure of companies and services provided to millions of people. Given their importance, understanding the resilience of OSS projects is paramount. A primary reason for OSS project failure is the shock caused by the dropout of a core developer, which can jeopardize productivity and project survival. Using a difference-in-differences (DiD) analysis, this study investigates the repercussions of this shock on the productivity of 8,234 developers identified among 9,573 OSS GitHub projects. Our findings reveal the indirect impact of the core developer's dropout. The remaining developers experienced a 20% productivity drop. This observation is troubling because it suggests that the shock might push other developers to drop out, putting the collaboration structure of the project at risk. Also, projects with higher productivity before the shock experienced a larger drop-down after the shock. This points to a tradeoff between productivity and resilience, i.e., the ability of OSS projects to recover from the dropout of a core developer. Our findings underscore the importance of a balanced approach in OSS project management, harmonizing productivity goals with resilience considerations.

CCS CONCEPTS

• **Software and its engineering → Software creation and management**; **Collaboration in software development**; • **Human-centered computing → Empirical studies in collaborative and social computing**.

KEYWORDS

Online Collaboration Networks, Resilience, Causal Inference

ACM Reference Format:

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 Proceedings of the ACM Web Conference 2024 (WWW '24 Companion), May 13–17, 2024, Singapore, Singapore.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3589335.3651559

WWW '24 Companion, May 13–17, 2024, Singapore, Singapore

ACM ISBN 979-8-4007-0172-6/24/05
https://doi.org/10.1145/3589335.3651559

1 INTRODUCTION

Estimated 60% of global websites rely on Open Source Software (OSS) [9]. A Forbes survey of the Global Fortune 2000 companies [2] found that 30% of these companies consider OSS projects 'extremely critical' to their team's success. These two numbers underscore the importance of the stability and performance of these projects key for the digital ecosystem. However, OSS projects frequently face shocks [3], which may jeopardize the projects' security, functionality, and overall performance and eventually cause them to fail.

In this work, we study how the dropout of a core developer, henceforth referenced as a *shock*, effects the OSS project. Specifically, we address two research questions: (**RQ1**): How does the dropout of a core developer impact the *productivity* of the remaining developers? (**RQ2**): Are projects with higher initial productivity more *resilient* to such shocks? To answer these research questions, we need to solve three methodological problems: (i) to identify OSS projects suffering from the specified shock, (ii) to compare the productivity of projects before and after the shock, and (iii) to ensure that observed changes in productivity can indeed be attributed to the shock. To do so, we rely on causal inference techniques, specifically on a two-stage matching process and a Difference-in-Difference analysis.

Findings. We find that the dropout of core developers reduces the project's productivity two-fold: First, because of the missing contributions of the core developer, and second because of the *additional impact* on the remaining developers. Our analysis reveals a decline in productivity of the remaining developers of up to 20% compared to those in non-exposed projects (**RQ1**).

This diminished productivity, persisting for at least 16 weeks, indicates that projects indeed had to struggle to recover from the shock. This highlights the importance of project *resilience*, defined as its ability to withstand and recover from shocks [19]. Our analysis reveals that small, but highly productive projects experience a more significant reduction in productivity after the shock. Hence, we observe a *tradeoff* between productivity and resilience: Highly productive projects are less resilient because their productivity incurs a larger drop-down in case of a shock (**RQ2**).

2 RELATED WORK

Why OSS developers leave projects. There are various reasons why developers leave OSS projects. They have found a new job or

lost interest in the project [11], or become involved in a different project [1]. Most OSS projects consist of a small number of core developers and a large number of peripheral developers. While turnover among peripheral developers is common, the loss of a core developer can significantly harm the project because of the loss of knowledge [10]. As a result, Coelho and Valente [3] found that over 30% of the studied OSS projects terminated after their main developers left.

The resilience of OSS projects. The question of whether OSS projects are capable of recovering from the loss of a core developer is nowadays addressed using the term resilience. Various factors have been identified to impact the resilience of OSS projects [3], including contributor motivation, project sustainability, and community management strategies. Crowston and Howison [5] have explored the social structure of OSS projects, highlighting the importance of collaboration and communication for project success. Similarly, Herbsleb [7] examined the role of turnover in OSS projects and emphasized the resulting challenges. Our work extends this line of research by explicitly focusing on the shock on the projects' productivity after the sudden dropout of a core developer.

