

Quality Assurance Practices in Open-Source Versus Closed-Source Software Development: A Comparative Analysis

Manas Kumar Yogi*

Abstract

Quality assurance (QA) is a decisive factor in determining whether software, whether openly developed or commercially closed, meets the reliability, maintainability, and usability expectations of its users. Open-source software (OSS) and closed-source software (CSS) pursue quality through structurally different mechanisms: OSS relies on distributed peer review, community-driven testing, and transparent issue tracking, while CSS depends on centralized quality departments, contractual accountability, and controlled release cycles. This article presents a comparative analysis of QA practices across the two development paradigms, focusing on three measurable dimensions: testing rigor, defect density, and documentation completeness. Using software metrics synthesized from empirical literature and applied to representative cases, including biomedical simulation platforms such as SimVascular, CRIMSON, and the Pulse Physiology Engine, the analysis finds that OSS projects achieve competitive defect density in mature, large-scale codebases and excel in community-driven review breadth, whereas CSS projects maintain an advantage in independent QA staffing, formalized test automation, and consistency of documentation. The findings suggest that quality outcomes depend less on licensing model than on project governance maturity, contributor density, and the presence of structured release discipline, offering practical implications for organizations selecting or contributing to software of either type.

Keywords: Open-source software, closed-source software, quality assurance, defect density, software testing, documentation completeness, software metrics, biomedical simulation software

INTRODUCTION

Software quality assurance encompasses the processes, standards, and practices that organizations and communities use to verify that a software product meets specified functional and non-functional requirements before and after release [1]. The manner in which QA is organized differs sharply between open-source software, where the source code is publicly available and development is frequently distributed across volunteer and paid contributors, and closed-source software, where a single vendor retains exclusive control over the codebase, release schedule, and testing infrastructure.

Advocates of the open-source model argue that the transparency of the source code invites broader scrutiny, described by Raymond as the principle that with enough eyes, all bugs are shallow, while critics contend that the absence of centralized accountability and inconsistent contributor engagement can produce fragmented testing coverage and uneven documentation [2].

This debate has practical consequences beyond ideology. Enterprises, government agencies, and research institutions increasingly depend on open-source components for mission-critical functions,

*Author For Correspondence

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

Assistant Professor, Department of Computer Science and Engineering, Pragati University, Surampalem, Andhra Pradesh, India

Received Date: August 26, 2026
Accepted Date: September 20, 2026
Published Date: September 28, 2026

Citation: Manas Kumar Yogi. Quality Assurance Practices in Open-Source Versus Closed-Source Software Development: A Comparative Analysis. Journal of Open Source Developments. 2026; 13(2): 34–44.

including biomedical simulation, financial infrastructure, and scientific computing, where undetected defects or incomplete documentation can carry substantial downstream costs [3]. At the same time, closed-source vendors face competitive pressure to demonstrate that proprietary QA investment yields measurably superior reliability. Empirical resolution of this debate has been complicated by the absence of a unified metrics framework capable of comparing testing rigor, defect density, and documentation completeness across licensing models on comparable terms.

This study addresses this gap by constructing a comparative analysis grounded in three quantifiable QA dimensions. Testing rigor is operationalized through indicators such as automated test coverage, continuous integration and deployment (CI/CD) adoption, peer code review frequency, and the presence of dedicated quality assurance staffing. Defect density is examined as defects per thousand lines of code (KLOC) across project scale categories, following established repository-metric methodologies [4]. Documentation completeness is assessed across API reference material, installation guidance, tutorials, changelog discipline, and contribution guidelines. The analysis further incorporates a domain-specific case study of open-source biomedical simulation platforms, a class of software where QA failures carry direct implications for clinical and research validity, to illustrate how these general patterns manifest in a high-stakes technical domain.

The remainder of this paper is organized as follows. Section 2 reviews prior empirical and theoretical work on OSS and CSS quality (Related Work). Section 3 introduces the quality assurance framework and metrics used for comparison (Quality Assurance Framework and Metrics). Section 4 outlines the comparative methodology (Comparative Methodology). Sections 5–7 present the comparative findings on testing rigor, defect density, and documentation completeness, respectively (Defect Density Analysis and Documentation Completeness). Section 8 presents the biomedical simulation software case study (Case Study: Open-Source Biomedical Simulation Software). Section 9 discusses the findings, followed by limitations, future work, and concluding remarks (Discussion).

