Distinguishing Software Aging from Stability in Open Source Projects: An Empirical Analysis of Activity Indicators

Tassio Sirqueira[1], **Jessica Faciroli**[2], **Vera Werneck**[1]

[1]Institute of Mathematics and Statistics – Rio de Janeiro State University (UERJ)
Rio de Janeiro – RJ – Brazil

[2]Faculty of Economic Sciences – Rio de Janeiro State University (UERJ)
Rio de Janeiro – RJ – Brazil

{tassio.sirqueira,vera}@ime.uerj.br, jessica.faciroli@uerj.br

Abstract. *Open-source software projects may experience variations in development activity over time. However, a decline in activity does not necessarily indicate software aging, as mature projects may require fewer changes over time. This study aims to distinguish software aging from stable maturity by analyzing activity indicators empirically. We conducted a longitudinal analysis of three open source projects, examining monthly trends in commits, active contributors, issues, and releases. The results show that stable projects exhibit low but consistent activity, whereas aging projects are characterized by abrupt and sustained declines across multiple indicators. The joint analysis of longitudinal activity indicators enables distinguishing between stability and software aging.*

1. Introduction

Open-source software (OSS) projects play an important role in contemporary software ecosystems, supporting critical infrastructures, widely adopted platforms, and long-lived applications across diverse domains. Their openness and collaborative nature have fostered large, heterogeneous communities, enabling rapid innovation and long-term sustainability. At the same time, the public availability of development artifacts makes OSS repositories a valuable source for empirical research, allowing scholars to investigate software evolution, developer behavior, and maintenance practices at scale. By analyzing version control systems, issue trackers, and release histories, researchers can observe how software projects evolve over extended periods, identifying patterns of growth, stabilization, and decline [Mukala 2025].

A recurring challenge in OSS research is assessing project health and sustainability. A wide range of indicators derived from development activity, social interactions, and ecosystem-level characteristics exist to capture different dimensions of project evolution [Mens and Goeminne 2011, Franco-Bedoya et al. 2014, Jansen 2014]. Metrics such as commit frequency, number of active contributors, issue resolution rates, and release regularity are frequently used as proxies for vitality and maintenance effort. In many empirical studies, sustained reductions in these indicators are interpreted as early signs of software aging or abandonment. However, this interpretation is not always straightforward, as it overlooks important contextual and temporal factors.

In practice, many widely deployed OSS projects intentionally evolve at a slower pace. Once a project reaches a stable architecture and a large user base, development

efforts may shift from feature expansion to maintenance, compatibility, and incremental improvements. In such cases, reduced development activity may reflect stability rather than degradation. Studies on breaking changes and long-term maintenance policies show that mature projects often prioritize conservatism over rapid evolution in order to preserve ecosystem stability and user trust [Bogart et al. 2021]. Consequently, low activity levels alone are insufficient to characterize the health or longevity of an OSS project.

Investigations into developer inactivity reveal that temporary disengagement is common among OSS contributors and does not necessarily indicate permanent departure or project decline [Calefato et al. 2022]. From an ecosystem perspective, low activity at the project level may coexist with high overall robustness and continued relevance within a broader software ecosystem [Jansen 2014, Franco-Bedoya et al. 2014]. Furthermore, studies on software aging emphasize that degradation should be understood as a systemic, sustained process characterized by consistent deterioration across multiple dimensions rather than isolated metric fluctuations [Yazvinskyi et al. 2024].

Many empirical analyses still struggle to distinguish software aging from stable maturity using interpretable activity indicators observed over long time horizons. Reliance on individual indicators or short observation windows often leads to oversimplified or misleading conclusions. Comparative studies that explicitly contrast projects with different lifecycle outcomes are particularly rare, limiting our ability to understand how aging manifests in practice and to develop systematic approaches for identifying it [Wu 2024, Juntura 2025].

In this paper, we directly address this core challenge by conducting a longitudinal empirical analysis of three widely used OSS projects: (i) WordPress, (ii) jQuery, and (iii) AngularJS, each representing a distinct evolutionary trajectory. By analyzing monthly activity indicators extracted from GitHub repositories, including commits, active contributors, issue management, and releases, we examine how development activity evolves over multiple years to enable clear distinctions between software aging and stable maturity. Rather than proposing predictive or machine-learning-based approaches, our study emphasizes interpretability by demonstrating how temporal patterns and the joint analysis of simple indicators can illuminate meaningful distinctions among active growth, stable maturity, healthy aging, and terminal aging.

From a conceptual standpoint, an important distinction must be made between software aging and stable maturity. Software aging refers to a process of progressive degradation that affects a project's ability to evolve, be maintained, or attract contributors over time. This process is typically characterized by persistent declines across multiple dimensions, including development activity, community engagement, and release practices. In contrast, stable maturity describes a lifecycle state in which a project remains actively used and maintained while evolving at a deliberately slower pace. In such cases, reduced activity is not a symptom of degradation but rather an outcome of architectural stabilization, conservative maintenance policies, and long-term ecosystem integration. Failing to distinguish between these states may lead to incorrect assessments of project health and misguided conclusions about software sustainability.

Methodologically, this distinction raises important implications for how project health is assessed. While some studies increasingly explore complex predictive models

and advanced analytics, such approaches often reduce interpretability and require extensive data preprocessing and calibration. In contrast, simple activity indicators—such as commits, active contributors, issue management, and releases—are readily available across most OSS repositories [Mukala 2025]. When analyzed longitudinally and in combination, these indicators provide sufficient contextual information to capture sustained trends and structural changes in project evolution. A longitudinal perspective is particularly critical, as it enables the differentiation between temporary fluctuations and systematic decline, thereby supporting more transparent and robust interpretations of software aging and stability.