3 DATA

GitHub Data. We utilize GHTorrent [6], to collect over 125 million repositories from GitHub with their commits, developers, and co-editing information. As most of these projects are not collaborative (e.g., 67% have a single contributor or 56% of the projects on GitHub are forks of original repositories), we discard projects with the following criteria detailed by Kalliamvakou et al. [8]: (1) repositories with only one developer, (2) repositories with fewer than fifty commits, (3) repositories whose duration between the oldest and newest commit is less than 100 days, and (4) repositories that were forked from pre-existing projects. After the filtering, we are left with 4.5 million projects.

As we study the impact of a core developer dropping out, we seek OSS projects with one particularly active developer. To identify these projects, we compute for each of the 4.5 million projects (1) the normalized skewness of the commits distribution per developer, (2) the project duration, and (3) the activity period of the core developer in the project. Following this procedure, we identify 9,573 projects with an extremely active developer (e.g. skewness higher than 0.8).

Treatment and Control. Our study aims to understand how the dropout of the core developer from an OSS project impacts the productivity of remaining developers in the same project. To quantify this impact, we compare the productivity of developers in OSS projects where the core developer left (*treatment group*) to the productivity of developers in projects that were not exposed to such a shock (*control group*).

The projects for the treatment group are identified by setting a threshold for the time lapsed after the core developer's last commit. To determine this threshold, we adopt the method by Scholtes et al. [18] who demonstrated that after 42 weeks without a commit by a developer, the probability of further contributions to the project drops to 10%.

Thus, we establish a minimum time threshold of 42 weeks (294 days) between the core developer's last commit and our analysis date. We identified 101 projects where the core developers left with 8,324 *treated* developers. Each developer in this *treatment group* is associated with a single project.

Covariates	
Project Size	*(1) Number of developers*: those that made one commit in the project. *(2) Team size*: number of developers that edited other developers' code. *(3) Number of co-editing interactions.*
Collaboration Structure	*(1) Diameter of the collaboration network*: it measures the ease of collaboration between developers on a project. *(2) Density of the network*: intensity of the collaboration between developers. *(3) Clustering Coefficient*: characterize the presence of sub-teams. *(4) Mean Foreign Modification Ratio*: tendency of developers to edit each others' code.
Work Overload	*(1) Commits Skewness*: quantifies the distribution of commits among developers to determine if the workload disproportionately falls on specific individuals. *(2) Harmonic Centralization*: Dependence of a project on a few central contributors.
Developers' Covariates	
Ego Network	*(1) the Number of Distinct Connections*: it measures the size of a developer's collaboration neighborhood. *(2) The Sum of Edge Weights*: it quantifies the collaboration intensity of a developer. *(3) Number of Ego Components*: it measures how much each individual bridges different groups of the network.
External Effort Activity	*(1) Focus Score*: A metric to assess a developer's effort distribution across projects. It is the weighted average of the cosine similarity between the embeddings of the project associated with a developer and other projects where the developer contributed, with weights being the number of commits in each project [12].

Table 1: Covariates used in the two-stage matching

Productivity Outcome. To measure developer productivity, we use *Halstead's effort* metric. It proxies the effort required to write a program, based on the number of operators and operands. Specifically, if N_1 is the total number and n_1 the distinct number of *operators*, while N_2 is the total number and n_2 the distinct number of *operands*, the effort is calculated as the product of the program's difficulty ($D = \frac{n_1}{2} \times \frac{N_2}{n_2}$) and its volume ($V = (N_1 + N_2) \times \log_2(n_1 + n_2)$). A decrease in this metric indicates more complex code, which is potentially harder to maintain and can also imply a higher likelihood of errors or bugs. Recent studies favor Halstead's effort as a preferred productivity metric among managers [3]. Additionally, we use two productivity metrics complementary to Halstead's effort: (1) *Levenshtein distance* of character edits in a commit and (2) *Cyclometric complexity*, reflecting the number of independent paths through the code. Because the results are similar to Halstead's effort, we focus on the latter.

4 METHODS

To estimate the causal impact of a core developer's dropout on the productivity of the remaining developers, we combine matching procedures and difference-in-differences methods [17].

