ScaleQuality Code Maturity Model (SQCM)

An evidence-based code maturity model for the era of AI-generated software

Version 2.1 · September 2026

Erik Fernandes · ScaleQuality

Scoring calibration described: v2.2

DOI: 10.5281/zenodo.22781322

Canonical English edition

Abstract

The ScaleQuality Code Maturity Model (SQCM) organizes observable software evidence into five domains: Security, Reliability, Maintainability, AI Code Durability, and Supply Chain. Its purpose is to support decisions about a specific codebase and its measured artifacts, with explicit limits on what was observed. It is not a certification of security, a replacement for an organizational maturity appraisal, or a predictor of incidents validated against a longitudinal population.

Version 2.1 consolidates the methodology and documents the implementation audited in September 2026, including score calibration **v2.2**. The central change is the use of measured test coverage as a continuous input, distinguished from structural test evidence. The revision explains coverage provenance, partial test execution, repository scope, missing evidence, and the conditions under which scores can be compared. It also corrects earlier claims about evidence monotonicity and reproducibility. Nineteen focused internal test suites, comprising 194 passing tests, support the implementation contracts examined here; they do not establish predictive validity or general scanner precision and recall.

Publication note. This document supersedes SQCM v2.0, DOI 10.5281/zenodo.21879222. The English text is canonical; the Portuguese PDF is its translation. Paper versions identify publications. A run's scoreVersion identifies its scoring calibration. The two sequences are deliberately distinct.

1. Construct, scope, and evidence

SQCM defines code maturity as a versioned assessment of the condition and supporting engineering practices observable in source code, its history, and available analysis artifacts. Its unit is a measured repository or a project assembled from repository analyses. An organization's processes and the runtime behavior of its deployed service are different units of analysis.

Established work supplies useful, complementary foundations: product quality models [1][2], technical debt and maintainability models [3][4][5], organizational process maturity [6], and delivery performance research [7]. SQCM combines selected signals for an operational code assessment. It does not claim to replace those frameworks or demonstrate conformity with their requirements merely by producing a score.

No questionnaire response enters the code-score composer described here. A separate product workflow can compare a team's declared assessment with code evidence and adjust its team-level result. That workflow, organization and business-unit averages, delivery metrics, and financial projections are not additional SQCM code domains and are not validated by this paper.

An evidence item must be interpreted with its origin and scope. A report generated by an instrumented test run, a report supplied by a CI client, a static finding, a Git-history heuristic, and a model-assisted inference are different kinds of evidence. The score is a decision aid built from those inputs; it does not turn them into equivalent observations.

Automatic discovery is the default for repository assessment. Supported toolchains, accessible dependencies, test commands, report paths, and exclusions still affect whether measurement succeeds. Explicit project configuration is available for some of these conditions. Earlier references to zero configuration describe the default entry path, not universal execution of any repository's private test infrastructure.

2. The five domains

Domain	Principal evidence	Interpretation boundary
Security	Static weaknesses, detected secrets, security practices, and infrastructure misconfiguration	A finding is not a complete exploitability assessment; absence of findings is not proof of security.
Reliability	Reliability findings, test practices, and measured coverage when available	Executed lines and detected practices do not establish test effectiveness or production reliability.
Maintainability	Maintainability findings, code duplication, engineering practices, and coverage evidence	A composite assessment is not a direct estimate of future maintenance cost.
AI Code Durability	Survival of code attributed to AI in a bounded Git-history window	Attribution and survival are proxies; neither identifies all AI use nor establishes correctness.

Domain	Principal evidence	Interpretation boundary
Supply Chain	Known dependency and supplied container-image advisories	Advisory presence is measured; runtime reachability and exploitability are not established.

2.1 Security

Security combines findings and evidence of practices rather than treating every check as an interchangeable defect. Current-tree secrets and secrets found only in history have different treatment. A critical exposure can cap the overall result even when other domains are strong. Infrastructure findings contribute to Security but do not, by themselves, trigger that critical-exposure cap.

This distinction corrects an earlier shorthand: one detected current secret does not necessarily put the **Security domain itself** in its lowest band. In a controlled otherwise-clean fixture, a current critical secret produced Security 60 and capped the overall score at 55. The finding's severity, the domain band, and the overall cap describe different things.