Contributions. This paper makes three contributions. First, we provide an operational and interpretable distinction between *terminal aging* and *stable maturity* in OSS projects based on multi-indicator longitudinal patterns (commits, contributors, and issue handling), without relying on predictive models. Second, we report a comparative longitudinal case study of three widely used OSS projects with contrasting trajectories (AngularJS, jQuery, and WordPress), illustrating how isolated indicators can be misleading when not interpreted temporally and contextually. Third, we complement the descriptive analysis with statistical validation, quantifying trend differences, stability/variability, and socio-technical coupling in the MA12 series. Formally, for a monthly indicator x_t, the MA12 at time t is defined as $\text{MA12}_t = \frac{1}{12}\sum_{i=0}^{11} x_{t-i}$, which smooths short-term fluctuations while preserving long-term evolutionary patterns.

The paper is organized as follows. Section 2 presents the background and related work on software aging, OSS evolution, and activity-based indicators. Section 3 describes the methodology, including project selection, data collection, and analysis procedures. Section 4 reports and discusses the empirical results, comparing the observed activity patterns across the analyzed projects and interpreting them in light of software aging and stability. Finally, Section 5 concludes the paper and outlines directions for future work.

2. Background and Related Work

The study of software evolution has long recognized that software systems are dynamic artifacts that continuously change over time. Lehman's laws of software evolution establish that systems operating in real-world environments must evolve or risk becoming progressively less useful [Lehman 1996]. More recent work has reframed software evolution through data-driven and fact-based perspectives, emphasizing that changes across versions should be treated as first-class entities for analysis. Wu [Wu 2024] formalizes this view by proposing a fact-based approach in which software evolution is analyzed through differential facts, enabling systematic reasoning about long-term change and compatibility.

In open-source software (OSS), evolutionary processes are particularly observable due to the public availability of development artifacts. Repository data exposes not only technical changes but also social and organizational dynamics. Mens and Goeminne [Mens and Goeminne 2011] analyze the evolution of social aspects in OSS ecosystems, showing that contributor participation and collaboration structures evolve alongside the codebase. Complementarily, Jansen [Jansen 2014] argues that assessing the health of OSS requires moving beyond isolated project metrics, as ecosystem-level interactions play a central role in long-term sustainability.

2.1. Software Aging and Project Health

Software aging has been investigated as a phenomenon associated with declining vitality, maintainability, or responsiveness of software systems. Yazvinskyi et al. [Yazvinskyi et al. 2024] provide an explicit empirical analysis of software aging indicators in the OpenStack ecosystem, demonstrating that sustained declines in commits, active contributors, and issue-handling activity are associated with aging-related risks. In this way, software aging manifests through the joint behavior of multiple indicators rather than through single metrics observed in isolation.

At the same time, interpreting activity decline remains challenging. Jansen [Jansen 2014] shows that low development activity does not necessarily indicate poor health, particularly in mature ecosystems where stability and reliability are prioritized. Franco-Bedoya et al. [Franco-Bedoya et al. 2014] reinforce this perspective by proposing a quality model for OSS ecosystems that integrates technical and social dimensions, suggesting that sustainability cannot be inferred solely from development intensity.

2.2. Mining Software Repositories and Activity Indicators

Mining software repositories has become a standard methodological approach for studying OSS evolution. Mukala [Mukala 2025] surveys data analytics techniques applied to open-source repositories and identifies commits, issue tracking data, contributor activity, and release histories as the most commonly used indicators of development dynamics. These indicators are widely adopted due to their availability, consistency across projects, and suitability for longitudinal analysis.

However, the relationship between activity metrics and project health is not straightforward. Calefato et al. [Calefato et al. 2022] investigate the inactivity of OSS core developers on GitHub and show that disengagement often stems from personal, organizational, or contextual factors rather than project failure. This challenges interpretations that equate declining contributor activity with abandonment or decay, underscoring the need for contextual and temporal analysis when using repository-derived indicators.

2.3. Stability, Maturity, and Misinterpretation of Decline

The distinction between software aging and stable maturity has been discussed in studies focusing on governance and change management. Bogart et al. [Bogart et al. 2021] analyze policies and practices related to breaking changes in open-source ecosystems and show that mature projects often adopt conservative evolution strategies. These strategies reduce visible development activity while preserving backward compatibility and user trust, indicating stability rather than degradation.

From a broader socio-technical perspective, Juntura [Juntura 2025] examines the evolution of openness in OSS and large-language-model ecosystems, arguing that participation and openness may decline over time due to legal, economic, and organizational pressures. Importantly, this decline does not necessarily imply technical degradation, but reflects shifts in sustainability strategies and power structures within ecosystems.

2.4. Positioning of This Study

Existing research indicates that activity-based indicators must be interpreted within their temporal and contextual settings. Sustained and abrupt declines across multiple indicators

have been associated with software aging [Yazvinskyi et al. 2024], whereas mature and stable projects may exhibit prolonged periods of low but consistent activity [Jansen 2014, Bogart et al. 2021]. Despite this recognition, there remains a lack of empirical studies that explicitly contrast software aging and stable maturity using simple, interpretable, and longitudinal indicators across comparable OSS projects.

This study addresses this gap by conducting a comparative longitudinal analysis of widely used OSS projects with distinct lifecycle trajectories. Grounded in a fact-based perspective on software evolution [Wu 2024], the analysis jointly examines multiple activity indicators over time, avoiding reliance on complex predictive models. By empirically distinguishing patterns of stability from terminal aging, the study contributes to a more nuanced understanding of OSS evolution and supports more informed interpretations of project health.