Two-Stage Matching. Following [15], we pair each developer from the treatment group with a developer from the control group. This two-stage matching allows us to (1) account for selection bias and (2) balance the distribution of covariates of the treatment and

control groups. This mitigates the risk that observed differences in post-shock productivity are caused by project or developer covariates and not by the dropout of the core developer.

We assign each project j to either the treated ($s_j = 1$) or the control ($s_j = 0$) group, depending on whether the project experienced the shock. For each project we calculate the propensity of experiencing such a shock, $\mathbb{P}[s_j = 1|X_i]$, given pre-event project characteristics X_j. Next, we match each project with $s_j = 1$ with a subset of control projects, C'_j, from the control group C such that $C'_j = \text{argmin}_{k \in C} ||\mathbb{P}[s_j|X_j] - \mathbb{P}[s_k|X_k]|| \leq 0.005$.

In a subsequent step, we match each developer i from the treated project j with a developer from a the subset of control projects C'_j, using the nearest neighbour algorithm to find the closest match. We validate the effectiveness of our two-stage matching procedure by calculating the absolute standardized mean differences (SMD) for the covariates between the treatment and control groups. The values were below the standard threshold of 0.1 for all matching covariates. This demonstrates that the matching procedure effectively controls for potential biases arising from differences in observed covariates between the treatment and control groups.

Difference-in-Difference. For the analysis, we use the sample of matched developers from 16 weeks before and after the shock to estimate the impact of this event on the developer productivity with the following difference-in-differences (DiD) model:

$$\log(Y_{it}) = \beta_0 + \beta_1 \text{Treated}_i + \beta_2 \text{Period}_t + \beta_3(\text{Treated}_i \times \text{Period}_t) + u_i + p_i + \epsilon_{it} \quad (1)$$

where Y_{it} is the outcome (e.g., Halstead's effort) for developer i in period t. Treated_i denotes whether developer i was exposed to the shock, and Period_t indicates whether the current week t is before (0) or after the core developer's departure. u_i captures the fixed effects for developer i, and p_i the fixed effects for the project to which developer i contributed. ϵ_{it} is the error term.

The reliability of the DiD analysis relies on the validity of the parallel trends assumption. It posits that were there no shock, the differences in outcome between the treated and control post-event would be equal to the pre-event differences, leading to a zero difference-in-differences estimate. The coefficient in the model capturing this causal effect is β_3. It isolates the causal impact of the core developer's dropout on the productivity of remaining developers, accounting for any pre-existing differences.

Additionally, we take address the problem of self-selection bias. Specifically, we account for time-invariant developer and project characteristics (such as project maturity and collaborative attitude), which are presumed stable over our observation window of 20 weeks (four weeks before and sixteen weeks after the shock). Using fixed effects, we effectively control for potential biases due to these characteristics, thereby reducing the risk of omitted variable bias.

5 RESULTS

5.1 RQ1: The Effect of Core Developers Dropout

Overall Interaction Effect. We assess the consequences of a core developer's dropout on the remaining developers' productivity using the DiD regression model, detailed in eq. (1). The study spans 32 weeks, evenly divided into pre and post-shock periods of 16 weeks each. Our DiD analysis reveals that, after adjusting for pre-shock differences, treated developers faced a decline of 19% of Halstead's effort (similar results hold for Levenshtein distance and Cyclomatic complexity with a decline of 13.1% and 19.3%, respectively). This decline underscores the pivotal role core developers play in the project's overall productivity. They are not merely prolific contributors; their presence significantly influences the output of other developers.

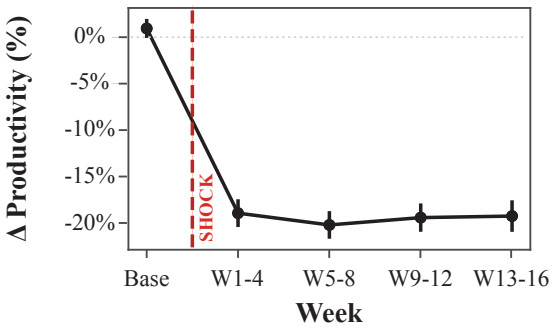

Figure 1: Estimated DiD effect of losing the core developer on other developers' productivity measured via Halstead's Effort. Effects are shown for the five four-week chunks (one for the pre-shock and four for the post-shock period). Error bars represent the 99% C.I.