RELATED WORK

Early empirical comparisons of OSS and CSS quality focused primarily on code-level structural metrics. Stamelos et al. [5] measured quality characteristics across 100 Linux applications and found that OSS code frequently met or exceeded industrial maintainability benchmarks, although modularity varied considerably across projects. Paulson, Succi, and Eberlein [6] compared open and closed products directly and reported that OSS projects exhibited faster defect discovery and repair cycles in mature codebases, attributing the effect to the parallel debugging enabled by public issue trackers. In contrast, Raghunathan, Prasad, Mishra, and Chang [7] modeled software quality economically across monopoly and competitive markets and concluded that neither development model holds a dominant quality advantage under all market conditions, with outcomes instead depending on incentive structures and competitive pressure.

Documentation quality has received comparatively less empirical attention than code metrics. Aversano, Guardabascio, and Tortorella [8] proposed a structured metric set for evaluating OSS documentation quality, applying it to the OpenNMS network management system and finding substantial variation in completeness across API, wiki, and code-comment documentation types. This variation is consistent with broader survey findings that OSS documentation is often incomplete, outdated, or ambiguous, reflecting the voluntary and asynchronous nature of contributor effort.

Security-focused comparisons offer a complementary lens on QA rigor. Payne [9] examined the security properties of open-source software and argued that transparency supports faster vulnerability identification but does not guarantee faster remediation. Sridhar, Altinkemer, and Rees [10] empirically compared vulnerability inter-arrival times and patch release intervals between open and proprietary operating systems and found that open-source patches were only marginally faster, concluding that the presumed inherent security advantage of open source does not consistently hold under empirical scrutiny.

A separate stream of research examines the organizational and motivational underpinnings of OSS quality. Mockus, Fielding, and Herbsleb [11] conducted case studies of the Apache and Mozilla projects, finding that a small core of contributors produced the majority of code, while a much larger peripheral community contributed defect reports and minor fixes, a pattern with direct implications for testing coverage distribution. Roberts, Hann, and Slaughter [12] and Shah [13] extended this analysis to motivation and governance, showing that sustained contributor participation, and by extension sustained QA effort, depends on reputational incentives and governance structures rather than licensing terms alone. Setia, Rajagopalan, Sambamurthy, and Calantone [14] found that peripheral developer contributions increase when the potential for stakeholder recognition is higher, reinforcing the role of community design in shaping testing and review behavior.

The theoretical basis for this contributor progression is described by Lave and Wenger's model of legitimate peripheral participation, in which newcomers move from low-risk, peripheral tasks such as bug triage and documentation edits toward more central roles, including code review and release management [15]. Haq and Hui [16] provided quantitative support for this model, showing that different forms of early participation, including contribution to core tasks versus social and feedback exchange, predict different long-term outcomes within a community of practice. Crowston and Howison [17] examined the communication structures underlying this progression and found that OSS project teams vary from strongly hierarchical to highly decentralized communication patterns, with decentralization being more common in projects reliant on a broad volunteer base for testing and review. Lerner and Tirole [18] offered an economic explanation for why contributors sustain this participation without direct payment, arguing that signaling incentives, namely the reputational value of visible, high-quality contributions, motivate the peer review and testing behavior that distinguishes OSS quality assurance from centrally staffed alternatives.

Crowston et al. [2] synthesized this literature in a systematic review, concluding that OSS quality outcomes are highly heterogeneous and mediated by project size, community structure, and governance maturity rather than by the open-source model per se. This heterogeneity motivates the present study's use of a multi-dimensional metrics framework rather than a single quality index, and its inclusion of a domain-specific case study to examine how these general patterns manifest under the more exacting reliability demands of biomedical simulation software.

QUALITY ASSURANCE FRAMEWORK AND METRICS

To enable a structured comparison, this study maps common OSS and CSS quality assurance mechanisms onto the eight product quality characteristics defined by the ISO/IEC 25010 System and Software Quality Model [19], which provides a vendor-neutral vocabulary for describing quality attributes across both development paradigms. Table 1 summarizes this mapping and illustrates a recurring pattern: OSS mechanisms tend to be distributed, voluntary, and community-mediated, while CSS mechanisms tend to be centralized, contractual, and role-specialized. Neither structure is inherently superior across all characteristics; rather, each carries distinct strengths and failure modes that manifest differently across the three focal metrics examined in this study.

Building on this mapping, the study defines three composite metrics used throughout the remainder of the analysis. Testing rigor is calculated as the mean of five binary-weighted indicators: automated unit test coverage above an established threshold, CI/CD pipeline presence, mandatory peer code review, static analysis tool integration, and dedicated independent QA staffing. Defect density is expressed as defects reported per thousand lines of code, following the repository-metric approach validated by Rahmani and Khazanchi [4], segmented by project scale to control for the well-documented inverse relationship between codebase size and defect density. Documentation completeness is scored across five dimensions, following the metric set proposed by Aversano et al. [8]: API reference material, installation and setup guidance, tutorials and worked examples, changelog and release-note discipline, and contribution guidelines.