3. Methodology

This study adopts an empirical, comparative methodology grounded in publicly available data from OSS repositories. The design follows a fact-based and longitudinal perspective on software evolution [Wu 2024], using observable activity indicators to distinguish software aging from stable maturity. The workflow is summarized in Figure 1.

We conduct a multiple-case study with three widely used OSS projects selected for relevance, longevity, and contrasting lifecycle trajectories: (i) WordPress, (ii) jQuery, and (iii) AngularJS. This selection enables direct comparison between stable maturity and terminal decline patterns, consistent with ecosystem-oriented perspectives on OSS health and evolution [Jansen 2014, Mens and Goeminne 2011].

Data were collected from GitHub repositories through automated scripts and publicly accessible APIs, following established repository-mining practices [Mukala 2025]. We extracted monthly aggregated metrics covering the full available history: commits, active contributors, issues opened, issues closed, and releases. To reduce short-term noise and emphasize structural trends, we computed a 12-month moving average (MA12) for each indicator, consistent with longitudinal analyses of aging indicators [Yazvinskyi et al. 2024]. Indicators were interpreted jointly and over time, since single metrics can be misleading when detached from temporal context [Bogart et al. 2021, Jansen 2014].

3.1. Lifecycle State Operationalization

We operationalize *terminal aging* as a sustained collapse of activity across multiple indicators, consistent with the notion of irreversible decline in evolutionary capacity discussed in Lehman's laws of software evolution [Lehman 1996]. Concretely, terminal aging is characterized by MA12 trends converging to near-zero levels for an extended period (at least 12 consecutive months) in both commits and active contributors, combined with a marked reduction in issue activity.

We interpret *stable maturity* as a lifecycle state in which development activity is low to moderate but persistent over time. In this case, MA12 commits, and contributors remain consistently non-zero, and issue dynamics stay bounded, reflecting maintenance-oriented development under conservative evolution strategies, rather than degradation.

3.2. Research Questions and Statistical Analysis

To complement the descriptive time-series inspection, we address three research questions using simple and interpretable statistical analyses applied to MA12 indicators. We validate the distinction using (i) cross-case contrast, (ii) multi-indicator consistency, and (iii) statistical tests on the MA12 series (RQ1–RQ3).

RQ1. *Do projects exhibit statistically different long-term trends in activity over time?* To answer this question, we fit linear models with a project–time interaction using the MA12 series as dependent variables (commits and active contributors). Statistically significant interaction terms indicate that projects have different slopes (i.e., different long-term trends).

RQ2. *Is stability (as opposed to aging) associated with lower variability in longitudinal activity?* We operationalize stability through the coefficient of variation (CV) computed over the MA12 series (standard deviation divided by mean). Dispersion differences by lifecycle state are assessed using Levene's test.

RQ3. *How strongly are technical activity and community participation coupled over time?* We measure socio-technical coupling via Spearman correlation between commits (MA12) and active contributors (MA12) per project, reporting 95% confidence intervals and significance tests.

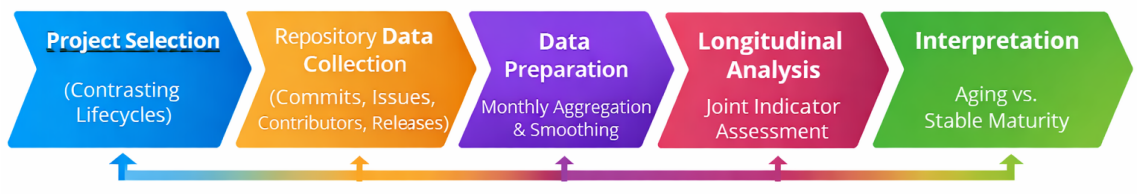

Figure 1. Methodological process adopted to distinguish software aging from stable maturity through longitudinal analysis of activity indicators.

3.3. Replication Package

To support replicability, we make available a replication package containing: (i) the collected datasets (monthly aggregated indicators) for all analyzed projects, (ii) the Python scripts used to collect and preprocess repository data, and (iii) the R scripts used to generate all figures, tables, and statistical analyses reported in this paper. The replication package is publicly available at `https://anonymous.4open.science/w/Research-Data-2F03/CIbSE2026/`.

4. Results and Discussion

This section presents and discusses the empirical results obtained from the longitudinal analysis of three large open-source software projects: (i) AngularJS, (ii) jQuery, and (iii) WordPress. The analysis is based on monthly activity metrics, smoothed with a 12-month moving average (MA12) to capture long-term trends and mitigate short-term fluctuations.

4.1. Commit Activity Dynamics

Commit activity, measured through MA12, reveals sharply distinct lifecycle trajectories across the three projects, as shown in the Figure 2.

