Certance.

V2026.6 · OCTOBER 2026

# The Certance Standard

The quality engineering standard for agentic software development: clean code, classical testing levels, hard gates and the operating model, for teams where AI writes most of the code.

**Audience:** engineers, quality engineers, engineering and quality leaders, risk and internal audit

**By:** Certance Advisory · certance.eu

DOCUMENT CONTROL

| | |
|---|---|
| **Status** | Current. Checked against the sources listed in the companion document in October 2026. Next review by March 2027, or earlier if a cited regulation or source changes. Every change is listed in the document history. |
| **Companion document** | The Certance Standard: Evidence and Guidance holds the evidence behind the rules, the regulatory map, worked examples and the adoption plan. Numbering runs across both documents: Chapters 1 to 14 and Appendices A and B here, Chapters 15 to 18 in the companion document. The Standard can be applied without the companion document. |
| **Rights** | © 2026 Certance sp. z o.o., trading as Certance Advisory. Licensed under Creative Commons Attribution-NoDerivatives 4.0 International (CC BY-ND 4.0): you may copy and share this Standard in full, or quote its clauses unchanged, crediting Certance Advisory; you may not publish an altered version. Certance also permits firms to copy the templates in Appendix B and fill them in for their own use. Licence text: creativecommons.org/licenses/by-nd/4.0. |
| **Name** | Certance™ and The Certance Standard™ are trademarks used by Certance sp. z o.o. A firm may state that it conforms to the Standard as Using this Standard sets out, or name the clauses it applies. It may not imply that Certance has certified or endorsed it, or call an altered version The Certance Standard. |
| **Cite as** | Certance Advisory. The Certance Standard.<br>Version 2026.6, October 2026. certance.eu/certance-standard. |

# CONTENTS


The Standard on one page . . . 4

Using this Standard . . . 5

PART I · FOUNDATIONS

01 Principles . . . 6

02 The change: five hard gates . . . 7

PART II · CODE AND TESTS

03 Clean code when agents write it . . . 10

04 Unit tests and test data . . . 11

05 Integration and contracts . . . 12

06 End-to-end and business journeys . . . 13

07 Non-functional quality . . . 15

PART III · PIPELINE AND RELEASE

08 Continuous integration gates . . . 17

09 Release and production . . . 18

PART IV · PEOPLE AND AGENTS

10 The quality engineer and developer operating model . . . 20

11 Running the agent roster . . . 21

PART V · AI INSIDE THE PRODUCT

12 Testing AI inside the product . . . 24

PART VI · EVIDENCE AND THE RELEASE DECISION

13 Evidence for supervisors . . . 27

14 Release and residual risk . . . 28

APPENDICES

A The Daily Rules · gate ownership and evidence · self-check . . . 31

B Adopter's templates . . . 35

REFERENCE

G Glossary . . . 39

S Sources . . . 40

H Document history . . . 41

# The Standard on one page

The Certance Standard sets out how a team proves the quality of its software, with evidence others can check, when AI writes most of the code and runs inside the product. This page states it in twelve statements: the five principles of Chapter 1 and seven requirements that the later chapters develop. Each statement links to the section that explains it. Chapter 14 sets when a release is tested enough and who accepts the risk left, and the Daily Rules in the Appendix gather what a team applies every day.

PRINCIPLES · CHAPTER 1

| | | |
|---|---|---|
| 1 | **AI is an amplifier of your engineering system.** Harden the system before scaling the agents. | 1.1 · p. 6 |
| 2 | **Every merge rests on behavioural evidence.** A green pipeline shows only that the tests passed, and the tests may have moved with the change. | 1.2 · p. 6 |
| 3 | **Every AI-assisted change carries its own proof and a named owner.** Five hard gates on each: provenance, test integrity, behavioural evidence, contract check, blast-radius match. | 1.3 · p. 6 |
| 4 | **Every change declares its provenance.** Every change traceable to its author, human or agent, with the access it held at the time. | 1.4 · p. 7 |
| 5 | **People keep the judgement.** Humans scope the journeys and design the coverage; agents implement. "Seems fine" is a stop signal. | 1.5 · p. 7 |

REQUIREMENTS · CHAPTERS 2 TO 14

| | | |
|---|---|---|
| 6 | **Production is cheap; verification is the asset.** Size your process for the review queue. | 3.2 · p. 10 |
| 7 | **Most of the cost of code comes after it is written.** Accept generated code only at a maintenance cost the business can carry. | Ch. 3 · p. 10 |
| 8 | **The specification is an artefact.** The human owns what must be true; the agent owns the implementation; a model never validates its own work. | 4.1 · p. 11 |
| 9 | **Agents are governed infrastructure.** Isolation, least privilege, configuration as code, output reviewed like third-party code. | Ch. 11 · p. 21 |
| 10 | **Quality is the evidence you can hand to someone else.** A colleague, a customer or a supervisor must be able to check it. | Ch. 13 · p. 27 |
| 11 | **Evidence must outlive the sprint.** Stored, linked to its release build, and kept for a retention period the firm sets and records. | 13.1 · p. 27 |
| 12 | **AI inside the product is tested with the journey it serves.** Evaluation sets are protected tests, pass rules hold over repeated runs, and every model change goes through the gates. | Ch. 12 · p. 24 |

# Using this Standard

### What it covers

The Standard sets rules for software that AI helps to write, test, run or decide: the code and tests agents write, the agents themselves, and AI features inside the product. The rules apply to every change, whoever made it. A rule that covered only declared AI work could be avoided by not declaring it. The Standard is engineering guidance and gives no legal advice or compliance opinion.

### How to read the rules

Most rules start with a number in blue, such as `6.2.1`. The release exit criteria in 14.1 are numbered by tier, such as `T1.3` for the third Tier 1 criterion, and the five gates in 2.1 are G1 to G5. The text that opens a chapter or section, before its first numbered rule, can also set a rule; cite it by the chapter or section number, such as 12.5. Notes, examples and figures explain the rules.

### The firm's test policy

Some rules leave a value to the firm, such as a performance budget, a retention period or a review cap. The firm's test policy sets these values, and B.1 lists them all. The policy may be stricter than the Standard. It is never weaker for a Tier 1 journey.

### Claiming conformance

A firm conforms when every rule is met for the systems and teams in scope, or has a deviation approved as B.1 sets out. Five rules allow no deviation: G1 Provenance; security scans on AI-assisted changes, 7.1; the testing rule, 1.6; risk acceptance by an owner outside the team that built the change, 14.2; and the Tier 1 exit criteria, 14.1, which a test policy can never weaken. Every deviation has an owner and an end date. The conformance statement names the edition, the scope and every open deviation. Someone outside the team that built the systems in scope checks it against the evidence in A.2, and the statement says whether that person works inside or outside the firm. The self-check in A.3 is a quick first look; conformance rests on the evidence.