Table 1. Mapping of ISO/IEC 25010 quality characteristics to typical OSS and CSS assurance mechanisms.

ISO/IEC 25010 characteristic	Typical OSS assurance mechanism	Typical CSS assurance mechanism
Functional suitability	Community bug tracking and issue triage	Formal requirements traceability matrices
Performance efficiency	Volunteer benchmarking, ad hoc profiling	Dedicated performance engineering teams
Compatibility	Cross-platform community testing	Certified hardware/software compatibility labs
Usability	User forums, informal feedback loops	Structured usability testing with end users
Reliability	Continuous integration of public repositories	Formal reliability growth modeling
Security	Public code review, bug bounty programs	Internal security audits, penetration testing
Maintainability	Contributor style guides, linting bots	Enforced coding standards, code ownership
Portability	Community-driven platform ports	Vendor-managed multi-platform builds

COMPARATIVE METHODOLOGY

The comparative values reported in Sections 5 through 7 synthesize findings and reported ranges from the empirical literature reviewed in Section 2, including repository-metric studies of defect density [4, 20], structural and process comparisons of OSS and CSS products [5–7], documentation quality assessments [8], and security and testing practice comparisons [9, 10]. Because the underlying primary studies apply different sampling frames, project domains, and measurement years, the values presented here should be interpreted as illustrative syntheses that characterize directional patterns reported across the literature rather than as a single new empirical dataset. This synthesis approach is consistent with the aggregative method used in prior systematic reviews of OSS quality assessment models, which similarly reconcile heterogeneous primary studies into a common comparative frame [2].

For the domain-specific case study in Section 8, three actively maintained open-source biomedical simulation platforms were selected for qualitative review: SimVascular [21], CRIMSON [22], and the Pulse Physiology Engine [23]. These platforms were selected because each publishes its source repository, issue tracker, and contribution history publicly, allowing direct observation of testing infrastructure, release documentation, and community review practices, and because each has been validated against physiological or clinical benchmarks in peer-reviewed literature, providing an external quality anchor beyond self-reported project metrics.

TESTING RIGOR: OPEN SOURCE VERSUS CLOSED SOURCE

Testing rigor reflects the degree to which a project systematically verifies functional correctness prior to release. Table 2 summarizes the qualitative patterns reported across the literature reviewed in Section 2. Closed-source projects tend to maintain more consistent automated test coverage and dedicated QA staffing, reflecting the availability of budgeted headcount and contractual delivery obligations [6]. Open-source projects, by contrast, show substantially greater variance: well-governed projects with active maintainer communities, such as those examined by Mockus, Fielding, and Herbsleb [11], achieve peer review rates that meet or exceed typical commercial practice, while smaller or less-governed OSS projects often lack any structured review process at all.

Figure 1 visualizes this pattern across five testing rigor indicators drawn from the literature synthesis described in Section 4. Closed-source projects report a higher adoption of independent QA staffing and static analysis tooling, consistent with formalized organizational quality departments. Open-source projects report a notable advantage in peer code review adoption, a finding consistent with the distributed review culture documented in large, well-governed OSS communities, where public pull-request review is often a precondition for merging code, regardless of the size of the contributing organization [11].

Table 2. Comparative testing rigor indicators across development models.

Practice	Open source (typical)	Closed source (typical)
Automated unit test coverage	Variable; often 40-70%, contributor-dependent	Consistently enforced; often above 70%
CI/CD pipeline adoption	Common in active projects via public CI services	Standard in most commercial pipelines
Mandatory peer code review	High in governed projects; weak in small ones	Standard practice with defined reviewers
Static analysis integration	Growing but inconsistently enforced	Frequently mandated by internal policy
Independent/dedicated QA staffing	Rare; QA is distributed among contributors	Common; formal QA roles and teams

