Journal of Electrical Engineering and Computer (JEECOM)
Vol. 8, No. 1 (2026), DOI: 10.33650/jeecom.v8i1.15699
p-ISSN: 2715-0410; e-ISSN: 2715-6427

From Active to Zombie: A Literature Review on Bus Factor's Role in Open Source Project Lifecycle Decline

Beni Nurfauzi [1*], Kusrini Kusrini [2]
[1,2] Department of Informatics Engineering, AMIKOM University of Yogyakarta, Yogyakarta, Indonesia

Article Info

Article history:

Received April 04, 2026
Revised April 28, 2026
Accepted April 29, 2026

Keywords:

Bus Factor
Open Source Software
Zombie Phase
Project Abandonment
OSS Abandonment

ABSTRACT

Open Source Software (OSS) is now a critical global infrastructure. However, its sustainability is threatened by knowledge concentration among few core developers, a vulnerability known as the Bus Factor. This systematic literature review examines the role of Bus Factor dynamics in driving OSS projects toward the zombie phase, a deceptive intermediate state wherein repositories remain administratively active while genuine human-driven development has ceased. Following a PRISMA-based selection process, 14 peer-reviewed studies were identified, quality-assessed, and thematically categorized to answer four research questions concerning Bus Factor measurement, lifecycle decline, computational prediction methods, and contributor behavioral dynamics. The review traces the evolution of Bus Factor measurement across four methodological generations and constructs a formal causal framework linking knowledge concentration to project death via sleeping developer behavior and zombie phase onset. Empirical evidence confirms that 89.65% of OSS projects experience total core developer loss, with only 27.07% recovering successfully. Critically, dynamic Bus Factor fluctuation remains entirely absent from existing abandonment prediction models. This review identifies this absence as the central research gap and identifies several candidate methodological directions for future empirical investigation. These findings establish a cohesive theoretical framework to guide proactive OSS sustainability research.

Corresponding Author:
Beni Nurfauzi,
AMIKOM University of Yogyakarta, Jl. Ring Road Utara Condongcatur, Yogyakarta, 55281, Indonesia
Email: b.nurfauzi@students.amikom.ac.id

1. INTRODUCTION

Studies show that more than 50% of OSS projects on GitHub become inactive within their first four years, with the probability of surviving longer than five years dropping below 50% [1]. In one analysis of widely-used npm packages, 15% became abandoned over a six-year observation window [2]. Another study of 1,932 GitHub projects found that 16% were abandoned, though 41% of those were later revived by new maintainers [3], [4]. Unlike commercial software maintained by dedicated organizational structures, OSS projects depend on voluntary, distributed contribution networks whose long-term stability is difficult to predict, measure, or guarantee [5], [6]. As a result, the field has increasingly turned toward formal frameworks capable of conceptualizing and quantifying the human-centric vulnerabilities that govern whether a project survives or declines [7], [8]. Central among these frameworks is the Bus Factor, a construct that formally captures a project's structural dependence on a small cohort of irreplaceable core developers [9], [10].

The conceptual importance of the Bus Factor intersects directly with OSS lifecycle decline, as most projects rely on critically small teams vulnerable to disengagement and knowledge loss [8], [11], [12]. However, existing research has not systematically traced the causal pathway linking these dynamics to specific decline states [7], [13], particularly the undertheorized "zombie phase." This dangerous intermediate state is characterized by deceptive peripheral activity that obscures a complete halt in genuine human-driven coding from downstream dependents [3], [11], [14]. Because individual studies have only fragmentarily examined knowledge concentration, developer disengagement, and abandonment as isolated phenomena [7], a systematic synthesis is required to map the full causal chain from architectural vulnerability to lifecycle decline.

ISSN: 2715-6427

Furthermore, the methodological evolution of Bus Factor measurement spanning informal, coverage, decay, and network-based frameworks has not been taxonomically synthesized in relation to its implications for lifecycle prediction [10], [13], [15].

Therefore, this literature review systematically examines the intersection of Bus Factor dynamics and OSS project decline to construct a cohesive theoretical framework mapping knowledge concentration to zombie phase onset. The remainder of this paper is structured as follows: Section 2 describes the systematic review methodology; Section 3 synthesizes findings across Bus Factor measurement evolution, lifecycle decline mechanics, computational prediction approaches, and contributor behavioral dynamics; and Section 4 concludes by identifying central research gaps and outlining future empirical directions.

2. METHOD

2.1. Research Questions

This literature review is guided by a systematic inquiry into the mechanisms of open-source software decline, specifically focusing on knowledge concentration and developer attrition. To systematically evaluate the intersection of these domains, four primary research questions were formulated.

1. RQ1: How is Bus Factor defined and measured in OSS projects?
2. RQ2: What is the relationship between Bus Factor fluctuation and OSS project lifecycle decline toward the zombie phase?
3. RQ3: What computational methods have been used to predict or detect project abandonment in OSS?
4. RQ4: How do contributor commit dynamics and sleeping developer behaviour influence OSS project health?

2.2. Search Strategy

A systematic literature search was executed across Google Scholar, Semantic Scholar, and the ACM Digital Library to capture peer-reviewed research in computer science and software engineering. The retrieval was guided by four distinct search string clusters queried verbatim as follows:

1. CLUSTER 1: ("bus factor" OR "truck factor" OR "knowledge concentration") AND ("open source" OR "OSS" OR "GitHub").
2. CLUSTER 2: ("zombie project" OR "project abandonment" OR "inactive project" OR "project sustainability") AND ("open source software").
3. CLUSTER 3: ("contributor turnover" OR "commit activity" OR "developer dropout") AND ("open source" OR "GitHub").
4. CLUSTER 4: ("survival analysis" OR "random survival forest" OR "project health") AND ("open source software").

2.3. Inclusion and Exclusion Criteria

To ensure a high-quality and relevant corpus, the literature selection process was guided by strict peer-reviewed parameters from 2014 to 2026 focusing on Bus Factor metrics and open-source software sustainability. The comprehensive breakdown of these specific inclusion and exclusion criteria is detailed in Table 1.

Table 1. Inclusion and Exclusion Criteria for Literature Selection

Criteria	Description
Include	- Peer-reviewed journals, conference proceedings
	- Publication year: 2014–2026
	- Topics: Bus Factor, OSS health, project lifecycle, contributor dynamics, survival analysis in SE
	- Language: English
Exclude	- Grey literature (blogs, non-peer-reviewed reports)
	- Duplicate records across databases
	- Full text unavailable
	- Irrelevant after abstract screening

2.4. PRISMA Selection Process

The study selection process adhered to the PRISMA 2020 guidelines, with the complete screening and inclusion phases detailed in Figure 1.

