Developer Lifecycles and Language Ecosystem Dynamics on GitHub: A Large-Scale Analysis of a Follow Network

Flyxion

March 29, 2026

Abstract

We present a large-scale empirical analysis of developer activity on GitHub using a dataset constructed from a follow network exceeding one million accounts. Unlike prior work focused on repositories, this study adopts a developer-centered perspective, tracking activity lifecycles at the level of individual users. Using metadata collected via the GitHub GraphQL API in a multi-day batched crawl, we classify accounts as active or inactive based on recent commit activity and associate each developer with a primary programming language inferred from their most recently updated repository. The present dataset covers approximately 370,000 accounts—roughly 37% of the full follow graph—with collection ongoing under API rate constraints. Analysis of this sample reveals a characteristic developer activity half-life of approximately ten years, strong variation in activity across programming language ecosystems, and evidence of generational turnover in software development communities. Cross-scale comparisons across pilot, intermediate, and full dataset batches confirm that the relative stratification of language ecosystems is stable, while the global activity rate converges downward as coverage extends further from the high-visibility core of the network. These findings provide a quantitative framework for understanding the temporal structure of developer participation in open source systems.

1 Introduction

The growth of open source software has transformed software development into a large-scale, globally distributed process. GitHub, as one of the central platforms for this activity, provides an unprecedented observational window into developer behavior, collaboration patterns, and the evolution of programming language ecosystems.

Most empirical studies of GitHub rely on repository-centric datasets such as GitHub Archive or BigQuery mirrors. While these datasets capture detailed event histories, they primarily describe *artifacts* rather than *actors*. As a result, they provide limited insight into the lifecycle of individual developers.

This paper adopts a complementary perspective by analyzing GitHub as a population of developers. Rather than tracking repositories, we track users: their entry into the system, their persistence over time, and their association with programming language ecosystems. The central questions we address concern what fraction of developers remain active over time, whether there is a characteristic lifetime of developer activity, how activity varies across programming language ecosystems, and what structural patterns emerge at scale. To answer these questions, we construct a large dataset derived from a GitHub follow network and analyze activity using survival-style methods and ecosystem-level aggregation.

2 Related Work

Empirical studies of GitHub have traditionally focused on repository dynamics, collaboration networks, and event streams [Kalliamvakou et al., 2014, Gousios, 2014]. These studies have provided valuable insights into pull request workflows, project growth, and contribution patterns.

More recent work has explored developer behavior, including onboarding, retention, and collaboration structures [Zhou and Mockus, 2016, Joblin et al., 2017]. However, most analyses rely on indirect measures of activity derived from repository events rather than explicit modeling of developer lifecycles. Survival analysis has been applied in related contexts, such as developer retention in open source communities and user churn in online platforms [Vasilescu et al., 2015], suggesting that participation in software ecosystems exhibits measurable temporal structure.

The present work differs from prior studies in two key respects. First, it uses a developer-centered dataset derived from a follow network rather than repository events. Second, it explicitly models activity as a function of account age, enabling estimation of developer lifetimes and ecosystem-specific retention.

3 Dataset Construction

3.1 Sampling Strategy

The dataset is constructed by crawling the follow graph of a GitHub user with an exceptionally large number of followed accounts, on the order of one million. This produces a population of developers that is not uniformly random but is instead

biased toward socially visible or interconnected accounts. While this introduces sampling bias, it also yields a population enriched for active and engaged developers, making it particularly suitable for studying sustained participation.

3.2 Data Collection

Data is collected using the GitHub GraphQL API in batched queries. For each account, we retrieve the account creation timestamp t_{create}, the number of followers, the number of public repositories, the most recently pushed repository, the timestamp of the most recent push t_{push}, and the primary language of that repository. Accounts that cannot be resolved as users—such as organizations or deleted accounts—are excluded from analysis.

3.3 Collection Infrastructure and Operational Constraints

The crawl is implemented in Bash using the `gh` CLI with GraphQL batching, processing twenty accounts per API request to balance throughput against per-query cost. Follow-graph pagination is handled via cursor-based enumeration, with each page retrieving one hundred followed accounts. Batch results are written to per-batch JSON cache files, enabling the process to be interrupted and resumed without data loss; already-completed batches are skipped on restart.

Rate limit management is handled automatically: each response includes the remaining point budget and the reset timestamp. When the remaining budget falls below a configurable threshold (default 100 points), the script sleeps until the reset window passes plus a small padding interval. Transient API failures trigger exponential-backoff retries up to a configurable maximum, after which the script exits with a non-zero status.

In practice, the full crawl of approximately 370,000 accounts required sustained operation over several days, with multiple interruptions due to network instability, API quota exhaustion, and process crashes. The crash-recovery design—batch-level caching of raw JSON and append-only accumulation in the CSV and text output files—ensured that completed work was preserved across restarts. The total collection effort amounted to several thousand API requests distributed across multiple days of wall-clock time.

3.4 Incomplete Coverage and Dataset Status

An important operational caveat applies to all results in this paper. The source follow graph contains approximately one million accounts. The present analysis covers 370,447 of these, representing roughly 37% of the full population. The remaining accounts have not yet been queried due to API rate constraints; completing the full crawl at the current rate would require approximately ten additional days of continuous operation.

The reported statistics should therefore be interpreted as results from a large but incomplete sample of the follow graph, rather than a census. The incompleteness is not random: accounts are crawled in follow-list order, and later portions of the

follow list may differ in their demographic and ecosystem composition. Whether this ordering introduces systematic bias relative to a uniformly random incomplete sample is not known.

