

Balancing Quality Assurance and Scalability in Modern Open-Source Software: A Review

Jajimoggala Snehitha¹, Manas Kumar Yogi^{2,*}

Abstract

The development of modern open-source software has increased rapidly and now is being used as critical infrastructure across the world. As the software expands, maintaining high quality while supporting large scale has become a major challenge and tension globally. Traditional quality assurance methods are no longer sufficient when open-source software (OSS) projects are handled with millions of users, developers, and downloads. This review discusses four main issues: sustainability of package registers when used excessively; security challenges in software supply chains and software bill of materials (SBOM) adoption; the impact of AI-based tools on software quality; and human factors such as maintainer burnout. This study shows that the increasing scalability tends to create new quality risks. Due to the rapid rise in OSS, the process of its development, sharing, and maintenance has undergone a significant transformation. This is the greatest challenge as the company needs to strike the proper balance between quality assurance and expansion. Testing, security checks, and review of code are all still largely a human burden and community support.

Keywords: Open-source software (OSS), AI-assisted software development, technical debt, code quality, maintainer burnout, software bill of materials (SBOM)

INTRODUCTION

Over the past decade, open-source software (OSS) has changed significantly. Earlier, it was mainly developed by small communities and used by limited technical users. However, today, OSS has become a core part of the global digital economy [1]. All present-day applications rely on open-source components for development and functionality. Research indicates that approximately 97% of AI models use open-source code in one way or another. OSS is widely used worldwide, with popular package registries making trillions of downloads annually [2]. The development, sharing, and maintenance of OSS has evolved in many ways owing to the rapid increase in the growth of OSS.

The current biggest challenge is finding a compromise between maintaining high-quality and scalable

software. Quality is also achieved through factors such as testing, security, and reviewing code, which are highly dependent on human intervention and community support [3]. Scalability requires automation, strong systems, and more resources to deal with increasing users and usage. Most open-source projects are volunteer-based, which means they do not have sufficient resources to deal with fast development. As a result, security risks can spread easily, maintainers may face burnout, and the infrastructure may not be able to sustain high demand.

This review explains the new challenges that arise when attempting to balance quality assurance and scalability in modern OSS. As these projects

*Author For Correspondence

Manas Kumar Yogi
E-mail: manas.yogi@gmail.com

¹Student, Department of Computer Science and Engineering, Pragati Engineering College (A), Surampalem, Andhra Pradesh, India

²Assistant Professor, Department of Computer Science and Engineering, Pragati Engineering College (A), Surampalem, Andhra Pradesh, India

Received Date: March 09, 2026

Accepted Date: April 18, 2026

Published Date: April 30, 2026

Citation: Jajimoggala Snehitha, Manas Kumar Yogi. Balancing Quality Assurance and Scalability in Modern Open-Source Software: A Review. Journal of Open Source Developments. 2026; 13(1): 1–7p.

grow rapidly, several problems have started to occur. First, package registries, such as npm and PyPI, handle a large number of downloads caused by automation and AI-generated code, which makes it difficult for them [4]. This raises concerns regarding long-term sustainability. Second, supply chain security has become more important because a small mistake can affect many systems, mainly after incidents such as Log4Shell and rules such as the European Union Cyber Resilience Act. Third, AI tools help developers write code faster, but they can also create problems in the long term. Open-source project maintainers usually experience stress and workload. With the increase in such projects, it becomes even harder to maintain the software at a high standard, secure it, and make the community active [5].

This review is a compilation of various aspects of OSS to provide a clear picture of the present state of affairs. The combined consideration of all these areas assists students, researchers, and developers in realizing the primary issues that current open-source projects are grappling with [6]. It also highlights possible ideas and directions that can help improve these systems in the future.