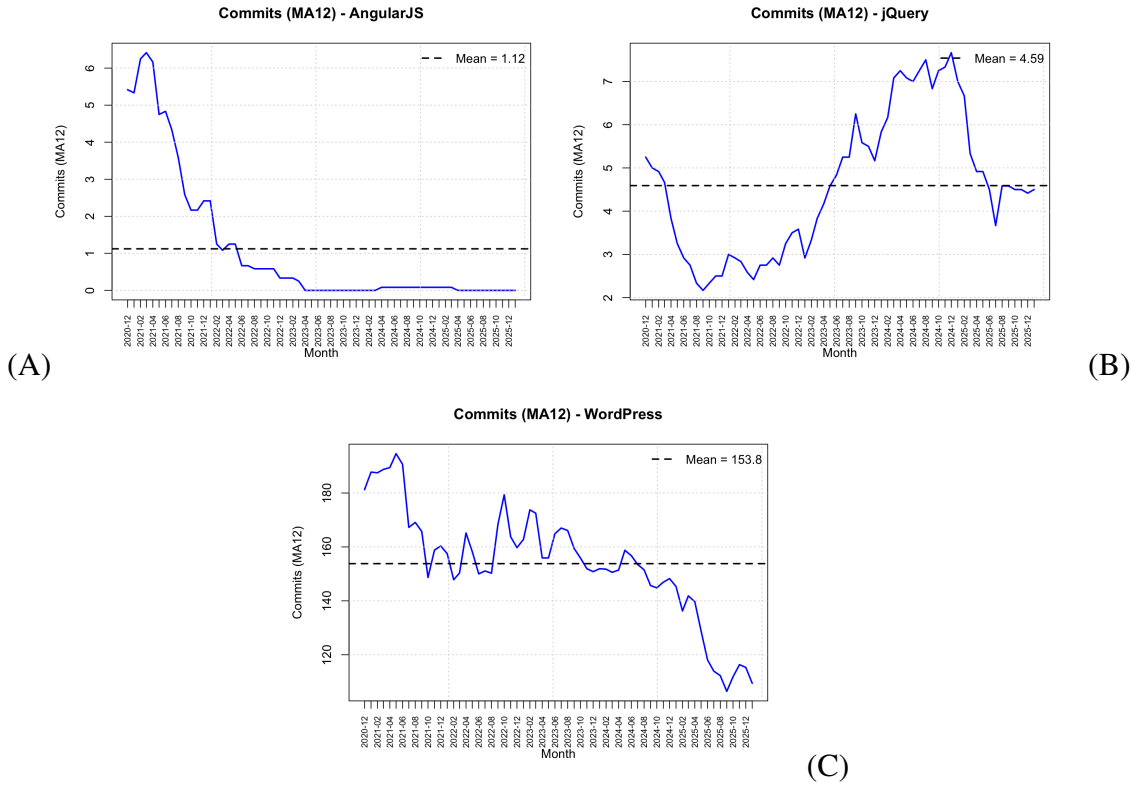

Figure 2. Monthly commit activity (MA12) for AngularJS, jQuery, and WordPress.

AngularJS (Figure 2A) exhibits a rapid and persistent decline in commit activity. After an initial phase of moderate development, the MA12 curve converges toward zero and remains flat for an extended period. This pattern indicates a structural cessation of development rather than a temporary slowdown, characterizing a clear case of terminal aging.

jQuery (Figure 2B) presents a more nuanced trajectory. After an initial decline, commit activity partially recovers and stabilizes around a lower but non-zero mean. This behavior suggests a transition toward stable maturity, in which development continues at a reduced yet persistent pace, typical of mature infrastructure libraries.

WordPress (Figure 2C) maintains the highest absolute level of commit activity throughout the observed period. Although a gradual downward trend is visible in later years, MA12 values remain well above zero, indicating sustained development. This pattern reflects continuous large-scale evolution rather than aging in the strict sense.

4.2. Evolution of Active Contributors

Contributor dynamics reinforce the distinctions observed in commit activity, as shown in the Figure 3.

AngularJS (Figure 3A) shows a sharp collapse in the number of active contributors, converging toward zero and remaining there. The synchronized decline of contributors and commits provides strong evidence of terminal aging, consistent with observations of project abandonment [Mens et al. 2005, Cotroneo et al. 2014].

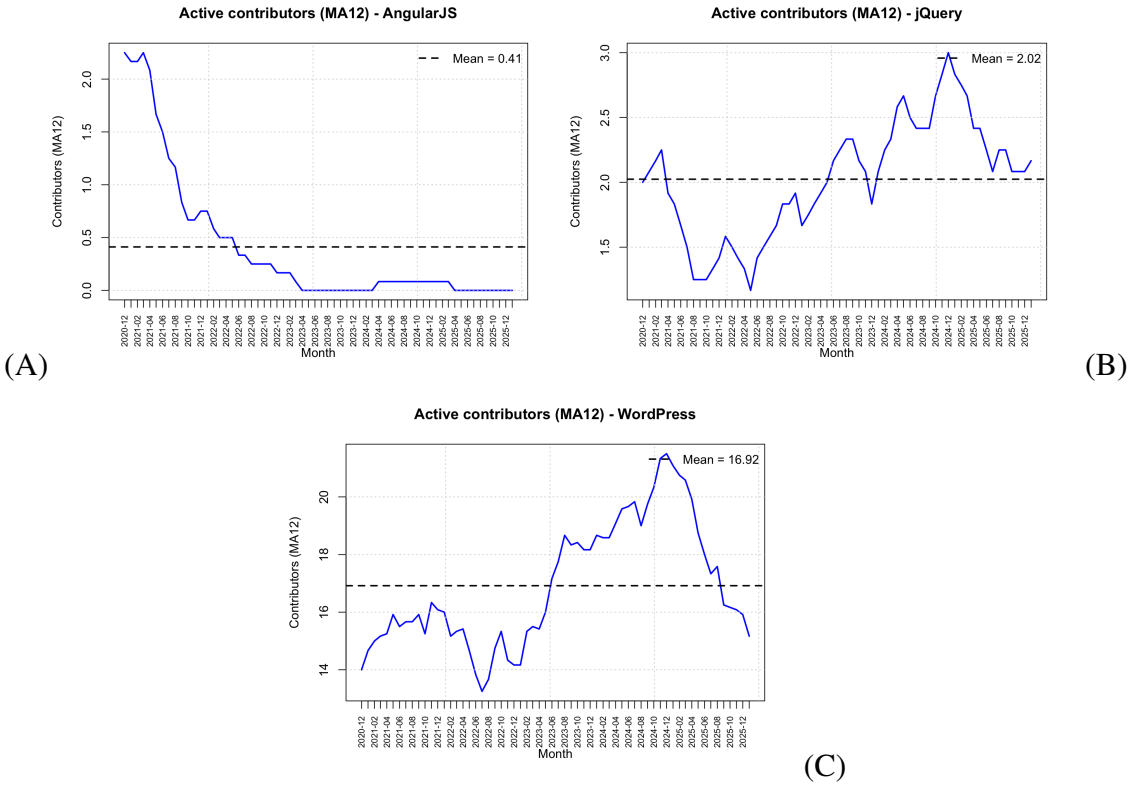

Figure 3. Active contributors (MA12) for AngularJS, jQuery, and WordPress.