Nonetheless, the sample of 370,000 accounts is large enough to yield stable aggregate estimates, and cross-scale comparisons across intermediate batches suggest that the key qualitative findings are robust. Section 4 presents this validation in detail.

3.5 Activity Classification

We define a binary activity indicator:

$$A = \begin{cases} 1 & \text{if } t_{\text{push}} \geq t_{\text{now}} - \Delta \\ 0 & \text{otherwise,} \end{cases}$$

where $\Delta = 365$ days. Thus, a developer is classified as **active** if they have pushed to a public repository within the past year, and **inactive** otherwise.

3.6 Language Assignment

Each developer is assigned a primary programming language based on the most recently updated repository. While this is a coarse proxy, it captures the developer's current ecosystem affiliation. Accounts with no identifiable primary language are assigned to a special category denoted `None`.

3.7 Dataset Size

At the time of analysis, the dataset contains 370,447 total accounts, of which 215,442 are classified as active and 155,005 as inactive. This scale is sufficient to produce stable estimates of aggregate behavior and ecosystem-level structure.

4 Cross-Scale Validation

Because the dataset was assembled incrementally over multiple days, intermediate snapshots are available at substantially smaller scales. Comparing results across these scales provides an informal consistency check on the stability of the findings.

Three scale points are available: a pilot batch of approximately 960 accounts collected during initial testing, an intermediate batch of approximately 30,000 accounts representing the first several hours of the full crawl, and the present dataset of 370,447 accounts.

Table 1 summarizes key aggregate statistics across these three batches.

Several patterns are immediately apparent. The global active fraction declines monotonically from the pilot to the full dataset, moving from 0.800 to 0.597 to 0.582. This is consistent with the sampling bias interpretation: the earliest accounts crawled are concentrated near the most-followed, most-active users, producing an inflated activity rate. As the crawl extends further into the follow graph, the population

Statistic	Pilot ($n \approx 960$)	Intermediate ($n \approx 30{,}000$)	Full ($n \approx 370{,}000$)
Active fraction	0.800	0.597	0.582
Approx. half-life	n/a	10 years	10 years
TypeScript ratio	0.901	0.846	0.820
Python ratio	0.892	0.728	0.692
Rust ratio	0.940	0.863	0.831
Ruby ratio	0.641	0.481	0.430
Java ratio	n/a	0.490	0.502
None ratio	0.586	0.371	0.356

Table 1: Key statistics compared across three dataset scales. "n/a" indicates quantities not estimable due to insufficient sample size at that batch level.

diversifies toward less active and less connected developers, and the global activity rate converges toward a lower value.

Individual language activity ratios show a similar convergence pattern. TypeScript moves from 0.901 in the pilot to 0.820 in the full dataset; Python from 0.892 to 0.692; Rust from 0.940 to 0.831. The ordering across languages, however, is remarkably stable. In every batch, Rust and TypeScript rank among the highest activity languages; Ruby and the `None` category rank among the lowest. The relative stratification that is the primary empirical finding of this paper appears to stabilize well before the full dataset scale.

The survival curve analysis was not tractable at the pilot scale due to sparse coverage of account age bins, but the intermediate batch already produces a clear half-life estimate of ten years, confirmed exactly by the full dataset. The survival curve shape—including the gradual decline over the first decade and the late-age upturn attributable to survivorship bias—is qualitatively identical across the two larger batches. The intermediate batch shows $S(10) = 0.500$ and $S(18) = 0.500$, while the full dataset shows $S(10) = 0.535$ and $S(18) = 0.713$, reflecting the dilution of the high-persistence tail as the population expands.

The activity ratio of the `None` category declines monotonically across scales, from 0.586 to 0.371 to 0.356, reinforcing the interpretation that peripheral or inactive users are less likely to have maintained public repository engagement and that this population grows disproportionately as the crawl moves further from the high-visibility core of the network.

These cross-scale comparisons provide informal evidence that the main findings are not artefacts of the specific accounts processed. They also characterize the direction of residual bias: the full dataset likely still overestimates true activity rates and language-specific ratios relative to what would be observed with complete coverage of the one-million-account follow graph, and more substantially relative to GitHub's full user population.

5 Methodology

5.1 Activity Ratio

For each programming language L, we define the activity ratio:

$$R(L) = \frac{N_{\text{active}}(L)}{N_{\text{active}}(L) + N_{\text{inactive}}(L)}.$$

This metric estimates the probability that a developer associated with language L is currently active.

5.2 Account Age and Survival Function

We define account age as $\tau = t_{\text{now}} - t_{\text{create}}$. Accounts are grouped into integer year bins, and the fraction of active accounts is computed for each bin. We approximate a survival function:

$$S(\tau) = P(\text{active} \mid \text{age} = \tau),$$

which represents the probability that a developer remains active at age τ. The point at which $S(\tau) \approx 0.5$ provides an estimate of the *activity half-life*.

5.3 Observed Lifetime

For each account with valid timestamps, we define an observed lifetime:

$$\ell = t_{\text{push}} - t_{\text{create}}.$$

This is a lower-bound estimate of the true lifetime, as currently active users are right-censored. Median lifetimes are computed for each language ecosystem.

6 Results

6.1 Global Activity Rate

Across the dataset of 370,447 accounts, approximately 58.2% are classified as active under the one-year activity threshold. This is notably higher than estimates derived from repository-centric datasets, reflecting the sampling bias inherent in follow networks toward socially connected or visible developers.

6.2 Language Ecosystem Activity

Table 2 presents activity ratios for the most common programming languages in the dataset. The results exhibit a clear stratification of ecosystems. Modern languages such as Rust and TypeScript show the highest activity ratios, indicating strong ongoing engagement and relatively few legacy inactive accounts. In contrast, older ecosystems such as Ruby and Java display substantially lower activity ratios, reflecting the accumulation of inactive users over time.