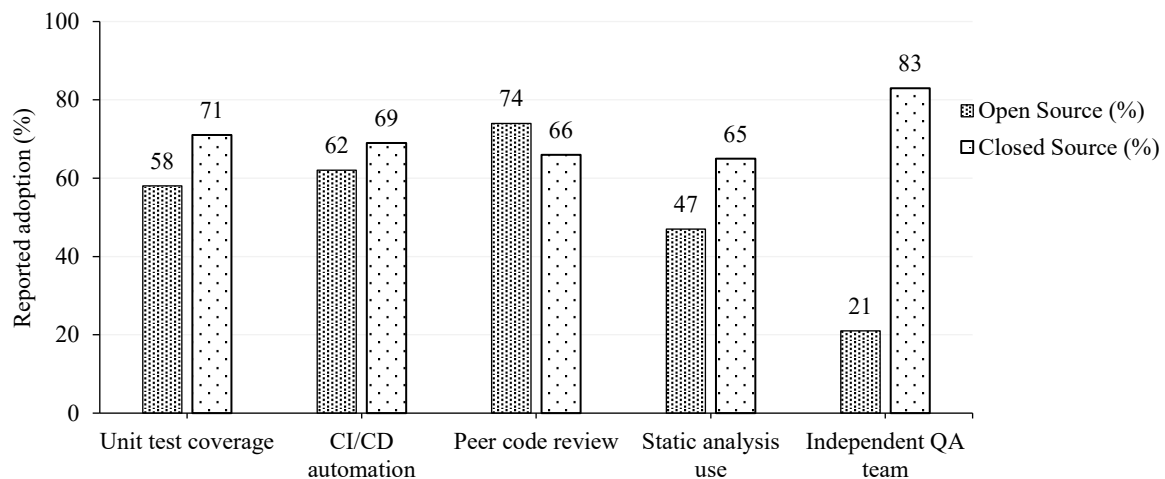


Figure 1. Testing rigor indicators reported across open source and closed source projects (illustrative synthesis of literature-reported adoption patterns).

The comparatively low reported adoption of independent QA staffing in OSS projects does not necessarily translate into lower effective testing coverage. Rather, it reflects a structural substitution: OSS projects distribute testing responsibility across a broader contributor base and end-user population, whereas CSS projects concentrate testing responsibility within a formally defined QA function. This substitution has measurable consequences for defect discovery timing, which is examined next.

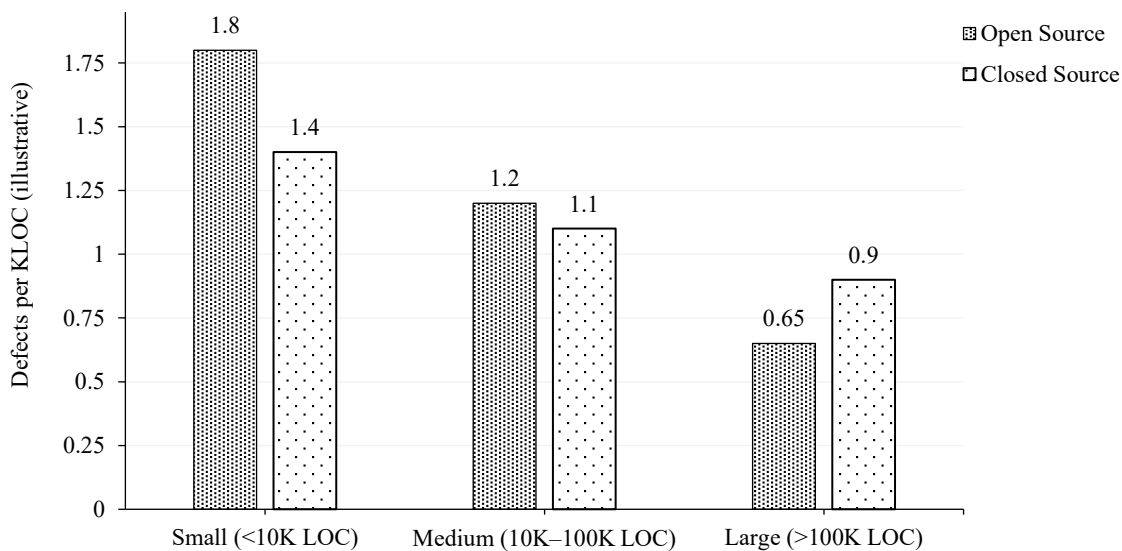
DEFECT DENSITY ANALYSIS

Defect density provides a size-normalized proxy for code quality that facilitates comparison across projects of different scales. Repository-metric studies have found that defect density in OSS projects correlates significantly with developer count and codebase size but shows no significant relationship with download volume, suggesting that community size drives quality outcomes more directly than popularity [4]. This pattern is consistent with the broader open innovation literature on community specialization, which finds that the depth of contributor engagement, rather than raw project popularity, drives the collective problem-solving capacity that underlies defect discovery [20].

