ScaleQuality Code Maturity Model (SQCM)

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

Version 3.0 · October 2026

Erik Fernandes · ScaleQuality

Scoring calibrations described: v2.2 (default) and mainframe-v1.0

DOI: 10.5281/zenodo.23106322

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 3.0 extends the model in two directions. First, it introduces scoring profiles and the first profile beyond the default one: **SQCM Mainframe 1.0**, a separate ruler for code estates that are mostly COBOL and job control, with its own domains, metric bands, weights, and level locks, calibrated on nine public COBOL repositories. Second, it documents **license evidence**: how the instrument inventories third-party components, evaluates their licenses against an organization's policy, and produces exportable records. License results remain outside every numerical score. The default profile and its score calibration **v2.2** are unchanged and are carried over from v2.1, which this edition supersedes.

Publication note. This document supersedes SQCM v2.1, DOI 10.5281/zenodo.22781322. The English text is canonical; the Portuguese PDF is its translation. Paper versions identify publications. A run's scoreVersion identifies its scoring calibration. Two scoring sequences now exist: v2.2 for the default profile and mainframe-v1.0 for the Mainframe profile. Publication and scoring sequences remain 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.

Scoring profiles. A scoring profile is a complete ruler: domains, metrics, bands, weights, composition rules, and level rules. A run is scored by exactly one profile, and the run records it through its score version. Profiles share the 0-100 scale and the level bands so that a portfolio can be read on one axis, but scores produced by different profiles are not comparable as measurements (section 5.3). Sections 2 to 6 describe the default profile; section 7 describes the Mainframe profile.

Evidence outside the score. Some evidence supports decisions without entering any score. License evidence is the principal case: whether a license is acceptable depends on how the software is used and on the organization's policy, so the same component can be acceptable for one organization and not for another. Section 8 describes that evidence and why it stays outside the numerical result.

2. The five domains of the default profile

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.
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**; section 8 describes that evidence.

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.
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. Every scoring profile uses the same bands. A profile can add level locks that hold the level below the band its score reaches; the Mainframe profile does (section 7.5). 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.

5.3 Two scoring rulers

A run scored by the Mainframe profile records scoreVersion mainframe-v1.0; a run scored by the default profile records v2.2. The two numbers share a scale and level bands, not a measurement: a mainframe 80 and a default 80 are produced from different domains, metrics, and weights. The comparability rule of section 5.2 applies across profiles exactly as it applies across calibrations of one profile, and the same evolution logic that suppresses a delta across scoring versions treats a profile change as a version change.

A project can change profile when its composition changes, for example when modern code added during a migration comes to dominate the lines read. Its series then restarts under the new ruler. Runs of mainframe projects recorded before the Mainframe profile was calibrated carry no score and are not rescored retroactively.

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.

None of these general-purpose analyzers parses COBOL, JCL, or CICS definitions. Mainframe sources are read by the dedicated reader described in section 7.3.

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. The Mainframe profile (SQCM Mainframe 1.0)

7.1 Why a separate profile

The default profile was calibrated on repositories built with package managers, test runners, CI pipelines, and Git history. A mainframe estate is organized differently. Shared record layouts live in copybooks rather than packages. Programs are started by job control (JCL) and by transaction definitions rather than by an application entry point. Unit tests are rare. Sources are frequently exported from a library manager without Git history.

Scored with the default ruler, such a system receives a number whose inputs mean something else: dependency manifests are absent, test frameworks are absent, and the general analyzers do not parse COBOL. The Mainframe profile replaces those inputs with signals specific to the platform and keeps the shared scale and level bands. Industry analysts expect more than 70% of mainframe exit projects started in 2026 to fall short of their intended benefits, citing overestimation of generative AI [35]. This paper does not test that prediction; the profile measures the estate a migration starts from.

7.2 When the profile applies

The profile applies to a project when every measured repository contains mainframe source and when COBOL code lines, JCL lines, and PL/I lines make up at least half of the lines read in those repositories. COBOL is counted in code lines, with comments excluded; JCL, PL/I, and files with 34 common modern-language extensions are counted in all their lines. Assembler modules are inventoried but not counted. Projects that fail either condition keep the default profile, including mixed estates in which modern code dominates and projects in which one repository has no mainframe source.