Language	Active	Inactive	Activity Ratio
Rust	4,805	976	0.831
TypeScript	27,307	5,974	0.820
Lua	1,555	390	0.799
Dart	1,372	556	0.712
Go	5,676	2,365	0.706
Shell	5,352	2,323	0.697
Python	40,229	17,920	0.692
Kotlin	1,831	848	0.683
HTML	16,455	8,400	0.662
Swift	1,477	809	0.646
JavaScript	20,087	13,404	0.600
C++	6,821	4,736	0.590
C#	3,612	2,793	0.564
PHP	2,749	2,303	0.544
Java	6,297	6,246	0.502
Ruby	1,582	2,098	0.430

Table 2: Activity ratios for major programming languages, ordered by activity ratio.

6.3 The Null Language Population

A significant fraction of accounts are associated with no identifiable primary language, with $N_{\text{None}} = 95{,}447$ and $R_{\text{None}} = 0.356$. This group dominates the inactive population, accounting for nearly 40% of inactive accounts. It likely corresponds to users with no public repositories, archived projects, or non-code usage of the platform. The markedly lower activity ratio of this group suggests that repository presence is a strong predictor of continued participation.

6.4 Distribution of Languages Among Active and Inactive Accounts

The distribution of languages differs significantly between active and inactive populations. Among active accounts, the leading languages are Python at 18.67%, followed by the `None` category at 15.75%, TypeScript at 12.67%, JavaScript at 9.32%, and HTML at 7.64%. Among inactive accounts, the distribution is dominated by `None` at 39.68%, with Python at 11.56%, JavaScript at 8.65%, HTML at 5.42%, and Java at 4.03%. The contrast highlights the central role of the `None` category in accounting for inactivity and suggests that language ecosystems with active repository maintenance strongly correlate with continued engagement. Figure 1 illustrates this contrast directly.

6.5 Developer Survival Curve

Figure 2 presents the estimated survival function of developer activity as a function of account age, and Table 3 provides the underlying numerical values. The survival

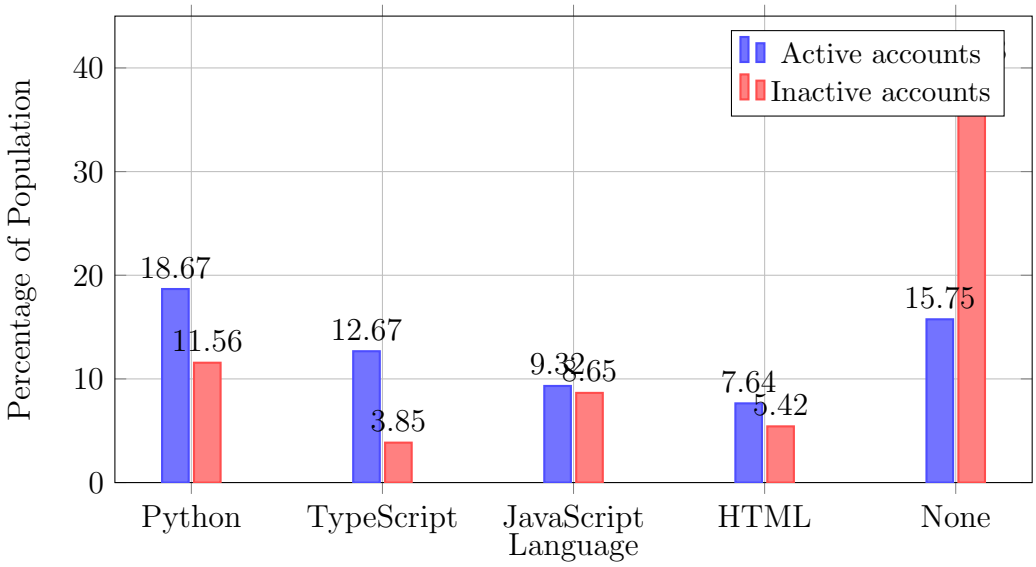

Figure 1: Language distribution among active and inactive accounts. The `None` category is disproportionately concentrated among inactive accounts, while TypeScript shows a strong skew toward active participants.

curve exhibits a gradual decline over the first decade, followed by stabilization and an eventual increase for older accounts. The point at which $S(\tau) \approx 0.5$ occurs near $\tau \approx 10$ years, suggesting a characteristic developer activity half-life of approximately ten years. The increase in activity fraction for older accounts is attributable to survivorship bias: early adopters who remain active are disproportionately persistent developers.

6.6 Language Activity Ratios

Figure 3 displays the activity ratios for the twenty-five largest language populations in the dataset. The ordering reveals a clear temporal gradient: recently established ecosystems cluster toward the top, while mature or legacy ecosystems occupy the lower range.

6.7 Median Developer Lifetimes

Median observed lifetimes by language reveal additional structure. Languages associated with smaller or more specialized communities tend to exhibit longer lifetimes. Emacs Lisp developers show a median lifetime of 12.20 years, followed by Elixir at 11.95, Clojure at 11.70, and Common Lisp at 10.73. Vim Script developers exhibit a median of 10.40 years, while Go and Rust developers show lifetimes of approximately 8.35 and 8.30 years respectively. This difference is interpretable not as reduced retention among Go and Rust developers, but rather as an artefact of limited historical depth in younger ecosystems whose developers have simply had less elapsed time to accumulate long observed lifetimes.