Infrastructure checks inspect repository artifacts such as container definitions and orchestration templates. Selected baseline checks are moderated when they express architectural intent rather than a defect. This is calibration of the instrument, not a determination that a deployment complies with a security standard.

2.2 Reliability and Maintainability

These domains combine static findings and observed engineering practices. Coverage supplies one input to both; section 3 defines its meaning. Finding a test or mutation-testing framework is evidence of its presence, not proof that an effective suite ran, and not a measured mutation score.

Duplication is measured over selected source formats, with tolerance for limited repetition. Generated assets, translation catalogs, and lockfiles can otherwise dominate the denominator. The v2.0 development record reported 8.5% duplication before restricting source scope and 1.7% afterward on the same subject. Those values illustrate a historical scope error, not a new benchmark or a universally applicable correction factor.

Research on churn, ownership, and knowledge concentration provides relevant context [10][11][12][13]. It does not imply that SQCM currently computes every metric those studies discuss. In the audited composition, Git conventions and durability evidence are used; a truck-factor measure is not an input to the code score.

2.3 AI Code Durability

The durability engine examines a 90-day history window and allows 14 days for newly introduced code to settle before evaluating survival. It separates attributed AI activity, ordinary human activity, and automation according to its heuristics. The AI domain uses the survival percentage of the attributed AI cohort. Human survival is a comparison supplied alongside it, not a denominator by which the AI score is normalized.

Missing history, no recent commits, a cohort still awaiting settlement, and no AI attribution have distinct states. They must not be read as either 0% or 100% durability. The

implementation also bounds blame processing, with a default limit of 500 files. On large repositories this can leave part of a cohort unexamined and bias survival downward; the output should not be interpreted as exhaustive whole-repository lineage.

Code survival is not correctness. Necessary refactoring can reduce survival; retained defective code can increase it. Authorship attribution depends on metadata and recognizable patterns and can miss unmarked assistance. This domain is not an inventory of external AI services, a measure of model governance, or proof that a specific person used an assistant.

Studies of assistant security [15][16], commercial observations of churn and duplication [17][18][19], controlled productivity work [20], and delivery surveys [21][22] motivate studying these questions but differ in population, methods, and potential bias. Adoption reports [23][24][25] provide historical context only. A recent empirical preprint examines issues associated with verified AI-authored commits [26]; a separate study combines a review of 109 papers, workshops, and patch experiments [27]. Neither validates SQCM's weights or establishes that its durability score predicts incidents.

2.4 Supply Chain

Dependency analysis matches components against advisory data [32]. The domain combines severity-weighted, deduplicated advisory units through a bounded decay function. Its calibration avoids the early saturation in which small advisory backlogs collapsed almost every subject to the minimum. Exact internal coefficients are not published.

When a CI client supplies a scan of a built container image, its advisory evidence can contribute to the same domain. A repository's container recipe is not a measurement of the deployed image. License obligations are reported separately and **do not change the numerical score**; acceptability depends on usage and policy.

SLSA and OpenSSF Scorecard offer related assurance frameworks [28][29], while the cited US executive order and EU regulation provide historical and jurisdiction-specific policy context [30][31]. SQCM does not certify compliance with them. A known vulnerable package can be unreachable at runtime; advisory presence alone cannot settle that question. Conversely, lack of a demonstrated call path does not prove the component harmless.

3. Coverage: what was measured, where, and how

3.1 Four evidence paths

Path	What happens	What the result can support
Connected-repository diagnosis	A supported runner checks out a pinned revision, runs tests, and reads a fresh report.	Coverage of the instrumented scope at that revision, subject to execution status and exclusions.
CI report import	The client packages available reports and sends them with the analysis.	Coverage reported by the client's environment; not independent attestation of that environment.

Path	What happens	What the result can support
Agent before/after validation	Paired receipts check revision, report freshness, test outcomes, report identity, and scope consistency.	A stricter coverage comparison for the validated pair.
Structural fallback	Test/module mapping and available practice evidence are examined when runtime coverage is unavailable.	Evidence of test structure or posture, not a measured percentage of executed source lines.