INFRASTRUCTURE SUSTAINABILITY AND THE TRAGEDY OF THE COMMONS

The Scale Crisis in Package Registries

Open-source package registries have grown at an extremely fast rate because developers worldwide use them to download software libraries. Maven Central, PyPI, npm, and NuGet support trillions of downloads every year. As this depicts the large popularity of OSS, it also gives us something to fear: Most downloads are not necessarily due to new developments or innovations [7]. Rather, they occur because modern software is dependent on a significantly large number of other packages, automated systems repeatedly load libraries, and in some cases, build systems are not configured correctly. Therefore, the servers that host such registries experience extremely high traffic; therefore, it is harder to manage and maintain such registries [8].

Maven Central traffic analysis reveals that a few systems are responsible for most downloads. In fact, 82 percent of the usage is served by less than 1 percent of the IP addresses, and a majority of the traffic is associated with a number of larger cloud providers [9]. This implies that large businesses are highly dependent on open-source infrastructure to operate their services. However, the expense of sustaining such systems, such as bandwidth, storage, and security, is normally managed by foundations and volunteers instead of the companies that utilize them the most. Because most organizations think that these resources are free and not limited, automated development systems end up downloading packages repeatedly, resulting in a needless load. This is commonly called a tragedy of the commons, whereby the resources shared are exploited to an extent that ensures that the infrastructure is difficult to support in the long term.

Synthetic Growth Versus Organic Innovation

According to researchers, there are two types of open-source software development: organic and synthetic growth. Organic growth occurs when developers genuinely develop new software and download packages to create applications. In contrast, synthetic growth occurs because automated systems, such as CI/CD pipelines and machine learning processes, continue to download the same dependencies repeatedly. According to reports released by Sonatype, many new development systems are faster and more convenient, resulting in frequent and repeated downloads [10]. As a result, registries such as Maven Central experience extremely high traffic. These platforms are designed to store and share software packages and not to support such volumes of automated traffic, which places additional stress on their infrastructure.

A good example of how major open-source platforms may be challenged by severe security issues is the node package manager (NPM) ecosystem. There are approximately 500,000 releases of npm with critical vulnerabilities, which is far too many to examine manually [11]. Unethical users also posted approximately 171,740 malicious packages to propagate malicious code. The popularity of npm among developers across different parts of the world makes it a perfect target for hackers.

Toward Sustainable Infrastructure Models

The increasing pressure of open-source registries is driving the exploration of new solutions that will enable such systems to be more sustainable. Tiered access models are being tested on platforms such as npm and Maven Central. Under this model, free users can still consume the services, while large companies with a high download volume may have to contribute money. This was also raised in an open letter by the Open-Source Security Foundation in late 2025, which stated that it is not economically viable in the long term to have large-scale commercial use of open-source infrastructure without having the necessary support [12]. The technical community has highlighted the importance of caching and using local repository managers to reduce pressure on public registries like Maven Central and npm. Instead of downloading the same packages every time a project is built, organizations can store them locally and reuse them. Companies that skip caching and repeatedly download from public registries create unnecessary traffic and increase the strain on open-source infrastructure.

SUPPLY CHAIN SECURITY AND THE SOFTWARE BILL OF MATERIALS

The Software Bill of Materials Imperative

Events such as the 2020 SolarWinds attack or the Log4Shell vulnerability demonstrated the risk of software dependency bugs. A single defect in a single system may easily spread to thousands of other related systems in a short time. This forced governments to act, and measures such as the U.S. Executive Order 14028 and the EU Cyber Resilience Act were introduced to ensure that software components are better tracked and managed in a safer manner [13].

A software bill of materials (SBOM) is a comprehensive list of all elements, libraries, and dependencies used to establish a software program. It assists organizations in knowing what their software contains and makes it easier to identify vulnerable components and resolve security problems in a short period [14]. As modern applications rely on many external libraries, SBOMs can be useful for enhancing transparency and security. However, they have not been adopted uniformly. Some of the challenges are the lack of common standards for SBOM formats, difficulties in creating SBOM formats automatically, and challenges in integrating SBOM analysis within normal software development processes.