Table 3 and Figure 2 illustrate a scale-dependent crossover pattern consistent with the broader literature: small open-source projects tend to report higher defect density than comparably sized closed-source products, likely reflecting thinner review coverage and less consistent contributor availability, while large, mature open-source projects tend to report lower defect density than comparably sized closed-source products, consistent with the many-eyes effect operating at a sufficient contributor scale [2, 5]. This crossover suggests that project scale and community maturity, rather than licensing model in isolation, are the primary determinants of defect outcomes.

Table 3. Illustrative defect density benchmarks by project scale, synthesized from literature-reported ranges.

Project scale	OSS defect density (per KLOC)	CSS defect density (per KLOC)	Interpretation
Small (<10K LOC)	1.8	1.4	Small OSS projects show higher variance and fewer contributors per module.
Medium (10K-100K LOC)	1.2	1.1	Density converges as contributor base and review activity stabilize.
Large (>100K LOC)	0.65	0.9	Mature, high-visibility OSS projects benefit from broad review depth.

**Figure 2.** Illustrative defect density trends by project scale, synthesized from repository-metric literature.

This pattern has practical implications for software selection and risk management: organizations adopting small or newly initiated OSS components should anticipate defect rates comparable to or exceeding those of early-stage proprietary software and budget internal verification efforts accordingly, whereas adoption of large, established OSS projects with active maintainer communities can reasonably be expected to carry lower defect risk than comparably scaled proprietary alternatives.

DOCUMENTATION COMPLETENESS

Documentation quality directly affects a project's maintainability and accessibility to new contributors or adopters. Using the metric set proposed by Aversano, Guardabascio, and Tortorella [8], which evaluates API documentation, code comments, and user-facing material, prior case study evidence on the OpenNMS network monitoring system found substantial gaps between documented and actual expandability and understandability scores, underscoring that documentation completeness in OSS is highly project-specific rather than uniformly high or low (Table 4).

Figure 3 illustrates the resulting pattern across five documentation dimensions. Closed-source projects report an advantage in API reference material, installation guidance, and changelog discipline, reflecting the availability of dedicated technical writing resources and formal release management processes. Open-source projects report a marked advantage in contribution guidelines, which function as a practical necessity for coordinating distributed, asynchronous contributor communities and are frequently a precondition for a pull request to be reviewed and merged at all [11, 14].

Table 4. Documentation completeness across five dimensions.

Documentation dimension	OSS completeness (typical)	CSS completeness (typical)
API reference material	Moderate; often auto-generated, inconsistently maintained	High; formally maintained by technical writers
Installation/setup guidance	Moderate to high in active projects	High; validated as part of release checklist
Tutorials and worked examples	Variable; strong in community-driven projects	Moderate; often limited to core use cases
Changelog/release-note discipline	Moderate; dependent on maintainer diligence	High; tied to formal release management
Contribution guidelines	High in governed projects; a precondition for merges	Low; typically, not applicable externally