The supported report readers cover JaCoCo, LCOV, Cobertura, Istanbul, Clover, and Go coverage. Supporting a format does not guarantee execution of every language, build system, or test harness that can emit it. Configuration, generated files, test selection, credentials, and external services can affect the run and denominator.

In the connected diagnosis flow, runtime measurement is attempted before the final verdict when enabled. The selected repository and revision are checked. Failed, unavailable, or mismatched execution falls back to structural evidence with a reason. Completion of a diagnostic job is not, by itself, evidence that the entire test suite passed.

3.2 Full and partial test execution

The general coverage runner may accept a fresh report produced by passing tests even when part of the suite failed, with a partial-execution note. Such a percentage describes the executed scope and should be read with that note. It is not interchangeable with coverage from a complete successful suite.

The paired agent-validation path applies stronger acceptance criteria: reports must be fresh, tests must have passed, failures and incomplete or unsupported execution must not be treated as valid measurements, and the compared scope must remain consistent. If a comparable pair cannot be established, measured coverage is not credited as a verified before/after improvement. These stricter criteria must not be generalized to every coverage number displayed by the platform.

3.3 Continuous input to score calibration v2.2

For a line-coverage report, the underlying proportion is:

coverage = covered instrumented lines / total instrumented lines.

The report's scope and exclusions determine the denominator. SQCM normalizes the accepted percentage to a proportion between 0 and 1. In score calibration v2.2, a valid measured report contributes continuously to Reliability and Maintainability. A qualitative test-posture cap used for structural evidence is not substituted for that measured proportion.

The previous categorical treatment could stop distinguishing improvements once coverage reached a bucket boundary. The current path preserves differences above that boundary. The following controlled compositor fixture holds the other inputs fixed; these are synthetic regression results, not measurements of customer repositories.

Measured coverage	Reliability	Maintainability	Overall
70%	84	92	93
80%	86	93	94
90.5%	89	94	95
100%	91	95	96

This is not a universal conversion table. Other findings, practices, domain availability, rounding, and critical-exposure caps affect the result. In a capped fixture, higher coverage improved its contributing domain scores while the overall score remained 55. A higher percentage does not guarantee a visible increase in the rounded overall result.

Coverage is also not test effectiveness [8]. Executing every line can coexist with weak assertions or missing behavioral cases. Mutation testing provides a different perspective [9], but SQCM must distinguish a detected mutation framework from an actually executed mutation experiment.

3.4 Multi-repository scope

The connected runtime-coverage path examined in this revision targets the primary successfully analyzed repository. It does not execute every connected repository's suite merely because a project contains multiple repositories.

When architectural evidence is merged, accepted report-based coverage takes precedence over structural estimates. Ratios in the selected group are combined arithmetically rather than weighted by the total instrumented lines of every repository. Consequently, a project-level percentage is not necessarily whole-project line coverage. A defensible comparison requires identifying the included repositories and reports; adding a repository can change the scope as well as the number.

4. Composition, levels, and missing evidence

Domain scores and the overall result use a 0-100 scale. The overall composition combines available domain scores and applies calibration rules, including critical-exposure caps. Security, Reliability, and Maintainability form the required core for a ready snapshot in the audited composer. AI Code Durability and Supply Chain contribute when measurable; their absence does not prevent that core snapshot from being ready.

Unavailable optional domains are excluded from composition and the remaining contributions are normalized. This corrects the older description that required all four original domains before a temporal snapshot could exist. A numerical zero is a measured low result; an unavailable value is not a numerical result. State and provenance fields must accompany interpretation. Not every legacy projection preserves this distinction equally, so a scalar alone is insufficient evidence of a completed measurement.

The five overall bands are separated at 40, 60, 80, and 90. These boundaries and the internal weights are design calibration, influenced by established maturity conventions [6], not thresholds estimated from a longitudinal outcome study. Confidence categories indicate

evidence conditions; they are not calibrated probabilities that software is safe or that a prediction is correct.