### Citing a clause

Write Certance Standard 2026.6, clause 6.2.1. For a passage without its own number, cite its section, such as 14.2. Rule numbers stay fixed within an edition, and the document history records any change a later edition makes.

### Two documents

This Standard holds the rules and can be used on its own. A second document, The Certance Standard: Evidence and Guidance, explains why each rule exists, maps the rules to EU and UK regulation and sets out how to adopt them. Its chapters are numbered 15 to 18, and every reference to it says "in the companion document".

# 01 Principles

Five principles set how the Standard decides. All five answer one problem: an assistant can produce changes faster than a team can verify them, Ch. 15 of the companion document. Where no specific rule covers a situation, the principles decide it.

1. **AI is an amplifier of your engineering system**, 1.1
2. **Every merge rests on behavioural evidence**, 1.2
3. **Every AI-assisted change carries its own proof and a named owner**, 1.3
4. **Every change declares its provenance**, 1.4
5. **People keep the judgement**, 1.5

> 1.6 ONE RULE APPLIES TO ALL TESTING
>
> **No agent or tool may weaken a test to make it pass.** When a test fails, the code gets fixed. A test that has been changed, skipped or deleted to get a green build is worse than a failing one, because nobody can see that the protection has gone. This rule applies across all testing work in the Standard. Four controls enforce it: gate G2, Test integrity, 2.1; test-first delivery and mutation score, Ch. 4; protected paths and the re-run rule, Ch. 8; and the limits on healer agents, Ch. 11.

## 1.1 AI is an amplifier of your engineering system

**AI multiplies existing engineering conditions.** The payback on AI-assisted development depends on code review and on the strength of the engineering system around the tools. Teams with disciplined review, version control and quality engineering standards get faster and stay safe. Teams without them ship more defects, and ship them sooner. Harden the system before scaling the agents.

## 1.2 Every merge rests on behavioural evidence

**A passing pipeline shows only that the current tests passed.** When an assistant edits code and tests in the same change, the tests can move with the code, and a green run then no longer shows that the behaviour is right, Ch. 15 of the companion document. Evidence has to show what the code does: run output, a trace, a before-and-after demonstration, or a new test that fails on the old code. An agent's report that the task is done does not count. Gate G3, Behavioural evidence, enforces this on every AI-assisted change, 2.1.

## 1.3 Every AI-assisted change carries its own proof and a named owner

**An AI-assisted change must arrive with its own proof, and a named person answers for it.** When a developer directs the agent, that developer answers, Ch. 10. When an agent works on its own, such as a background agent opening a pull request or a healer fixing a test overnight, its named owner on the agent roster answers. Agents nobody owns are retired, so no change arrives without someone accountable for it, Ch. 11. A change without proof does not merge, Ch. 2. In review of human-written code, trust in a familiar author often stands in for evidence. An agent cannot earn that trust: it carries no accountability for outcomes, and it learns from past incidents only what someone writes into its instruction files, Ch. 3.

### 1.4 Every change declares its provenance

**Every change declares whether and where AI assisted.** That declaration is what governance stands on: attribution is what lets a team route review by risk, measure what the tools change, and run an honest postmortem when something ships wrong. **Assume AI took part unless the change says otherwise:** a change that reaches review with no provenance declaration does not merge until its author adds one, and until then it is reviewed as an AI-assisted change. This holds at every tier, because gate G1 cannot be waived.

### 1.5 People keep the judgement

**Humans scope the journeys and design the coverage; agents implement.** Whoever builds a change, person or agent, never approves it or accepts the risk it leaves, Ch. 2 and 14.2. **"Seems fine", "looks good to me" or an AI reviewer's approval stops the review, and nothing merges on it**, Ch. 8. Each signals pattern-matching: recognising the shape of sound code without verifying what it does. To continue, the approver explains the change to the explainability bar, Ch. 3, or asks for the evidence G3 requires, Ch. 2. The scenario in Chapter 15 of the companion document shows the failure: a change judged sound from the code side, with a wrong structure change inside it.

## 02 The change: five hard gates

Every AI-assisted change passes five gates before it merges:

- **G1 Provenance:** the change declares whether it is AI-assisted, which tool was used, and which parts the tool produced.
- **G2 Test integrity:** the tests were not changed to fit the code.
- **G3 Behavioural evidence:** the change shows what the code does, beyond a passing pipeline.
- **G4 Contract check:** changed data structures still match what their consumers expect.
- **G5 Blast-radius match:** the change touches only what it says it touches.

The table in 2.1 sets out each gate in full. G1 applies to every change, with or without AI, and is never waived. G2 to G5 can be waived only by the approver named for that gate, who must be someone other than the author or the requester of the change. The approver records the reason in the exception log. A waiver on a Tier 1 path also needs the release owner's approval, 14.2, because it moves risk onto customers of that journey. Every month a reviewer outside engineering, such as the risk function, leads a review of the log, and the engineering lead attends to explain each entry. Appendix A.2 lists who enforces each gate and who may approve an exception.

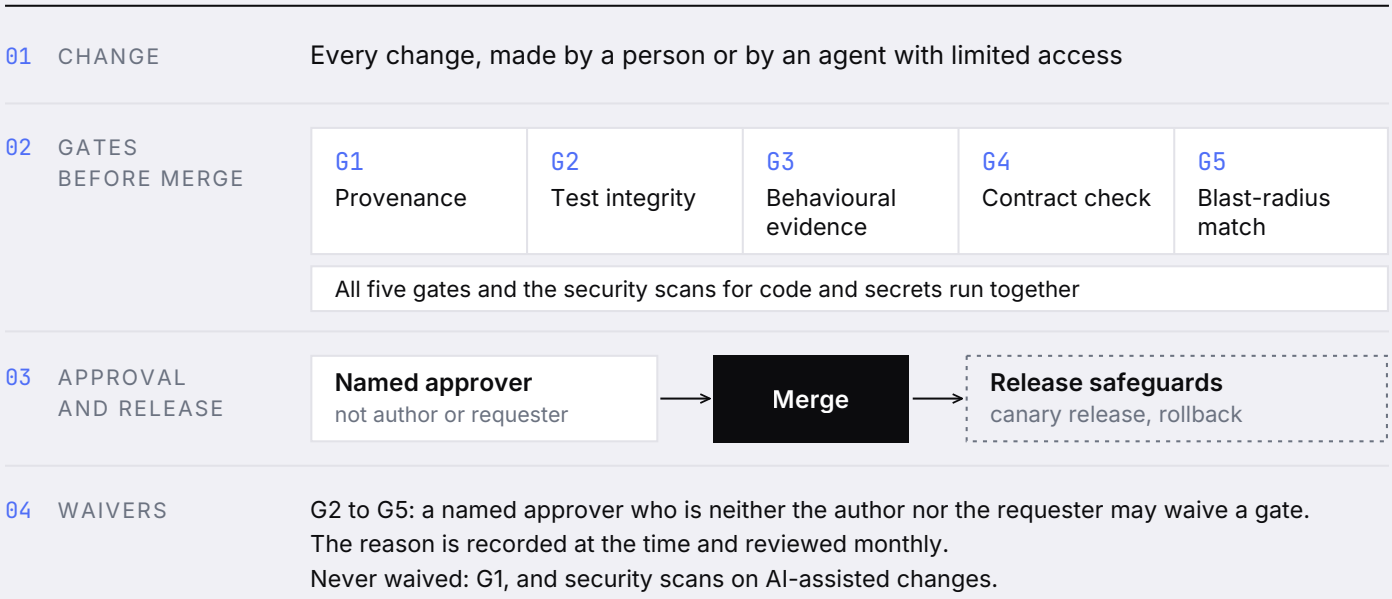



Figure 2.1 · How a change reaches merge. G1 checks every change; G2 to G5 apply to every AI-assisted change.