7.3 What the reader reads

No public analyzer in the engine understands these formats, so the profile uses a dedicated lexical reader. It reads text. It does not compile, expand copybooks into programs, resolve macros, or execute anything. Fixed-format COBOL is read in the code area of columns 8 to 72, with the column-7 indicator honored for comment, debugging, and continuation lines [36]. Free-format sources are recognized by their source-format directive.

Source	Recognized by	What is extracted
COBOL programs	Common program extensions, or exported members whose header contains IDENTIFICATION DIVISION and PROGRAM-ID	Program identifiers, including nested programs; COPY and EXEC SQL INCLUDE; static and dynamic CALL; CICS LINK and XCTL; declared files and DB2 tables; paragraphs, sections, GO TO, and ALTER; error-handling counts
Copybooks	Common copybook extensions	Which programs and copybooks include each one; a fingerprint of its code
JCL jobs and procedures	Extensions, or a JOB or PROC statement in the header	Steps and the programs they start, procedures, data sets, and passwords written in the JCL
CICS resource definitions	Definition files, or DFHCSDUP input inside a job [37]	Transactions and the programs they start

Source	Recognized by	What is extracted
BMS maps	Map files, or map assemblies inside a job [38]	Mapsets and maps; the symbolic description map generated from a map, which programs copy, counts as present
Assembler and PL/I	Their extensions	Module names, from the main entry label or the file name; inventoried, not analyzed

Entry points are the programs started from outside the code: JCL steps, including the application program named in an IMS region-controller step and a program named by a DB2 RUN PROGRAM command; procedures; and CICS transactions. Reachability is computed from those entry points through static CALL, LINK, and XCTL targets. A call or transfer through a variable is resolved only when the name of a program of the repository appears as a literal constant in the calling program, in a VALUE clause or a MOVE statement. Otherwise it remains dynamic and is counted as such.

A program that no static reference reaches is reported as **unreferenced**, never as dead code, because dynamic dispatch cannot be excluded lexically. With no entry point at all, reachability is not computed and the unreferenced share is unavailable.

A paragraph is counted as **dead** only when control cannot reach it: no PERFORM, PERFORM THRU range, or GO TO names it, its section is not performed, and it cannot be reached by falling through from the start of the procedure division, from a GO TO target, or within a performed section. Fall-through stops at a paragraph that leaves the flow through GOBACK, STOP RUN, EXIT PROGRAM, GO TO, or a CICS RETURN or XCTL. Paragraphs named with an -EXIT suffix are not counted.

System and vendor modules, such as utility programs, Language Environment, CICS, DB2, IMS, and MQ interfaces, and z/OS services, are excluded from the missing-source list by name patterns. Calls between nested programs of one source stay inside that source. **Generated programs** are recognized by generator markers in their headers; they remain in the inventory but are excluded from the metrics of section 7.4. **Unit tests** are recognized only by explicit markers: COBOL Check suites, identified by the .cut file convention of that project [39], and programs that call the IBM zUnit runtime [40]. A file name containing "test" is not evidence of a test.

Each program and copybook also receives a fingerprint of its code area that is stable across comment edits and sequence renumbering. A modernization workflow outside the score uses it to detect which legacy objects changed between two measurements.

7.4 Domains, metrics, and bands

Each metric maps linearly to 0-100 between the value that scores 100 and the value that scores 0, clamped at both ends. A coverage metric read on fewer than five statements does not count. A domain with no countable metric is excluded and the remaining weights are renormalized, as in the default profile.