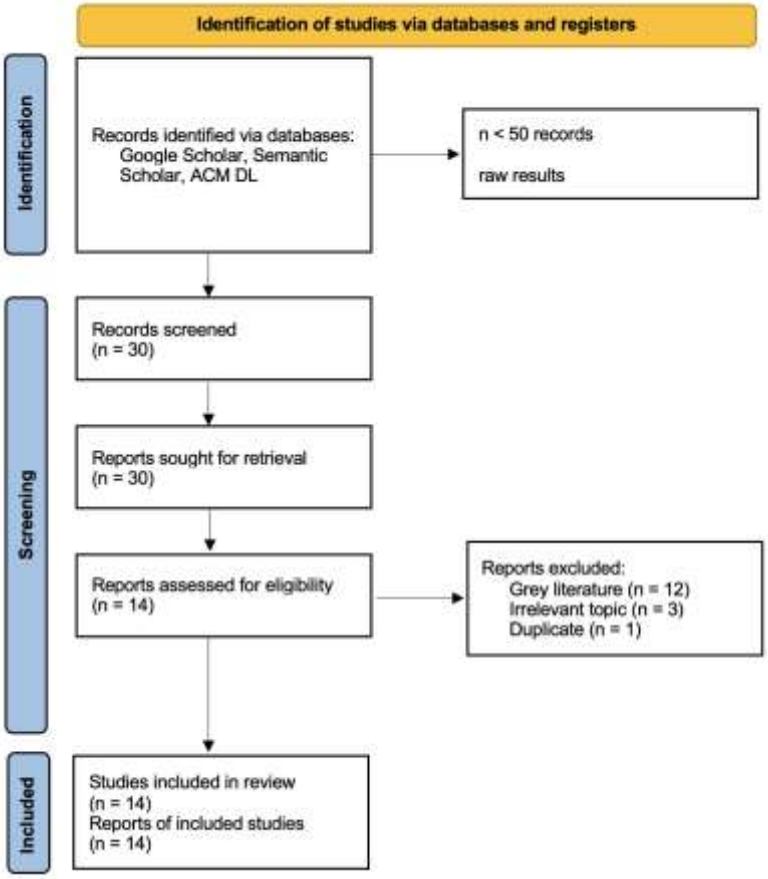

Figure 1. Identification of Studies via Databases and Registers

The initial database search yielded fewer than 50 records, which was reduced to 30 unique records after duplicate removal. During the title and abstract screening, all 30 records were assessed for relevance to Bus Factor dynamics and open-source project decline. Following a comprehensive full-text evaluation, 16 articles were excluded due to irrelevant subject matter or being grey literature (such as non-peer-reviewed preprints, working papers, and theses), leaving a final corpus of 14 included studies.

2.5. Quality Assessment

To ensure a rigorous foundation for exploring the Bus Factor in open-source software (OSS) lifecycle decline, the 14 selected papers were evaluated across four dimensions: relevance to the RQs, dataset quality/scale, methodological novelty, and alignment with current sustainability trends. This assessment filters out outdated or tangential literature, ensuring a high-quality corpus. The systematic scoring of each paper across five specific criteria and their overall quality ratings are detailed in Table 2.

Table 2. Quality Scoring Table

No	Author(s) & Year	Dataset Scale	Study Type Rigor	Methodological Clarity	Generalizability	RQ Contribution	Overall Quality
1	Cury & Avelino [10] (2024)	Medium	Medium	High	High	High	High
2	Robinson et al. [16] (2022)	High	Medium	High	Medium	High	High
3	Ait et al. [1] (2022)	High	High	High	High	High	High
4	Yin et al. [17] (2022)	Medium	Medium	High	Medium	High	Medium
5	Jamieson et al. [18] (2024)	Medium	Medium	High	High	High	High

6	Xiao et al. [19] (2023)	High	High	High	High	High	High
7	Khan & Filkov [20] (2024)	Medium	Medium	High	High	High	High
8	Qin et al. [21] (2025)	High	Medium	High	Medium	High	High
9	Liu et al. [22] (2026)	High	High	High	Medium	High	High
10	Tulili et al. [23] (2025)	High	High	High	Medium	High	High
11	Segundo et al. [24] (2025)	Low	Low	High	Medium	High	Medium
12	Bhattacharya et al. [13] (2025)	Medium	Medium	High	High	High	High
13	Nourry et al. [11] (2025)	High	High	High	High	High	High
14	Calefato et al. [25] (2022)	Medium	Medium	High	High	High	High

Table 2 demonstrates the strong overall quality of the corpus, with 12 High and two Medium ratings. The dataset scale is robust, with half of the studies analyzing thousands of repositories or entire ecosystems. Methodological clarity and RQ relevance scored universally High, ensuring reproducibility. Study designs range from rigorous empirical analyses to machine learning, with only one study utilizing conceptual simulation. Although some ecosystem-specific papers show Medium generalizability, the heavy reliance on large-scale data ensures high reliability. To map conceptual coverage, Table 3 categorizes each study by its thematic contributions.

Table 3. Thematic Categorization Table

No	Author(s) & Year	Primary Theme	Secondary Theme	Justification
1	Cury & Avelino [10] (2024)	Bus Factor Measurement	OSS Lifecycle Decline	Introduces the Knowledge Islands framework to visualize the spatial concentration of expertise, advancing traditional Bus Factor measurement to identify specific codebase vulnerabilities.
2	Robinson et al. [16] (2022)	Abandonment Prediction	OSS Lifecycle Decline	Compares Frequentist and Bayesian Cox Proportional-Hazards models, demonstrating the rigidity of traditional survival methods when analyzing project abandonment.
3	Ait et al. [1] (2022)	OSS Lifecycle Decline	Abandonment Prediction	Formally establishes the deceptive "zombie phase" of OSS projects by empirically analyzing repository survival rates on a massive scale.
4	Yin et al. [17] (2022)	Contributor Dynamics	OSS Lifecycle Decline	Combines institutional and socio-technical network analysis to demonstrate how communication and governance norms influence long-term project sustainability.
5	Jamieson et al. [18] (2024)	Contributor Dynamics	OSS Lifecycle Decline	Examines how value-related discussions predict contributor turnover, highlighting the impact of ethical conflicts on project decline.
6	Xiao et al. [19] (2023)	Abandonment Prediction	Contributor Dynamics	Employs longitudinal regression and survival analysis to map how early contributor participation patterns determine sustained project activity.
7	Khan & Filkov [20] (2024)	Abandonment Prediction	None	Addresses the accuracy-interpretability tradeoff by using the ReACTs framework and LIME to translate machine learning predictions into actionable sustainability advice.
8	Qin et al. [21] (2025)	Contributor Dynamics	Abandonment Prediction	Investigates the departure of company-sponsored contributors using initial behavior features, highlighting the distinct turnover patterns of corporate actors.
9	Liu et al. [22] (2026)	Contributor Dynamics	None	Profiles contributors in popular ML libraries based on granular workload compositions and active shifts, identifying behavioral trajectories over time.

