Early Forecasting of Developer Inactivity in Open Source Projects

Sam Utzinger
swu3@nau.edu
Northern Arizona University
Flagstaff, AZ, USA

ABSTRACT

Open Source Software (OSS) projects rely on the sustained participation of core developers, making them vulnerable when contributors take extended breaks or disengage. Prior work has characterized inactivity and developer breaks, but maintainers still lack practical tools to anticipate them. We model developer disengagement as a temporal prediction problem against GitHub user contributions. Preliminary results on OSS repositories show that our approach correctly flags 65–75% of 7-day windows preceding inactivity transitions, suggesting that GitHub data can support early-warning systems for OSS developers' breaks. This research contributes a foundation for automated early-warning systems that help maintainers anticipate and mitigate project risks, representing a step toward a broader understanding of developer sustainability in OSS communities.

ACM Reference Format:
Sam Utzinger. 2026. Early Forecasting of Developer Inactivity in Open Source Projects. In *Proceedings of ACM/IEEE International Conference on Software Engineering — Student Research Competition (ICSE 2026 SRC).* ACM, New York, NY, USA, 3 pages. https://doi.org/XX.XXXX/XXXXXXX.XXXXXXX

1 INTRODUCTION

Open Source Software (OSS) enables anyone to inspect, modify, and enhance code, creating communities that rely on transparency, distributed collaboration, and collective innovation [9, 10]. These communities operate in decentralized, generally loosely connected contributions, limited co-worker awareness, and a decentralized peer-production community [16]. In these projects, it is usual that a small group of core contributors disproportionately concentrates expertise and decision-making authority. Avelino et al. [3] show that, in 65% of 133 popular GitHub projects, the truck factor (minimum number of developers whose loss would impair the project) is at most two. Prior work further shows that all core developers take breaks, and nearly half disengage for a year or more [5, 14]. When these breaks occur unexpectedly, projects face knowledge silos, stalled issues, slow pull-request processing, delayed releases, and in some cases abandonment [3, 6, 16]. For infrastructure-critical ecosystems—such as the Linux kernel, OpenSSL, and the npm and PyPI package managers—the impact of such disruptions can propagate broadly across the software supply chain [1, 13, 18].

Despite extensive research on truck factor, developer knowledge concentration, socio-technical networks, and inactivity patterns [2, 3, 5, 7, 12, 14, 17], maintainers still lack operational tools that help them anticipate harmful breaks before they happen. Evidence shows that when core developers step away, 16% of projects are abandoned, and only some recover through the emergence of new core members [3, 6]. Existing studies focus on characterizing why or how disengagement happens, but they do not translate the findings into early-warning mechanisms. As a result, maintenance remains reactive, detecting inactivity only after its consequences.

Our work addresses this gap by translating these concepts into a practical, end-to-end early-warning pipeline. We hypothesize that daily engagement signals (e.g., commits, pull requests, issues) contain predictive signals of developer disengagement. Building on prior research on break labeling, socio-technical metrics, project health indicators, and developer knowledge concentration [2, 3, 5, 7, 12, 17], we built a machine-learning pipeline that (1) labels developer activity states, (2) extracts temporal activity, (3) trains predictive models to estimate the risk of inactivity, and (4) presents the results in a proactive dashboard for maintainer interventions. Guided by this gap, we investigate the following question:

RQ. How can developer activity traces be algorithmically modeled into early-warning signals of impending inactivity?

2 METHOD

We model developer disengagement as a temporal prediction problem. Given a sequence of developer activity events $A = \{a_1, a_2, \ldots, a_n\}$ over time T, our goal is to predict whether a developer will be inactive within the next k days. In this study, we construct a new time-series dataset that integrates behavioral, social, and contribution-based indicators derived from the GitHub repositories. We summarize the four stages of our method in the following.

Stage 1: Data Collection We collect fine-grained activity traces from GitHub to characterize real developer behavior. We seeded our study with a small set of repositories used in prior work on developer inactivity [5] and for which we can obtain contextual maintainer feedback. This convenience sample enabled rapid piloting of the extraction pipeline.

We gathered granular contribution data beyond basic counts. In this study, we focused on mapping contributors' rhythm from GitHub, grouped into three main activities: two coding-related (commits and pull requests) and one non-coding (issues). We collected additional contextual signals, including pull-request comments and reviews, per-file commit metadata (lines added and deleted), and issue timeline events (labeling, assignments). This yields a detailed chronological trace of each contributor's involvement in the project.