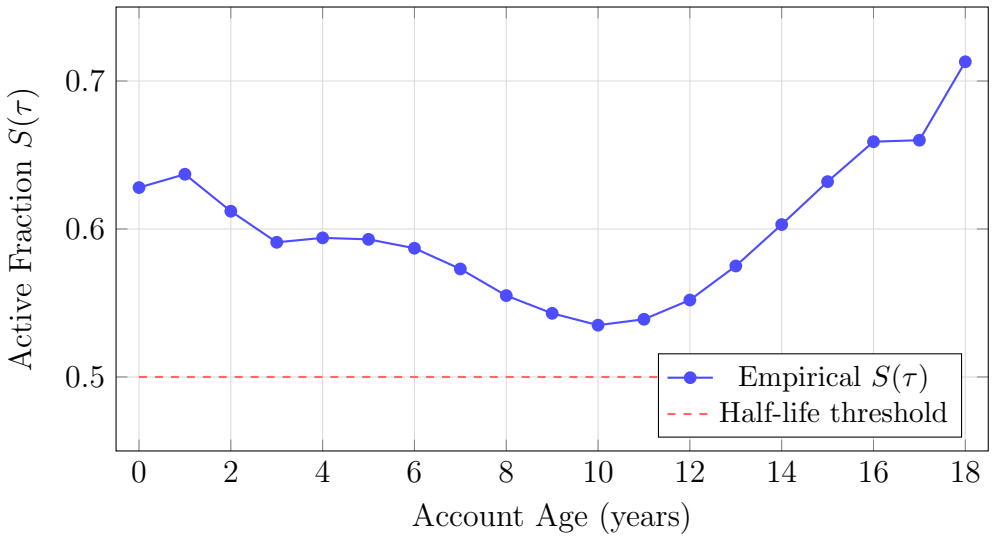

Figure 2: Empirical survival function $S(\tau)$: fraction of accounts remaining active as a function of account age in years. The dashed line marks the $S(\tau) = 0.5$ threshold, intersected near $\tau \approx 10$ years. The upturn for accounts older than twelve years reflects survivorship bias among persistent early adopters.

6.8 Emerging Ecosystems

Several newer ecosystems exhibit exceptionally high activity ratios: Astro at 0.90, MDX at 0.84, Rust at 0.83, and TypeScript at 0.82. These values indicate that these ecosystems are still in a growth phase, characterized by a relatively small inactive population and a high ratio of new entrants to legacy participants.

6.9 Intermediate Batch Survival Curve

Figure 4 presents the survival curve from the intermediate 30,000-account batch alongside the full dataset curve, enabling direct comparison. The intermediate batch shows a sharper descent, with $S(10) = 0.500$ exactly—the half-life threshold—and a less pronounced late-age upturn. This difference reflects the higher proportion of high-persistence long-term developers in the full dataset, which has a richer tail of accounts from the early platform era. The qualitative structure, however, is identical: monotonic decline over the first decade, near-threshold crossing near ten years, and a survivorship-driven upturn among the oldest accounts.

7 Discussion

7.1 Developer Activity as a Survival Process

The observed survival curve suggests that developer activity on GitHub can be modeled as a stochastic process with a characteristic decay timescale. The empirical half-life of approximately ten years indicates that participation is neither short-lived nor permanent, but instead follows a structured lifecycle.

Age (years)	Active Fraction $S(\tau)$
0	0.628
1	0.637
2	0.612
3	0.591
4	0.594
5	0.593
6	0.587
7	0.573
8	0.555
9	0.543
10	0.535
11	0.539
12	0.552
13	0.575
14	0.603
15	0.632
16	0.659
17	0.660
18	0.713

Table 3: Estimated survival function $S(\tau)$ of developer activity by account age.

This behavior is consistent with a hazard-based interpretation. Let $h(\tau)$ denote the hazard rate of becoming inactive at age τ. The relatively smooth decline of $S(\tau)$ over the first decade suggests a moderately increasing hazard rate during early and mid-career stages, followed by stabilization among long-term participants. The increase in $S(\tau)$ for large τ is not indicative of renewed activity but rather reflects survivorship bias: developers who remain active after many years constitute a selected subset with lower effective hazard rates.

7.2 A Generative Model of Developer Participation

The empirical results suggest that developer activity can be modeled as a birth-death process on a population of participants. Let $N(t)$ denote the number of active developers at time t within a given ecosystem, and let $n(\tau, t)$ denote the density of developers of age τ at time t. The evolution of the active population can be approximated by

$$\frac{dN}{dt} = \lambda_{\text{entry}}(t) - \int_0^\infty \lambda_{\text{exit}}(\tau)\, n(\tau, t)\, d\tau,$$

where $\lambda_{\text{entry}}(t)$ is the rate at which new developers join and $\lambda_{\text{exit}}(\tau)$ is the age-dependent rate at which active developers become inactive.

In steady-state conditions, the activity ratio $R(L)$ can be interpreted as propor-