Domain	Metric	Scores 100 at	Scores 0 at
Reliability	Declared files with FILE STATUS	100%	50%
Reliability	CICS data commands with RESP or NOHANDLE, or in a program with HANDLE CONDITION or HANDLE ABEND	100%	50%
Reliability	SQL statements checked through SQLCODE or SQLSTATE within 15 code lines, or WHENEVER SQLERROR	100%	50%
Modernizability	Referenced names without source in the checkout	0%	25%
Modernizability	Programs no entry point reaches	5%	50%
Modernizability	Programs with a dynamic call or transfer and no resolved target	0%	20%
Modernizability	Entry points known through JCL or CICS definitions	yes	no
Maintainability	GO TO statements per thousand code lines	2	25
Maintainability	Repeated logic, in windows of 8 normalized statements	10%	50%
Maintainability	Dead paragraphs	2%	20%
Maintainability	Programs over 2,000 code lines	0%	25%
Maintainability	Median of the average code lines per paragraph	30	80
Security	Dynamic SQL (PREPARE, EXECUTE IMMEDIATE) per thousand code lines	0	1
Security	Password written in JCL	none	any
Testability	Unit-test sources per program	0.5	0

Reliability averages its three coverages weighted by the number of statements each one read, so 40 SQL statements count more than 5 declared files. CICS data commands are READ, WRITE, REWRITE, DELETE, the browse commands, and the queue commands; transfers such as LINK and XCTL are not counted. **Modernizability** and **Maintainability** are unweighted means of their available metrics. **Security** is zero when a password is written in JCL; otherwise it follows the dynamic SQL band. **Testability** follows the ratio of unit-test sources to programs.

Application programs are the inputs to the error-handling, GO TO, size, paragraph, and ALTER metrics; generated programs and unit-test programs are excluded. Repeated logic is computed over the procedure divisions of application programs, because data declarations repeat by nature when several programs read the same record layout.

The missing-source share divides the referenced names that have no source in the checkout by the distinct names referenced by programs through static calls, transfers, and copybooks. Programs that only a JCL step or a CICS definition names are counted as missing when absent, but are not part of the denominator. The share is therefore conservative and can exceed 100% in a checkout that contains mostly job definitions. Aligning the denominator with every reference kind is planned for profile 1.1.

7.5 Weights, composition, and level locks

Domain	Weight
Reliability	30
Modernizability	25
Maintainability	25
Security	10
Testability	10

The overall score is the weighted mean of the available domains. Security weighs less than in a modern repository because the reader currently has two security signals, and both read zero in every system of the calibration corpus. A domain that measures little would otherwise lift the score without information. Security still acts through a lock.

The level follows the shared bands at 40, 60, 80, and 90, subject to two locks:

Level	Additional condition
3 and above	At most 5% of referenced names without source; no ALTER statement in application programs; no security signal; at least one known entry point
4 and above	The level 3 conditions, at least one unit-test source, and continuous measurement enabled for the project

A system that fails a level 3 condition stays at level 2 whatever its score. ALTER modifies the target of a GO TO at run time, so the control flow cannot be read from the source [42]. It was an obsolete element of the 1985 COBOL standard and was removed in the standard's 2002 revision, although IBM Enterprise COBOL still accepts it [41].

The engine cannot observe whether a project measures continuously. It therefore records the level without continuous measurement, together with the level the project reaches with it, and the service that holds the project configuration applies the second when continuous measurement is enabled. For a mainframe estate, continuous measurement means a pipeline when the code is kept in Git, or a connection that sends each promotion from a library manager. That connection had not been validated against a customer installation at the time of writing. A system measured only from uploaded archives has no continuous measurement and therefore stays at level 3 or below.

AI Code Durability and Supply Chain are not part of this profile. Git history is often absent in mainframe sources, and dependency manifests do not exist. The default profile's practice

risks about missing tests, type layers, and CI configuration are not reported under this profile either, because the corresponding mainframe evidence is read directly.

7.6 Calibration corpus and results

The bands were set on 30 September 2026 using nine public repositories. They were chosen because they are publicly available and contain mainframe COBOL, most of them with JCL, CICS, or DB2. The revisions below were recorded so that the reading can be repeated; the results were reproduced at these revisions on 2 October 2026 with identical scores. Two repositories, a vendor demonstration and a synthetic benchmark, are not named (see below).