We augment the original DiD equation eq. (1) to consider multiple follow-up periods. In this expanded model, the dependent variable Y_{it} represents the productivity of developer i at time t, while the categorical variable Period_t indicates one of five four-week periods, comprising one pre-shock and four post-shock periods.

Figure 1 presents the results of this time-stratified DiD analysis. It shows the difference in productivity between treated and control developers relative to pre-shock differences. Importantly, the absence of a statistically significant pre-shock difference indicates the effectiveness of our two-stage matching procedure. Figure 1 reveals that the productivity drop observed in the first four weeks after the shock remains for at least sixteen weeks. This persistence suggests that OSS projects do not recover quickly from the loss of a core developer.

5.2 RQ2: Productivity Tradeoff

The previous result prompted us to examine the issue of project resilience, i.e. the ability of a project to withstand shocks and recover from them [19]. In our case resilience is proxied by the productivity of the remaining developers. I.e. a project is resilient if the productivity can reach its pre-shock level quickly. Our goal is to determine whether projects with higher pre-shock productivity are better equipped to cope with the dropout of a core developer.

To address this question, we first categorize each project into one of five productivity levels (ranging from P1 to P5) based on the quintiles of their pre-shock productivity. We then assign each developer to the productivity level of the project they contributed to. In our regression analysis, stratified by project productivity levels, we find that the magnitude of the productivity drop depends

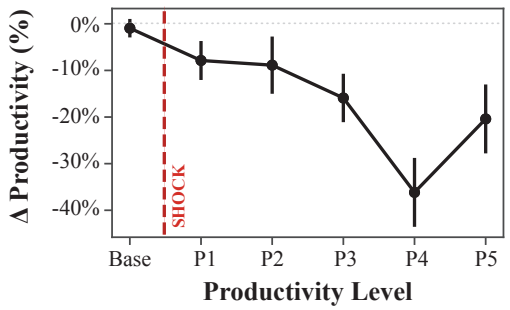

Figure 2: Productivity change relative to projects' pre-shock productivity levels: The y-axis illustrates the drop in developers' productivity across distinct pre-shock productivity levels (P1 to P5). Projects in the 60th to 80th percentile (P4) witness the steepest decline. Error bars denote the 99% C.I.

on a project's overall pre-shock productivity. Interestingly, there exists an intermediate range of pre-shock productivity where the decline is most pronounced, suggesting lower resilience in these cases. As shown in Figure 2, projects falling within the 60th to 80th percentile of productivity (P4) exhibit the largest drop (36.1% in the 99th Confidence Interval). We obtain a similar trend for the other two measures. To conclude, we find a negative relation between a project's base productivity and its developers' ability to withstand shocks, such as the dropout of core developers. We refer to this as the *productivity-resilience* tradeoff.

6 DISCUSSION

Our findings underscore an important dynamic in shocked OSS projects: While the dropout of a core developer leads to a direct loss in contributions, it hampers a project's productivity also indirectly in a two-fold manner. Firstly, we see a decrease of overall productivity after the shock, which lasts for at least 16 weeks. Secondly, we observe that this decrease predominantly affects the most active and most collaborative developers. Such a situation is worrying because it may increase the risk of these highly collaborative developers dropping out. This can lead to dropout cascades and eventually to the collapse of the project. Notably, projects with higher productivity appear to be more susceptible to these compounding effects, which we denoted as the productivity-resilience tradeoff.

To improve the resilience and long-term stability of OSS projects, the project management needs to understand this tradeoff. This means the focus should not be on short-term productivity goals, but rather on retaining essential knowledge in the team. This knowledge is mostly concentrated in active and productive developers, which however are the most affected ones in case of a shock.

Limitations and Future Work. When filtering our data, we have applied a rather strict procedure to identify projects where core developers dropped out. In case of misidentification, we may have incorrectly assigned developers to the control group instead of the exposed group. Assuming that misassigned developers do not suddenly become more productive than developers correctly identified as 'shocked', our findings represent a conservative estimate of the true effect. This is a classical limitation of causal studies [14]. Additionally, drawing from observational data has its challenges, primarily due to potential confounding effects. To counteract this, we have implemented a two-stage matching approach at the project and the developer level and integrated fixed effects into our models. Finally, while our study is focused on the analysis of productivity, future works should investigate the relation between productivity drops or other disruptions resulting in the *failure* of a project. Also, inspired by recent analysis of social media discussions [4, 16] future work might consider to analyze discussions exploiting recent success in natural language processing [13, 20].