No	Author(s) & Year	Primary Theme	Secondary Theme	Justification
10	Tulili et al. [23] (2025)	Contributor Dynamics	OSS Lifecycle Decline	Performs a 23-year longitudinal analysis on the Gentoo ecosystem, correlating developer sentiment with turnover and component stability.
11	Segundo et al. [24] (2025)	Bus Factor Measurement	None	Shifts from static evaluation to proactive mitigation by simulating how Bus Factor-guided task allocation can stabilize software development teams.
12	Bhattacharya et al. [13] (2025)	Bus Factor Measurement	None	Proposes a Dynamic Knowledge Decay model using psychological retention principles to more accurately capture partial and fading code familiarity over time.
13	Nourry et al. [11] (2025)	OSS Lifecycle Decline	Bus Factor Measurement	Quantifies Truck Factor Developer Detachment (TFDD) across 36,000+ repositories, providing definitive empirical evidence on the frequency of core developer loss.
14	Calefato et al. [25] (2022)	Contributor Dynamics	OSS Lifecycle Decline	Employs state transition modeling to prove that developers exhibiting prolonged "sleeping" behavior have a high probability of permanent project abandonment.

The thematic distribution presented in Table 3 reveals a strong concentration in Contributor Dynamics and Abandonment Prediction Methods, which together represent the primary focus of 9 out of the 14 studies. This dominance highlights a prevailing consensus in the software engineering community that OSS sustainability is fundamentally a socio-technical challenge driven by human behaviors, early participation signals, and the necessity to forecast contributor disengagement. OSS Lifecycle Decline serves as a critical secondary theme across many studies, firmly establishing the empirical reality of the zombie phase and core developer loss. Conversely, Bus Factor Measurement is the least represented as a primary theme, with only three dedicated studies. This distribution exposes a critical gap in the current state of the field: while the community excels at analyzing contributor turnover and building predictive ML models, there remains a structural disconnect in integrating advanced, dynamic Bus Factor measurements directly into those predictive frameworks. To contextualize the rapid methodological advancements within this domain, Table 4 maps the chronological evolution of the research focus across the five-year inclusion period.

Table 4. Research Trend Mapping Table

Year	Papers Published	Dominant Theme	Emerging Focus	Notable Gap
2022	4	Lifecycle Decline & Contributor Dynamics	Formalizing the "zombie phase" and establishing state-transition models for sleeping developer behaviors.	Lack of predictive ML frameworks utilizing dynamic socio-technical features.
2023	1	Abandonment Prediction	Linking initial onboarding signals directly to long-term project survival probabilities.	Focus restricted to surface-level activity metrics rather than structural codebase knowledge.
2024	3	Abandonment Prediction & Contributor Dynamics	Enhancing ML model interpretability (LIME/ReACTs) and analyzing value-related contributor turnover.	Lack of temporal decay considerations in structural and visual measurements.
2025	5	Bus Factor Measurement & Contributor Dynamics	Shift toward dynamic knowledge decay models, proactive simulations, and ecosystem-scale corporate turnover analysis.	Absence of integrated survival models that actively ingest dynamic Bus Factor data.
2026	1	Contributor Dynamics	Highly granular profiling of contributor workloads in specialized, high-stakes domains (e.g., ML OSS).	Limited integration of specialized domain profiles back into general abandonment prediction.

As outlined in Table 4, the chronological trajectory of OSS sustainability research reveals a clear methodological evolution from early descriptive survival diagnostics toward advanced, dynamic predictive architectures. However, a persistent research gap remains across this timeline: existing frameworks

ISSN: 2715-6427

continuously fail to integrate dynamic Bus Factor fluctuations as a real-time predictor variable. Synthesizing this corpus confirms that the 14 selected studies provide a methodologically rigorous and temporally relevant foundation excelled in dataset quality and methodological novelty to successfully answer the four RQs and identify critical gaps in current OSS survival forecasting.

3. RESULTS AND DISCUSSION

The systematic literature review process yielded a final corpus of 14 peer-reviewed studies, selected from an initial pool of 30 records following the PRISMA-based screening procedure described in Section 2. These studies collectively address the four research questions formulated in this review, spanning publication years from 2022 to 2026 and drawing on empirical analyses of large-scale GitHub repository datasets, contributor behavioral data, and computational modeling frameworks. Table 5 presents a structured synthesis of the included studies, summarizing each paper's methodological approach, primary dataset, and key contribution to the understanding of Bus Factor dynamics, OSS lifecycle decline, contributor turnover, and abandonment prediction.

Table 5. Summary of Included Studies

No	Author(s)	Year	RQ Addressed	Method / Approach	Dataset / Source	Key Finding
1	Cury & Avelino [10]	2024	RQ1, RQ2	Knowledge Islands Framework	GitHub repositories	Visualizes knowledge concentration as hierarchical directory trees; identifies isolated 'islands' of expertise at file/module level
2	Robinson et al. [16]	2022	RQ3	Kaplan-Meier + Cox PH (Frequentist & Bayesian)	Open-source Python projects on GitHub	Bayesian Cox PH outperforms frequentist Cox; R^2=0.19 highlights rigidity of proportional hazards assumption in OSS contexts
3	Ait et al. [1]	2022	RQ2, RQ3	Empirical survival rate analysis	19,000+ GitHub projects	Zombie repositories are deceptive precursors to project death; 18% of downstream projects respond with avg 13.5-month delay
4	Yin et al. [17]	2022	RQ4	Institutional analysis + socio-technical network analysis	OSS communities (CSCW dataset)	Social governance structures and network centrality directly predict long-term OSS sustainability beyond simple activity metrics
5	Jamieson et al. [18]	2024	RQ4	Granger causality analysis and impulse response functions	Logs of GitHub issues and commits from 52 Decentralized Web (DWeb) projects	Value-related discussions significantly predict contributor turnover; discussions about respectfulness predict increased departure and decreased newcomer onboarding, while discussions about social power predict better retention.
6	Xiao et al. [19]	2023	RQ3, RQ4	Longitudinal regression + survival analysis	31 GitHub projects (FSE 2023)	First 3–6 months of contributor participation are highly predictive of long-term sustained activity; organizational diversity is key
7	Khan & Filkov [20]	2024	RQ3, RQ4	ReACTs framework (ML + LIME explainability)	OSS projects on GitHub	Evidence-based ReACTs translate opaque ML predictions into actionable developer advice; AUC 0.72–0.84 on sustainability tasks
8	Qin et al. [21]	2025	RQ4	XGBoost classifier on initial behavior features	Company-sponsored OSS contributors	Initial contribution behavior (commit type, frequency, scope) predicts which company-sponsored developers will stop contributing
9	Liu et al. [22]	2026	RQ4	Contributor profile analysis	Popular ML libraries on GitHub	Significant divergence in retention rates between company-sponsored and volunteer