Stage 2: Data Enhancement Using the information collected in *Stage 1*, we built a per-developer daily sequence of activity events and socio-technical data, tied to their activity state. Each daily developer record is produced through three steps:

(i) Daily state and response creation. We adopted the activity-state model from Calefato et al. [5], labeling each day as *Active*, *Non-Coding*, *Inactive*, or *Gone*. To avoid look-ahead bias, we modify the original labeling so that each state is derived solely from past information. The response variable is created by shifting the

ICSE 2026 SRC, April 2026, Rio de Janeiro, Brazil
2026. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/XX.XXXX/XXXXXXX.XXXXXXX

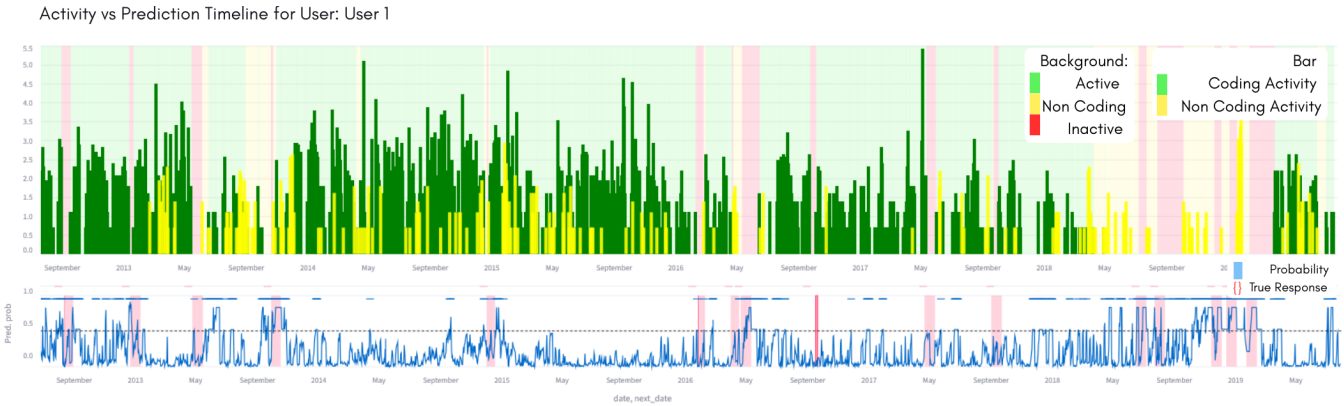

Figure 1: Top activity timeline coding vs. non-coding with background as state bands. The bottom shows ground-truth look-ahead windows and model prediction probabilities. User was selected for their contribution variability.

inactivity label forward by N days, marking windows in which a transition to inactivity begins.

(ii) Behavioral aggregation. We aggregate a developer's raw GitHub events into daily behavioral features, producing a unified, chronological trace for each user.
(iii) Socio-technical enrichment. We contextualize individual behavior within the broader project ecosystem by computing OSS sustainability metrics validated in prior work [2, 5, 7, 17]. A subset of these metrics is chosen based off a bias-variance tradeoff.

Stage 3: Modeling We frame the modeling task as a horizon-aware binary classification problem. For each developer-day pair $(user, day)$, the model predicts whether the developer will become inactive within the next k days.

We trained a supervised learning model implementing a transition function $F(X_t) \rightarrow y_{t+k}$ where X_t represents recent core developer activity and y_{t+k} indicates future inactivity transition. We employed a random forest classifier with lagged features which provides predictive performance. We further aim to design a time series approach that supports early forecasting of disengagement and parameter interpretation.

Stage 4: Evaluation We evaluate robustness, correctness, biases, usefulness, and generalizability using established practices in temporal modeling and empirical software engineering [4, 8, 11]. The evaluation comprised three components:
(i) Data validation. We verified that the modified labeled data created in *Stage 2* reflects real developer activity and contains no temporal leakage. This includes checking predictor timestamps, detecting look-ahead effects, and comparing our labeling outcomes with the original algorithm from Calefato et al. [5].
(ii) Temporal and cross-entity generalization. To evaluate generalizability beyond individual contributors or projects, we will employ multiple cross-validation strategies and nonstandard accuracy metrics tailored for our model assumptions. To examine independence and how unseen developers or repositories affect our model we implement three data splits: time-blocked cross-validation, Leave-One-User-Out CV, and Leave-One-Project-Out CV.l [4, 11, 15]. A hyperparameter tuning stage will assess model complexity, feature subsets, and period length to balance flexibility with interpretability.
(iii) Holdout-period evaluation. To ensure robustness and practical utility, we will evaluate the model on newly collected developer activity data by assessing performance across different developers.