Repository	Kind	Revision	COBOL code lines	Programs
aws-samples/aws-mainframe-modernization-carddemo	Demonstration	59cc6c2fd7eb	32,562	44
cicsdev/cics-genapp	Demonstration	f6f3f4b2580d	6,794	31
Vendor demonstration (not named)	Vendor demonstration	withheld	26,083	81
IBM/Bank-of-Z	Demonstration	ff2a59181bd8	23,697	43
etalab/taxe-fonciere	Published production code	6bd40b2d78ea	2,997	6
Synthetic benchmark (not named)	Synthetic benchmark	withheld	6,177	42
walmartlabs/zECS	Published service	6d6bcbbc89c9	2,670	5
IBM/dbb	Samples	ce2c1be58eb0	4,621	6
IBM/zOS-Client-Web-Enablement-Toolkit	Samples	5117ca15e45e	1,319	1

COBOL code lines include copybooks and exclude comments; the corpus totals 106,920 [43]. The taxe foncière code is the property-tax computation published by the French tax administration. zECS is a caching service for z/OS published by engineers of a retailer, without a statement about production use. The vendor demonstration is distributed under a license that restricts its use to demonstrations, and the benchmark is a public repository of synthetic code generated with a language model and published without a license file; neither is named here. The benchmark and the two sample collections were used to exercise the reader; they are not evidence about operating estates. One sample collection, IBM/dbb, has more modern than mainframe lines and would be scored by the default profile under the rule in section 7.2.

System	Rel.	Mod.	Maint.	Sec.	Test.	Score	Level by score	Level
CardDemo	97	85	83	100	0	81	4	2
GenApp	94	100	70	100	0	81	4	3
Vendor demonstration	40	75	88	100	0	63	3	3
Bank of Z	88	63	67	100	0	69	3	2
taxe foncière	n/a	33	100	100	0	62	3	2
Synthetic benchmark	79	65	100	100	0	75	3	2
zECS	100	92	94	100	0	87	4	3
DBB samples	n/a	39	97	100	0	63	3	2
Web Enablement Toolkit	n/a	67	89	100	0	70	3	2

The level shown is the level without continuous measurement, since no public repository in the corpus measures continuously. Several observations shaped the calibration:

- **The bands separate systems for the stated reason.** The vendor demonstration has the lowest Reliability, 40, because 39% of its CICS data commands and 32% of its SQL statements are not followed by error handling. zECS reaches 100.
- **Locks matter more than the mean for a migration decision.** CardDemo scores 81 and stays at level 2: it contains four ALTER statements, and its CICS definitions name six programs whose source is not in the repository. Bank of Z and the taxe foncière code expose no entry point the reader can resolve: Bank of Z ships no CICS definitions and its single job starts a PL/I program, and the taxe foncière code ships no JCL.
- **Missing context lowers Modernizability.** The taxe foncière code calls versions of its computation modules for earlier years whose sources are not in the repository; 59% of its referenced names are missing.
- **Two domains do not yet discriminate.** Security read 100 and Testability read 0 for every system. Both results are correct for this corpus, but their bands cannot be calibrated until systems with those signals are measured.

7.7 What calibration corrected in the reader

Calibration on public code was first a test of the reader. Every number that did not match the source was checked in the file before a rule changed:

Correction	Effect in the corpus
CICS definitions installed by a DFHCSDUP job are read	GenApp: transactions 0 to 25, entry points 0 to 10

Correction	Effect in the corpus
Dead paragraphs honor performed sections and fall-through	Bank of Z: dead paragraphs 59.1% to 9.6%
Copybooks generated from BMS maps count as present	GenApp: one missing copybook to none
Calls between nested programs stay inside the source	Web Enablement Toolkit: 22 missing names to none; DBB: 9 to none
System modules (CICSPlex SM, Workload Simulator, z/OS web enablement, Health Checker, C runtime) are excluded	GenApp: 5 missing programs to none
PL/I sources are inventoried	Bank of Z: a PL/I program called by the code is no longer missing
Generated programs leave the maintainability inputs	DBB: 45 GO TO per thousand lines to 0
Tests require explicit markers	GenApp: 5 files named like tests, which are screens of the application, no longer count

Findings confirmed in the source were kept, for example a CICS STARTBR without error handling in the vendor demonstration, sequential files without FILE STATUS in Bank of Z, and the ALTER statements in CardDemo. Names that remained missing were checked to be absent from the repositories, such as programs defined in CardDemo's CICS definitions and a copybook the benchmark copies but does not contain.