The implementation separates score composition from narrative generation. Model-assisted qualitative assessment can supply structural test-posture evidence when measured reports are absent. Therefore, the end-to-end pipeline is not accurately described as containing no inference. The claim is narrower: accepted measured coverage is not replaced by a model's invented percentage.

5. Measurement boundaries and comparability

5.1 Adding evidence can change the model's available scope

Version 2.0 stated that supplying an image report could only lower a score and that withholding evidence could not improve it. That general guarantee is withdrawn.

For a fixed set of measured domains and other fixed inputs, adding adverse advisory evidence does not improve the Supply Chain domain. But making a previously unavailable domain measurable also changes the overall composition. A controlled fixture with Supply Chain initially unavailable changed from overall 95 to 96 when clean image evidence introduced a measured domain. Omitting adverse evidence can likewise make an assessment look better. No general resistance to selective evidence omission is established by the score formula.

The appropriate comparison is therefore **score plus scope plus provenance**. A change in coverage source, measured domain set, repository set, scanner depth, exclusions, or scoring version may explain a delta without any corresponding change in the underlying code.

5.2 Reproduction is more than checking out the same commit

Composition is deterministic for fixed accepted inputs and calibration. Collecting those inputs again from the same commit is a different operation. Advisory databases change, Git-history windows move with the clock, tool and rule versions change, and qualitative model-assisted assessments may vary.

A reproducible record should identify revision, repository and artifact scope, report identity and exclusions, execution outcome, analysis time, tool/rule versions, scoring version, and relevant external-data versions. These are methodological requirements for a defensible replay, not a claim that every present export already captures every item completely.

Completed diagnostic results have persistence guards against late replacement, and the recorded scoring version travels with the run. Dedicated evolution-report logic suppresses a delta across scoring versions. However, not all summary charts currently enforce the same comparison restrictions. Readers should inspect the versioned records before treating every visual trend as a like-for-like improvement. This publication does not silently relabel or recalculate historical runs.

6. Static analysis depth and artifact scope

Fast analysis combines selected static checks, secret detection, dependency analysis, duplication, infrastructure checks when applicable, and practice evidence. Deep analysis additionally uses a code-property-graph engine [33] with a proprietary taint specification. The audited selection chooses the dominant supported language between Java and JavaScript/TypeScript; it is not a claim that every module of a polyglot repository receives the same deep treatment.

The catalogs contain ten Java and seven JavaScript/TypeScript vulnerability classes. When deep execution is unavailable, fast analysis can still produce findings. A completed fast result must not be interpreted as successful execution of every deep check. Provenance granularity remains a limitation; it does not universally establish per-query execution success.

Licensing constraints informed the choice of analysis infrastructure. The historical CodeQL terms cited in [34] should be consulted for their actual conditions; availability to some users does not imply an unrestricted right to redistribute a hosted commercial analysis service. This paper does not offer a general licensing opinion.

A CycloneDX SBOM can accompany a repository result. Its availability depends on successful collection and transport limits; the presence of an export option alone does not prove that a complete inventory was produced. Production performance, live application behavior, and runtime exploitability remain outside this code diagnosis. Executing tests in a controlled runner does not extend that boundary to production telemetry.

7. Validation and results

7.1 September 2026 implementation audit

The revision followed interface contracts, service calls, scoring, persistence, integration boundaries, failure handling, and focused regression tests. Production task definitions and running image identifiers were inspected separately from local code. This confirms the audited deployment snapshot; it does not constitute new end-to-end tests of every customer integration.

Validation group	Suites	Passing tests
Composition, coverage, upload, states, and agent deltas	7	36
Receipts, revision checks, report discovery/freshness, and missing suites	6	83
Persistence, CI projection, coverage, cards, and MCP contracts	5	71
Paper-specific controlled examples and counterexamples	1	4
Total	19	194

The four paper-specific tests exercise continuous coverage, a critical-exposure cap, the effect of making Supply Chain measurable, a current secret, and sensitivity to structural posture within their assertions. These are implementation regression checks written and

run by the vendor. They are not an independent validation sample, a security penetration test, or a study correlating SQCM with field outcomes.