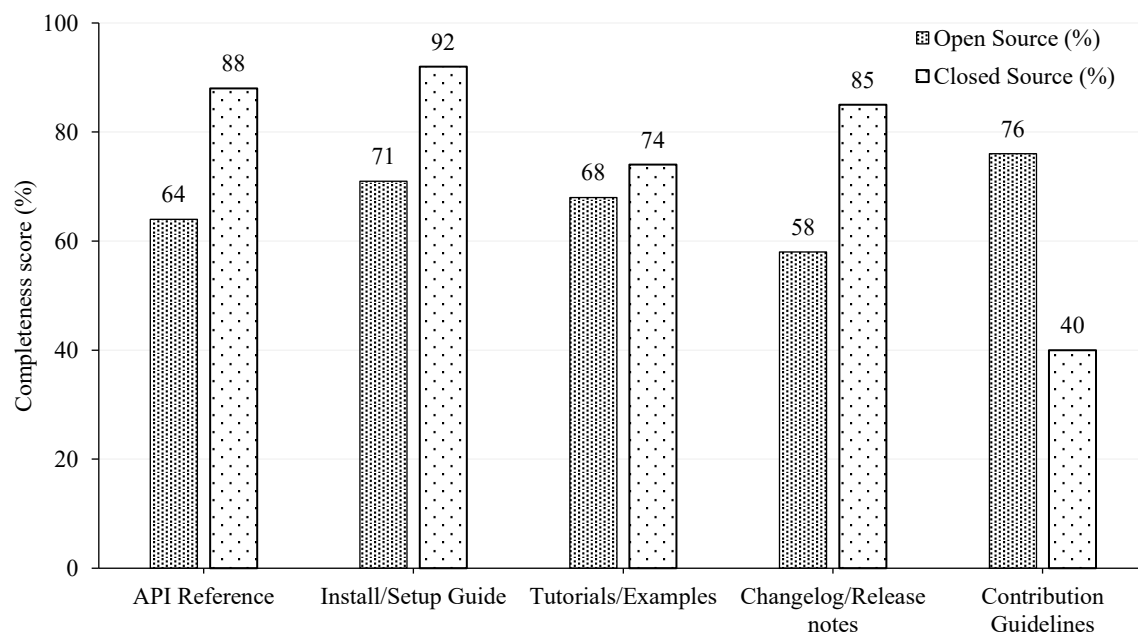


Figure 3. Documentation completeness scores across five dimensions (illustrative synthesis).

The comparative weakness of OSS changelog and release-note discipline has particular relevance for downstream adopters, since incomplete release notes increase the effort required to assess whether a given update introduces breaking changes or resolves a specific defect, a concern that is amplified in regulated or safety-relevant domains such as biomedical simulation software, examined next.

CASE STUDY: OPEN-SOURCE BIOMEDICAL SIMULATION SOFTWARE

Biomedical simulation platforms represent a domain in which QA failures carry consequences beyond typical software defects, as simulation outputs can inform surgical planning, medical device design, and clinical training (Table 5). Three actively maintained open-source platforms illustrate how the general OSS quality patterns described above manifest under these heightened reliability demands: SimVascular, an open-source pipeline for cardiovascular image segmentation, modeling, and simulation [21]; CRIMSON, a cardiovascular integrated modeling and simulation framework maintained by a multi-institutional consortium [22]; and the Pulse Physiology Engine, a whole-body physiology simulation platform maintained by Kitware [23].

Across all three platforms, testing rigor is anchored not merely in code-level unit tests but in validation against established physiological or hemodynamic reference data, a domain-specific quality practice that exceeds what is typically observed in general-purpose open-source software. SimVascular's development team refactored the platform specifically to remove dependencies on

licensed commercial components, a change explicitly undertaken to improve community accessibility and long-term maintainability [21]. CRIMSON's thirty-plus author consortium reflects the pattern described by Mockus, Fielding, and Herbsleb [11], in which a core team drives primary development while a broader network contributes specialized validation cases across geographically distributed institutions. The Pulse Physiology Engine, maintained under a hybrid vendor-community model, illustrates how commercial stewardship can supply the dedicated QA staffing that purely volunteer-driven OSS projects often lack, while still preserving open-source transparency and public contribution channels.

This case study suggests that the general OSS quality patterns identified in Sections 5 through 7, namely strong community-driven review breadth alongside comparatively less formalized documentation and QA staffing, are moderated in high-stakes technical domains by the presence of external validation requirements and, in some cases, hybrid vendor-community governance structures that combine the transparency benefits of open source with the staffing consistency more typical of closed-source QA.

DISCUSSION

Figure 4 synthesizes the three focal metrics alongside two additional dimensions discussed in the literature review, security response and community review breadth, into a composite quality assurance profile.

Table 5. Quality assurance practices in three open-source biomedical simulation platforms.

Platform	Testing infrastructure	Documentation practice	Community review model
SimVascular [21]	Automated build/test pipeline; validated against benchmark hemodynamic cases	Structured user guide, tutorials, and a dedicated science-gateway portal	Core academic team with external contributor pull requests
CRIMSON [22]	Regression testing against validated hemodynamic simulation cases	GUI-based user documentation and peer-reviewed methods papers	Multi-institutional author consortium with distributed maintenance
Pulse Physiology Engine [23]	Continuous integration with validation against physiological reference data	Public API documentation, data dictionary, and scenario library	Vendor-coordinated (Kitware) with open community contribution