No	Author(s)	Year	RQ Addressed	Method / Approach	Dataset / Source	Key Finding
						contributors in high-profile ML OSS libraries
10	Tulili et al. [23]	2025	RQ4	Ecosystem-level longitudinal analysis	OSS ecosystem dataset	Turnover, retention, and growth follow distinct ecosystem-level patterns; peripheral contributor loss precedes core team collapse
11	Lisan & Norris [9]	2025	RQ1	Agent-based simulation using Truck Factor metric	Simulated software development teams	Bus Factor-guided task allocation reduces knowledge concentration; simulation validates dynamic redistribution strategies
12	Bhattacharya et al. [13]	2025	RQ1	Dynamic Knowledge Decay Model (exponential decay)	GitHub repositories	Exponential retention decay model captures partial knowledge; more accurately reflects real-time Bus Factor than static coverage
13	Nourry et al. [11]	2025	RQ2, RQ4	Empirical survival + transition probability analysis	36,000+ GitHub repositories	89.65% of projects experience full core developer loss; only 27.07% survive; 70% of detachments within first 3 years
14	Calefato et al. [25]	2022	RQ4	Survival analysis + state transition modeling	OSS core developers on GitHub	Breaks exceeding 12 months yield 54% probability of permanent departure; sleeping behavior inflates perceived Bus Factor resilience

3.1. The Architecture of Knowledge Concentration: Defining and Measuring Bus Factor in OSS

The conceptualization and quantification of the Bus Factor have evolved through five distinct generations (Table 6). Generation 0 began as an informal "hit by a truck" heuristic that conceptualized human dependency but lacked a mathematical foundation. To address this, Generation 1 introduced coverage-based greedy heuristics to formalize static code ownership based on commits, though it ignored developer turnover and knowledge decay. Generation 2 resolved this by introducing dynamic models with exponential decay functions, but operated primarily at the file level. To capture broader structural topologies, Generation 3 utilized network-based Knowledge Islands to visualize interconnected codebase vulnerabilities, though it remained a purely retrospective diagnostic tool. Finally, Generation 4 deployed proactive agent-based simulations for task reallocation, but introduced high computational complexity and remains isolated from downstream ecosystem survival forecasting.

Table 6. Taxonomy Table

Generation	Approach Name	Core Innovation	Limitation Addressed from Previous	New Limitation Introduced	Representative Study
Generation 0	Informal Heuristic	Conceptualized single-point human dependency risk using a qualitative metaphor.	Addressed the lack of initial awareness regarding human-centric architectural risk.	Lacks mathematical formalization and cannot be computationally scaled or measured.	Foundational Concept
Generation 1	Coverage-Based Greedy Heuristic	Algorithmic counting models using linear repository commit history thresholds.	Overcame the qualitative and unmeasurable nature of Generation 0.	Assumes contributor knowledge is completely static, permanent, and immune to temporal erosion.	Lisan & Norris (2024)

ISSN: 2715-6427

Generation	Approach Name	Core Innovation	Limitation Addressed from Previous	New Limitation Introduced	Representative Study
Generation 2	Decay-Based Dynamic Model	Dynamic time-series knowledge calculations using exponential decay functions.	Overcame the static time-slice assumption by modeling memory retention and loss over time.	Fails to map the socio-technical structural communication topologies between contributors.	Bhattacharya et al. (2025)
Generation 3	Network-Based Knowledge Islands	Bipartite structural networks visualizing the spatial concentration of expertise.	Resolved the individualistic, non-structural limitations by visualizing broader codebase vulnerabilities.	Operates purely as a retrospective diagnostic framework rather than an active testing environment.	Cury & Avelino (2024)
Generation 4	Agent-Based Simulation	Simulates distributed task-allocation dynamics using autonomous virtual agents.	Overcame the passive, retrospective analytical limitations to allow proactive mitigation strategies.	High computational complexity; isolated from downstream predictive lifecycle models.	Lisan & Norris / Agent-Based Study (2025)

The evolution of the Bus Factor reveals a clear trajectory toward higher temporal and structural sensitivity, moving from static commit-counting to non-linear time decay [13] and multi-agent simulations [9]. However, a critical gap persists: none of these advanced configurations have been integrated as time-varying predictors within modern abandonment models, isolating risk metrics from active health forecasting. This vulnerability stems from the volunteer-driven nature of OSS, where unequal knowledge distribution is structurally inevitable [26]. Contributors gravitate toward specific modules, leading to knowledge concentration siloed within a small core team [8]. This critical dependence means their sudden absence creates a knowledge vacuum that paralyzes project maintenance [6].

Originally conceptualized via the informal "hit by a truck" scenario, this phenomenon is interchangeably termed Bus Factor, Truck Factor, or Knowledge Concentration Index [6]. Regardless of nomenclature, the goal is identical: quantifying a project's resilience by identifying the minimum threshold of critical developers required to sustain operations [6]. Empirical literature operationalizes this through three primary computational approaches:

- Coverage-Based: Uses a strict greedy heuristic, ranking developers by authorship and iteratively removing top contributors until maintained files drop below a 50% threshold. A project has a Bus Factor of one if removing a single developer orphans over half the repository.
- Decay-Based (Dynamic Knowledge Decay Model): Inspired by psychological memory retention, this approach posits that multiple developers gain partial knowledge through collaboration. It applies an exponential retention fall-off curve to penalize older contributions and reward sustained investment, reflecting current resilience rather than a historical snapshot.
- Network-Based (Knowledge Islands): Extends the Bus Factor into a structural visualization tool. It maps the codebase as a hierarchical directory tree to highlight isolated "islands" of concentrated technical understanding, pinpointing exact modules at risk of abandonment.

Despite these advancements, significant limitations remain. A central tension persists over whether the Bus Factor should be treated as a static snapshot or a continuous dynamic signal. Furthermore, a profound lack of measurement standardization and universally accepted decay parameters across software engineering [27] makes cross-ecosystem comparisons highly challenging. This fragile architecture of knowledge concentration directly influences how its collapse dictates OSS lifecycle decline.

3.2. From Knowledge Loss to Project Death: Bus Factor Collapse and OSS Lifecycle Decline

The socio-technical decline taxonomy of OSS projects progresses linearly through four stages, each defined by a critical research gap (Table 7). It begins with localized knowledge concentration, where isolated architectural expertise creates a brittle codebase vulnerable to Bus Factor collapse (Cury & Avelino, 2024); however, current metrics fail to quantify the exact velocity and pathways of this structural collapse. This vulnerability triggers developer disengagement, where remaining maintainers face severe operational strain and enter prolonged inactive periods [25], though empirical frameworks cannot yet predict which specific maintainers will succumb to this socio-technical stress. When this hibernation compounds into the total loss of the core team, the project enters a deceptive zombie phase where development stalls entirely [11], [25], yet the field lacks clear threshold telemetry to identify the precise trigger point of this system-wide state. Finally, this unstable phase terminates in project death, driven by unmanaged technical debt and a sharp drop in survival probability that forces repository deprecation [1] despite this clear trajectory, modern models still fail by treating project extinction as a spontaneous event rather than the predictable climax of a long-term zombie phase.