The audited implementation identifiers and fixture outcomes are recorded in the supplementary validation note, with illustrative coverage rows. Internal source code, exact scoring coefficients, and proprietary rule catalogs are not released with this paper. The supplement therefore supports traceability but not unrestricted independent reproduction of the full proprietary instrument.

7.2 Historical performance observations

SQCM v2.0 reported three fast CI upload-to-verdict observations on one commit: **5.25, 5.29, and 7.17 seconds**, using a two-vCPU, eight-GB staging container. They were a small historical sample, not a latency distribution. The reported coverage-read stage must not be interpreted as a complete execution of arbitrary test suites in that time.

Those observations were not rerun for v2.1. Connected coverage can require dependency installation and test execution, and its orchestration can wait up to approximately 20 minutes before fallback. The older five-second example is therefore not an end-to-end latency promise for the current connected-repository flow.

The v2.0 deep-analysis record reported approximately 33, 67, and 101 seconds on eight, four, and two virtual CPUs respectively for its 277-source-file Java subject. These figures also remain historical. The two earlier measurement lessons remain useful: CPU contention can distort per-stage attribution, and polling resolution can distort apparent completion time. Neither is a substitute for a new benchmark campaign.

7.3 Historical detection calibration

The prior publication reported 23 classified findings in a deliberately vulnerable Java application, five genuine findings in a vulnerable Node application, and zero findings in a 49-file reference Java application. It also reported no false positives in its examined internal services and frontend. The earlier refinement exercise retained five known-positive paths while reducing reported false positives from 137 to zero on its calibration subject.

These are retained as descriptions of the v2.0 development record, not reproduced results for v2.1. Teaching applications and author-selected internal code do not establish general precision, recall, or absence of vulnerabilities. Runtime encoding and type-resolution failures found during that work reinforce the need to validate the distributed analysis artifact and its environment together.

8. Limitations and research agenda

The most consequential open question is predictive validity. Neither the score bands nor the domain weights have been established by a longitudinal study linking SQCM levels to incident rates, remediation cost, or maintenance outcomes. Cost-of-quality estimates [14] motivate the problem but do not calibrate repository-level financial predictions.

Further limitations include incomplete static-analysis coverage, uncertain AI attribution, bounded history processing, potentially partial test execution, client-supplied reports, changing advisory databases, model-assisted fallback evidence, multi-repository aggregation, and incomplete enforcement of comparability in some presentations.

Repository access is also not access to every deployed artifact or external SaaS implementation.

The validation agenda is to publish a broader independent defect corpus, a version-pinned latency campaign across sizes and toolchains, fuller provenance and comparison controls, clearer whole-project coverage accounting, and a longitudinal cohort with documented confounders. These are research and implementation goals, not capabilities established by this publication.

9. Version history and corrections

Publication	Principal contribution	Permanent prior record
v1.0	Four-domain code-maturity construct and AI durability	10.5281/zenodo.21251863
v2.0	Supply Chain, artifact boundaries, and historical instrument measurements	10.5281/zenodo.21879222
v2.1	Consolidated method; measured coverage in score v2.2; provenance, scope, and comparability corrections	This publication

The corrections are substantive: measured coverage is continuous rather than a high-coverage bucket; core snapshot readiness requires three domains rather than all original four; optional evidence can change the normalized overall score; repeat collection is not guaranteed identical solely by commit; a critical secret's severity is distinct from its domain band; and bibliographic rationale is distinguished from metrics actually implemented. References [26] and [27] identify their updated versions and publication status.

The five-domain construct remains. The contribution of this revision is a more precise account of what the instrument measures and of the conditions under which its numbers can be used responsibly.

References

The 34-item bibliography is retained for continuity. References differ in evidential role: standards and conceptual foundations, empirical studies, vendor reports, historical adoption reporting, and tool or policy documentation. Inclusion does not imply independent validation of SQCM.