Regulatory Pressures and Open-Source Stewards

The European Union Cyber Resilience Act introduces new responsibilities for organizations, known as OSS stewards. These groups support and maintain open-source projects but do not directly earn money from them. Under this law, such organizations are expected to create security policies, manage how vulnerabilities are reported and fixed, and cooperate with regulators when security issues occur. This is a major change from the traditional open-source model, in which maintainers usually provide software without any guarantees and are not held responsible for security problems.

Some projects under the Linux Foundation, such as the Civil Infrastructure Platform, Yocto Project, and Zephyr Project, have developed methods to improve security and meet compliance requirements. These include creating long-term security plans, using semantic versioning to clearly show software updates, and applying tools such as the OpenSSF Scorecard to identify possible security issues.

Tooling for Supply Chain Security

The Open-Source Security Foundation aims to improve the security of OSS by creating helpful tools and standards. The OpenSSF Scorecard is a major instrument employed to test whether open-source projects practice good security measures, including managing dependencies appropriately, reporting vulnerabilities, and closely reviewing code. Standards such as SPDX 3.0 and the Open Chain framework assist open-source license rules to ensure that organizations produce SBOMs and ensure that they comply with open-source license rules [15]. Overall, these tools and frameworks assist in enhancing transparency and increasing the safety of software supply chains.

A popular study reveals that there are still several challenges in evaluating the quality of OSS. Their research found that different tools use different quality models, and there is no common standard for

evaluation. In addition, many tools are difficult for beginners to use. Therefore, it can be difficult for developers and organizations to properly assess the quality of OSS.

AI-DRIVEN DEVELOPMENT AND TECHNICAL DEBT ACCUMULATION

The Productivity Promise of AI Assistants

Large language models (LLMs) are computer systems that can understand and write computer code. They can perform tasks such as writing code, debugging, and documentation, often working side by side with human programmers. GitHub Copilot and Codex are tools that assist developers in operating with greater speed by proposing and writing code. The Stack Overflow Developer Survey 2024 indicated that approximately 76% of developers already use these AI tools or intend to use them in the future, demonstrating that they are becoming essential software development tools [15].

Although large language models are useful for writing code, researchers have questioned the quality of the code produced by LLMs. Research on early systems such as OpenAI Codex discovered that AI-generated code can be vulnerable to security attacks in some cases, malfunctions, and unwritten code (code smells). It may also end up being complicated when it does not have to be so. For this reason, researchers doubt that the value of AI coding tools will be the same in more sophisticated AI systems.

Longitudinal Evidence from Cursor Adoption

A major study analyzed 806 open-source repositories that adopted Cursor to understand the impact of AI coding assistant usage. The researchers compared the usage of the Cursor in the project and similar projects that did not use the Cursor using the difference-in-differences method. They would be able to quantify the real implications of AI tools for the development of OSS by observing how these projects evolved over the years.

The researchers discovered that projects recorded significant growth in productivity in the first month after the introduction of Cursor, where three to five times the volume of code was added. However, this growth was temporary and began to fade after two months. Meanwhile, there was a 30+ percent increase in code quality issues, and it also experienced a 41 percent growth in code complexity that brought about an increase in technical debt. Researchers discovered through the Panel Generalized Method of Moments that such technical debt in the future slows down development; that is, the immediate gains of AI tools may create maintenance issues in the long term [16].

Implications for Quality Assurance

The results have implications for the quality assurance of AI-assisted software development. Sophisticated AI tools, known as Agentic AI Systems, can produce a large amount of code and modify multiple files simultaneously. Therefore, human reviewers find it more difficult to check each line of code as they usually do with human-written code. Consequently, certain mistakes or quality problems in AI-written code might go undetected, posing a problem for software quality maintenance [17].

According to researchers such as He et al., quality assurance should be regarded as an important aspect of AI-powered development tools [5]. They proposed that AI tools are not only used to produce code, but also to verify its quality, propose tests, and indicate potential technical debt. The lack of such features can lead to short-term speedy organizations and long-term maintenance and software quality issues.