The gates run together, in the pipeline and in review, beside the security scans of Chapter 7, and a person then approves the merge. The merge approver is neither the author nor the requester of the change, and for agent work the person who directed the agent counts as its author, 1.3. For a change on a Tier 1 path, the change approver also sits outside the team that implemented it. Release safeguards follow the merge, Ch. 9.

## 2.1 The gates

| GATE | RULE | WHY IT EXISTS |
|---|---|---|
| **G1 · Provenance** | The change declares: AI-assisted yes or no, which tool, which parts. A pull-request template field is enough, and a pull request opened by a background agent already declares itself through its author account. Platforms can make the declaration checkable, for example with a commit tag for AI-assisted contributions or agent commits that name the person who assigned the task. Where the platform's audit log marks agent actions, it cross-checks the declaration, 11.1. Trivial tool use does not count as AI assistance: spelling fixes, identifier completion, mechanical renames and reformatting. Anything a tool wrote or restructured does count. Cannot be waived. | Attribution enables measurement, routing and honest postmortems |
| **G2 · Test integrity** | An assistant that modified production code must not silently modify tests, fixtures or expected data in the same change. Same-change test edits are flagged and reviewed as their own question: did behaviour legitimately change, or did the test move to make the change pass? | Self-consistent changes defeat the purpose of a test suite |

| GATE | RULE | WHY IT EXISTS |
| --- | --- | --- |
| **G3 · Behavioural evidence** | The change carries proof of behaviour beyond continuous integration (CI) status: run output, a trace, a before-and-after demonstration, or a new test failing on the old code. The evidence carries the build identifier and the date it was produced. | "Tests pass" is weakest exactly when tests moved with the change |
| **G4 · Contract check** | Any change to a serialised structure (JSON, Avro, Protobuf, API payload, event shape, file format) is validated against its consumers before merge. Schema validation minimum; consumer-driven contracts where consumers are known. | Catches a breaking structure change before it reaches a consumer |
| **G5 · Blast-radius match** | The artefact inventory matches the stated intent. "Refactor function X" that also touches a schema fails by definition. New dependencies count as artefacts: a package added outside the stated intent fails the same way, see phantom dependencies, Ch. 3. | Agents cross boundaries humans mentally separate |

# 03 Clean code when agents write it

**Most of the cost of code comes after it is written.** Every later change, incident and audit has to read it again. An agent makes the first version cheap and leaves that later cost with the business, so the decision that matters at review is whether the business can afford to maintain what merges. The rules in 3.2 set that bar.

Clean-code discipline assumed the person who wrote a line had thought about it. When an agent writes the line, the reviewer is the first person to think about it. Agent-written code is held to the same quality as code written by hand. The engineer's work shifts: into bounding what the agent is asked to do, 3.2, and into selecting what survives review, 3.1 and 16.3 in the companion document.

## 3.1 The explainability bar

3.1.1 **No line merges that its owner cannot explain: why this line, why here, and what breaks without it.** "It works" is not an explanation. An approver who cannot answer those questions withholds approval until they can. Automation bias erodes this rule first, so check it on every review.

## 3.2 Rules

3.2.1 **Reject working code that is not maintainable.** Working is only the entry condition. Code is read again under incident pressure, so accept only code that an engineer outside the change can follow.

3.2.2 **Give the agent small, bounded tasks.** Name the files. State the non-goals ("do not touch tests, fixtures or schemas"). A safe agent task is one whose diff can be narrated in full. The bound matters most for background agents: a task handed to an unattended agent is the last decision made before review, so scope it for a run that nobody will watch.

3.2.3 **Review and approve a plan before any change.** Ask the assistant for its intended change list and hold it to the explainability bar: what it changes, what it must leave untouched, and what would make it wrong. A plan that merely "seems fine" triggers the stop signal of Principle 1.5. The plan stage is where a wrong approach is cheapest to catch, before any code exists, so reject a flawed one here. Approval sets the direction. It does not vouch for the result, so the behavioural evidence in G3 still has to hold, Ch. 2. File the approved plan with the change record.

3.2.4 **Write the conventions into the repository's instruction files, and enforce them in continuous integration (CI).** Keep the conventions, banned patterns and code structure in instruction files that are versioned and reviewed like code. Keep the file short and specific: write down the conventions an agent would not guess, and leave out general overviews of the repository. An instruction file guides the agent and cannot force it, so anything that must hold belongs in a CI gate, Ch. 8.

3.2.5 **Hold the review-time floor: AI diffs get more minutes per line than human diffs.** Equal review time for AI diffs signals rubber-stamp approval.

3.2.6 **Size the work to the review queue.** The test policy sets, for each tier, the largest agent-written change a reviewer is given and the number of open agent-written pull requests each reviewer may hold, B.1. Work above the cap waits until a reviewer is free.

# 04 Unit tests and test data

Unit tests are the first barrier, and an agent can weaken them without anything turning red. Coverage records which lines ran and says nothing about whether an assertion would fail if the code were wrong, so 4.3 measures a suite by its mutation score, the share of small deliberate code changes the suite catches. Tests generated from agent-written code encode what the code does, bugs included. The human-written specification in 4.1 closes that gap, and 4.2 lists the four ways AI-written tests fail.

## 4.1 The human writes the test before the agent writes the code

4.1.1 **Write, or explicitly approve, the failing test before the agent writes the code that passes it.** The human owns the specification and the agent owns the implementation. An agent that writes both the code and its tests will validate its own mistakes. Tests generated after the code are derived from the code; they verify what the AI already produced, and the team can no longer tell intentional behaviour from incidental.

4.1.2 Mechanical enforcement: commit the tests before the agent implements, then `git diff tests/` after the agent's change. An empty diff shows that no test file was touched. Extend the check to test configuration and snapshot files, because an agent that cannot edit a blocking test can still skip it in the configuration or regenerate the snapshot it fails against. This turns gate G2, Test integrity, 2.1, from a review judgement into a check a pipeline can run.