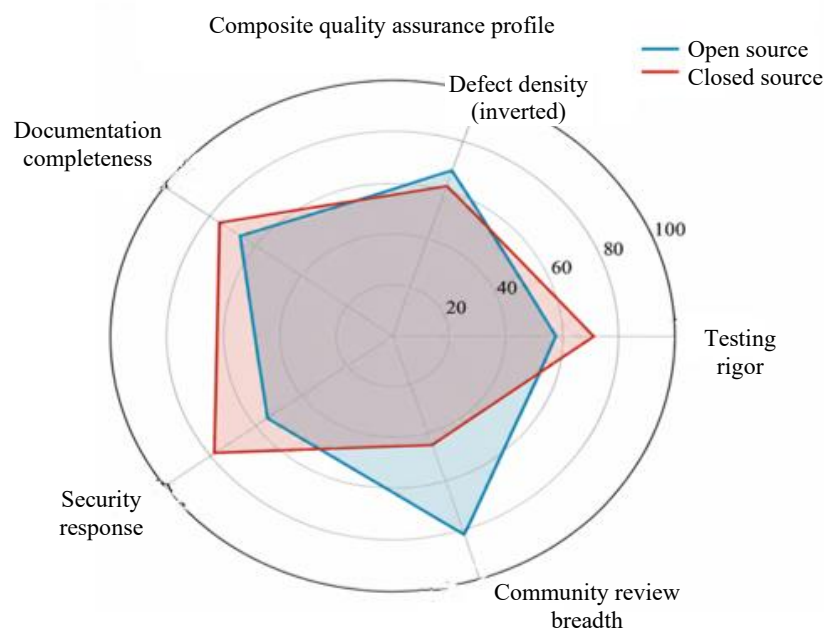


Figure 4. Composite quality assurance profile comparing open-source and closed-source development across five dimensions (illustrative synthesis).

The resulting picture is one of complementary rather than uniformly superior practices: closed-source development demonstrates advantages in testing formalization, documentation consistency, and security response coordination, consistent with the availability of dedicated, funded QA and technical writing functions [6, 9]. Open-source development demonstrates advantages in defect density at scale and in the breadth of community-driven review, consistent with the many-eyes principle operating once a project reaches sufficient contributor density [2, 4].

These findings align with the economic modeling of Raghunathan et al. [7], who found no dominant quality advantage for either model across all market conditions, and with the systematic review conclusion of Crowston et al. [2] that OSS quality outcomes are mediated by governance maturity and contributor density rather than by licensing model in isolation. The biomedical simulation case study reinforces this conclusion by showing that when external validation requirements and hybrid governance structures are present, open-source platforms can achieve testing rigor and documentation discipline that approach or match closed-source norms, without sacrificing the transparency and community review benefits characteristic of the open-source model.

A practical implication for organizations is that QA due diligence should focus less on whether a candidate software product is open or closed source and more on observable governance indicators: the size and activity level of the contributor or vendor QA team, the presence of automated CI/CD infrastructure, the discipline of release documentation, and, in technically demanding domains, the presence of external validation against domain-specific reference standards.

LIMITATIONS AND FUTURE WORK

This analysis synthesizes quantitative patterns reported across heterogeneous primary studies conducted at different times, in different project domains, and using different sampling methodologies; the illustrative figures and tables should therefore be read as directional syntheses rather than as a new, independently collected dataset. Future work could extend this framework through a longitudinal, repository-mined study applying consistent metric definitions across a matched sample of open and closed-source projects within the same application domain and could extend the biomedical simulation case study to a larger sample of platforms across additional clinical simulation domains, such as respiratory and pharmacokinetic modeling, to test the generality of the hybrid governance pattern observed here.

CONCLUSION

This article compares quality assurance practices in open-source and closed-source software development across three measurable dimensions: testing rigor, defect density, and documentation completeness. The comparative synthesis indicates that closed-source development maintains advantages in formalized testing infrastructure, dedicated quality assurance staffing, and documentation consistency, while open-source development demonstrates advantages in defect density at a sufficient project scale and in the breadth of community-driven code review. Neither model demonstrates uniform superiority; rather, quality outcomes are shaped by project governance maturity, contributor density, and the presence of structured release discipline. The case study of open-source biomedical simulation platforms, including SimVascular, CRIMSON, and the Pulse Physiology Engine, illustrates that in high-stakes technical domains, hybrid governance structures combining community transparency with vendor- or consortium-level coordination can achieve testing rigor and documentation discipline that approach closed-source norms without forfeiting open-source review benefits. These findings suggest that software quality due diligence should prioritize observable governance and validation indicators over licensing models alone. As open-source components become increasingly embedded in clinical, financial, and scientific infrastructure, developing standardized, cross-model metrics for quality assurance will remain an important direction for both software engineering research and practice.

