ScaleQuality Code Maturity Model (SQCM)

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

Version 1.0 · July 2026

Author: Erik Fernandes · Founder, ScaleQuality

Contact: erik.fernandes@scalequality.io · scalequality.io

DOI: 10.5281/zenodo.21251863

Executive summary

Organizations of every size have accelerated software production with generative AI. The speed is real and measurable. The ability to trust what gets produced has not kept up. Between late 2024 and early 2026, the share of new code generated by AI at one of the world's largest technology companies jumped from roughly 25% to 75%, according to public statements by its leadership [23] [25]. Over the same period, industry research documented rising code duplication, a steep decline in refactoring, and persistent delivery instability in teams that adopted AI [17][18][19][21][22].

This paper introduces the **ScaleQuality Code Maturity Model (SQCM)**, a model that answers a question existing models do not: **"can I trust this code?"**. SQCM defines *code maturity* as a measurable construct at the repository level, assessed exclusively from **observable evidence** (structure, manifests, version history, tests, automation, and static analysis), with no questionnaires and no self-declaration, in minutes and with zero configuration.

The model evaluates four domains: three inherited from the established software quality tradition (**Security**, **Reliability**, and **Maintainability**, present in both ISO/IEC 25010 [1] and ISO/IEC 5055 [2]) and one novel domain that no prior standard covers: **AI code durability**, the measure of how much AI-generated code survives over time rather than being reworked or abandoned. The verdict is expressed on a five-level ladder (Reactive, Initial, Structured, Managed, Optimized), with per-domain scores, an explicit confidence level for every measurement, and reproducibility guaranteed by score versioning.

SQCM is not a defect detector, and it is not an organizational process model. It is a measurement instrument for the maturity of the code artifact itself, designed for the practical decision teams face every day in the AI era: what can this repository withstand, and what is missing for it to withstand more.

---

1. The problem: code is born faster than the ability to trust it

Three facts, taken together, define the current moment in software engineering.

First: AI adoption in code writing is massive and accelerating. In October 2024, Google's CEO stated that more than 25% of the company's new code was generated by AI and reviewed by engineers [23]. In April 2025, Microsoft stated a figure between 20% and 30% [24]. By April 2026, the figure stated by Google was already 75% of new code, up from 50% the previous fall [25]. These are self-reported numbers, without public methodology, and the definition of "AI-generated" is elastic (it includes accepted autocomplete suggestions). Even so, as a signal of scale and speed of change, they are unequivocal: in 18 months, the declared share tripled.

Second: the speed gain is real. In a controlled experiment run by GitHub and Microsoft researchers, developers with access to GitHub Copilot completed an implementation task 55.8% faster than the control group [20]. It is a single-task study, conducted by interested parties and published as a preprint, but the directional effect has been replicated in the industry's lived experience. SQCM does not start from skepticism about speed: speed is the reason adoption happened.

Third: what gets produced at that speed has a documented risk profile. In a systematic 2022 evaluation, roughly 40% of the programs generated by GitHub Copilot across 89 scenarios built around the most critical weaknesses in the CWE catalog contained vulnerabilities [15]. The scenarios were designed around high-risk patterns and the data reflects the 2021-era models, so the number should be read as a foundational milestone rather than a current rate. More concerning than the rate itself is the behavioral effect documented by Stanford researchers in a study published at ACM CCS 2023: participants with access to an AI assistant wrote significantly less secure code in four out of five tasks while simultaneously **believing more strongly** that their code was secure [16]. The risk is not just worse code. It is worse code accompanied by more confidence.

The economic consequence of poor software quality was known before generative AI: the Consortium for Information & Software Quality estimated the cost of poor software quality in the United States at no less than US$ 2.41 trillion in 2022, of which approximately US$ 1.52 trillion in accumulated technical debt [14]. The figure aggregates distinct categories (operational failures, cybercrime enabled by defects, technical debt) and is an industry estimate, not a peer-reviewed result. But the order of magnitude makes precision unnecessary: code quality is a problem measured in billions, and code production has just been multiplied.

The question that emerges from this landscape is not "does AI write good or bad code?". It is a different, more operational one: **given a concrete repository, today, what can it withstand?** Can it withstand production? Change? The very code AI generated in it last week? That question needs a ruler. The next section shows why the existing rulers do not answer it.

2. Why the existing rulers do not answer "can I trust this code?"

The tradition of measuring software quality and maturity is rich, and SQCM builds on it. But each existing family of models answers a different question from the one at hand.

