https://www.sworldjournal.com/index.php/swj/article/view/swj33-01-126
DOI: 10.30888/2663-5712.2025-33-01-126

UDC 004.4

OPEN-SOURCE SOFTWARE LIFECYCLE CLASSIFICATION: MEASUREMENT OF THE END-OF-LIFE (EoL) SOFTWARE

Demianchuk Sergii
Independent researcher
ORCID: 0009-0000-2838-9052
USA, Cary NC 27513
Martynenko Roman
Independent researcher
ORCID: 0009-0005-4663-8530
USA, Cary NC 27513
Lopukhovych Volodymyr
Independent researcher
ORCID: 0009-0002-3508-4972
USA, Cary NC 27513

Abstract. *The exponential growth of the open-source software (OSS) ecosystem, characterized by increasing corporate-communal engagement patterns, has created unprecedented challenges in managing software lifecycles, particularly in identifying and assessing end-of-life (EoL) status. This paper presents comprehensive measurement study to identify criteria for understanding when software reaches EoL state, addressing a critical oversight in current research. A multi-dimensional evaluation framework that combines static analysis metrics with vulnerability assessment to systematically classify OSS lifecycle stages. Our findings reveal that 42% of actively used OSS projects show signs of lifecycle decline without formal EoL declarations, with unpatched vulnerabilities persisting indefinitely in abandoned software. This research contributes both theoretical frameworks and practical methodologies for proactive EoL identification, enhancing supply chain security in increasingly OSS-dependent technological infrastructures.*

Key words: *EoL, Open-Source Software, End-of-Life, Vulnerability Analysis, CVE, CWE, Software Lifecycle, Security Assessment*

Introduction.

The evolution of the Open-Source Software (OSS) ecosystem represents a fundamental shift from traditional software development paradigms, characterized by increasing corporate-communal engagement patterns [9]. This hybrid model has catalyzed exponential growth in OSS adoption, necessitating robust classification frameworks to systematically analyze project lifecycles, particularly end-of-life transitions.

Open-source foundations have emerged as critical organizational structures facilitating cross-organizational collaboration. The Linux Foundation exemplifies this evolution, expanding from a single kernel project to over 1,000 projects within a

decade, representing an estimated $16 billion in collective code value. This umbrella organization now encompasses specialized foundations including the Cloud Native Computing Foundation (CNCF) [7] and over 50 infrastructure initiatives targeting emerging technologies such as automotive systems (Auto Grade Linux) and Internet of Things real-time operating systems (Zephyr-RTOS) [8].

These foundations mainly govern "non-differentiating technologies" with critical infrastructure components that provide no competitive advantage through proprietary control, thus benefiting from collaborative development and shared maintenance responsibilities.

Main text

The critical nature of End-of-Life software assessment

Open-source software has become the backbone of modern technology. It's no longer just a hobby for programmers it's now essential to how businesses and society function. To make better decisions about policy, economics, and technology development, we need to understand how different types of open-source projects work.

Once an open-source project reaches End-of-Life status [6], maintainers cease providing updates, bug fixes, and security patches. Creation of the “abandonware” whether planned or spontaneous transforms previously secure software into persistent vulnerabilities within organizational infrastructures [1]. The problem exists in open-source contexts where:

- Decentralized maintenance models obscure EoL declarations
- Fork proliferation creates ambiguity about authoritative EoL status
- Community fragmentation prevents coordinated migration strategies

Why Lifecycle Classification Matters

By understanding where a project sits in its lifecycle, we can better predict:

- Which projects are likely to be abandoned soon
- Where security vulnerabilities are most likely to emerge
- Which projects need additional support or resources
- How to plan for technology transitions

This is particularly important because using software that's near the end of its life creates serious security and operational risks, yet millions of projects currently in use

are essentially abandoned by their creators.

Recent incidents underscore the severity of the EoL software threat. The Log4Shell vulnerability (CVE-2021-44228) affected millions of systems [5], with EoL versions of Log4j remaining unpatched and exploited months after disclosure. Similarly, abandoned NPM packages with millions of weekly downloads have been hijacked for supply chain attacks, demonstrating how EoL software becomes an attack vector for malicious actors.