## 4.2 The four traps of AI-written tests

1. **Tests that assert nothing.** They execute code and pass with any implementation. Judge them by their assertions: a coverage number cannot show what a test checks.
2. **Shared blind spots.** The same model wrote the code and the tests; both miss the same edge. Ask for edge cases before asking for test code, and add at least one hand-written adversarial case per module.
3. **Over-mocking.** Every boundary mocked until no real behaviour is exercised. Mock only real boundaries, such as external services and the system clock. A defect a mock hides surfaces later, in integration, where it costs more to find, Ch. 5.
4. **Test deletion.** The quietest failure: an agent removes or skips the test that blocked it. The `git diff tests/` check exists for exactly this.

## 4.3 Rules

4.3.1 **Anchor generated tests to the codebase.** Name the framework and its version in the repository's instruction file, and point the agent at one neighbouring test file as the style reference.

4.3.2 **Keep tests on observable behaviour.** Reject any test that mirrors private implementation details.

4.3.3 **State properties as well as examples on Tier 1 calculation, parsing and validation code.** A person writes the property, such as "a transfer between two accounts leaves their total unchanged"; property-based testing and fuzzing then generate the inputs an agent's examples leave out.

4.3.4 **See every generated test fail once.** Break the code or invert the assertion. A test never seen to fail gives no evidence that it can catch a defect.

4.3.5 **Measure suite health by mutation score, and gate on it.** Report it upward and retire line coverage from the same reports. In every pull request, run it on the changed code and block the merge if the score on that code drops against the previous run; run the full suite nightly and trend the result on a dashboard. No single score is a pass mark: the gate is no regression.

## 4.4 Test data

The rules in this section apply at every test level, to people and agents alike.

4.4.1 **Use synthetic data by default.** Build it from an edge-case catalogue drawn from defect history and production metadata: field formats, value distributions, currency and status mix and volumes, with no record content.

4.4.2 **Never store production records as test fixtures, in any form.** Raw, masked and pseudonymised extracts all count, and the exception below covers a single test occasion and never becomes a stored fixture: removing names still leaves account numbers, payment references, amounts and dates that can identify a customer.

4.4.3 **Treat any use of production data as a time-limited exception decided outside the test team.** The data protection officer and the chief information security officer, or the firm's equivalents, approve it in writing with its purpose, scope, environment, controls and end date. It is recorded in the exception log and in the test data exception record, B.3, reported to the firm's information and communication technology (ICT) risk management function, and deleted at the end date.

4.4.4 **Limit what an agent may read and send.** An agent works on synthetic data and code only, and holds no production data, 11.1. The agent roster records, for each agent, which data classes it may read and what its model vendor may keep or use from prompts and outputs under the contract. The data protection officer and the compliance function decide whether those vendor terms are acceptable.

# 05 Integration and contracts

Integration tests against real dependencies expose the over-mocked AI tests of Chapter 4, 5.1. Data contracts make a breaking structure change, like the example in Chapter 15 of the companion document, impossible to merge silently, 5.2. When an agent changes both sides of a boundary in one task, each side gets its own review, 5.3.

## 5.1 Real boundaries

5.1.1 **Test every component at least once against its real dependency:** the real database engine, the real queue, the real HTTP surface (a containerised instance counts). Agents default to mocking everything because mocks make tests pass quickly. A component whose tests never touch a real boundary is untested at this level, whatever its coverage says.

## 5.2 Data contracts

5.2.1 **Give every serialised structure that crosses a boundary (API payloads, events, files, shared JSON) a contract.** Use JSON Schema as the minimum, and consumer-driven contracts (each consumer records what it expects from the provider) where consumers are known. Contracts are verified in continuous integration (CI) on both sides, so a breaking structure change cannot merge silently, whoever or whatever authored it. Additive changes pass; renames and removals fail until every consumer agrees.

## 5.3 Rules

5.3.1 **Validate the schema of every serialised structure in the repository, internal ones included.** Internal consumers break the same way external ones do.

5.3.2 **Record consumer-driven contracts for every boundary with a known consumer.** The provider build fails when it breaks a recorded expectation.

5.3.3 **Treat contract files as protected paths, Ch. 8.** An agent may propose a contract change and only a human may approve it. Keep the versioning rules (additive changes are safe, breaking changes are versioned) with the contract files.

5.3.4 **Review each side of a boundary on its own.** When an agent changes both sides in one task, treat it as two changes with two reviews. An agent that writes both sides of a boundary checks its work against itself, the same weakness gate G2, Test integrity, 2.1, guards against in unit tests.

# 06 End-to-end and business journeys

End-to-end (E2E) tests cost the most to run and maintain of any functional tests, so they belong on the journeys the business depends on. Left to choose, an agent scripts whatever is easiest to reach. Developers implement journey tests with agents, Ch. 10, and 6.2 sets out who owns the inventory of journeys and their tiers. Coverage is reported per journey, with what each test asserts, 6.1.

## 6.1 Journey tiers

| TIER | JOURNEYS | FAILURE CONSEQUENCE |
|---|---|---|
| **Tier 1** | Money movement, transaction execution, credit and lending decisions, regulatory reporting, authentication and step-up verification | Direct financial loss or regulatory breach |
| **Tier 2** | Dashboards and balances, statements and history, notifications, onboarding | Silent data corruption or client trust collapse; the tier most easily passed off as covered: a suite can verify the page renders without checking the numbers |
| **Tier 3** | Internal operations views, search, third-party feed integrations | Operational friction |

Coverage reporting at this level is a journey table: journey, tier, covered or partial or uncovered, and what "covered" asserts. A single percentage cannot carry those four facts. A Tier 2 dashboard test that only asserts the page loads is partial, and the table says so.

| JOURNEY | TIER | COVERAGE | WHAT THE TEST ASSERTS |
|---|---|---|---|
| Card payment | 1 | **Covered** | The payment reaches the processor and the balance falls by the amount |
| Sign-in with a passkey | 1 | **Uncovered** | No end-to-end test |
| Monthly statement | 2 | **Partial** | The page loads; the figures are never checked |

| JOURNEY | TIER | COVERAGE | WHAT THE TEST ASSERTS |
|---|---|---|---|
| Transaction search | 3 | **Covered** | Results match the filter |

A constructed example. It shows the format and describes no real firm.