Product quality models (ISO/IEC 25010). ISO/IEC 25010:2023 defines the current software product quality model, with nine characteristics including Security, Reliability, and Maintainability [1]. It is the field's canonical vocabulary, and SQCM adopts it deliberately. But 25010 is a taxonomy: it says *what* to observe, not *how* to measure it from a real repository, and it does not produce a graded verdict.

Automated source code quality measurement (ISO/IEC 5055). ISO/IEC 5055:2021, developed by CISQ, was the first ISO standard to define quality measures taken automatically from the internal structure of source code, across four characteristics: Security, Reliability, Performance Efficiency, and Maintainability [2]. It is SQCM's most direct methodological ancestor: it establishes that measuring quality straight from code, without human intermediation, is a standardizable practice. Two limits, however: 5055 measures **present quality** (existing structural violations), not the maturity of the practices that produce future quality; and it contains no provision whatsoever for AI-generated code, a phenomenon that postdates its publication.

Maintainability ratings from static analysis (SIG/TÜViT). The Software Improvement Group's practical maintainability model, published academically in 2007 and operationalized as a certification with 1-to-5-star ratings [5], is SQCM's closest commercial precedent: it proves that assigning a **graded score** to code, derived from static analysis and anchored in a standard (ISO 25010), is market-accepted and certifiable. The differences define the space SQCM occupies: the SIG model covers a single domain (maintainability), traditionally operates in an assisted-evaluation context, and has no AI code dimension at all.

Quantified technical debt (SQALE, Maintainability Index). The SQALE method formalized the quantification of technical debt as remediation cost derived from code analysis [3], and is the conceptual basis of widely adopted tools. The Maintainability Index, proposed in 1992-1994, showed that composite indices computed from code have decades of academic tradition [4]. The trajectory of this family, however, points in the direction SQCM follows: single, opaque indices lost ground to multidimensional models with explicit characteristics, exactly the design of ISO 25010 and the SIG model [1][5].

Process maturity (CMMI). CMMI, now at version 3.0, consolidated the idea of maturity in five levels [6]. Its conceptual heritage is undeniable, and SQCM's ladder is indebted to it. But CMMI assesses the **organizational process**: policies, institutional practices, management capability. Two organizations at the same CMMI level can host repositories in radically different states. CMMI answers "does this organization work maturely?"; it does not answer "is this specific code trustworthy?". Moreover, formal appraisals are multi-week investments, incompatible with the decision cadence of the AI era.

Delivery metrics (DORA). The DORA research program demonstrated, with the longest-running longitudinal research of its kind, that observable technical capabilities predict delivery performance and organizational outcomes [7]. SQCM adopts that epistemology (observable capabilities as predictors) but points the lens at the artifact: DORA measures the team's delivery flow; SQCM measures the state of the code that flow produces.

In summary: we know how to measure present code quality (5055), grade maintainability (SIG), quantify debt (SQALE), assess process (CMMI), and measure delivery (DORA). What none of these instruments answers is the composite question the AI era made urgent: **does this repository, right now, with this share of AI-generated code, have the practices and the structure that predict trustworthy behavior over time?** That is SQCM's question.

---

3. The construct: code maturity

3.1 Definition

Code maturity is the property of a repository that predicts its behavior over time: its resistance to incidents, its capacity to absorb change safely, and the rate at which work invested in it survives rather than being reworked.

Three distinctions delimit the construct:

1. **Maturity is not instantaneous quality.** Quality, in the ISO 5055 sense, is a snapshot: the structural violations present in the code today [2]. Maturity is prognosis: it includes the snapshot, but also weighs the practices that determine the next snapshots (the existence and effectiveness of tests, integration automation, change discipline legible in the history). A repository can have few violations today and no practice preventing its degradation tomorrow.
2. **Code maturity is not process maturity.** The object is the repository, not the organization [6]. SQCM's unit of measurement lets a single organization see repositories at different levels, because that is what reality contains.
3. **Maturity necessarily includes the AI-authorship dimension.** In an era when most new code may be machine-generated [25], a maturity model that does not distinguish how generated code behaves over time is measuring the past.

3.2 The four domains

SQCM evaluates four domains. Three are inherited directly from the ISO tradition [1][2]:

- **Security**: does the repository expose exploitable patterns? Does it contain versioned credentials? Do the weaknesses present correspond to known vulnerability categories?

- **Reliability**: is the code's behavior verified by tests? Do critical paths have coverage? Is there continuous integration catching regressions before delivery?
- **Maintainability**: does the structure sustain change? Is there typing, discernible architectural patterns, lint and formatting discipline, a legible commit history?

The fourth domain is the model's original contribution:

- **AI code durability**: of the AI-generated lines introduced into the repository, how many survive? What is the rework rate on them? How does that survival compare to human-authored code in the same repository?