[1] ISO/IEC 25010:2023. *Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - Product quality model.* International Organization for Standardization, 2023. https://www.iso.org/standard/78176.html

[2] ISO/IEC 5055:2021. *Information technology - Software measurement - Software quality measurement - Automated source code quality measures.* International Organization for Standardization, 2021 (developed by CISQ/OMG). https://www.iso.org/standard/80623.html · https://www.it-cisq.org/standards/code-quality-standards/

[3] Letouzey, J.-L. "The SQALE method for evaluating Technical Debt". *Third International Workshop on Managing Technical Debt (MTD 2012)*, IEEE, 2012, pp. 31-36. DOI: 10.1109/MTD.2012.6225997. See also: Letouzey, J.-L.; Ilkiewicz, M. "Managing Technical Debt with the SQALE Method". *IEEE Software* 29(6), 2012, pp. 44-51. https://ieeexplore.ieee.org/document/6225997

[4] Oman, P.; Hagemeister, J. "Construction and testing of polynomials predicting software maintainability". *Journal of Systems and Software* 24(3), 1994, pp. 251-266. DOI: 10.1016/0164-1212(94)90067-1. https://doi.org/10.1016/0164-1212(94)90067-1

[5] Heitlager, I.; Kuipers, T.; Visser, J. "A Practical Model for Measuring Maintainability". *QUATIC 2007*, IEEE, pp. 30-39. DOI: 10.1109/QUATIC.2007.7. Operationalization: *SIG/TÜViT Evaluation Criteria Trusted Product Maintainability.* https://dl.acm.org/doi/10.1109/QUATIC.2007.7 · https://www.softwareimprovementgroup.com/wp-content/uploads/SIG-TUViT-Evaluation-Criteria-Trusted-Product-Maintainability.pdf

[6] CMMI Institute / ISACA. *Capability Maturity Model Integration (CMMI) V3.0*, 2023. https://cmmiinstitute.com/ · https://cmmiinstitute.com/learning/appraisals/levels

[7] Forsgren, N.; Humble, J.; Kim, G. *Accelerate: The Science of Lean Software and DevOps.* IT Revolution Press, 2018. ISBN 978-1942788331. Research program: https://dora.dev/

[8] Inozemtseva, L.; Holmes, R. "Coverage Is Not Strongly Correlated with Test Suite Effectiveness". *ICSE 2014*, ACM. DOI: 10.1145/2568225.2568271. https://dl.acm.org/doi/10.1145/2568225.2568271

[9] Just, R.; Jalali, D.; Inozemtseva, L.; Ernst, M. D.; Holmes, R.; Fraser, G. "Are Mutants a Valid Substitute for Real Faults in Software Testing?". *FSE 2014*, ACM. DOI: 10.1145/2635868.2635929. https://dl.acm.org/doi/10.1145/2635868.2635929

[10] Rahman, F.; Devanbu, P. "How, and Why, Process Metrics Are Better". *ICSE 2013*, IEEE. https://ieeexplore.ieee.org/document/6606589/

[11] Nagappan, N.; Ball, T. "Use of Relative Code Churn Measures to Predict System Defect Density". *ICSE 2005*, ACM, pp. 284-292. DOI: 10.1145/1062455.1062514. https://dl.acm.org/doi/10.1145/1062455.1062514

[12] Bird, C.; Nagappan, N.; Murphy, B.; Gall, H.; Devanbu, P. "Don't Touch My Code! Examining the Effects of Ownership on Software Quality". *ESEC/FSE 2011*, ACM, pp. 4-14. DOI: 10.1145/2025113.2025119. https://dl.acm.org/doi/10.1145/2025113.2025119

[13] Avelino, G.; Passos, L.; Hora, A.; Valente, M. T. "A Novel Approach for Estimating Truck Factors". *ICPC 2016*, IEEE. https://arxiv.org/abs/1604.06766

[14] Krasner, H. / CISQ. *The Cost of Poor Software Quality in the US: A 2022 Report.* Consortium for Information & Software Quality, December 2022. https://www.it-cisq.org/the-cost-of-poor-quality-software-in-the-us-a-2022-report/

[15] Pearce, H.; Ahmad, B.; Tan, B.; Dolan-Gavitt, B.; Karri, R. "Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions". *IEEE Symposium on Security and Privacy (S&P) 2022*. https://arxiv.org/abs/2108.09293