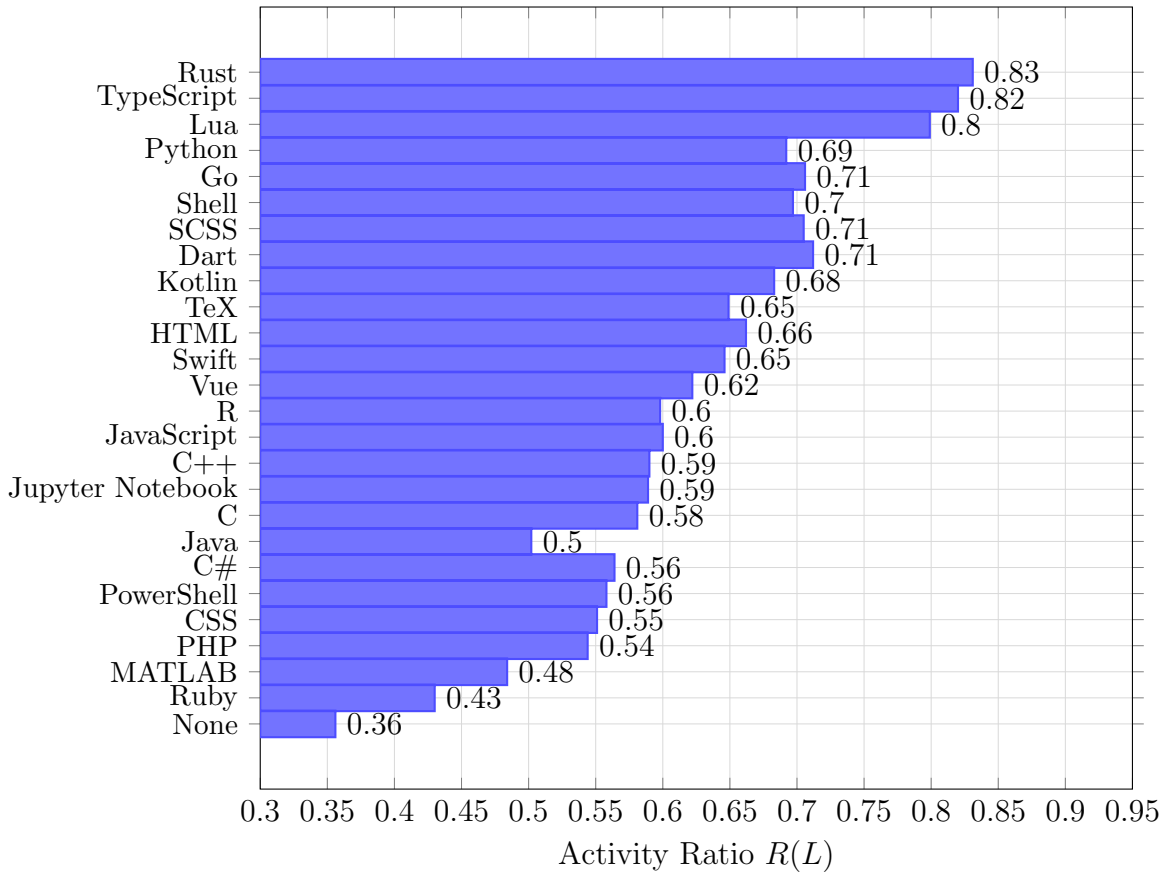

Figure 3: Activity ratios $R(L)$ for the twenty-five largest language populations in the dataset. Languages are ordered by ratio from lowest (None, Ruby) to highest (TypeScript, Rust), revealing a temporal stratification that tracks ecosystem age and growth phase.

tional to the balance between these two processes:

$$R(L) \sim \frac{\lambda_{\text{entry}}}{\lambda_{\text{entry}} + \lambda_{\text{exit}}}.$$

Rapidly growing ecosystems are characterized by $\lambda_{\text{entry}} \gg \lambda_{\text{exit}}$, while declining ecosystems satisfy the opposite inequality. This model provides a mechanistic interpretation of the observed stratification of programming languages and connects the cross-sectional activity ratio to underlying population dynamics.

7.3 Ecosystem Age and Activity Ratios

The activity ratio $R(L)$ serves as a proxy for the temporal phase of a programming language ecosystem, reflecting the balance between entry and exit processes. Under this interpretation, high $R(L)$ indicates growing ecosystems with high entry relative to exit, intermediate $R(L)$ indicates mature and stable ecosystems, and low $R(L)$ indicates declining ecosystems with accumulated inactive users. This framework explains the observed ordering: Rust and TypeScript occupy a growth phase, Python

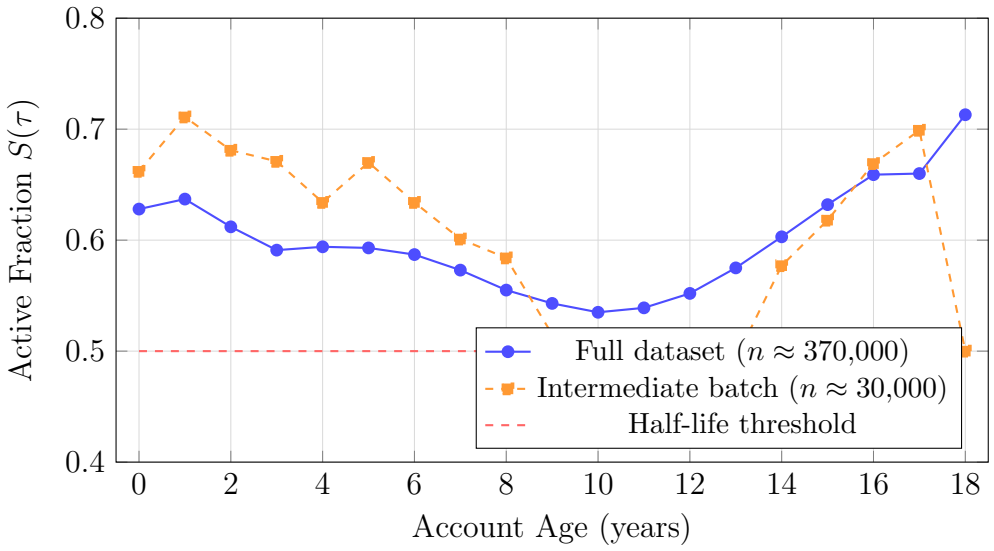

Figure 4: Survival curves for the intermediate ($n \approx 30{,}000$) and full ($n \approx 370{,}000$) datasets. Both curves cross the $S(\tau) = 0.5$ threshold near $\tau = 10$ years. The intermediate batch shows a slightly sharper decline and a less pronounced late-age upturn, consistent with its smaller representation of very long-tenured developers.

and Go a mature phase, and Java and Ruby a declining phase relative to the snapshot captured by this dataset.