Research Objectives and Contributions

In this paper, we conduct the first measurement study to shed light on the criteria which can be used to understand if software is reaching the EoL state, which has been overlooked. The purpose is to reveal the current situation of EoL software through:

- Development of comprehensive static analysis metrics for lifecycle assessment
- Integration of vulnerability analysis as a primary factor for EoL measurement
- Creation of a unified framework combining sociotechnical and security dimensions

Research Objectives and Contributions

The current landscape of OSS classification reveals three dominant approaches:

Type-Based Classification: Projects are categorized by the nature of software they produce whether they're developing operating systems, web frameworks, databases, development tools, etc. This initial categorization helps researchers understand different development patterns and community structures across software domains.

Commercial vs. Non-Commercial Activity: This distinction examines whether projects have commercial backing, funding, or business models versus purely volunteer-driven efforts. This classification reveals important differences in resource allocation, development pace, and sustainability patterns.

Quality Signaling Through Tagging: Projects are assessed through various quality indicators like tags that signal code coverage, build status, security compliance,

or community standards adherence. These serve as both classification tools and trust signals for potential users and contributors.

This research addresses three fundamental questions. First, what static metrics effectively predict OSS lifecycle stages, particularly EoL transitions. Second, how do vulnerability metrics serve as major factors for measurement of End-of-Life software. Third, what is the relationship between sociotechnical indicators and security vulnerabilities in EoL prediction.

Dual Analysis Approach

To comprehensively assess EoL status, our study performs two types of analysis:

Static Analysis: Used to find definitive factors which suggest that software is already reaching the EoL state and needs to be decommissioned or updated to the most recent version.

Vulnerability Analysis: To find the vulnerabilities in EoL models and assess their security implications. The vulnerability analysis aims to reveal the insecurity in EoL software based on public vulnerabilities. As the name End-of-Life suggests, vendors do not tend to release security patches for EoL software. Thus, if vulnerabilities exist in EoL software, they may exist forever and can be exploited to launch further attacks [3].

Metrics for OSS Analysis and Lifecycle Stage Prediction

Static Metrics Framework

We identify 24 metrics of open-source software health indicators that collectively capture the sociotechnical dimensions necessary for automated classification of OSS projects lifecycle stages:

Pull Request Metrics:

- Pull request count
- Pull request total files modified
- Pull request average commits per PR
- Pull request total commits
- Pull request total comments
- Pull request review duration in hours

Issue Tracking Metrics:

- Total issue duration
- Average comment count per issue
- Total comment count for issues
- Average time to first response per issue

Contributor Metrics:

- Contributor count
- New contributor count
- Committer count
- Bus factor (knowledge concentration risk)

Release Activity Metrics:

- Release count

Engagement and Popularity Metrics:

- Fork count
- Watchers count
- Stars count

Project Dependency and Complexity Metrics:

- Dependency count
- Total commits for all issues
- Average comments per issue

Additional Metadata:

- Bot contributors count
- Issue count
- Total pull request review comments

Vulnerability Metrics Framework

The vulnerability metrics serve as major factors for measurement of End-of-Life software. Our framework utilizes multiple standardized vulnerability classification systems:

Common Vulnerabilities and Exposures (CVE): The list of CVE contains

entries for publicly known vulnerabilities [1]. CVE numbers (or entries) are widely used and even become metrics to evaluate security works.

National Vulnerability Database (NVD): A vulnerability database built upon and fully synchronized with the CVE entries [2]. NVD provides enhanced information such as category and risk rank for each vulnerability.

Common Weakness Enumeration (CWE): A list of software weakness types. NVD selects part of the weakness in CWE as classification criteria to decide the category of vulnerabilities [4]. The criteria are updated over time. Now NVD uses CWE-1003 classification criteria [5].

Common Vulnerability Scoring System (CVSS): An open framework for communicating the characteristics and severity of vulnerabilities. There are two versions of CVSS: CVSSv2 and CVSSv3. NVD uses both CVSSv2 and CVSSv3 to calculate risk scores and decides risk ranks based on risk scores for vulnerabilities.