Tiers classify journeys. To rank coverage gaps that compete for remediation effort, score each on two five-point scales and multiply them: business consequence, how much a silent failure on that journey costs, and detection delay, how long it would run before anything noticed.

| SCORE | CONSEQUENCE OF AN UNDETECTED FAILURE | DETECTION DELAY WITH NO TEST GUARDING IT |
|---|---|---|
| **5** | Direct financial loss, regulatory breach or harm to clients | Silent for weeks or months; no monitoring covers the path |
| **4** | Significant reputational damage or loss of client trust; escalations likely | Days; a user has to notice and report it |
| **3** | Service degraded for some clients; operations affected | Hours; an alert or a user report follows within the day |
| **2** | Internal friction or edge cases; quickly recovered | Minutes; error pages or obvious breakage, likely monitored |
| **1** | Cosmetic; no material business impact | Instantly; monitoring catches this class of failure |

6.1.1 **Read the product against fixed bands:** 20 to 25 Critical, 12 to 19 High, 6 to 11 Medium, 1 to 5 Low.

6.1.2 **Never band a Consequence 5 gap below High.** The product still sorts the list: Consequence 5 with Detection Delay 2 scores 10, bands High, and ranks below any score of 20 or more.

6.1.3 **Route every Consequence 4 or 5 gap to its named risk owner outside delivery, whatever its score,** 14.2.

6.1.4 **Keep likelihood in its own column, on its own five-point scale:** 1 rare, with no known trigger; 3 possible, with a trigger seen in testing or elsewhere; 5 expected, with a trigger that occurs in normal use within a release cycle. Detection delay is elapsed time until a failure becomes visible and absorbs no likelihood. The scales are ordinal: the product ranks gaps and means no probability.

6.1.5 **Use likelihood for two things only:** to order work among gaps with the same score, and to inform the owner who accepts the risk, 14.2. It never changes a score, a band or the floor.

A modest-consequence gap that would go undetected for a quarter can outrank a high-consequence gap an alarm would catch in minutes. The model ranks dashboard-accuracy gaps high: the consequence is moderate, but nothing alarms when a displayed number is wrong.

## 6.2 Rules

6.2.1 **Give quality engineering ownership of the journey inventory and its tiers.** It writes them with product and, where it applies, risk, and reviews them quarterly. Agents never add, remove or re-tier journeys.

6.2.2 **Make Tier 1 journeys assert outcomes.** The balance changed correctly, the order reached the venue, the report contains the right figures. A screen that renders proves none of them.

6.2.3 **Make Tier 2 tests assert that the data is correct.** "The dashboard shows the sum of these fixtures" is an assertion; "the dashboard shows" only checks that something rendered.

6.2.4 **Limit end-to-end tests to the journeys in the inventory, and keep them stable.** Use deterministic waits, accessible locators and seeded state. Flakiness at this level erodes the only signal leadership sees.

6.2.5 **Test authentication journeys as Tier 1.** They include passkey and multi-factor sign-in steps, and they are testable: test tooling provides virtual authenticators, so "we cannot test login" never justifies leaving the journey untested.

6.2.6 **Explore every new Tier 1 and Tier 2 flow before release.** A person runs a time-boxed exploratory session against a written charter that names the journey and the risks from 6.3. It finds behaviour nobody specified, the blind spot that generated code and generated tests share. Record the charter, the findings and the tests the findings produced.

### 6.3 Risk analysis for each change

The tiers rank the journeys that exist. A change brings risks of its own, and an agent writes tests for what it was asked to build, which can leave those risks without a test. The developer and the quality engineer list them before the tests are written.

6.3.1 **Write a short risk analysis for every change that touches a Tier 1 or Tier 2 journey.** Name the journeys it affects, what could fail, and what the failure would look like to a customer if nothing alarmed. Score each risk on the 6.1 scales, with likelihood in its own column.

6.3.2 **Trace every risk to the tests that cover it.** Record each risk's coverage state, covered, partial or uncovered, with the tests behind it. A Consequence 4 or 5 risk with no passing test stays open into the release decision, 14.1.

6.3.3 **Let agents propose risks, and keep the list with the quality engineer.** The quality engineer owns the analysis, and the developer's tests answer it, 10.1.

6.3.4 **File the analysis with the change.** It stays with the pull request or work item, and the release record lists every risk still untraced, 14.1.

## 07 Non-functional quality

Agents change the volume and provenance of code, while load, latency and attackers behave as they always have. The non-functional levels keep their classical shape, with agentic-era adjustments to security, performance, accessibility and resilience.

### 7.1 Security

7.1.1 **Run static application security testing (SAST) and secret scanning on every change, and never waive them on AI-assisted changes.** A single finding may be dismissed only with a recorded reason (false positive, used in tests, mitigated, or accepted risk), approved by the named security role. Keep the platform's bypass lists empty or limited to emergency-access accounts, and check that one dismissed finding does not approve the same finding in later changes. AI-authored code receives security gating at least as strict as third-party code.

7.1.2 For agent systems themselves, use the Open Worldwide Application Security Project (OWASP) Top 10 for Agentic Applications as the scoping reference, read with the OWASP Top 10 for Large Language Model (LLM) Applications. The gates in Chapter 2 and the roster rules in Chapter 11 are the controls a team applies to the risks these lists name.

## 7.2 Performance

7.2.1 **Keep the classical performance instruments: budgets on critical paths, load tests on Tier 1 journeys, and a baseline against which regressions are measured.** Performance assertions belong in continuous integration (CI) where the journey allows it.

## 7.3 Accessibility

7.3.1 **Run automated accessibility checks in CI, and give every new user-interface flow a human pass.** The automated checks catch the mechanical faults, and the human pass covers the rest. Accessible locators in end-to-end tests double as an accessibility signal: when a test cannot find an element by role, a screen reader cannot either.

## 7.4 Resilience

7.4.1 **Test that every Tier 1 journey survives the failure of what it depends on, and recovers within a time the test policy sets.** Fault-injection and failover tests make the failure happen on purpose: a dependency that times out, returns errors or disappears, a database that fails over, a queue that backs up. Error handling, timeouts and retries are code an agent writes and the happy-path tests never exercise, so they need tests of their own. Run the failure tests on every change to a Tier 1 journey's error handling, timeouts or dependencies, and on every release build, 14.1.

7.4.2 Where the firm is subject to operational resilience rules, 17.1 in the companion document, the resilience owner sets the scenarios and the impact tolerances. Engineering supplies the test results: which dependency failed, how the journey behaved, and how long recovery took against the tolerance. Whether the results show the firm can remain within its tolerances is for the resilience owner and the compliance function to decide.

# 08 Continuous integration gates

Chapter 2's gates depend on reviewer discipline, and discipline fades under load. This chapter moves as much of the gate set as possible into pipeline mechanisms that fire without anyone needing to remember them.