ISO/IEC 5055 defines four characteristics, including Performance Efficiency [2]. SQCM v1.0 deliberately does not include performance: it depends on workload and execution context, and cannot be measured honestly from static repository signals in a zero-config assessment. Section 8 records this boundary as an explicit limitation and future work.

3.3 Why AI durability is a first-class domain

The available evidence, read together, describes a consistent pattern: AI-generated code ages differently.

GitClear's report series, covering hundreds of millions of changed lines between 2020 and 2026, documented: churn (lines reverted or substantially changed within two weeks) growing from 3.1% in 2020 to 5.7% in 2024; duplicated blocks appearing 8 times more frequently in 2024; the share of copy-pasted code rising from 8.3% to 12.3% between 2021 and 2024, overtaking refactored ("moved") code for the first time, which fell from 24.1% in 2020 to 9.5% in 2024 and 3.8% in the 2026 edition; and legacy code maintenance activity dropping 74% [17][18][19]. The due caution applies: GitClear is a commercial code-analysis company, the data is correlational, and the methodology is proprietary. What gives the series weight is its directional consistency across three consecutive annual editions and the specificity of the numbers.

On the academic front, the largest empirical study to date of verifiably AI-authored code in the wild (302.6 thousand commits from five assistants, across 6,299 GitHub repositories) found more than 484 thousand introduced issues, 89.3% of them code smells, with more than 15% of each assistant's commits introducing at least one issue; and, critically for the durability construct, **22.7% of the issues introduced by AI survive to the latest version of the repositories** [26]. The study is a preprint (March 2026), with the corresponding caveat, but its central finding converges with the industry evidence: what AI introduces tends to stay.

On the delivery front, DORA research documented in 2024 that increased AI adoption was associated with a 1.5% drop in throughput and a 7.2% drop in delivery stability [21]. In 2025, with roughly 90% of ~5,000 respondents using AI, the relationship with throughput turned positive, but the negative relationship with stability **persisted**, and the report's central thesis became that AI amplifies the existing system: organizations without quality controls turn change volume into

instability [22]. Reading the 2024-to-2025 evolution faithfully matters: the speed paid off; stability remains the price.

Finally, a systematic review of 108 studies on the quality of LLM-generated code, combined with industry workshops, found that academic research concentrates on security and performance while **maintainability is the most understudied characteristic**, despite being practitioners' declared priority [27]. There is, therefore, a recognized gap between what gets studied and what teams need to measure. SQCM's durability domain is an operational response to that gap.

3.4 How SQCM defines durability

Durability in SQCM is measured as a **survival rate**: of the AI-attributed lines introduced into the repository within an analysis window, the fraction that remains in the current code, compared against the equivalent rate for human-authored lines in the same repository, with per-tool decomposition where the origin is identifiable.

Two design decisions deserve record. First, the metric is always **relative** (rates and proportions, never absolute volumes): the defect-prediction literature established that absolute churn measures are poor predictors, while normalized relative measures predict defect density with high accuracy [11]. Second, AI authorship attribution is heuristic (based on commit signals and contribution patterns) and the model treats it as such: it feeds a domain with an explicit confidence level, never a per-line accusation (section 6 details the confidence treatment).

---

4. The method: from observable evidence to verdict

4.1 Principle: evidence, not declaration

SQCM evaluates only what can be observed in the repository. There is no questionnaire, no self-assessment, no declarative input of any kind in the score's composition. This choice has a cost (there are valuable practices that leave no observable trace) and the model accepts it in exchange for a property no declarative assessment has: **the score does not depend on what the team believes about itself**. The Stanford study cited in section 1 is the argument in one sentence: in the AI era, declared confidence and actual security diverge [16].

The epistemology is the same one validated by the DORA program: observable capabilities as outcome predictors [7]. And every family of signal SQCM reads has published empirical validity:

- **Test signals**: the presence and proportion of tests are informative, but SQCM follows the literature in not treating coverage as a sufficient proxy for effectiveness. The correlation between coverage and suite effectiveness, controlling for suite size, is low to moderate [8]. Coverage therefore enters as a partial signal with explicit confidence, and the presence of **mutation testing** (whose mutant detection correlates with real fault detection [9]) is treated as a higher-tier signal.

- **History signals**: process metrics extracted from version control predict defects better than static code metrics [10]. SQCM reads the history: commit discipline, contribution distribution (whose relationship with failures is documented [12]), knowledge concentration estimable via truck factor [13], and the line-survival signals that feed durability [11].
- **Structural and automation signals**: the presence and configuration of continuous integration, static typing, lint and formatting, containerization, infrastructure as code, and an architectural pattern discernible from the directory structure.
- **Static analysis signals**: security, reliability, and maintainability findings produced by established code and secret scanners, with per-finding provenance recorded (which tool produced which evidence).