jQuery (Figure 3B) displays a reduction followed by stabilization of contributor activity around a small core group. This pattern is characteristic of mature projects that rely on a limited but stable maintenance team, as discussed by Mockus et al. [Mockus et al. 2002].

WordPress (Figure 3C) maintains a substantially larger and more persistent contributor base. Although fluctuations are observed, contributor MA12 remains consistently high, reflecting organizational resilience and sustained community engagement.

4.3. Issue Balance: Opened vs. Closed Issues

Issue balance MA12 (opened minus closed issues) provides insight into backlog dynamics and governance capacity, as shown in the Figure 4

AngularJS (Figure 4A) exhibits early volatility followed by convergence to zero. This convergence does not reflect effective backlog control, but rather the near disappearance of issue activity, consistent with the decline in both development and community participation.

jQuery (Figure 4B) shows oscillations around zero with a small positive mean. This behavior indicates that, over time, the project opens and closes a comparable number of issues, maintaining a bounded backlog and reflecting episodic maintenance cycles.

WordPress (Figure 4C) presents an almost perfectly neutral issue balance throughout the observed period. Here, a near-zero balance reflects highly structured governance and institutionalized maintenance processes, rather than inactivity.

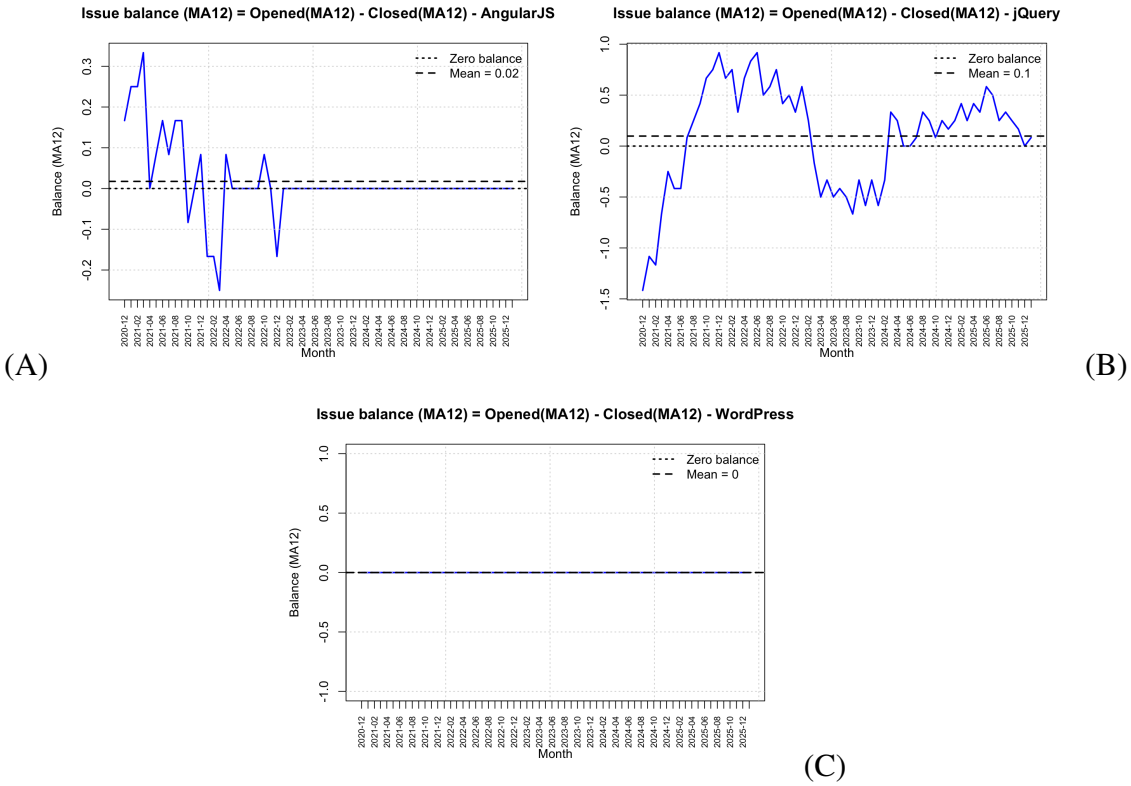

Figure 4. Issue balance (Opened–Closed, MA12) for AngularJS, jQuery, and WordPress.

4.4. Issue Close/Open Ratio

The issue close/open ratio computed using a 12-month moving average (MA12) provides a complementary perspective on software evolution by capturing the long-term balance between demand for maintenance (issues opened) and system responsiveness (issues closed), as shown in the Figure 5.