HUMAN FACTORS: GOVERNANCE, BURNOUT, AND STEWARDSHIP

The Maintainer Sustainability Crisis

Linux studies have revealed that most successful open-source projects are funded by foundations or companies. Organizations also assist in other key activities, such as dealing with bugs, user queries, legal matters, and keeping the project on track. Without this institutional backing, projects are likely to experience difficulties in coping with development and project governance as they expand.

There are a large number of open-source projects that rely on a few maintainers, many of whom are now becoming burnt out. Although open-source licenses do not compel the maintainers to maintain or continue updating the software as many companies and users want them to, it is expected that they will continue to maintain and keep the project active. This strain builds impractical expectations that may result in motivational loss and withdrawal of maintainers, thus slowing down projects or project stagnation.

Community Health Metrics and Governance Evolution

The CHAOSS (Community Health Analytics Open Source Software) project offers indicators that are used to gauge the health of an open-source community. It assesses projects based on numerous indicators in various fields. Responsiveness to problems within 24 hours, more than 80% pull request merge rate, over 60% of first-time contributors who remain active, and from over countries are some of the key metrics [17]. These metrics assist in identifying the activity, management, and sustainability of an open-source project.

With the growing number of open-source projects, more appropriate governance is required to regulate contributors and decisions. The Apache Software Foundation adheres to the meritocracy system, according to which a contributor can be promoted to become a committer and finally a project management committee member owing to their contributions. Similarly, the Python community in 2018 reformed its leadership structure and no longer has a single leader but instead a Steering Council, indicating that the governance structure can be transformed without destroying the culture of the projects.

Commercial Stewardship as an Emerging Model

The end-of-life announcement of Ingress NGINX in 2026 showed that even widely used open-source projects may not be maintained indefinitely. This has made many organizations realize the risks of relying only on community support. This has led to a situation where most companies are resorting to commercial steward vendors that offer patches, security updates, and long-term maintenance to maintain stability and minimize the chances of finding themselves with unsupported software [18].

This is because OSS is like a puppy, as described by an analyst. This implies that it may not be charged initially; however, it takes time, effort, and resources to maintain, secure, and update. Organizations may also need to eventually replace it if the project becomes unsupported. This idea highlights that the real challenge is not the license cost but the total cost of ownership over the software's lifecycle.

CONCLUSION

OSS has evolved significantly. Initially, the majority of OSS projects were small initiatives predominantly run by volunteer communities. However, today, quite a number of these projects have taken a very significant role and serve as critical infrastructure for companies and organizations worldwide. However, the support infrastructure, including funding, technical support, and human resources, has not expanded at an equal pace. This has made quality control and the development of open-source projects a significant issue, with pressure on open-source infrastructure, such as package registries, being one of them. These packages are commonly utilized by many companies using automated tools such as CI/CD pipelines and AI development tools; however, they do not provide sufficient resources to support them. Another major concern is software supply chain security. The hacks of SolarWinds and Log4Shell showed that a single vulnerability in widespread software could affect thousands of companies. Owing to these risks, governments and companies are paying more attention to transparency and security practices, including storing more detailed information about software components.

Simultaneously, new technologies, such as AI-based coding solutions, have created new opportunities and issues. The tools can build up the speed of development in the short term but can also contribute to more technical debts and lower code quality when not used cautiously. Improper

governance and maintainer burnout are among the issues in most open-source projects. To solve these problems, the open-source ecosystem needs to be better funded, there should be better quality assurance, better governance, and more responsible usage of emerging technologies.