3 CURRENT RESULTS AND DISCUSSION

Our preliminary results show that developer inactivity can be anticipated using activity traces extracted from GitHub. They show a 65%–75% accuracy in correctly flagging 7-day windows before a developer transition to inactivity. This model uses other developers inside the same repository as training data, adopting leave-one-out CV. Figure 1 shows a representative developer's activity alongside with our labels and predicted probability contrasted with the windows used as response. The visualization shows both signal and noise: many transitions appear as very narrow "slivers" of INACTIVE days, and some low-activity stretches remain unlabeled. Probability spikes cluster around several of these windows, supporting our claim that temporal patterns in GitHub activity contain useful information about forthcoming disengagement.

For the "Rdatatable/data.table" repository, using other core developers as training data, our model correctly identified 72% of the 7-day windows preceding an inactivity transition in the repo. In total, there were 147 such windows; the model correctly identified 106 of them by generating at least one correct alert within each window. Using a 60% probability threshold, the model produced 4,868 total alerts overall, of which 4,715 occurred outside inactivity windows and 153 occurred inside the windows.

The ability to forecast when a developer will transition to inactivity has substantial potential. Our results show that GitHub user data is effective as a foundation for proactive project monitoring and supports prior claims [5] that the temporal patterns in this data can be modeled to anticipate contributor disengagement. With proactive project monitoring, the ability to flag early warnings when a developer is disengaging becomes feasible, enabling managers to adopt preventive strategies to deal with developer breaks. This establishes that disengagement can be detected before it occurs using reproducible features already present in repository data.

REFERENCES

[1] Adam Alami, Raúl Pardo, and Johan Linåker. 2024. Free open source communities sustainability: Does it make a difference in software quality? *Empirical Software Engineering* 29, 5 (2024), 114.

[2] Pedro Arantes, Felipe Soupinski, and Awdren Fontão. 2023. Social networks during software ecosystems' death. In *2023 IEEE/ACM 11th International Workshop on Software Engineering for Systems-of-Systems and Software Ecosystems (SESoS)*. IEEE, 9–12.

[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] Christoph Bergmeir and José M Benítez. 2012. On the use of cross-validation for time series predictor evaluation. *Information Sciences* 191 (2012), 192–213.

[5] 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.

[6] 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.

[7] Otávio Cury and Guilherme Avelino. 2024. Knowledge Islands: Visualizing Developers Knowledge Concentration. *arXiv preprint arXiv:2408.08733* (2024).

[8] Davide Dell'Anna, Fatma Basak Aydemir, and Fabiano Dalpiaz. 2023. Evaluating classifiers in SE research: the ECSER pipeline and two replication studies. *Empirical Software Engineering* 28 (2023), 3.

[9] The Linux Foundation. 2017. What is Open Source Software? https://www.linuxfoundation.org/blog/blog/what-is-open-source-software. Accessed: 2025-11-06.

[10] GitHub. 2024. What is Open Source Software (OSS)? https://github.com/resources/articles/what-is-open-source-software. Accessed: 2025-11-06.

[11] Trevor Hastie, Robert Tibshirani, Jerome Friedman, et al. 2009. The elements of statistical learning.

[12] Steffen Herbold, Aynur Amirfallah, Fabian Trautsch, and Jens Grabowski. 2019. A systematic mapping study of developer social network research. *arXiv preprint arXiv:1902.07499* (2019).

[13] Manuel Hoffmann, Frank Nagle, and Yanuo Zhou. 2024. The value of open source software. *Harvard Business School Strategy Unit Working Paper* 24-038 (2024).

[14] Giuseppe Iaffaldano, Igor Steinmacher, Fabio Calefato, Marco Gerosa, and Filippo Lanubile. 2019. Why do developers take breaks from contributing to OSS projects? A preliminary analysis. *arXiv preprint arXiv:1903.09528* (2019).

[15] Ron Kohavi et al. 1995. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai*, Vol. 14. Montreal, Canada, 1137–1145.

[16] 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 *IFIP International Conference on Open Source Systems*. Springer, 116–129.

[17] CHAOSS Project. 2025. Community Health Analytics in Open Source Software (CHAOSS). https://chaoss.community. Accessed: 2025-11-06.

[18] Jiayi Sun. 2024. Sustaining scientific open-source software ecosystems: challenges, practices, and opportunities. In *Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings*. 234–236.