## 8.1 The gate set

The Gate served column names what each mechanism backs: G1 to G5 are the five hard gates defined in Chapter 2, and a chapter reference marks a mechanism that serves that chapter's controls as a whole.

| MECHANISM | WHAT IT ENFORCES | GATE SERVED |
|---|---|---|
| Schema and contract jobs | No serialised-structure change merges without validation against consumers | G4 |
| Protected paths | Fixtures, schemas, contract files, migrations, expected-data files, evaluation sets of AI features, agent definitions and settings, and pipeline configuration require a second explicit approval when changed alongside production code; enforced by code-ownership rules in the repository | G2, G4 |
| Diff heuristics | Flag or block artefact-boundary crossing, file counts beyond stated intent, ownership-boundary crossing, new dependencies outside stated intent, and changes to both sides of a contract boundary in one change | G5 |
| Test-diff check | `git diff` on tests, test configuration and snapshot files against the pre-implementation commit; non-empty means the flag from G2 fires | G2 |
| Security scanning | Static application security testing (SAST) and secret scanning run on every change and cannot be waived on AI-assisted changes; a single finding is dismissed only with a recorded reason | Ch. 7 |
| Dependency check | Every new package is verified to exist in the registry and to be on the approved list before merge; an unknown or newly registered package blocks the change | G5, Ch. 3 |
| Software bill of materials | Every release build produces a software bill of materials (SBOM) listing every component and its version, with a signed attestation of the source and process that produced the build; a component that has not passed the dependency check blocks the release | G5, Ch. 7 |

| MECHANISM | WHAT IT ENFORCES | GATE SERVED |
|---|---|---|
| Mutation gate | Mutation score on changed code in every pull request, blocking a drop against the previous run; the full suite nightly, trending on a dashboard | Ch. 4 |
| Risk-tiered review routing | High blast radius, protected paths or low provenance confidence route to a senior reviewer automatically; the test policy may set stricter review by path, such as two security reviewers on authentication files | Ch. 3 |
| Roster evaluation | Any change to an agent's instructions, tools, permissions or model pin re-runs the roster's evaluation set; a regression blocks the change | Ch. 11 |

## 8.2 Rules

8.2.1 **Use AI review assistants as pre-filters, and keep a named person as the approver.** AI review assistants produce agent output, which earns the same suspicion as any other, so the human whose name is on the approval still has to meet the explainability bar, Ch. 3. An AI reviewer also increases the risk of automation bias, so its verdict is recorded as assistance in the G1 Provenance declaration. An AI review never counts as an independent review.

8.2.2 **Turn every gate that can be a pipeline job into one.** Agents produce more change each week than manual discipline can keep up with.

8.2.3 **Protect the pipeline configuration.** It is itself a protected path: an agent that can edit CI can remove every gate in this chapter in a single change.

8.2.4 **Treat a pipeline that passes only on a re-run as failed.** A re-run shows only that the failure did not repeat, so the change waits until someone records why the first run failed, and automatic retries are capped so nothing merges by repetition. Quarantine an unreliable test only with a named owner and a deadline, and count it as failed in release exit status while it is out, 14.1. By the deadline the test is fixed, or deleted because the behaviour it checked no longer exists, with the reason approved under gate G2. Off Tier 1 paths the owner may extend the deadline once, with a signature. On a Tier 1 path only the release owner may extend it, and the Tier 1 exit criterion stays Not met until the test is fixed and passes. Block fixed waits added to steady a flaky test (one that passes and fails on the same code): they weaken the test silently, which gate G2 exists to stop, Ch. 2 and Ch. 11.

8.2.5 **Keep the pull-request run within the time the test policy sets.** Agent volume multiplies runs, so when the full suite would exceed that time, run on the pull request the tests the change reaches, and run the full suite at least nightly and on every release build. Give each change its own short-lived test environment, and record in the test policy any system that cannot run in one. Use virtual services for dependencies that cannot run there, kept honest by the contract checks in 5.2.

# 09 Release and production

Every earlier barrier can miss a defect, and this chapter is built on that assumption. Feature flags and canary releases limit how many users a missed defect reaches, and automatic rollback reverses a release when an agreed threshold is breached, 9.1 and 9.3. Every incident then feeds back into the system, 9.2.

## 9.1 Progressive delivery

9.1.1 **Define thresholds and rollback conditions before the rollout starts.** A threshold set during an incident is chosen under pressure and is easily loosened.

9.1.2 Feature flags separate deploy from release: code is deployed switched off and verified in production infrastructure. A configuration change then turns it on for progressively larger groups, and turns it off as fast when it goes wrong. Canary releases expose the new version to a small share of live traffic first. The canary is judged against a baseline captured from the stable version: error rate, response time for the slowest requests (the 95th and 99th percentiles), and saturation, meaning how close the system is running to its limits.

## 9.2 Production as a feedback instrument

9.2.1 **Instrument production so every incident can be traced.** Metrics show what changed, traces show where it happened and logs help explain why. Every incident feeds two artefacts back into the system: a regression test at the lowest test level that would have caught it, Ch. 4 to 6, and a postmortem entry that records whether AI authored the change and which chapter's barrier failed. Postmortems name the layer that failed and leave individuals out.

## 9.3 Rules

9.3.1 **Release Tier 1 journeys behind flags with canary stages.** Make no exceptions for "small" changes, because gate G5, Blast-radius match, 2.1, exists to check that a change described as small really is small.

9.3.2 **Roll back automatically on a guardrail breach.** Auto-rollback is the last mechanical gate, designed on the assumption that every earlier barrier failed.

9.3.3 **Add an AI-attribution field to the incident taxonomy.** Without it, incidents cannot be compared by provenance, 10.3.4.

# 10 The quality engineer and developer operating model

When agents write most of the code, the work that needs people shifts to deciding what to verify, designing the barriers and reading the evidence. Both roles change with it. The developer turns the business requirement into an executable specification, the tests written before the agent writes the code, and answers for every merged line. The quality engineer owns what gets verified and how, and turns the evidence into risk language for leadership, 10.1. Record these assignments so every verification task has a named owner.

## 10.1 Who owns what

| CONCERN | DEVELOPER | QUALITY ENGINEER |
|---|---|---|
| Unit specs and code | Owns specs (test-first) and implementation via agents; narrates every merged line | Audits spec quality; runs mutation sampling; teaches the four traps |
| Integration and contracts | Implements against contracts | Owns the contract suite and its versioning rules |
| Journeys and end-to-end tests | Implements journey tests, agent-assisted | Owns the journey inventory, the tiers and what "covered" asserts. Developers and their agents write the journey tests to that design |
| Gates and pipeline | Works within them | Designs them, tunes them, and prepares the exception log for the monthly review led outside engineering, Ch. 2 |
| Independence of test evidence | Does not re-run or review their own Tier 1 journey tests for release | Arranges the re-run or review of Tier 1 journey tests by someone outside the team that built the change, 14.1; Tier 2 and Tier 3 tests are reviewed by someone other than their author |
| AI features in the product | Implements the feature, its prompts and its tools; runs the evaluation set on every change | Owns the AI feature inventory, the evaluation sets and their graders, the pass rules and the red-team plan, Ch. 12 |
| Evidence upward | Provides per-change evidence | Translates it: journey coverage tables and risk language for managers and leadership |

## 10.2 Pairing with developers and their agents

10.2.1 Quality engineers review the prompts and agent instruction files as well as the code those agents produce. A rule in an instruction file shapes every later change the agent makes, so the instruction file is a primary quality engineering artefact and deserves more review than any single test case.