7.4 Hazard Function Estimation

Let $S(\tau)$ denote the empirical survival function. The corresponding discrete hazard function is approximated by

$$h(\tau) \approx 1 - \frac{S(\tau + 1)}{S(\tau)},$$

which estimates the probability that a developer becomes inactive between age τ and $\tau + 1$, conditioned on being active at age τ. The gradual decline of $S(\tau)$ implies a relatively low but initially increasing hazard rate over the first decade.

A natural candidate for parametric modeling is the Weibull distribution:

$$S(\tau) = \exp\left(-(\lambda\tau)^k\right),$$

where $k > 1$ corresponds to increasing hazard. Taking logarithms yields

$$\log(-\log S(\tau)) = k \log \tau + k \log \lambda,$$

suggesting estimation of k and λ via linear regression on transformed data. Preliminary inspection indicates k slightly greater than 1, implying mild aging effects in developer participation. For $k = 1$ the model reduces to exponential decay with constant hazard, while $k < 1$ would indicate a decreasing hazard characteristic of strong early selection. The empirical data suggests a mixture of mildly increasing hazard and survivorship bias, resulting in the non-monotonic observed survival curve.

7.5 Generational Structure of Programming Languages

The results suggest that programming language ecosystems evolve in generational waves, each characterized by rapid adoption, sustained activity, and eventual saturation. A simplified periodization consistent with the data places Ruby, PHP, and early JavaScript as dominant ecosystems from roughly 2008 to 2013, with Python and modern JavaScript frameworks dominant from 2014 to 2018, and Rust, Go, and TypeScript as the leading ecosystems from 2019 to the present. As each generation matures, its activity ratio declines due to the accumulation of inactive participants. Newer ecosystems exhibit higher ratios because they have not yet accumulated such historical mass.

7.6 Language Migration and Ecosystem Transitions

Although this study assigns a single primary language to each developer, the observed generational structure strongly implies that developers migrate between ecosystems over time. This process can be conceptualized as a transition system on a discrete state space of languages. Let $P(L_i \to L_j)$ denote the probability that a developer transitions from language L_i to language L_j over a given time interval. While direct estimation of P requires longitudinal data, the cross-sectional patterns observed here imply dominant transition pathways such as Ruby to JavaScript or TypeScript, Java to Kotlin, Objective-C to Swift, and JavaScript to TypeScript. These transitions reflect technological shifts and platform evolution rather than purely individual preferences.

7.7 Niche Ecosystems and Retention

Languages such as Emacs Lisp, Clojure, and Common Lisp exhibit unusually long median lifetimes. Such ecosystems are characterized by smaller community size, higher specialization, and stronger identity or tooling dependence. These properties may lead to lower exit rates, resulting in longer observed developer lifetimes. This suggests a trade-off between ecosystem size and retention: large ecosystems attract many participants but also experience higher turnover, while niche ecosystems retain fewer but more persistent developers.

7.8 Entropy and Ecosystem Aging

The accumulation of inactive accounts within a language ecosystem can be interpreted as a form of entropy increase. As ecosystems mature, they accumulate historical artifacts—repositories, accounts, and code—that no longer participate in active development. Let $I(t)$ denote the number of inactive accounts in an ecosystem. In the absence of removal processes, $dI/dt \geq 0$, leading to a monotonic dilution of the active fraction and a corresponding decline in activity ratio. New ecosystems begin with low entropy and high activity ratios, which decline as the system ages. This entropic interpretation is consistent with the observed ordering of ecosystems and provides a thermodynamic analogy for the dynamics of participation.

7.9 Network Effects and Visibility Bias

The use of a follow network introduces a structural bias toward developers who are socially visible. This inflates the observed activity rate and amplifies the representation of ecosystems associated with infrastructure, tooling, and open source leadership. More formally, let w_i denote the probability that developer i is included in the sample. Then

$$w_i \propto f(F_i, A_i),$$

where F_i is follower count, A_i is activity, and f is an increasing function of both quantities. The observed estimator of the survival function accordingly overestimates the true survival probability. Correcting for this bias would require reweighting observations or incorporating independent sampling strategies.

7.10 Activity as Constraint Satisfaction

An alternative interpretation of developer activity frames continued participation as a form of ongoing constraint satisfaction. Developers remain active as long as they can maintain alignment between personal interest, available time, economic incentives, and ecosystem relevance. In this view, inactivity arises not from a single cause but from the gradual accumulation of constraint violations. This perspective is consistent with the observed gradual decay of activity rather than abrupt transitions and connects the empirical survival curve to an underlying process of resource and motivation dynamics.

7.11 Ecosystem Collapse: The Case of Objective-C

The intermediate batch data reveals an extreme case that deserves separate treatment. Objective-C exhibits an activity ratio of approximately 0.15 in the 30,000-account batch—the lowest of any language with a sample size sufficient for reliable estimation. This figure places it far below the general range of 0.35–0.83 observed for most other languages and warrants interpretation as qualitatively distinct.