4.2 Score composition

Each domain receives a 0-to-100 score, composed from the pertinent signals according to three publishable principles (the exact weights are the model's property and stay out of this paper, for the same reason a commercial appraisal's internal scoring questionnaire is not public):

1. **Severity-weighted, per-domain scoring.** The same finding weighs differently depending on the domain it affects: a high-severity security weakness weighs more in the Security domain than an informational style finding weighs in Maintainability. The weighting is monotonic in the severity reported by the scanner.

2. **Penalty caps per evidence class.** No single class of static analysis findings can, by itself, zero out a domain. The model imposes caps preventing volume of low-severity findings from dominating the verdict.

3. **Deliberate exceptions for critical evidence.** The exception to the previous principle is explicit and documented: the presence of **versioned credentials in the repository** (secrets) lowers the Security domain to the critical band on its own. A repository with exposed keys is not a mature repository, regardless of any other virtue.

Domain scores aggregate into an overall score, and the status bands are public and visible in the product: below 60, critical; 60 to 79, attention; 80 or above, healthy.

4.3 Zero-config as requirement, not convenience

An SQCM assessment starts from a pointer (repository URL, provider connection, or file upload) and produces the verdict in minutes, with no configuration. This is not merely user experience: it is an epistemological requirement of the model. Assessments requiring extensive configuration measure, in part, the quality of the configuration. Assessments requiring weeks measure a repository that has already changed. The decision window of the AI era (in which 18 months tripled the share of generated code [23][25]) demands that measurement fit inside the change cycle it intends to measure.

---

5. The ladder: five maturity levels

The overall score converts into a level from 1 to 5, with names describing the repository's state:

Level	Name	Band	Meaning
1	**Reactive**	< 40	The repository lacks the minimum structures of trust. Quality, where it exists, is individual, invisible effort. Incidents are discovered by users.
2	**Initial**	40 to 59	Fragments of practice exist (some tests, some automation), without systematicity. Behavior under change is unpredictable.
3	**Structured**	60 to 79	The fundamental practices exist and operate: tests with real presence, continuous integration, discernible structural discipline. The repository sustains routine change.
4	**Managed**	80 to 89	Practices are consistent and verifiable across all domains, with strong test and automation signals. The repository sustains change under pressure.
5	**Optimized**	≥ 90	Measurable excellence across the four domains, including advanced verification practices. The repository is a strategic asset, not a risk liability.

The thresholds are published deliberately. They are visible in the product (the score and the level appear together), and hiding them would be false opacity. The conceptual heritage of the five-level ladder comes from CMMI [6]; the difference lies in the object (the repository, not the organization) and in the method (observable evidence, not appraisal).

Each level transition is accompanied, in the product, by the concrete list of what would unlock it: which open risks, which absent practices, which domain is holding the advance back. The ladder is not just classification; it is a prioritized backlog.

---

6. Confidence, unmeasured states, and reproducibility

Three properties of SQCM exist to protect the verdict's honesty, and they are as much a part of the model as the score itself.

Explicit per-domain confidence. Each domain carries a confidence level (high, medium, low) reflecting the density and quality of the available evidence. A repository without accessible git history produces a durability domain with low confidence; AI-assisted readings (such as the directional coverage posture) are always labeled as such and never raise confidence above what deterministic evidence sustains. The motivation comes from the literature itself: if coverage alone does not guarantee effectiveness [8], an honest model must state how confident it is, measurement by measurement.

Unmeasured states do not score. When a domain's evidence is insufficient, the domain is declared unmeasured, and the overall verdict reflects it. SQCM does not interpolate, does not assume averages, and does not convert absence of evidence into a neutral score. Temporal trends are only recorded when all four domains are effectively measured, so the historical series never mixes incomplete measurements with complete ones.

Reproducibility through score versioning. Every verdict records the model version that produced it, and past verdicts are immutable: the model's evolution (new weight versions, new signals) applies to future measurements and never rewrites history. Two runs of the same model version over the same repository state produce the same result. This discipline is what makes it possible to compare a repository's trajectory over time knowing the ruler did not silently move mid-journey.

The role of AI inside the assessment engine itself also deserves record, because the distinction matters: AI is used as a **signal extractor and synthesizer** (for example, in the directional reading of test gaps), always with a confidence label; the score's composition, the weights, the caps, and the thresholds are deterministic. SQCM uses AI to read evidence; it does not use AI to opine without evidence.