## 10.3 Team disciplines

10.3.1 **Track review time against diff size, by provenance.** This anti-rubber-stamp metric is reviewed monthly; converging AI and human review times are the early warning. Until the measure is automated, sample a dozen merged AI-assisted changes each quarter and record the review time each received.

10.3.2 **Keep the team's skills sharp.** Each quarter, rotate hand-written in-depth tests on critical paths; run assistant-free exercises on the systems the team must reason about unaided. The risk is atrophy as well as wrong code: a team that stops reasoning about its systems unaided loses the ability to review what its agents produce.

10.3.3 **Teach the named failure modes.** Reward hacking, self-consistent changes, shallow assertions, automation bias, blast-radius blindness. Engineers spot in review the failure modes they can name, and read past the ones they cannot.

10.3.4 **Measure delivery outcomes, split by provenance.** Track the five delivery measures of Google's DevOps Research and Assessment programme: change lead time, deployment frequency, failed deployment recovery time, change fail rate and deployment rework rate. Add escaped defects, the defects found in production that a test level should have caught, 9.2. Report each for AI-assisted and human-only changes separately. The split shows a team whether AI is raising throughput at the cost of stability.

# 11 Running the agent roster

This chapter treats agents as infrastructure: agent infrastructure meets the same reliability and testing standards as any internal production system. In practice that means configuration held as code, least privilege and a named owner for every agent, 11.1, and an evaluation that every harness change must pass before it lands, 11.2.

This Standard assumes AI is already present across the organisation: in the code, the tests, the delivery pipeline, the product and the software suppliers provide. This chapter governs the agents that write code and tests. Chapter 12 governs AI inside the product customers use.

## 11.1 Rules

11.1.1 **Hold every agent's configuration as code.** The harness is code: agent definitions, models, system instructions, tool lists and permissions live in the repository, versioned and changed by review.

11.1.2 **Give each agent the least privilege its task needs.** Each agent gets the narrowest tool, credential and network set its task needs: what it may delete, whom it may contact, and where it may publish. Development agents never hold production access. An agent that implements code gets read-only access to the tests, and on a Tier 1 journey at least one acceptance check sits where that agent can neither read nor edit it. Read-only test access is necessary and insufficient on its own: gate G3, Behavioural evidence, and review catch the rest, 2.1.

11.1.3 **Treat agent output as third-party code.** Review it with the same suspicion, scanning and gates as third-party code, or stricter, 7.1.

11.1.4 **Separate the agent that makes a change from the one that approves it.** The agent that generates a change does not approve it. The agent that heals a failing test does not decide the test was wrong. A heal that changes what a test asserts, or which element or data it checks, is a test edit: a person reviews it under gate G2 even when the change touches no production code, and the record states what the test checked before and after. A heal that only updates how a test finds the same element is listed in the run log. Cross-checks between agents reduce noise, and the final judgement stays with a person, 8.2.1.

11.1.5 **Keep the roster inventory current, owned and pinned.** Pin model versions and review the pins on vendor updates. Keep an inventory of which agents exist, what each may touch and who owns it, and retire agents nobody owns, 1.3.

11.1.6 **Keep a reviewable record of every agent session.** Agent sessions produce reviewable artefacts: plans, diffs, run logs, and where tooling supports it, annotated recordings of what the agent did.

11.1.7 **Export the platform's agent audit logs and check them against the G1 Provenance record.** Export the logs into the firm's own retention, because platforms keep them for a limited time, and use them to check the G1 field: an agent-authored change declared as "no AI" is a finding. Protect the agent definitions held in the repository with push rules, so the roster inventory matches what runs. The G1 field remains the record for AI assistance the platform cannot see, such as an engineer working with an editor assistant.

11.1.8 **Record each agent's runtime controls before it joins the roster.** The other rules govern an agent before it runs. For each agent, the roster inventory records which controls act while it runs, such as policy hooks that block a tool call, live interception and a kill switch, and which the vendor does not offer.

11.1.9 **Run every coding agent in an isolated environment.** Its network is off or limited to an allow-list the test policy sets, it holds no production access and no customer data, and no secret sits anywhere it can read. Isolation sets the damage an agent can do when it goes wrong or is misled.

11.1.10 **Keep approval-bypass modes switched off.** A setting that lets an agent act without a person's approval, such as auto-approve or skipping permission checks, stays off on every machine and runner that holds credentials. Agent settings files are protected paths, 8.1, so changing one needs a second approval. Deleting data, rewriting version-control history, publishing and contacting customers each need a person's approval every time.

11.1.11 **Treat everything an agent reads as untrusted.** Issues, web pages, documents, dependency files and tool output can carry instructions written by someone else. An agent session that reads such content holds no secrets, no production access and no permission to change the pipeline, the tests or its own settings.

11.1.12 **Give every agent a way to stop and report.** Each harness lets the agent stop and report a task it finds contradictory or impossible. The stop is recorded, goes to the person who assigned the task, and counts as a pass in roster evaluation, 11.2. After the number of failed fix attempts the test policy sets, the agent hands the task back to a named person.

11.1.13 **Reserve the decisions for people.** An agent may propose a change, run tests, retry within the cap and propose a quarantine. Merging, waiving a gate, re-tiering a journey, releasing and accepting risk always need a named person, and every agent action appears in the session record, 11.1.6.

## 11.2 Evaluating the roster

Chapter 8 gates the code an agent writes, and this section gates the agent itself. The unit under evaluation is the harness and the model together, so a score reported for a model says little about that model inside a team's own harness.

11.2.1 **Re-run the evaluation set on every harness change.** Any edit to an agent's instructions, tools, permissions or model pin re-runs the roster's evaluation set before it lands, and a regression blocks it the way a failing test blocks code.

11.2.2 **Build the evaluation set from the team's own failures.** Start with twenty to fifty real tasks from this repository, drawn from what agents have got wrong. Admit a task only when two engineers would reach the same pass or fail verdict on it.

11.2.3 **Judge agents on the team's own codebase; public benchmark scores say little about it.**

11.2.4 **Grade the outcome, including the diff.** A run passes only if the task is done and the resulting change clears G2 and G5, Blast-radius match, 2.1: no test moved, nothing touched outside the stated scope. Include bait tasks whose easy path is to weaken a test, and tasks that cannot be done as specified; the pass condition is that the agent declines, or stops and reports, 11.1.12, as the testing rule, 1.6, requires.