Table 7. Taxonomy Table

Causal Link	Supporting Studies	Nature of Gap	Gap Severity
Knowledge Concentration → Bus Factor Collapse	Cury & Avelino (2024); Bhattacharya et al. (2025)	Structural Blind Spot: Lack of dynamic models to map the precise structural velocity and path of repository knowledge concentration.	Medium
Bus Factor Collapse → Sleeping Developer Behavior	Calefato et al. (2022)	Behavioral Disengagement Gap: Inability to predict individual developer hibernation as a direct consequence of socio-technical strain.	High
Sleeping Developer Behavior → Zombie Phase	Calefato et al. (2022); Nourry et al. (2025)	Latency & Threshold Uncertainty: Absence of quantitative mathematical indicators defining when developer breaks trigger a zombie state.	High
Zombie Phase → Project Death	Ait et al. (2022); Robinson et al. (2022)	Abandonment Integration Gap: Failure of survival models to incorporate time-varying zombie metrics, treating death as a static snapshot event.	High

The identified empirical gaps across this causal chain form a compounding systemic blind spot that severely restricts OSS risk management; because existing models isolate each stage of decline, they fail to track how risks flow from initial architectural imbalances down to ultimate project abandonment. Consequently, resolving the disconnect between knowledge concentration and final failure forecasting is the field's most urgent priority. This trajectory is best conceptualized through modern lifecycle models that recognize an insidious intermediate "zombie phase" a deceptive state where genuine, human-driven coding ceases but peripheral activities like automated bots or issue comments continue [6], [28]. This phase is deeply rooted in human capital: a sudden Bus Factor decline creates an immediate knowledge vacuum that accrues insurmountable technical debt, making it far more dangerous to global software supply chains than explicit abandonment because downstream dependents are lured into a false sense of security [6], [14]. Large-scale empirical evidence substantiates this threat, revealing that 89.65% of over 36,000 GitHub repositories experience a total core developer detachment (TFDD) during their lifecycle, with 70% of these detachments occurring within the first three years, where survival probabilities subsequently plummet to just 27.07% [6]. Synthesis matrices confirm that Bus Factor dynamics are most critical during project inception where initial detachment risks are highest and ultimate decline [1], [19]. Conversely, empirical coverage is weakest during the transitional growth and maturity phases where localized knowledge subtly concentrates [23]. This imbalance implies that community interventions must be executed proactively during growth and maturity; waiting until the decline phase, when developers already exhibit prolonged sleeping behavior, is consistently too late to prevent a repository from slipping into an irreversible zombie phase [25] and exposing modern software supply chains to sudden project death.

ISSN: 2715-6427

Table 8. Synthesis Matrix Table

Lifecycle Phase	Definition	Bus Factor Criticality	Key Empirical Evidence	Observing Study	Consequence if Unaddressed
Inception (0–3 years)	The foundational period of a project characterized by early developer participation and initial code commits.	High	70% of all total core developer detachments occur within the first three years; early participation metrics strongly predict long-term sustainability.	Nourry et al. (2025); Xiao et al. (2023)	The project suffers premature extinction before establishing a viable community or structural resilience.
Growth	A period of expanding repository activity where new contributors are onboarded and the codebase	Medium	Turnover, retention, and growth follow distinct ecosystem-level patterns, where the loss of peripheral contributors can precede core team collapse.	Tulili et al. (2025)	Severe dependency bottlenecks form, as new participants fail to convert into core maintainers capable of sharing the technical burden.
Maturity	Stabilized operations featuring consistent development rhythms, established community roles, and sustained ecosystem retention.	Medium	Components exhibit long-term stability and varying retention rates based on developer sentiments, yet continue to rely on a dangerously small core team.	Tulili et al. (2025)	Hidden vulnerabilities slowly accumulate as codebase complexity increases, cementing isolated knowledge silos.
Decline	A measurable drop in coding activity, marked by core developers reducing their commit frequency and pausing active participation.	High	Core developers enter a deceptive "sleeping" state; taking a break that exceeds 12 months yields a 54% probability of permanent departure.	Calefato et al. (2022)	Unmanaged collapse of the project's Bus Factor as the core team silently fractures without transferring knowledge.
Zombie Phase	A non-functional state where genuine human coding ceases, but the repository displays deceptive peripheral or automated bot activity.	High	89.65% of projects experience a total loss of their core development team, transitioning into a deceptive holding pattern before death.	Ait et al. (2022); Nourry et al. (2025)	Unpatched security vulnerabilities accumulate, posing a severe and silent risk to downstream software supply chains.
Project Death	The permanent cessation of all repository activities, resulting in explicit abandonment and formal deprecation.	Low (Post-mortem)	Over half of analyzed GitHub repositories die within their first four years, with survival probabilities dropping sharply over time.	Ait et al. (2022)	Forces disruptive, costly downstream migrations and triggers cascading failures across dependent OSS ecosystems.

The reviewed literature demonstrates that developer expertise decays non-linearly over time rather than disappearing abruptly upon departure [13]. Because technical familiarity erodes progressively, static Bus Factor measurements that rely strictly on historical commit counts are structurally inadequate, routinely overestimating resilience by artificially treating inactive contributors as current knowledge holders and blinding maintainers to silent architectural vulnerabilities. This descent toward the zombie phase is rarely abrupt; instead, core developer loss manifests in distinct commit patterns and "sleeping" behaviors where short, intense coding periods are separated by prolonged dormancy [16]. While recovery remains possible during initial pauses, rigorous survival analysis reveals a stark tipping point: if a developer's inactivity exceeds 12 months, the probability of resuming contributions drops to 21–26%, while permanent disengagement surges to 54%, inevitably pushing the repository into an irreversible zombie phase sustained only by peripheral chatter

or bots. Evaluating this collective evidence raises a critical analytical debate regarding whether Bus Factor collapse is the root cause of project decline or merely a late-stage symptom of underlying organizational failures such as community toxicity, conflicting inclusivity values, or shifting corporate sponsorship which precipitate burnout and deflate the metric before abandonment [6]. Resolving these causal ambiguities and tracking the unstudied post-exit micro-dynamics of the remaining peripheral community requires moving beyond static post-mortem analyses and integrating dynamic, predictive modeling frameworks.

3.3. Predicting the Unpredictable: Computational Approaches to OSS Abandonment Detection

Computational approaches to OSS abandonment prediction span three categories statistical survival models, machine learning classifiers, and deep learning or ensemble architectures each presenting distinct tradeoffs between handling right-censored lifecycle data, interpretability, and capturing non-linear covariate interactions [7], [15], [16]. While statistical models offer strong interpretability and native censoring support but fail to capture non-linear dynamics, deep learning models maximize predictive accuracy at the expense of transparency. However, a striking, systemic omission persists across this entire spectrum: the Bus Factor is excluded from the feature space of all evaluated frameworks. Existing models over-rely on surface-level activity counts (e.g., commit frequency and contributor counts) because the Bus Factor requires computationally expensive, time-aware analysis of file authorship that does not easily translate into traditional static, tabular features [15], [27]. Resolving this dominant methodological gap requires a structured comparative framework (as evaluated across six critical dimensions in Table 9) and the development of advanced survival architectures capable of continuously ingesting dynamic, time-varying covariates. Future predictive frameworks must actively transition from aggregate activity counts to real-time signals of knowledge distribution and temporal decay, directly mapping these fluctuations to the changing hazard rate of project abandonment to enable transparent, actionable community interventions.