7.8 Limits of the profile

The calibration is provisional. The corpus has nine repositories and about 107,000 COBOL code lines. One contains production code published by a public administration; the others are vendor demonstrations, a published service, a synthetic benchmark generated with a language model, and samples written to teach. Demonstration code can contain deliberate defects and tends to be cleaner than an operating estate of millions of lines. The bands come from judgment and from the distribution of this corpus; no outcome data, such as incidents or migration cost, has been linked to them.

The reading is lexical. COPY REPLACING, preprocessors, and macro expansion are not resolved; names built at run time stay dynamic unless a literal constant identifies them; Assembler and PL/I are inventoried without analysis. Tests are recognized through COBOL Check files and zUnit runtime calls; whether tests written for IBM's newer test tooling keep those calls was not verified. Sources must be readable as ASCII-compatible text, so EBCDIC members are expected to be converted on export. Files over 2 MB and checkouts over 20,000 files are partly skipped.

The security domain has two signals. The profile will be reviewed, as version 1.1, with the first customer estate measured under authorization, together with the missing-source denominator described in section 7.4.

8. License evidence outside the score

8.1 Why licenses stay outside the score

A license obligation is not a code defect. Whether a strong-copyleft or network-copyleft component is acceptable depends on how the software is distributed or offered and on the organization's policy; the same component can be acceptable in an internal tool and not in a distributed product. A score that moved with license findings would make the same code read differently in two organizations, and the instrument would be answering a legal question for the company.

SQCM therefore records license evidence and evaluates it against the organization's own policy, and no domain score of either profile changes because of a license. The code diagnosis also lists two low-severity advisories, one for strong-copyleft components and one for components whose license could not be resolved. They are visible in the risk list and carry no weight in any domain.

8.2 Inventory

The inventory reads the lockfiles found in the repository and lists every resolved dependency, direct or transitive, with its declared license as reported by the deps.dev service [44]. Versions come only from lockfiles. A manifest without a lockfile is reported as such, because its versions, and therefore its licenses, are not fixed. Supported formats are those of the bundled scanner; CocoaPods is not supported.

When the scanner returns no license for a package, the platform queries the package's public registry (npm, PyPI, and Maven Central at the time of writing) and, when needed, the license text shipped with the package. Declared strings are normalized to SPDX identifiers and expressions [45] for known spellings and canonical license URLs. Ambiguous names such as "BSD" are kept as written unless the package's own license text settles them, and references of the form "SEE LICENSE IN" become document-specific LicenseRef identifiers. What remains unresolved stays visible as unknown and is never guessed.

This resolution sends the names and versions of unresolved packages to those public registries; names a registry does not know are cached, so they are not sent again on every run. The scanner itself sends package coordinates to the deps.dev and OSV services to obtain licenses and advisories [32][44].

Files carried inside the repository are examined in two further ways. An SPDX-License-Identifier tag or a recognizable license statement at the head of a tracked file is recorded as a notice. A file identical to a file of a popular npm or PyPI package release is recognized by its SHA-256 digest in an index maintained by the platform; files shared by more than three packages and files under 512 bytes are ignored, so that trivial content does not produce matches.

Each repository keeps up to 25,000 components or 6 MB of inventory, with truncation flagged. The dependency stage, including registry resolution, has four minutes; when it times out, the run records no inventory rather than a partial one.