---

7. From score to action

A maturity model that ends at the score is a report. SQCM was designed to operate, and three mechanisms bridge measurement and change:

Risks with business impact. Every relevant finding is expressed as a risk, with severity, location, and a business-language translation of impact (what this code pattern can cause, and to whom). A verdict's risk list is the repository's work agenda, ordered by severity.

Continuous measurement with a cut-off line. The point-in-time verdict (the snapshot) can be promoted to continuous measurement (the film): the repository is re-assessed on every change, and the organization defines a **maturity line** below which the delivery pipeline fails the build. Maturity stops being observation and becomes an operational contract. Choosing the line is the organization's call; the model supplies the ruler and the mechanism.

Executable remediation. The risks identified by the verdict feed remediation agents that propose fixes as real pull requests (missing tests, continuous integration fixes, defect corrections), subject to human review. The details of that mechanism are outside this paper's scope; the architecturally relevant point is that the same evidence that produces the score produces the plan, and the same system that measures can execute part of the recovery, with a human in the approval loop.

---

8. Limitations and validation agenda

A model that claims to measure honesty must declare its own boundaries. SQCM v1.0's limitations, and what we intend to do about each:

1. Observable evidence has blind spots. Valuable practices that leave no trace in the repository (design review, pairing, incident management) do not enter the verdict. SQCM measures what the repository shows; organizations needing process assessment should use process instruments [6]. There is no plan to incorporate self-declaration: the blind spot is the price of the principle.

2. Performance is out of scope for v1.0. Of ISO 5055's four characteristics [2], Performance Efficiency is not assessed, because it cannot be measured honestly from zero-config static signals. Future work: incorporating performance signals when connectable execution telemetry exists, under the same explicit-confidence discipline.

3. AI authorship attribution is heuristic. The detection of AI-generated lines relies on commit signals and contribution patterns, and has an error rate. That is why durability operates on aggregate, comparative rates (AI versus human within the same repository), never per-line judgment, and carries explicit confidence. Future work: as generation tools adopt standardized provenance marking, the heuristic will be replaced by verifiable declarative attribution.

4. The v1.0 thresholds are design calibration, not an empirical result. The level bands (40/60/80/90) and the ladder's semantics derive from the five-level tradition [6] and from design experience, not from a proprietary longitudinal study. The validation agenda, in order: (a) correlate, in a cohort of continuously measured repositories, the SQCM level with observable outcomes (defect incidence, subsequent churn, change survival), in line with the prediction literature [10][11]; (b) publish the level distribution across repository populations for public calibration of the bands; (c) revise the weights in light of the results, incrementing the score version and preserving history (section 6).

5. The third-party evidence grounding the durability domain has known biases. The GitClear series comes from a company with a commercial interest in the topic [17][18][19]; the most recent academic studies are preprints [26][27]; the adoption numbers are executives' self-declarations [23][24][25]. This paper cites them with those qualifications because it is the evidence that exists; the agenda in item 4 exists precisely so that SQCM produces, over time, primary evidence of its own.

6. Correlation is not causation, here too. SQCM predicts; it does not prove that raising the score causes the outcomes. The capabilities-to-outcomes relationship the model relies on is the best documented in the field [7], and the same caution DORA declares about its surveys [21][22] applies to this model.

9. Conclusion

Software engineering has crossed scale transitions before: from artisanal assembly to processes (CMMI), from declared quality to quality measured in code (ISO 5055), from delivery intuition to delivery science (DORA). Each transition demanded a new ruler because the previous one measured the wrong question.

The current transition has a clear signature: code is now born faster than the ability to trust it. The existing rulers measure present quality, organizational process, or delivery flow. None answers the question teams, leaders, and investors ask when facing a repository in the AI era: **can this withstand?**

The ScaleQuality Code Maturity Model is our answer: code maturity as a first-class construct, measured by observable evidence, without declaration, in minutes, with the AI code durability dimension no prior standard covers, explicit confidence on every measurement, and a ladder that converts diagnosis into plan. Version 1.0 is the deliberate beginning of a program: measure, publish, validate against outcomes, recalibrate in future versions without rewriting the past.

The model is in operation at scalequality.io. The methodology will remain published, versioned, and open to scrutiny, because that is what separates a ruler from an opinion.

References

[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:2603.28592, March 2026 (preprint). https://arxiv.org/abs/2603.28592

[27] "Quality Assurance of LLM-generated Code: Addressing Non-Functional Quality Characteristics". arXiv:2511.10271, November 2025 (preprint). https://arxiv.org/abs/2511.10271