Table 9. Comparison of Methods

Method	Category	Handles Right-Censored Data	Interpretability	Captures Non-linear Effects	Bus Factor Support	Reported Performance	Key Limitation
Cox Proportional-Hazards (Frequentist)	Statistical Survival	✓ Native	✓ High	✗ No	✗ No	$R^2 = 0.19$	Strict proportional hazards assumption violated in dynamic OSS lifecycles; overly rigid for long-term prediction
Cox Proportional-Hazards (Bayesian)	Statistical Survival	✓ Native	✓ High	✗ No	✗ No	Outperforms Freq. Cox	Same proportional hazard constraints; computationally expensive; still excludes knowledge distribution features
Kaplan-Meier Estimator	Statistical Survival	✓ Native	✓ High	✗ No	✗ No	Descriptive only	Non-parametric but purely descriptive; cannot incorporate covariates or dynamic features like Bus Factor

ISSN: 2715-6427

Method	Category	Handles Right-Censored Data	Interpretability	Captures Non-linear Effects	Bus Factor Support	Reported Performance	Key Limitation
Random Forest / XGBoost	ML Classifier	✗ No	◑ Moderate (LIME needed)	✓ Yes	✗ No	AUC 0.72–0.84	Treats lifecycles as static classification; ignores right-censored data; Bus Factor excluded from feature space
Temporal Convolutional Network (TCN)	Deep Learning	✗ No	✗ Low	✓ Yes	✗ No	High accuracy	Black-box nature restricts actionable insights; requires large training data; no native censoring support
DeepHit Neural Network	Deep Survival	✓ Native	✗ Low	✓ Yes	✗ No	Competitive on 9/11 datasets	Opaque predictions unsuitable for community interventions; cannot integrate dynamic time-varying Bus Factor

Legend: ✓ = Fully supported ◑ = Partially supported ✗ = Not supported ★ = Proposed framework in this review

While the Random Survival Forest (RSF) is proposed as a candidate framework for future empirical validation, the scholarly discourse surrounding open-source abandonment has already transitioned significantly from descriptive survival diagnostics toward predictive, specialized socio-technical architectures. In 2022, the dominant paradigm focused on baseline survival diagnostics and phased, observable decline, utilizing classic survival models to map ecosystem death and analyze sleeping developer states [1], [16], [17], [25]. By 2023 and 2024, the focus shifted toward early foresight and explainable machine learning, demonstrating that long-term activity is predictable through early participation (Xiao et al., 2023) and utilizing frameworks like Knowledge Islands, XGBoost, and ReACTs to prove that predictive models need not remain opaque black boxes [10], [20], [21]. The literature from 2025 and 2026 marks the field's rapid maturation by focusing on dynamic temporal erosion, large-scale ecosystem turnover, and specialized contributor profiling; researchers introduced dynamic knowledge decay, agent-based simulations, and quantified massive corporate or core detachments [9], [11], [13], fundamentally dismantling the flawed assumption that developer expertise remains static during prolonged inactivity while proving that core developers exhibit highly heterogeneous behavioral patterns [22].

Table 10. Diachronic Evolution Table

Year	Papers	Dominant Research Paradigm	Key Methodological Innovation	Prevailing Assumption Challenged	Remaining Gap
2022	Robinson et al. (2022); Ait et al. (2022); Yin et al. (2022); Calefato et al. (2022)	Baseline Survival Diagnostics	Frequentist and Bayesian survival analysis alongside state transition modelling.	Abandonment is a sudden, binary event (proved it is a phased decline).	Models lacked predictive foresight and were primarily descriptive and retrospective.

Year	Papers	Dominant Research Paradigm	Key Methodological Innovation	Prevailing Assumption Challenged	Remaining Gap
2023	Xiao et al. (2023)	Early Participation Prediction	Longitudinal regression combined with predictive survival analysis.	Long-term sustainability can only be accurately measured using mature project metrics.	Focus remained restricted to flat activity metrics rather than deep structural knowledge distribution.
2024	Cury & Avelino (2024); Qin et al. (2024); Khan & Filkov (2024)	Explainable ML & Structural Mapping	XGBoost classifiers, LIME explainability frameworks, and bipartite network visualizations.	ML predictions must be opaque black boxes; simple commit counts equal knowledge.	Structural measurements lacked temporal and psychological decay considerations.
2025	Bhattacharya et al. (2025); Lisan & Norris (2025); Nourry et al. (2025); Tulili et al. (2025); Qin et al. (2025)	Dynamic Decay & Ecosystem Modeling	Exponential knowledge decay functions, agent-based simulations, and TCN models.	Developer expertise remains static and fully intact during periods of prolonged inactivity.	Dynamic architectural metrics remain isolated from downstream abandonment forecasting models.
2026	Liu et al. (2026)	Specialized Contributor Profiling	Granular tracking of workload composition and shift analysis in ML domains.	Core developers exhibit identical, homogenous participation and engagement habits.	Specialized behavioral trajectories are not yet integrated into real-time survival monitoring frameworks.

This diachronic analysis reveals a consistent, progressive evolution in open-source sustainability research: the field has systematically advanced from static, post-mortem descriptions toward dynamic, real-time prediction. Early foundational work primarily focused on modelling historical survival rates and defining theoretical states of decline. Subsequently, researchers rapidly adopted machine learning classifiers to forecast long-term activity, eventually culminating in sophisticated models of psychological knowledge decay and multi-agent socio-technical simulations. Yet, despite this remarkable methodological maturation, a profound structural blind spot persists across the entire five-year timeline. The integration of the Bus Factor as a dynamically fluctuating, time-varying predictor variable remains the persistent, unresolved frontier. While modern algorithms excel at analysing surface-level participation metrics and providing explainable community advice, they continually fail to natively ingest real-time codebase knowledge concentrations. Consequently, bridging this disconnect between dynamic architectural risk metrics and downstream abandonment forecasting stands as the most structurally important gap in the current literature.

3.4. The Human Dynamics of Decline: Contributor Behavior, Sleeping Patterns, and Project Health

The disengagement of core developers in open-source software can be formalized as a continuous state transition model that maps the trajectory of contributors from full engagement to total project abandonment. The Active state represents the healthy baseline condition where a core contributor contributes code regularly, exhibiting steady commitment, broad collaboration, and high contribution intensity (Xiao et al., 2023). When a developer's engagement begins to wane, they transition into the Peripheral state. This state is characterized by observable behavioral signals such as a reduced commit frequency, a narrowing collaboration scope, and a shift toward minor perfective maintenance rather than adaptive new feature development (Qin et al., 2025; Tulili et al., 2025).