The Objective-C case illustrates what may be termed *ecosystem collapse*: a condition in which a language's population is overwhelmingly composed of historical accounts whose owners have long since migrated to successor technologies. The transition from Objective-C to Swift, initiated by Apple's introduction of Swift in 2014, provides a concrete mechanism. Developers who built their careers on iOS and macOS development did not disappear; they transferred their ecosystem affiliation to the successor language, leaving their Objective-C repositories and accounts behind as inactive artifacts.

This pattern distinguishes ecosystem collapse from ordinary maturation. In a maturing ecosystem such as Ruby, inactive accounts accumulate gradually as each cohort ages. In a collapsing ecosystem, the inactive accumulation is concentrated and structurally driven by a singular platform or tooling decision external to the developer community. The resulting activity ratio is not merely low but reflects a historical rupture.

Formally, this corresponds to a sudden large increase in λ_{exit} at a specific point in time, followed by suppression of λ_{entry} as the successor ecosystem captures all new entrants. The steady-state activity ratio under these conditions converges to a value determined primarily by the fraction of the pre-collapse population that chose not to migrate—typically a small specialist tail.

7.12 Activity as Constraint Satisfaction

An alternative interpretation of developer activity frames continued participation as a form of ongoing constraint satisfaction. Developers remain active as long as multiple conditions remain jointly satisfied: personal motivation, available time, economic incentives, and technological relevance. Inactivity emerges not from a single discrete failure but from the gradual erosion of this alignment over time. When any one constraint is violated persistently—a language loses market relevance, a developer's professional context shifts, or time pressure from competing obligations intensifies—the probability of disengagement increases.

This framing is consistent with the observed survival curve. A memoryless model would produce exponential decay, in which the hazard rate is constant and does not depend on how long a developer has been active. The Weibull model with $k > 1$ captures a process in which the longer a developer has been active, the greater the cumulative probability that at least one of the sustaining constraints has eroded. The gradual nature of the decline, rather than a sharp transition, reflects the multi-dimensional character of the constraint set: partial erosion of one constraint can be compensated by strong alignment on others, producing continued activity even under unfavorable conditions.

7.13 The Role of Non-Code Accounts

The large `None` category plays a central role in the structure of inactivity. Its low activity ratio of 0.356 indicates that accounts without active repository engagement are substantially more likely to become inactive. This suggests that participation in code production—rather than mere presence on the platform—is a key determinant of long-term engagement. The disproportionate representation of this group among inactive accounts (39.68% of all inactive accounts) implies that any aggregate estimate of platform-wide activity rates depends critically on how this population is treated.

8 Limitations

8.1 Sampling Bias

The dataset is derived from a follow network and is therefore not a uniform sample of GitHub users. It is likely biased toward developers who are socially visible, connected, or of particular interest to the source account. This bias inflates the observed activity rate and may overrepresent certain ecosystems.

8.2 Activity Proxy

Activity is defined based on public repository pushes within a one-year window. This measure does not capture private repository activity, non-code contributions such as issues and reviews, or activity conducted outside the GitHub platform. Some developers classified as inactive may therefore remain active in other contexts not reflected in the data.

8.3 Language Attribution

Assigning a single language based on the most recent repository is a simplification. Many developers work across multiple languages, and the chosen proxy reflects only their most recent visible activity. Developers working primarily in private or multi-language contexts may be systematically misclassified.

8.4 Right-Censoring

Observed lifetimes are right-censored for active users. As a result, median lifetime estimates are conservative and may substantially underestimate true participation durations. A more accurate estimator would require Kaplan–Meier methods or equivalent longitudinal tracking.

9 Toward a Unified Model of Developer Ecosystems

The results suggest that developer ecosystems can be understood as dynamical systems characterized by interacting entry and exit processes, temporal survival dynamics, migration between language states, and monotonic accumulation of inactive mass. These components can be integrated into a unified model in which programming languages are viewed not merely as tools but as evolving attractors in a high-dimensional space of developer behavior. Under this model, the activity ratio of a language serves as an order parameter encoding its current position in a lifecycle from emergence through maturity to decline. Such a model would enable prediction of ecosystem growth, decline, and transition, and would provide a quantitative framework for understanding the evolution of software development at scale.

10 Conclusion

This study provides a large-scale, developer-centered analysis of activity on GitHub using a dataset of over 370,000 accounts, representing approximately 37% of a follow graph constructed from a highly connected source user. The dataset was assembled over several days of continuous operation under GitHub API rate constraints, with crash-recovery infrastructure enabling incremental progress across multiple sessions. Developer activity follows a structured lifecycle with a characteristic half-life of approximately ten years. Programming language ecosystems exhibit systematic

variation in activity ratios, reflecting their stage of evolution: newer ecosystems show high activity due to growth, while older ecosystems accumulate inactive participants over time. The extreme case of Objective-C, with an activity ratio near 0.15 in the intermediate batch, illustrates how platform-driven ecosystem transitions can produce conditions qualitatively distinct from ordinary maturation. A substantial fraction of inactive accounts are associated with no identifiable primary language, identifying repository engagement as the strongest observable predictor of long-term participation.

Cross-scale validation across pilot ($n \approx 960$), intermediate ($n \approx 30{,}000$), and full ($n \approx 370{,}000$) datasets confirms that the relative ordering of language activity ratios is stable across all scales, while absolute values converge downward as the sample diversifies away from the most-visible accounts. The results should be interpreted as representative of the socially connected stratum of the GitHub population rather than the platform as a whole.

These findings suggest that both individual participation and ecosystem dynamics can be understood within a unified temporal framework. GitHub, viewed at scale, is not a static repository of code but a dynamic system shaped by generational turnover, platform decisions, and evolving technological landscapes. Future work may extend this analysis by completing the full crawl of the one-million-account follow graph, incorporating longitudinal data, modeling transitions between ecosystems, applying Kaplan–Meier estimation to account for right-censoring, and integrating additional signals such as collaboration networks and contribution types.