8.3 Built artifacts and copied code

A CI client can add built artifacts, such as an unpacked image or a distribution directory. They are scanned in the root-filesystem mode of Trivy [46], which reads the metadata of installed packages, and enter the inventory as artifact components with the licenses the scanner reports, without further normalization. Their known vulnerabilities also enter the Supply Chain domain (section 2.4). Binaries without package metadata, such as statically linked native code or firmware, yield no component list, and nothing is claimed about them.

Detection of copied code is optional and off by default. When an organization enables it, the platform runs the SCANOSS client [47] with path obfuscation, so it sends fingerprints of the files, which are hashes of their content, instead of the source text. The operator of the free knowledge base describes snippet identification on that service, while high-precision snippet matching is offered on commercial plans [47]. In a test on 2 October 2026 against the free base, an unmodified file of a widely used library was identified as a whole-file match, while fragments of 120 and 200 lines of the same library embedded in other files were not identified. This paper therefore claims only whole-file identification for the free base. An organization can also point the client to a commercial knowledge base or to its own SCANOSS server, which the platform accepts as a custom HTTPS endpoint. Matches enter the inventory as snippet components.

8.4 Policy, exceptions, and the pull-request check

Each organization has one policy, which assigns every license a decision: allowed, review, or forbidden. Rules are set per family (permissive, weak copyleft, strong copyleft, network copyleft, other, and unknown) and can be overridden per license. Licenses are assigned to families by identifier. That grouping is a platform parameter, not a legal classification; for example, it places source-available licenses and non-commercial Creative Commons licenses with network copyleft. For an expression, OR takes the most favorable option and AND the least favorable; several declared licenses are treated as AND.

The default template allows permissive licenses, sends weak copyleft, strong copyleft, other, and unknown licenses to review, and forbids network copyleft. An organization can evaluate development-only dependencies with the same rules or flag them only when forbidden. Packages under the organization's internal name prefixes are allowed. Only owners and administrators change the policy, and every change is kept in an append-only revision history.

Decisions are recomputed whenever evidence is read. A policy change therefore applies to existing inventories without a new scan. The pull-request result described below is the exception: it is recorded with its run.

An owner or administrator can accept a component under review or forbidden, for one package (one version or all of its versions) or for one license, across the organization or in one project, with a written justification and an expiry of at most 365 days. An accepted component is shown as accepted, never as allowed, and returns to the policy decision when the exception expires.

In continuous measurement, a run that carries an inventory is compared with a baseline run: the latest completed run of the pull request's base branch when one is recorded, otherwise the latest completed run of the project; with no earlier run, every component counts as new. Forbidden components without an active exception that the baseline did not

contain are reported; components under review are never reported by this check. A version change counts as a new component. The default result is a warning. An organization can choose to block, and only then does the CI client fail the build. Without an inventory, no license result is produced. Runs sent by the CI client do not yet record their branch, so the base-branch comparison depends on runs recorded for that branch by other paths; when none exists, the fallback can compare a change with an earlier run of the same change.

8.5 Outputs

For each run, the platform exports an SPDX 2.3 document in JSON [45]; SPDX 2.3 extends the version standardized as ISO/IEC 5962:2021 [48]. The document has a root package per repository and one package per component, carrying its declared license expression and package URL. Concluded license, copyright text, and download location are recorded as NOASSERTION, because they are not observed. It also exports a third-party notices text grouped by license, with each license text included once; copyright lines are not invented when the lockfiles do not carry them. The CycloneDX SBOM of section 6 receives component licenses from the same inventory. A document generated by the production code from a test inventory was parsed and converted without errors by the official SPDX tools (spdx-tools 0.8.5), and a deliberately malformed license expression was rejected; this checks the format, not the completeness of any inventory.

The audit dossier includes, per project, the inventory judged by the policy in force when the dossier is generated, the SPDX document, and the notices. At organization level it includes the policy with its revisions in the period and the exceptions. Each file is listed with its SHA-256 digest in the dossier manifest.

8.6 Limits of license evidence

The evidence reports licenses as declared by package metadata, registries, and file headers. It does not read every license file of every package, establish the license a lawyer would conclude, or determine whether a component is shipped, linked, or used at run time. It is not legal advice. Lockfiles bound what is seen: unlocked manifests, unsupported formats, and binaries without metadata stay outside the inventory. Copied fragments are not claimed for the free knowledge base, and notices come only from file headers.