REFERENCES

1. Adewumi A, Misra S, Omoregbe N. A review of models for evaluating quality in open source software. *IERI Procedia*. 2013;4:88–92. doi:10.1016/j.ieri.2013.11.014.
2. Avgeriou P, Kruchten P, Ozkaya I, Seaman C. Managing technical debt in software engineering (Dagstuhl Seminar 16162). *Dagstuhl Rep*. 2016;6(4):110–138. doi:10.4230/DagRep.6.4.110.
3. Decan A, Mens T, Grosjean P. An empirical comparison of dependency network evolution in seven software packaging ecosystems. *Empir Softw Eng*. 2019;24(1):381–416. doi:10.1007/s10664-017-9589-y.
4. Goggins S, Lombard K, Germonprez M. Open source community health: analytical metrics and their corresponding narratives. 2021 IEEE/ACM 4th International Workshop on Software Health in Projects, Ecosystems and Communities (SoHeal), Madrid, Spain. 2021. p. 25–33. doi:10.1109/SoHeal52568.2021.00010.
5. He H, Miller C, Agarwal S, Kästner C, Vasilescu B. Speed at the cost of quality: how Cursor AI increases short-term velocity and long-term complexity in open-source projects [preprint]. 2025. arXiv:2511.04427. doi:10.48550/arXiv.2511.04427.
6. Garcia D, Mirakhorli MT, Dillon S, Laporte K, Morrison M, Lu H, et al. A landscape study of open-source tools for software bill of materials (SBOM) and supply chain security. 2025 IEEE/ACM 3rd International Workshop on Software Vulnerability Management (SVM), Ottawa, ON, Canada, 2025. p. 37–45. doi:10.1109/SVM66695.2025.00010.
7. Kapitsaki GM, Kramer F, Tselikas ND. Automating the license compatibility process in open source software with SPDX. *J Syst Softw*. 2017;131:386–401. doi:10.1016/j.jss.2016.06.064.
8. Lerner J, Tirole J. Some simple economics of open source. *J Ind Econ*. 2002;50(2):197–234. doi:10.1111/1467-6451.00174.
9. Mockus A, Fielding RT, Herbsleb JD. Two case studies of open source software development: Apache and Mozilla. *ACM Trans Softw Eng Methodol*. 2002;11(3):309–346. doi:10.1145/567793.567795.
10. Nocera S, Romano S, Di Penta M, Francese R, Scanniello G. On the adoption of software bill of materials in open-source software projects. *J Syst Softw*. 2025;230:112540. doi:10.1016/j.jss.2025.112540.
11. Pearce H, Ahmad B, Tan B, Dolan-Gavitt B, Karri R. Asleep at the keyboard? Assessing the security of GitHub Copilot’s code contributions. 2022 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 2022. p. 754–768. doi:10.1109/SP46214.2022.9833571.
12. Raman N, Cao M, Tsvetkov Y, Kästner C, Vasilescu B. Stress and burnout in open source: toward finding, understanding, and mitigating unhealthy interactions. *ACM/IEEE 42nd International Conference on Software Engineering: New Ideas and Emerging Results*, Seoul, South Korea, 2020. p. 57–60. doi:10.1145/3377816.3381732.
13. Schäfer M, Nadi S, Eghbali A, Tip F. An empirical evaluation of using large language models for automated unit test generation. *IEEE Trans Softw Eng*. 2024;50(1):85–105. doi:10.1109/TSE.2023.3334955.
14. Taşgetiren A, Tarhan AK. Challenges in evaluating OSS quality: results from SLR on quality evaluation tools. *Ann Comput Sci Inf Syst*. 2025:393–398. doi:10.15439/2025F2661.
15. Trinkenreich B, Santos F, Stol KJ. Predicting attrition among software professionals: antecedents and consequences of burnout and engagement. *ACM Trans Softw Eng Methodol*. 2024;33(8):1–45. doi:10.1145/3691629.
16. Williams L, Benedetti G, Hamer S, Paramitha R, Rahman I, Tamanna M, et al. Research directions in software supply chain security. *ACM Trans Softw Eng Methodol*. 2025;34(5):1–38. doi:10.1145/3714464.

17. Zahan N, Lin E, Tamanna M, Enck W, Williams L. Software bills of materials are required. Are we there yet? IEEE Secur Priv. 2023;21(2):82–88. doi:10.1109/MSEC.2023.3237100.
18. Zhao L, Elbaum S. Quality assurance under the open source development model. J Syst Softw. 2003;66(1):65–75. doi:10.1016/S0164-1212(02)00064-X.