Appendices

A Discrete Survival Estimation

Let $\tau \in \mathbb{N}$ denote account age in years. For each age bin τ, define

$$N(\tau) = N_{\text{active}}(\tau) + N_{\text{inactive}}(\tau), \qquad S(\tau) = \frac{N_{\text{active}}(\tau)}{N(\tau)}.$$

This defines a discrete cross-sectional estimate of the survival function $S(\tau) \approx P(\text{active} \mid \text{age} = \tau)$. Unlike classical longitudinal survival analysis, this estimator assumes that the distribution of account ages approximates a stationary population.

A.1 Discrete Hazard Function

The discrete hazard function is approximated by

$$h(\tau) \approx 1 - \frac{S(\tau+1)}{S(\tau)} = \frac{S(\tau) - S(\tau+1)}{S(\tau)},$$

which estimates the probability that a developer becomes inactive between age τ and $\tau + 1$, conditioned on being active at age τ. The gradual decline of $S(\tau)$ implies a relatively low but increasing hazard rate over the first decade.

A.2 Half-Life Estimation

The activity half-life $\tau_{1/2}$ is defined implicitly by $S(\tau_{1/2}) = 1/2$. Given discrete observations, we estimate

$$\tau_{1/2} \approx \min\{\tau : S(\tau) \leq 0.5\},$$

which in the present dataset occurs near $\tau \approx 10$ years.

A.3 Right-Censoring

For active accounts, the true lifetime ℓ^* satisfies $\ell^* \geq \ell = t_{\text{push}} - t_{\text{create}}$. Observed lifetimes are therefore right-censored, and the median lifetime computed from observed ℓ underestimates the true median. A more accurate estimator would apply the Kaplan–Meier procedure:

$$\hat{S}(\tau) = \prod_{\tau_i \leq \tau} \left(1 - \frac{d_i}{n_i}\right),$$

where d_i is the number of exits at time τ_i and n_i is the number at risk.

A.4 Activity Ratio as a Mixture Statistic

Let T denote the random variable representing developer lifetime. Then

$$R(L) \approx P(T > \Delta \mid L),$$

where $\Delta = 1$ year is the activity cutoff. Thus $R(L)$ estimates the survival probability beyond a fixed horizon, making it a special case of the survival function evaluated at a single time point.

A.5 Bias from Cross-Sectional Sampling

The estimator $S(\tau)$ is biased by the age distribution of accounts. Let $\pi(\tau)$ denote the probability of sampling an account of age τ. The observed survival function is

$$S_{\text{obs}}(\tau) = P(\text{active} \mid \tau, \text{sampled}),$$

which depends on $\pi(\tau)$. If older accounts are overrepresented, the survival curve may be biased upward at large τ. Combined with the visibility weighting described below, this produces upward bias both at large τ and in aggregate activity estimates.

A.6 Visibility-Weighted Sampling

Let w_i denote the inclusion probability of account i in the follow network, with $w_i \propto f(F_i, A_i)$ where F_i is follower count and A_i is activity level. The observed survival estimator becomes

$$S_{\text{obs}}(\tau) = \frac{\sum_i w_i \, \mathbf{1}_{\{\text{active},\, \tau_i = \tau\}}}{\sum_i w_i \, \mathbf{1}_{\{\tau_i = \tau\}}}.$$

Since w_i increases with activity, $S_{\text{obs}}(\tau)$ overestimates the true survival probability. Correcting for this bias requires either Horvitz–Thompson reweighting or an independent sampling frame.

A.7 Confidence Intervals

For each age bin, the standard error of $S(\tau)$ is approximated by

$$\text{SE}(\tau) = \sqrt{\frac{S(\tau)(1 - S(\tau))}{N(\tau)}}.$$

A 95% confidence interval is $S(\tau) \pm 1.96 \cdot \text{SE}(\tau)$. Given the large sample sizes in this dataset, these intervals are narrow, implying high statistical confidence in aggregate estimates.

A.8 Weibull Model Fit

The Weibull survival model $S(\tau) = \exp(-(\lambda\tau)^k)$ yields, upon logarithmic transformation,

$$\log(-\log S(\tau)) = k \log \tau + k \log \lambda,$$

enabling estimation of k and λ by linear regression on the log-log scale. When $k > 1$ the hazard is increasing (aging effect); when $k = 1$ the hazard is constant (memoryless exponential); when $k < 1$ the hazard is decreasing (early selection). The empirical data suggests k slightly greater than 1, indicating a mild aging effect consistent with gradual disengagement over the developer lifecycle.

A.9 Limit of Large Time

As $\tau \to \infty$, the observed survival function approaches

$$\lim_{\tau \to \infty} S(\tau) = P(\text{persistent developer}),$$

which represents the long-run fraction of the population composed of highly persistent users. The upward trend in $S(\tau)$ at large τ is therefore interpretable as convergence to this limiting value under a mixture model in which the population contains a subpopulation of developers with effectively zero exit hazard.

References

Eirini Kalliamvakou et al. The promises and perils of mining GitHub. In *MSR*, 2014.

Georgios Gousios. The GHTorrent dataset and tool suite. In *MSR*, 2014.

Minghui Zhou and Audris Mockus. Who will stay in the FLOSS community? *IEEE Transactions on Software Engineering*, 2016.

Bogdan Vasilescu et al. Gender and tenure diversity in GitHub teams. In *CHI*, 2015.

Mitchell Joblin et al. Classifying developers into core and peripheral roles. In *ICSE*, 2017.