License evidence can support a license compliance program, for example one organized according to ISO/IEC 5230 [49]. Producing it does not establish conformity with that standard or with any license.

9. Validation and results

9.1 September 2026 implementation audit

The v2.1 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 confirmed the audited deployment snapshot; it did 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 identifiers and fixture outcomes are recorded in the v2.1 supplementary validation note.

9.2 October 2026 additions

The Mainframe profile and the license evidence were examined in the same way as in section 9.1: interface contracts, service calls, persistence, and focused regression tests. The code was read at the production revisions of the two services that implement them on 2 October 2026 (ai-governance `9c00c8187d77`, insights `4a81e4055953`), and the tests below were run on those revisions with type checking disabled in the test runner.

Validation group	Suites	Passing tests
Mainframe reader and profile scoring	2	25
Mainframe level with continuous measurement, and score versions	1	3
License inventory, registry resolution, normalization, notices, file index, and copied-code scanning	8	62
License policy, exceptions, exported documents, and the pull-request check	4	37
Total	15	127

The calibration of section 7.6 was reproduced on 2 October 2026 by reading the nine repositories at the recorded revisions with the production reader. Every domain score, score, and level matched the values recorded on 30 September 2026. On the same day, the copied-code test of section 8.3 and the SPDX format check of section 8.5 were run with the clients and tools named there. As in section 9.1, these are regression checks written and run by the vendor, not an independent validation sample.

Internal source code, exact default-profile coefficients, and proprietary rule catalogs are not released with this paper. The supplement supports traceability but not unrestricted independent reproduction of the full proprietary instrument.

9.3 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 or v3.0. 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.

9.4 Historical detection calibration

The v2.0 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 later editions. 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.

10. Limitations and research agenda

The most consequential open question is predictive validity. Neither the score bands nor the domain weights of either profile 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 Mainframe profile adds the limits of section 7.8, and license evidence adds those of section 8.6.

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. For the Mainframe profile, it adds the measurement of customer estates under authorization, a PL/I reader, more security signals, and the validation of library-manager connections. These are research and implementation goals, not capabilities established by this publication.

11. 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	10.5281/zenodo.22781322
v3.0	Scoring profiles and SQCM Mainframe 1.0; license evidence outside the score	This publication

The default profile's construct, calibration v2.2, and the corrections recorded in v2.1 are unchanged. The contribution of this revision is a second ruler for systems the first one was not built to read, and a documented boundary for evidence that informs decisions without entering a score.

References

The bibliography of v2.1 is retained as items 1 to 34, and items from 35 onward are new in this edition. 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

[35] Gartner. "Gartner Predicts More Than 70% of Mainframe Exit Projects Will Fail Due to Overestimation of Generative AI's Capabilities". Press release, Stamford, June 18, 2026. Cited as industry context; the prediction concerns exit projects initiated in 2026 and is not tested here. https://www.gartner.com/en/newsroom/press-releases/2026-06-18-gartner-predicts-more-than-70-percent-of-mainframe-exit-projects-will-fail-due-to-overestimation-of-generative-ais-capabilities

[36] IBM. *Enterprise COBOL for z/OS 6.5 Language Reference*, "Reference format". https://www.ibm.com/docs/en/cobol-zos/6.5.0?topic=structure-reference-format

[37] IBM. *CICS Transaction Server for z/OS 6.x*, "Defining resources with DFHCSDUP" and "The DFHCSDUP DEFINE command". https://www.ibm.com/docs/en/cics-ts/6.x?topic=resources-defining-dfhcsdup · https://www.ibm.com/docs/en/cics-ts/6.x?topic=commands-dfhcsdup-define-command

[38] IBM. *CICS Transaction Server for z/OS 6.x*, "BMS macro DFHMSD" and "BMS macro DFHMDI". https://www.ibm.com/docs/en/SSJL4D_6.x/reference-applications/bms/dfhp47j.html · https://www.ibm.com/docs/en/SSGMCP_6.2.0/reference-applications/bms/dfhp47g.html