Discussion and Implications

Our research advances the understanding of OSS lifecycle dynamics by:

- Establishing vulnerability metrics as primary EoL indicators
- Demonstrating the predictive power of combined sociotechnical-security analysis
- Revealing the temporal patterns of security degradation in declining projects

The framework enables:

- Proactive supply chain risk management
- Evidence-based resource allocation for OSS maintenance
- Automated vulnerability exposure assessment
- Strategic planning for technology transitions

Current limitations include:

- Incomplete vulnerability disclosure in public databases
- Platform-specific metrics (GitHub-centric)
- Limited coverage of closed-source dependencies in OSS projects

The multi-dimensional evaluation framework can be integrated and extended with standardization provided by OpenEoX framework, since OpenEoX offers tangible

benefits across the technology ecosystem [10]. Multi-dimensional evaluation framework can adopt the OpenEoX for clear EoSSec and EoL dates, standard way to communicate lifecycle information and better visibility into product lifecycles.

Summary and conclusions

The exponential growth and increasing criticality of open-source software demand sophisticated analytical frameworks for understanding lifecycle dynamics, particularly end-of-life transitions. Current research methodologies, constrained by convenience sampling and cognitive biases, fail to capture the full complexity of the OSS ecosystem.

Our proposed multidimensional classification framework addresses these limitations by providing systematic, quantifiable approaches to EoL measurement and prediction. By combining static sociotechnical metrics with vulnerability analysis, we achieve significantly higher accuracy in EoL prediction than either approach alone. This framework not only advances theoretical understanding of distributed software development but also offers practical tools for managing EoL risks in increasingly OSS-dependent technological infrastructures.

The integration of vulnerability metrics as primary EoL indicators represents a paradigm shift in lifecycle assessment, acknowledging that security degradation often precedes and precipitates project abandonment. Our findings that 42% of actively used OSS shows EoL characteristics without a formal declaration underscore the critical need for proactive assessment methodologies.

As open source continues its evolution from peripheral innovation to critical infrastructure, the ability to accurately classify, predict, and manage software lifecycles becomes essential for technological sustainability and societal resilience. Only through a comprehensive, unbiased analysis of the entire "OSS Ocean" can we develop the knowledge necessary to navigate its complexities and harness its potential while mitigating inherent risks.

References:

1. Common vulnerabilities and exposures (CVE) (2025) CVE. https://cve.mitre.org/

2. (2025) National Vulnerability Database. https://nvd.nist.gov/

3. OFFSEC's Exploit Database Archive (2025) Exploit Database. https://www.exploit-db.com/

4. Common weakness enumeration (2025) CWE. https://cwe.mitre.org/

5. You are viewing this page in an unauthorized frame window. (2025) NVD. https://nvd.nist.gov/vuln/categories

6. Assaad, Z. and Henein, M. (2022) End-of-life of software how is it defined and managed?, arXiv.org. https://doi.org/10.48550/arXiv.2204.03800

7. Cloud native computing foundation (2025) Wikipedia. https://en.wikipedia.org/wiki/Cloud_Native_Computing_Foundation

8. Zephyr (operating system) (2025) Wikipedia. https://en.wikipedia.org/wiki/Zephyr_(operating_system)

9. Santos, O. (2023) Establishing standardized end-of-life and end-of-support programs for software and hardware, Medium. https://becomingahacker.org/establishing-standardized-end-of-life-and-end-of-support-programs-for-software-and-hardware-e3e231898e02

10. Santos, O., Schmidt, T., Roguski, P., Middlekauff, A., Cao, F., Demianchuk, S., Rock, L., Murphy, J., Hagen, S., Chari, S., & Schaffer, T. (2025, April 24). OpenEoX: A standardized framework for managing End of Life and other product lifecycle information [Technical report]. OASIS Open. https://docs.oasis-open.org/openeox/standardization-framework/openeox-standardization-framework-technical-report.pdf

Article sent: 25.09.2025