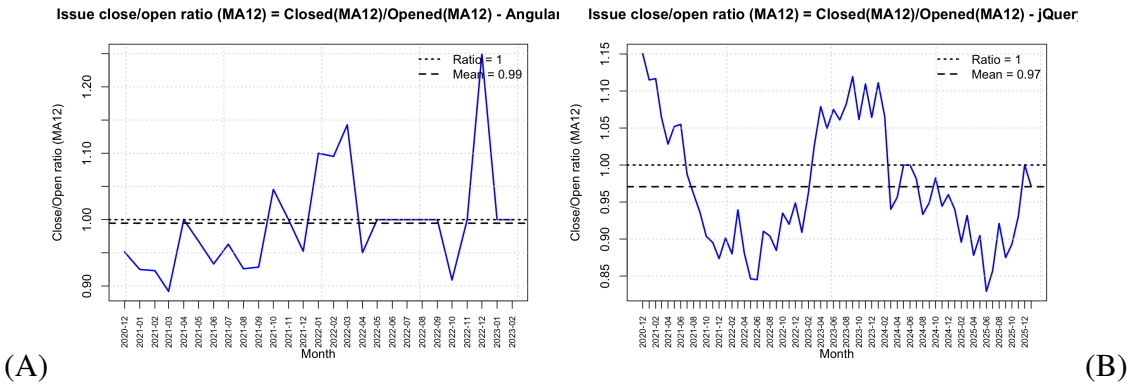

Figure 5. Issue close/open ratio (MA12) for AngularJS and jQuery.

For AngularJS (Figure 5A), the close/open ratio exhibits increasing instability over time, with pronounced oscillations around the equilibrium value of one and occasional spikes above it. As observed in the later periods, this behavior coincides with

a substantial reduction in the number of newly opened issues. In this context, a ratio close to or above one does not reflect improved maintenance effectiveness, but rather a diminishing inflow of new change requests. The unstable ratio, therefore, reinforces the interpretation of AngularJS as a project in terminal decline rather than in stable maintenance.

In contrast, jQuery (Figure 5B) displays a close/open ratio that remains consistently close to one throughout the analyzed period, despite moderate short-term fluctuations. These deviations correspond to transient phases of backlog accumulation or reduction, but the long-term equilibrium indicates that issue resolution closely follows issue creation. The sustained balance observed in jQuery supports its classification as a project in stable maturity, rather than decline.

For WordPress, the close/open ratio could not be computed from the analyzed repository data because issue tracking is largely handled outside GitHub for the selected repository.

4.5. Statistical Results for RQ1–RQ3

This section reports statistical results that complement the longitudinal visual inspection. All analyses are based on the MA12 series, consistent with the methodological goal of capturing structural trends rather than short-term fluctuations.

Project	Slope	SE	p-value
AngularJS	-0.080	0.048	0.097
jQuery	0.053	0.048	0.275
WordPress	-0.973	0.048	$< 10^{-4}$

Table 1. RQ1 (commits): estimated long-term trend (slope) on commits (MA12) per project using a linear model with project–time interaction.

Project	Slope	SE	p-value
AngularJS	-0.028	0.007	$< 10^{-4}$
jQuery	0.017	0.007	0.023
WordPress	0.077	0.007	$< 10^{-4}$

Table 2. RQ1 (contributors): estimated long-term trend (slope) on active contributors (MA12) per project using a linear model with project–time interaction.

RQ1: Long-term trend differences. Table 1 reports the estimated slopes for commits (MA12), while Table 2 reports slopes for active contributors (MA12), obtained from linear models with project–time interaction. The project-time interaction is significant for both commits and contributors ($p < 10^{-4}$), indicating that the projects follow distinct long-term trajectories.

For commits, WordPress shows a strong negative slope ($\beta = -0.973$, $p < 10^{-4}$), indicating a gradual reduction in commit intensity over time, yet remaining at high activity levels as shown in Figure 2. AngularJS presents a negative slope ($\beta = -0.080$) that is

weaker and not significant at $\alpha = 0.05$ ($p = 0.097$), reflecting that most of the decline is concentrated in specific periods rather than a steady linear decrease. jQuery shows a slightly positive but non-significant slope ($\beta = 0.053$, $p = 0.275$), consistent with a mature project whose activity fluctuates but does not collapse.

For contributors, the trajectories are more clearly differentiated. AngularJS exhibits a significant negative slope ($beta = -0.028$, $p < 10^{-4}$), consistent with a loss of community participation over time. jQuery shows a small but significant positive slope ($\beta = 0.017$, $p = 0.023$), indicating sustained contributor presence consistent with stable maturity. WordPress exhibits a significant positive slope in contributors ($\beta = 0.077$, $p < 10^{-4}$), reinforcing its sustained community engagement even as commit intensity decreases. These results are consistent with Lehman's view that software evolution depends not only on code change, but also on continued organizational capacity to sustain change [Lehman 1996].

Project	State	CV (Commits MA12)	CV (Contrib. MA12)
AngularJS	Terminal aging	1.660	1.580
WordPress	Stable maturity	0.136	0.130
jQuery	Stable maturity	0.358	0.224

Table 3. RQ2: stability proxy via coefficient of variation (CV) computed over the MA12 series for commits and contributors.

RQ2: Stability as lower variability. Table 3 reports the coefficient of variation (CV) computed over the MA12 series. AngularJS shows substantially higher variability for both commits and contributors ($\text{CV}_{commits} = 1.66$, $\text{CV}_{contributors} = 1.58$), whereas WordPress presents the lowest variability ($\text{CV}_{commits} = 0.136$, $\text{CV}_{contributors} = 0.130$), and jQuery remains intermediate ($\text{CV}_{commits} = 0.358$, $\text{CV}_{contributors} = 0.224$). Levene's test indicates statistically significant differences in dispersion by lifecycle state for commits and contributors ($p < 10^{-4}$). Since the terminal-aging class contains only one project in this study, these dispersion tests should be interpreted as exploratory; nonetheless, the CV values provide a transparent proxy for stability consistent with the distinction between stable maturity and terminal decline.