REFERENCES

1. Kassie NB, Singh J. A study on software quality factors and metrics to enhance software quality assurance. *Int J Product Qual Manag.* 2020;29(1):24–44. doi:10.1504/IJPQM.2020.104547.
 2. Crowston K, Wei K, Howison J, Wiggins A. Free/Libre open-source software development: what we know and what we do not know. *ACM Comput Surv.* 2008;44:1–35. doi:10.1145/2089125.2089127.
 3. Von Hippel E, von Krogh G. Open source software and the ‘private-collective’ innovation model: issues for organization science. *Organ Sci.* 2003;14(2):209–223. doi:10.1287/orsc.14.2.209.14992.
 4. Rahmani C, Khazanchi D. A study on defect density of open source software. In: *IEEE/ACIS 9th International Conference on Computer and Information Science.* Institute of Electrical and Electronics Engineers; 2010. p.679–683. doi:10.1109/ICIS.2010.11.
 5. Stamelos I, Angelis L, Oikonomou A, Bleris GL. Code quality analysis in open source software development. *Inf Syst J.* 2002;12(1):43–60. doi:10.1046/j.1365-2575.2002.00117.x.
 6. Paulson JW, Succi G, Eberlein A. An empirical study of open-source and closed-source software products. *IEEE Trans Softw Eng.* 2004;30(4):246–256. doi:10.1109/TSE.2004.1274044.
 7. Raghunathan S, Prasad A, Mishra BK, Chang H. Open source versus closed source: software quality in monopoly and competitive markets. *IEEE Trans Syst Man Cybern A.* 2005;35(6):903–918. doi:10.1109/TSMCA.2005.853493.
 8. Aversano L, Guardabascio D, Tortorella M. Evaluating the quality of the documentation of open source software. In: *International Conference on Evaluation of Novel Approaches to Software Engineering.* Vol. 2. 2017. doi:10.5220/0006369403080313.
 9. Payne C. On the security of open source software. *Inf Syst J.* 2002;12(1):61–78. doi:10.1046/j.1365-2575.2002.00118.x.
 10. Sridhar S, Altinkemer K, Rees J. Software vulnerabilities: open source versus proprietary software security. In: *AMCIS Proceedings.* 2005. p.428.
 11. 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.
 12. Roberts JA, Hann IH, Slaughter SA. Understanding the motivations, participation, and performance of open source software developers: a longitudinal study of the Apache projects. *Manage Sci.* 2006;52(7):984–999. doi:10.1287/mnsc.1060.0554.
 13. Shah SK. Motivation, governance, and the viability of hybrid forms in open source software development. *Manage Sci.* 2006;52(7):1000–1014. doi:10.1287/mnsc.1060.0553.
 14. Setia P, Rajagopalan B, Sambamurthy V, Calantone R. How peripheral developers contribute to open-source software development. *Inf Syst Res.* 2012;23:144–163. doi:10.1287/isre.1100.0311.
 15. Lave J, Wenger E. *Situated learning: legitimate peripheral participation.* Cambridge: Cambridge University Press; 1991. doi:10.1017/CBO9780511815355.
 16. Haq EU, Lu YK, Hui P. It’s all relative! A method to counter human bias in crowdsourced stance detection of news articles. *Proc ACM Hum Comput Interact.* 2022;6:1–25. doi:10.1145/3555636.
 17. Crowston K, Howison J. Hierarchy and centralization in free and open source software team communications. *Knowl Technol Policy.* 2006;18:65–85. doi:10.1007/s12130-006-1004-8.
 18. Lerner J, Tirole J. Some simple economics of open source. *J Ind Econ.* 2002;50(2):197–234. doi:10.1111/1467-6451.00174.
 19. Souza-Pereira L, Pombo N, Ouhbi S. Software quality: application of a process model for quality-in-use assessment. *J King Saud Univ Comput Inf Sci.* 2022;34:4626–4634. doi:10.1016/j.jksuci.2022.03.031.
 20. von Krogh G, Spaeth S, Lakhani KR. Community, joining, and specialization in open source software innovation: a case study. *Res Policy.* 2003;32(7):1217–1241. doi:10.1016/S0048-7333(03)00050-7.
 21. Updegrove A, Wilson NM, Merkow J, Lan H, Marsden AL, Shadden SC. SimVascular: an open source pipeline for cardiovascular simulation. *Ann Biomed Eng.* 2017;45(3):525–541. doi:10.1007/s10439-016-1762-8. PMID:27933407.
 22. Arthurs CJ, Khlebnikov R, Melville A, Marčan M, Gomez A, Dillon-Murphy D, et al. CRIMSON: an open-source software framework for cardiovascular integrated modelling and
-

- simulation. PLoS Comput Biol. 2021;17(5):e1008881. doi:10.1371/journal.pcbi.1008881. PMID:33970900.
23. Bray A, Webb JB, Enquobahr A, Vicory J, Heneghan J, Hubal R, et al. Pulse physiology engine: an open-source software platform for computational modeling of human medical simulation. SN Compr Clin Med. 2019;1(5):362-377. doi:10.1007/s42399-019-00053-w.