If a developer's engagement continues to decay, they enter the Sleeping state. In this state, the developer makes no code contributions, but their break duration remains under 12 months. During this critical window, recovery is still highly possible; empirical survival analysis demonstrates that sleeping developers have a 46% probability of transitioning into the Returned state, wherein they resume active coding or collaborative activities (Calefato et al., 2022). However, if the inactivity period exceeds the 12-month threshold, the developer transitions into the Gone state. Calefato et al. (2022) establish that crossing this temporal boundary yields a 54% probability of permanent departure, signifying a virtually irreversible loss of institutional knowledge.

ISSN: 2715-6427

Ultimately, individual developer trajectories aggregate into project-level consequences, culminating in the Zombie Trigger. This threshold is reached when sleeping or gone developers collectively drain the repository of its vital expertise, leaving no active core maintainers. The observable signal for this trigger is a complete Truck Factor Developer Detachment, an event that 89.65% of GitHub projects eventually experience, which actively pushes the repository into the terminal zombie phase (Nourry et al., 2025).

Table 11. State Transition Table

State	Observable Signals	Transition To	Transition Probability	Empirical Source
Active	Regular code commits, steady commitment, high contribution intensity.l	Peripheral	Not formally quantified	Xiao et al. (2023); Qin et al. (2025)
Peripheral	Reduced commit frequency, narrowing collaboration scope, perfective maintenance.	Sleeping	Not formally quantified	Qin et al. (2025); Tulili et al. (2025)
Sleeping	Zero code contributions; break duration remains < 12 months.	Returned	46% probability of resuming activity	Calefato et al. (2022)
Sleeping	Zero code contributions; break duration crosses the 12-month threshold.	Gone	54% probability of permanent departure	Calefato et al. (2022)
Gone	Continuous inactivity extending > 12 months.	Zombie Trigger	Dependent on the project's remaining Bus Factor	Nourry et al. (2025)
Returned	Resumed commits or non-coding collaborative activities.	Active	Not formally quantified	Calefato et al. (2022)
Zombie Trigger	Total loss of core development team (Bus Factor collapse).	Zombie Phase	89.65% of projects experience this detachment	Nourry et al. (2025)

The formalization of the sleeping developer lifecycle elevates contributor disengagement from a mere descriptive observation to a predictive theoretical framework. The synthesized literature demonstrates that developer departure is rarely spontaneous; rather, it follows a highly predictable state trajectory characterized by a gradual decay in commit intensity and collaboration scope [19], [21]. By establishing distinct behavioral phases particularly the critical 12-month threshold where temporary sleeping behavior solidifies into a 54% probability of permanent abandonment this model maps the exact human dynamics that precipitate a project's decline [25]. When multiple core contributors simultaneously traverse this trajectory toward the "gone" state, it initiates a catastrophic knowledge collapse [11]. Consequently, if this behavioral state model is actively monitored via dynamic Bus Factor tracking, it can serve as a robust early warning system, empowering maintainers to execute targeted interventions before the irreversible onset of the zombie phase.

3.5. Discussion

A cross-cutting synthesis reveals that open-source software (OSS) lifecycle decline is driven by an interconnected socio-technical system characterized by three themes: the structural fragility of knowledge concentration, the deceptive nature of dormant developer behaviors, and the rigid limitations of current predictive methodologies. OSS projects inherently rely on a dangerously small cohort of core experts [6]. When these individuals experience life or corporate transitions, they enter a "sleeping" state, causing a catastrophic

loss of institutional knowledge that drags the repository into a deceptive zombie phase (peripheral, non-coding activity) and, ultimately, project death. Despite this clear causal sequence, a profound measurement gap persists: the dynamic fluctuation of the Bus Factor remains systematically absent from predictive abandonment models. While current machine learning and statistical survival frameworks utilize static, surface-level activity counts, they fail to incorporate the actual distribution and non-linear temporal decay of technical expertise proved by frameworks like the Dynamic Knowledge Decay Model and Knowledge Islands.

This measurement deficiency is deeply rooted in a methodological gap. Traditional survival frameworks cannot accommodate non-linear, time-varying human disengagement, while basic machine learning classifiers fail to handle right-censored lifecycle data and lack transparent interpretability [7]. Bridging this divide requires advanced architectures—such as ensemble survival methods or explainable survival architectures capable of handling time-varying covariates, right-censored data, and interpretable feature outputs [27]. Resolving these gaps holds profound implications across the software ecosystem: it enables OSS maintainers to deploy early warning interventions (e.g., task redistribution and mentorship) before permanent core disengagement; allows downstream enterprise users to mitigate supply chain risks prior to cascading system failures [6], [9] and advances software engineering research by embedding dynamic knowledge decay into models that sustain global digital public goods.

4. CONCLUSION

In conclusion, a systematic review of fourteen studies confirms that open-source software (OSS) decline is driven by a socio-technical causal chain, evidenced by 89.65% of projects losing all core developers and a 54% permanent departure rate among 'sleeping developers' who break for over a year. Despite this fragility, current computational abandonment prediction models remain blind to the temporal decay of contributor expertise and dynamic Bus Factor fluctuations. To bridge this critical research gap, future empirical work must leverage promising frameworks like the Random Survival Forest to achieve three objectives: prospectively validate dynamic Bus Factor features on large-scale datasets, track real-time contributor fluctuations during the critical first three years of development, and establish a universally standardized Bus Factor measurement protocol. Advancing these directions will transform the Bus Factor from a retrospective diagnostic into a proactive early warning instrument to sustain global digital infrastructure.

ACKNOWLEDGEMENTS

The author wishes to express his deepest gratitude to Prof. Kusrini. for her invaluable guidance, continuous support, and insightful suggestions throughout the development of this research. Her expertise and encouragement have been instrumental in the completion of this journal article.

REFERENCES

[1] A. Ait, J. L. C. Izquierdo, dan J. Cabot, “An empirical study on the survival rate of GitHub projects,” dalam *Proceedings of the 19th International Conference on Mining Software Repositories*, Pittsburgh Pennsylvania: ACM, Mei 2022, hlm. 365–375. doi: 10.1145/3524842.3527941.

[2] G. Avelino, E. Constantinou, M. T. Valente, dan A. Serebrenik, “On the abandonment and survival of open source projects: An empirical investigation,” 19 Juni 2019, *arXiv*: arXiv:1906.08058. doi: 10.48550/arXiv.1906.08058.

[3] R. He, H. Ye, dan M. Zhou, “Revealing the value of Repository Centrality in lifespan prediction of Open Source Software Projects,” 13 Mei 2024, *arXiv*: arXiv:2405.07508. doi: 10.48550/arXiv.2405.07508.

[4] C. Osborne, P. Sharratt, D. Foster, dan M. Boehm, “A Toolkit for Measuring the Impacts of Public Funding on Open Source Software Development,” 9 November 2024, *arXiv*: arXiv:2411.06027. doi: 10.48550/arXiv.2411.06027.