Therefore, we mainly interpret RQ2 using per-project CV magnitudes (Table 3) and use Levene's test only as a supporting check rather than for population-level inference.

Project	Spearman ρ	CI low	CI high	p-value
AngularJS	0.998	0.994	1.000	$< 10^{-4}$
jQuery	0.926	0.852	0.963	$< 10^{-4}$
WordPress	-0.422	-0.591	-0.214	0.001

Table 4. RQ3: socio-technical coupling between commits (MA12) and active contributors (MA12) using Spearman correlation with 95% confidence intervals.

RQ3: Socio-technical coupling. Table 4 reports Spearman correlations between commits (MA12) and contributors (MA12), with confidence intervals. AngularJS and jQuery present strong positive coupling ($\rho = 0.998$ and $\rho = 0.926$, respectively; $p < 10^{-4}$), indicating that reductions in code activity are closely aligned with reductions in active contributors. WordPress shows a moderate negative correlation ($rho = -0.422$, $p = 0.001$), suggesting a different governance regime in which contributor presence does not translate linearly into commit intensity (e.g., coordination processes, change batching, or maintenance practices). This supports the broader interpretation that mature ecosystems may exhibit stable participation while technical activity evolves under structured processes [Jansen 2014, Bogart et al. 2021].

4.6. Robustness Check

To mitigate scale effects across projects of substantially different sizes, we performed an additional robustness check using the normalized MA12 series. For each project, indicators were divided by their historical maximum, yielding values in the $[0, 1]$ range.

The normalized trends preserve the qualitative patterns observed in the original analysis. AngularJS still exhibits a sustained convergence toward zero across indicators, jQuery maintains low but persistent activity, and WordPress remains stable at higher relative levels. These results indicate that the distinction between stable maturity and terminal aging is not driven solely by absolute project size, but reflects structural differences in longitudinal evolution.

4.7. Discussion and Threats to Validity

The evolution of commits and active contributors reveals distinct lifecycle trajectories across the analyzed projects. Following the concept of *software aging* by Cotroneo et al. [Cotroneo et al. 2014], sustained reductions in development activity are interpreted as structural signals rather than transient variations.

AngularJS exhibits a pronounced decline in both commits and active contributors, with MA12 values converging to zero in the final observation window. This pattern is consistent with a terminal aging phase, in which development effectively ceases, and the system no longer evolves. Such behavior aligns with Lehman's laws of software evolution, which state that systems that are no longer actively maintained tend to become progressively less adaptable and eventually obsolete [Lehman 1996].

In contrast, jQuery and WordPress show sustained activity throughout the observation period. Although their absolute levels of activity differ substantially, both projects maintain consistent commit flows and contributor participation, suggesting a state of stable maturity rather than decline. This distinction highlights that aging should not be conflated with low activity per se, but rather with persistent downward trends and loss of evolutionary capacity.

The analysis of issue opening and closing activity further supports these interpretations. For AngularJS, both metrics approach zero over time, reinforcing the characterization of a terminal lifecycle state. For jQuery and WordPress, issue activity remains balanced and continuous, indicating ongoing maintenance and governance processes.

To consolidate the longitudinal findings, Table 5 presents aggregated statistics for each project, along with their inferred lifecycle classification.

Project	State	Period	Months	Commits_{mean}	Contrib_{mean}	Open_{mean}	Close_{mean}	Rel_{mean}	Commits_{12m}
AngularJS	Terminal aging	2020–01 to 2026–01	73	1.36	0.53	0.96	0.95	0	0
jQuery	Stable maturity	2020–01 to 2026–01	73	4.74	2.05	5.86	5.90	0.15	54
WordPress	Stable maturity	2020–01 to 2026–01	73	151.29	16.51	0	0	0	1313

Table 5. Summary statistics per project and classification of lifecycle state.

For WordPress, issue statistics in Table 5 reflect the selected GitHub repository only and do not represent the project-wide issue tracking process.

Table 5 reinforces the qualitative observations from the time-series analysis. AngularJS shows negligible activity in the final 12 months, justifying its classification as terminally aging. In contrast, jQuery and WordPress maintain non-zero activity levels and consistent contributor engagement, supporting their classification as stable, mature systems. This distinction echoes prior findings in the literature that emphasize continuity of maintenance, rather than growth alone, as a defining characteristic of sustainable software evolution [Mens et al. 2005].

Threats to validity. We discuss threats to validity along common validity dimensions in empirical software engineering.

Construct validity. Our indicators (commits, active contributors, issues opened/closed, releases) are proxies of development and maintenance activity and do not directly measure user adoption, business relevance, or downstream ecosystem use. In addition, GitHub-based definitions may differ across projects (e.g., bot activity, mirrored repositories, or different governance processes). We mitigate this threat by jointly and longitudinally interpreting indicators rather than relying on single metrics, and by grounding the indicator selection in prior repository-mining work [Mukala 2025, Yazvinskyi et al. 2024]. A specific construct limitation concerns WordPress: issue management for the selected repository is largely conducted outside the GitHub issue tracker, which makes issue-based indicators for WordPress not directly comparable. We explicitly acknowledge this and restrict issue-based interpretations for WordPress to the repository context rather than treating zeros as an absence of maintenance.

Internal validity. The analysis relies on monthly aggregation and MA12 smoothing, which reduces short-term noise but may introduce a lag that delays the visibility of abrupt changes. Therefore, short-lived bursts or rapid transitions may be attenuated in the MA12 series. We mitigate this risk by combining MA12 results with visual inspection of long-term patterns across multiple indicators and by reporting robustness checks (normalized MA12) that preserve the qualitative conclusions.