[39] Open Mainframe Project. *COBOL Check*. Apache License 2.0; the repository is archived and the project is in Emeritus status. https://github.com/openmainframeproject/cobol-check

[40] IBM. *IBM Developer for z/OS 14.2*, "ZUnit overview". https://www.ibm.com/docs/en/SSQ2R2_14.2.0/com.ibm.etools.rdz.zunit.doc/topics/c_zunit_overview.html

[41] ISO/IEC JTC 1/SC 22. *ISO/IEC FCD 1989:2001, Information technology - Programming languages - COBOL.* Final Committee Draft, document N3204, January 15, 2001, Annex D.1. https://www.open-std.org/jtc1/sc22/open/n3204.pdf · IBM. *Enterprise COBOL for z/OS 6.5 Language Reference*, "ALTER statement". https://www.ibm.com/docs/en/cobol-zos/6.5.0?topic=statements-alter-statement

[42] Dijkstra, E. W. "Letters to the editor: go to statement considered harmful". *Communications of the ACM* 11(3), 1968, pp. 147-148. DOI: 10.1145/362929.362947. https://doi.org/10.1145/362929.362947

[43] Calibration corpus, read at the revisions listed in section 7.6 for measurement only; no source is redistributed with this paper. License as stated in each repository: aws-samples/aws-mainframe-modernization-carddemo (Apache-2.0), cicsdev/cics-genapp (EPL-2.0), IBM/Bank-of-Z (Apache-2.0), etalab/taxe-fonciere (CeCILL 2.1), walmartlabs/zECS (Apache-2.0), IBM/dbb (Apache-2.0), IBM/zOS-Client-Web-Enablement-Toolkit (Apache-2.0); a vendor demonstration under a demonstration-only license and a synthetic benchmark published without a license file, which are not named. https://github.com/aws-samples/aws-mainframe-modernization-carddemo · https://github.com/cicsdev/cics-genapp · https://github.com/IBM/Bank-of-Z · https://github.com/etalab/taxe-fonciere · https://github.com/walmartlabs/zECS · https://github.com/IBM/dbb · https://github.com/IBM/zOS-Client-Web-Enablement-Toolkit

[44] Google. *OSV-Scanner* documentation, "License scanning". https://google.github.io/osv-scanner/usage/license-scanning/ · Open Source Insights. *deps.dev API v3.* https://docs.deps.dev/api/v3/

[45] Linux Foundation and its Contributors. *The Software Package Data Exchange (SPDX) Specification, Version 2.3.* 2022. https://spdx.github.io/spdx-spec/v2.3/

[46] Aqua Security. *Trivy* documentation, "Rootfs" and "License scanning". https://trivy.dev/docs/latest/guide/target/rootfs/ · https://trivy.dev/docs/latest/guide/scanner/license/

[47] SCANOSS. *scanoss.py*, MIT License. https://github.com/scanoss/scanoss.py · SCANOSS documentation, "Platform Architecture Overview". https://docs.scanoss.com/en/latest/introduction/platform-architecture-overview · Software Transparency Foundation. *OSSKB.* https://www.softwaretransparency.org/osskb

[48] ISO/IEC 5962:2021. *Information technology - SPDX® Specification V2.2.1.* International Organization for Standardization, 2021. https://www.iso.org/standard/81870.html

[49] ISO/IEC 5230:2020. *Information technology - OpenChain Specification.* International Organization for Standardization, 2020. https://www.iso.org/standard/81039.html · https://openchainproject.org/license-compliance

---

ScaleQuality Code Maturity Model v3.0 · © 2026 Erik Fernandes Amaral · Licensed CC BY 4.0. Internal scoring weights of the default profile, detection heuristics, and the taint specification remain ScaleQuality property. The Mainframe profile's weights and bands are published in section 7 because they are also shown in the product. This document describes the methodology at the conceptual level.