[5] T. Dey, B. Fitzgerald, dan S. Daniel, “CROSS: A Contributor-Project Interaction Lifecycle Model for Open Source Software,” 17 September 2024, *arXiv*: arXiv:2409.08267. doi: 10.48550/arXiv.2409.08267.

[6] Y. Yehudi, C. Goble, dan C. Jay, “Individual context-free online community health indicators fail to identify open source software sustainability,” 9 Mei 2024, *arXiv*: arXiv:2309.12120. doi: 10.48550/arXiv.2309.12120.

[7] Y. Xu, R. He, H. Ye, M. Zhou, dan H. Wang, “Predicting Abandonment of Open Source Software Projects with An Integrated Feature Framework,” 29 Oktober 2025, *arXiv*: arXiv:2507.21678. doi: 10.48550/arXiv.2507.21678.

[8] Y. Yehudi, C. Goble, dan C. Jay, “Individual context-free online community health indicators fail to identify open source software sustainability,” Sep 2023.

[9] A. Lisan dan B. Norris, “Guiding Effort Allocation in Open-Source Software Projects Using Bus Factor Analysis,” 2024, *arXiv*. doi: 10.48550/ARXIV.2401.03303.

[10] O. Cury dan G. Avelino, “Knowledge Islands: Visualizing Developers Knowledge Concentration,” 16 Agustus 2024, *arXiv*: arXiv:2408.08733. doi: 10.48550/arXiv.2408.08733.

[11] O. Nourry, M. Kondo, S. Saito, Y. Iimura, N. Ubayashi, dan Y. Kamei, “Abandonment and Resilience: Understanding Core Developer Turnover in Open-Source Software,” *IEICE Trans. Inf. Syst.*, vol. E108.D, no. 11, hlm. 1412–1415, Nov 2025, doi: 10.1587/transinf.2025EDL8005.

[12] O. Nourry, M. Kondo, S. Saito, Y. Iimura, N. Ubayashi, dan Y. Kamei, “Myth: The loss of core developers is a critical issue for OSS communities,” 30 November 2024, *arXiv*: arXiv:2412.00313. doi: 10.48550/arXiv.2412.00313.

[13] A. Bhattacharya, S. N. Duggirala, A. Ramdas, dan Muthukumaran K, “A Novel Framework to Estimate Truck Factor in Software Repositories Using Dynamic Knowledge Decay Model,” dalam *Frontiers in Artificial Intelligence and Applications*, H. Fujita, A. Hernandez-Matamoros, dan Y. Watanobe, Ed., IOS Press, 2025. doi: 10.3233/FAIA250505.

ISSN: 2715-6427

[14] C. Miller, M. Jahanshahi, A. Mockus, B. Vasilescu, dan C. Kastner, "Understanding the Response to Open-Source Dependency Abandonment in the npm Ecosystem," dalam *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*, Ottawa, ON, Canada: IEEE, Apr 2025, hlm. 2355–2367. doi: 10.1109/ICSE55347.2025.00004.
[15] S. A. Piccolo, "Fast and Accurate Heuristics for Bus-Factor Estimation," 13 Agustus 2025, *arXiv*: arXiv:2508.09828. doi: 10.48550/arXiv.2508.09828.
[16] D. Robinson, K. Enns, N. Koulecar, dan M. Sihag, "Two approaches to survival analysis of open source Python projects," dalam *Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension*, Virtual Event: ACM, Mei 2022, hlm. 660–669. doi: 10.1145/3524610.3527871.
[17] L. Yin, M. Chakraborti, Y. Yan, C. Schweik, S. Frey, dan V. Filkov, "Open Source Software Sustainability: Combining Institutional Analysis and Socio-Technical Networks," *Proc. ACM Hum.-Comput. Interact.*, vol. 6, no. CSCW2, hlm. 1–23, Nov 2022, doi: 10.1145/3555129.
[18] J. Jamieson, N. Yamashita, dan E. Foong, "Predicting open source contributor turnover from value-related discussions: An analysis of GitHub issues," dalam *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, Lisbon Portugal: ACM, Feb 2024, hlm. 1–13. doi: 10.1145/3597503.3623340.
[19] W. Xiao, H. He, W. Xu, Y. Zhang, dan M. Zhou, "How Early Participation Determines Long-Term Sustained Activity in GitHub Projects?," dalam *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, San Francisco CA USA: ACM, Nov 2023, hlm. 29–41. doi: 10.1145/3611643.3616349.
[20] N. I. Khan dan V. Filkov, "From Models to Practice: Enhancing OSS Project Sustainability with Evidence-Based Advice," dalam *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*, Porto de Galinhas Brazil: ACM, Jul 2024, hlm. 457–461. doi: 10.1145/3663529.3663777.
[21] M. Qin, Y. Zhang, K.-J. Stol, dan H. Liu, "Who Will Stop Contributing to OSS Projects? Predicting Company Turnover Based on Initial Behavior," *Proc. ACM Softw. Eng.*, vol. 2, no. FSE, hlm. 2782–2805, Jun 2025, doi: 10.1145/3729393.
[22] J. Liu, H. Zhang, dan Y. Zou, "Understanding Open Source Contributor Profiles in Popular Machine Learning Libraries," *ACM Trans. Softw. Eng. Methodol.*, vol. 35, no. 3, hlm. 1–59, Mar 2026, doi: 10.1145/3747347.
[23] T. R. Tulili, A. Rastogi, dan A. Capiluppi, "Exploring turnover, retention and growth in an OSS Ecosystem," dalam *Proceedings of the 29th International Conference on Evaluation and Assessment in Software Engineering*, Istanbul Turkiye: ACM, Jun 2025, hlm. 887–897. doi: 10.1145/3756681.3757050.
[24] C. Segundo, M. Oliveira, dan E. Gonçalves, "An Agent-Based Simulation for Task Allocation in Software Development Teams based on the Truck Factor Metric," *ADCAIJ Adv. Distrib. Comput. Artif. Intell. J.*, vol. 14, hlm. e32600, Nov 2025, doi: 10.14201/adcaij.32600.
[25] F. Calefato, M. A. Gerosa, G. Iaffaldano, F. Lanubile, dan I. Steinmacher, "Will you come back to contribute? Investigating the inactivity of OSS core developers in GitHub," *Empir. Softw. Eng.*, vol. 27, no. 3, hlm. 76, Mei 2022, doi: 10.1007/s10664-021-10012-6.
[26] G. Von Krogh, S. Spaeth, dan K. R. Lakhani, "Community, joining, and specialization in open source software innovation: a case study," *Res. Policy*, vol. 32, no. 7, hlm. 1217–1241, Jul 2003, doi: 10.1016/S0048-7333(03)00050-7.
[27] S. A. Piccolo, P. D. Meo, G. Terracina, dan G. Greco, "The Theory and Practice of Computing the Bus-Factor," 8 Maret 2026, *arXiv*: arXiv:2603.07845. doi: 10.48550/arXiv.2603.07845.
[28] S. Lahtinen, "Open source project life cycle and survival," University of Helsinki, 2020.