Statistical conclusion validity. Our statistical analyses are intentionally lightweight and interpretability-oriented. However, time-series observations are temporally correlated, which may inflate statistical significance if treated as independent. We partially mitigate this by using the MA12 series (reducing high-frequency noise) and by triangulating statistical outputs with qualitative trend inspection. In addition, RQ2 compares dispersion by lifecycle state with a small number of cases (only one project classified as terminal aging), so its inferential results should be interpreted as exploratory; the CV values themselves remain useful descriptive stability proxies.

External validity. The study analyzes three projects selected to represent contrasting trajectories, which supports analytical generalization through case contrast but limits statistical generalization to the broader OSS population. Different domains, governance models, or hosting practices may yield different indicator behavior. Future work will expand the project set and incorporate complementary ecosystem-level signals to strengthen generalizability.

Replicability. To facilitate verification and reuse, all datasets and scripts are available in the replication package (Section 3.3).

5. Conclusion and Future Work

This paper investigated how longitudinal activity indicators can be used to distinguish software aging from stable maturity in open-source software projects. Through a comparative empirical analysis of three widely used OSS projects: (i) WordPress, (ii) jQuery, and (iii) AngularJS, we showed that declines in development activity should not be interpreted uniformly as indicators of degradation or abandonment. Instead, stable, mature projects may exhibit low but consistent levels of activity, whereas terminal aging is characterized by abrupt, persistent declines across multiple indicators.

The results provide empirical evidence that simple, interpretable metrics such as commits, active contributors, and issue-handling activity are sufficient to differentiate between healthy stability and terminal aging when analyzed jointly and over time. By relying on longitudinal patterns rather than isolated measurements or complex predictive models, the study reinforces prior arguments that context and temporal consistency are important for meaningful interpretations of project health in OSS ecosystems. The proposed analytical approach is transparent, reproducible, and applicable to a broad range of OSS projects, making it suitable for both research and practical monitoring scenarios.

Future work will extend this study in several directions. First, we plan to expand the analysis to a larger and more diverse set of OSS projects, covering additional application domains and governance models. Second, we intend to incorporate complementary indicators such as dependency evolution, user adoption signals, and broader ecosystem-level measures to further enrich the interpretation of stability and software aging. Finally, although this study deliberately avoids predictive modeling, we intend to investigate how the identified longitudinal patterns can be leveraged to design early-warning mechanisms or decision-support tools for OSS sustainability, while preserving interpretability and analytical transparency.

Acknowledgments

The authors gratefully acknowledge the support of Rio de Janeiro State University (UERJ) and the Carlos Chagas Filho Foundation for Research Support of the State of Rio de Janeiro (FAPERJ), through the Young Scientist Program of Our State (Processo nº E-26/204.525/2025; SEI-260003/020801/2025). This study was conducted within the scope of the EcoData Laboratory (https://www.ecodata.uerj.br/). This study was conducted within the scope of the EcoData Laboratory (https://www.ecodata.uerj.br/). Additionally, ChatGPT and Grammarly were used exclusively for language and grammar review.

References

Bogart, C., Kästner, C., Herbsleb, J., and Thung, F. (2021). When and how to make breaking changes: Policies and practices in 18 open source software ecosystems. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 30(4):1–56.

Calefato, F., Gerosa, M. A., Iaffaldano, G., Lanubile, F., and Steinmacher, I. (2022). Will you come back to contribute? investigating the inactivity of oss core developers in github. *Empirical Software Engineering*, 27(3):76.

Cotroneo, D., Natella, R., Pietrantuono, R., and Russo, S. (2014). A survey of software aging and rejuvenation studies. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 10(1):1–34.

Franco-Bedoya, O., Ameller, D., Costal, D., and Franch, X. (2014). Queso a quality model for open source software ecosystems. In *2014 9th International Conference on Software Engineering and Applications (ICSOFT-EA)*, pages 209–221. IEEE.

Jansen, S. (2014). Measuring the health of open source software ecosystems: Beyond the scope of project health. *Information and Software Technology*, 56(11):1508–1519. Special issue on Software Ecosystems.

Juntura, P. (2025). Degradation of openness? a comparative study on the evolution of open source in software and large language models. Master's thesis, School of Business, Aalto University, Espoo, Finland.

Lehman, M. M. (1996). Laws of software evolution revisited. In Montangero, C., editor, *Software Process Technology*, pages 108–124, Berlin, Heidelberg. Springer Berlin Heidelberg.

Mens, T. and Goeminne, M. (2011). Analysing the evolution of social aspects of open source software ecosystems. In *IWSECO@ ICSOB*, pages 1–14.

Mens, T., Wermelinger, M., Ducasse, S., Demeyer, S., Hirschfeld, R., and Jazayeri, M. (2005). Challenges in software evolution. In *Eighth International Workshop on Principles of Software Evolution (IWPSE'05)*, pages 13–22. IEEE.

Mockus, A., Fielding, R. T., and Herbsleb, J. D. (2002). Two case studies of open source software development: Apache and mozilla. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 11(3):309–346.

Mukala, P. (2025). Exploring data analytics on open-source software repositories: A review of development activities and tools for mining logs. *Available at SSRN 5391607*.

Wu, X. (2024). *A Fact-Based Approach to Software Evolution*. Doctoral thesis, School of Computer Science and Engineering, Nanyang Technological University, Singapore.

Yazvinskyi, Y., Bogatinovski, J., Cardoso, J., and Kao, O. (2024). On software ageing indicators in openstack. arXiv preprint arXiv:2404.16446.