REFERENCES

[1] 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.
[2] Adrian Bridgwater. 2023. The future for Open Source. https://www.forbes.com/sites/adrianbridgwater/2023/02/06/the-future-for-open-source/
[3] Jailton Coelho and Marco Tulio Valente. 2017. Why modern open source projects fail. In *Proceedings of the 2017 11th Joint meeting on foundations of software engineering*. 186–196.
[4] Francesco Corso, Giuseppe Russo, and Francesco Pierri. 2024. A Longitudinal Study of Italian and French Reddit Conversations Around the Russian Invasion of Ukraine. *arXiv preprint arXiv:2402.04999* (2024).
[5] Kevin Crowston and James Howison. 2003. The social structure of open source software development teams. (2003).
[6] Georgios Gousios and Diomidis Spinellis. 2012. GHTorrent: GitHub's data from a firehose. In *2012 9th IEEE MSR*. IEEE.
[7] James D Herbsleb. 2007. Global software engineering: The future of socio-technical coordination. In *FOSE'07*. IEEE.
[8] 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*. 92–101.
[9] Hila Lifshitz-Assaf Lifshitz-Assaf, Frank Nagle, and Frank Nagle. 2021. The Digital Economy Runs on Open Source. Here's How to Protect It. https://hbr.org/
[10] Mehvish Rashid, Paul M Clarke, and Rory V O'Connor. 2017. Exploring knowledge loss in open source software (OSS) projects. In *SPICE 2017*.
[11] Gregorio Robles, Jesus M Gonzalez-Barahona, and Israel Herraiz. 2009. Evolution of the core team of developers in libre software projects. In *2009 6th IEEE international working conference on mining software repositories*. IEEE, 167–170.
[12] Giuseppe Russo, Christoph Gote, Laurence Brandenberger, Sophia Schlosser, and Frank Schweitzer. 2023. Helping a friend or supporting a cause? disentangling active and passive cosponsorship in the US congress. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics, Volume 1: Long Papers*. Association for Computational Linguistics, 2952–2969.
[13] Giuseppe Russo, Nora Hollenstein, Claudiu Cristian Musat, and Ce Zhang. 2020. Control, Generate, Augment: A Scalable Framework for Multi-Attribute Text Generation. In *Findings of the Association for Computational Linguistics: EMNLP 2020*. 351–366.
[14] Giuseppe Russo, Manoel Horta Ribeiro, Giona Casiraghi, and Luca Verginer. 2023. Understanding online migration decisions following the banning of radical communities. In *Proceedings of the 15th ACM Web Science Conference 2023*. 251–259.
[15] Giuseppe Russo, Manoel Horta Ribeiro, and Robert West. 2023. Stranger Danger! Cross-Community Interactions with Fringe Users Increase the Growth of Fringe Communities on Reddit. *arXiv preprint arXiv:2310.12186* (2023).
[16] Giuseppe Russo, Niklas Stoehr, and Manoel Horta Ribeiro. 2023. Acti at evalita 2023: Overview of the conspiracy theory identification task. *arXiv preprint arXiv:2307.06954* (2023).
[17] Giuseppe Russo, Luca Verginer, Manoel Horta Ribeiro, and Giona Casiraghi. 2023. Spillover of antisocial behavior from fringe platforms: The unintended consequences of community banning. In *Proceedings of the International AAAI Conference on Web and Social Media*, Vol. 17. 742–753.
[18] Ingo Scholtes, Pavlin Mavrodiev, and Frank Schweitzer. 2016. From Aristotle to Ringelmann: a large-scale analysis of team productivity and coordination in Open Source Software projects. *Empirical Software Engineering* (2016).
[19] Steven M et al. Southwick. 2014. Resilience definitions, theory, and challenges. *European journal of psychotraumatology* (2014).
[20] Niklas Stoehr, Pengxiang Cheng, Jing Wang, Daniel Preotiuc-Pietro, and Rajarshi Bhowmik. 2023. Unsupervised contrast-consistent ranking with language models. *arXiv preprint arXiv:2309.06991* (2023).