[16] Perry, N.; Srivastava, M.; Kumar, D.; Boneh, D. "Do Users Write More Insecure Code with AI Assistants?". *ACM CCS 2023*. DOI: 10.1145/3576915.3623157. https://dl.acm.org/doi/abs/10.1145/3576915.3623157

[17] GitClear. *Coding on Copilot: 2023 Data Suggests Downward Pressure on Code Quality.* 2024. https://www.gitclear.com/coding_on_copilot_data_shows_ais_downward_pressure_on_code_quality

[18] GitClear. *AI Copilot Code Quality: 2025 Data Suggests 4x Growth in Code Clones.* 2025. https://www.gitclear.com/ai_assistant_code_quality_2025_research

[19] GitClear. *The Maintainability Gap: AI Code Quality in 2026.* January 2026. https://www.gitclear.com/the_ai_code_quality_maintainability_gap

[20] Peng, S.; Kalliamvakou, E.; Cihon, P.; Demirer, M. "The Impact of AI on Developer Productivity: Evidence from GitHub Copilot". arXiv:2302.06590, 2023 (preprint). https://arxiv.org/abs/2302.06590

[21] DORA / Google Cloud. *Accelerate State of DevOps Report 2024.* https://dora.dev/research/2024/dora-report/

[22] DORA / Google Cloud. *State of AI-assisted Software Development 2025.* September 2025. https://dora.dev/dora-report-2025/

[23] Fortune. "Google's code is now more than 25% AI-generated, CEO Sundar Pichai says". October 2024. https://fortune.com/2024/10/30/googles-code-ai-sundar-pichai/

[24] CNBC. "Satya Nadella says as much as 30% of Microsoft code is written by AI". April 2025. https://www.cnbc.com/2025/04/29/satya-nadella-says-as-much-as-30percent-of-microsoft-code-is-written-by-ai.html

[25] Semafor. "Google CEO says 75% of company's new code is AI-generated". April 2026. https://www.semafor.com/article/04/24/2026/google-ceo-says-75-of-companys-new-code-is-ai-generated

[26] Liu, Y.; Widyasari, R.; Zhao, Y.; Irsan, I. C.; Chen, J.; Lo, D. "Debt Behind the AI Boom: A Large-Scale Empirical Study of AI-Generated Code in the Wild". arXiv:2603.28592v2, revised April 26, 2026 (preprint). https://arxiv.org/abs/2603.28592v2

[27] Sun, X.; Ståhl, D.; Sandahl, K.; Kessler, C. "Quality Assurance of LLM-generated Code: Addressing Non-Functional Quality Characteristics". Journal of Systems and Software, 2026. DOI: 10.1016/j.jss.2026.112885. Author version: arXiv:2511.10271v2, March 12, 2026. https://doi.org/10.1016/j.jss.2026.112885 · https://arxiv.org/abs/2511.10271v2

[28] OpenSSF. *SLSA - Supply-chain Levels for Software Artifacts.* Open Source Security Foundation. https://slsa.dev/

[29] Open Source Security Foundation. *OpenSSF Scorecard.* https://securityscorecards.dev/

[30] The White House. *Executive Order 14028: Improving the Nation's Cybersecurity.* May 2021 (established software bill of materials expectations for federal software supply). https://www.federalregister.gov/documents/2021/05/17/2021-10460/improving-the-nations-cybersecurity

[31] European Union. *Regulation (EU) 2024/2847 of 23 October 2024 - Cyber Resilience Act,* on horizontal cybersecurity requirements for products with digital elements. https://eur-lex.europa.eu/eli/reg/2024/2847/oj

[32] Google. *OSV - Open Source Vulnerabilities.* Distributed vulnerability database and schema. https://osv.dev/ · https://ossf.github.io/osv-schema/

[33] Joern - Open-source code analysis platform based on Code Property Graphs. Apache License 2.0. https://joern.io/ · https://github.com/joernio/joern

[34] GitHub. *CodeQL - Terms and Conditions.* Licensing conditions depend on the permitted use and applicable agreement; cited as historical context for the instrument selection, not a general statement of current entitlement. https://github.com/github/codeql-cli-binaries · https://docs.github.com/en/code-security/codeql-cli

---