11.2.5 **Model judges assist, and a person makes the final call.** Calibrate each judge as 12.2.3 sets out, give it an "unknown" answer that routes to a human, and never let it be the only gate, 8.2.1.

11.2.6 **Keep every evaluation run.** Logs and scores, stored with the harness change they tested, are the record that each harness change was evaluated before it went into use.

# 12 Testing AI inside the product

This chapter governs AI features inside the product: a support assistant, an advice tool, a model that scores credit or flags fraud. It reuses the journey tiers and scoring of Chapter 6 and the release decision of Chapter 14, and adds what behaviour that changes from one run to the next requires.

## 12.1 Inventory and tiers

**List every AI feature customers use or that acts on their accounts, and tier it with the most critical journey it can act on.** An AI feature is any part of the product whose behaviour comes from a trained model: generated text, a score, a ranking or a decision.

12.1.1 **Record each AI feature in the journey inventory, 6.2.** For each: the journey and its tier, the model and its pinned version, the prompts, retrieval sources and tools it uses, the named owner, and the supplier where the model is bought in, including a model embedded in a vendor's product.

12.1.2 **Re-tier the feature when what it can do changes.** An assistant that explains a statement sits with Tier 2, the tier of statements and history; once it can start a payment, it sits with Tier 1. A model that decides or prices credit sits with Tier 1, as credit and lending decisions do in 6.1; a model that flags fraud takes the tier of the journey its decision acts on.

12.1.3 **Score AI feature gaps on the 6.1 scales, and expect long detection delays.** An AI feature fails by giving a fluent, plausible wrong answer, with no error page and no alert. Unless a production monitor, 12.5, watches for that failure, score its Detection Delay at 4 or 5.

12.1.4 **Keep the inventory with quality engineering.** It writes the inventory with product, risk and, where the firm has one, the model risk function. Agents never add, remove or re-tier AI features.

## 12.2 Evaluation sets are tests

**Give every AI feature an evaluation set, and protect it like a test suite.** An evaluation set holds the inputs, the expected behaviour or grading rule for each, and the quality each case checks: accuracy, answers grounded in the firm's own sources, refusal of requests outside the feature's scope, required disclosures, and consistent treatment across customer groups. 11.2 sets the same discipline for the agent roster; this section applies it to the product.

12.2.1 **Hold evaluation sets in version control as protected paths, 8.1.** A change to the feature's prompt, model or retrieval sources that also edits its evaluation set needs the second explicit approval protected paths require, and the reviewer treats the evaluation-set edit as its own question, as gate G2, Test integrity, does for tests, 2.1.

12.2.2 **Build the set from real failures.** Start with complaints, incidents, defects and cases staff overrode, add synthetic cases for the edges, and keep production customer records out of it, 4.4.

12.2.3 **Name the grader for every quality.** Use code-based checks where an exact rule exists, a model-based grader where judgement is needed, and human review of a sample. Before a model-based grader gates a release, and again whenever its model changes, calibrate it against human verdicts on a labelled sample: report chance-corrected agreement, such as Cohen's kappa, swap the order of the answers it compares to test for position bias, and run it at least three times on the same sample. A grader below the agreement the test policy sets, B.1, does not gate.

12.2.4 **Test relations as well as answers.** Metamorphic tests check that a rephrased question gets a consistent answer, and back-to-back tests compare the new version with the one in production; adversarial and drift tests follow in 12.4 and 12.5.

## 12.3 Pass rules over repeated runs

**State every release criterion for an AI feature over repeated runs.** The same input can pass on one run and fail on the next, so a single passing run shows the feature can behave and says little about whether it will.

12.3.1 **Run each evaluation case several times and write the pass rule into the test policy, B.1.** For a Tier 1 feature, a case passes only when every one of its k runs passes, written $\text{pass}^k$. Set the number of runs from the failure rate the risk owner will accept: with no failure in n runs, the failure rate is below 3 in n at 95% confidence, the statistical rule of three. Accepting at most 1 failure in 100 takes 300 clean runs across the evaluation set; 1 in 20 takes 60. For Tier 2, the policy sets a pass rate over k runs. Record k, the settings that affect variation, and the model version with every result.

12.3.2 **Set the thresholds when the release is planned.** A threshold lowered on release day is a deviation, and the acceptance record shows it as one, 14.1 and 14.2.

12.3.3 **Block a drop against the version in production.** A fall on any quality the evaluation set checks for a Tier 1 feature, 12.2, on the same evaluation set, blocks the release as a mutation-score drop blocks a merge, 8.1.

12.3.4 **Report results in customer terms.** "In four runs each of 50 balance questions, 3 answers gave a wrong figure" tells a risk owner how many customers would see a wrong number; an average score hides it.

## 12.4 Adversarial testing

**Attack every customer-facing generative feature before anyone else does.** Red teaming is structured adversarial testing: people and tools try to make the feature leak data, ignore its instructions, give advice it must not give, or act outside its permissions, including through prompt injection.

12.4.1 **Red-team before first release and after every model change.** The red team sits outside the team that built the feature.

12.4.2 **Automate the known attacks in continuous integration.** Include instructions hidden in the documents, emails and web pages the feature retrieves, attempts to extract other customers' data, and requests to reveal the feature's own instructions.

12.4.3 **Test the feature's permissions as well as its words.** A feature that can call tools gets the least privilege of 11.1, and a test shows that a request to use a tool outside its set fails.

12.4.4 **Log every finding with a severity until it is closed.** A finding above the severity the test policy sets as blocking stops the release, 14.1.

## 12.5 Model change and drift

**Treat a change to the model, its version, the prompts, the retrieval sources or the tools as a code change.** It goes through the gates G1 to G5 of Chapter 2 whether a person or an agent made it, and gate G5, Blast-radius match, covers every journey that uses the same model.

12.5.1 **Pin model versions, suppliers' included.** Where a supplier can change the model behind a feature, require notice of the change in the contract, and re-run the evaluation set before traffic moves. Where the supplier can neither pin nor notify, score the feature's Detection Delay at 5 until a monitor under 12.5.3 would catch the change, 12.1.

12.5.2 **Release model changes behind flags with canary stages, 9.1.** Judge the canary on evaluation scores as well as error rates.

12.5.3 **Monitor live behaviour for drift.** Grade a sample of live outputs against the evaluation rules, and track complaint tags, staff overrides, refusal rates and shifts in what customers ask. The time the monitor takes to raise an alert is the feature's Detection Delay, 12.1.

12.5.4 **Feed every production failure back into the evaluation set.** Each one becomes a case, as each incident becomes a regression test in 9.2.

## 12.6 Evidence for the model risk and compliance functions

**Hand the model risk and compliance functions the evidence, and leave the verdict with them.** Engineering supplies the inventory, 12.1; the evaluation sets, results and grader calibration, 12.2 and 12.3; the red-team findings and their closure, 12.4; the change and monitoring records, 12.5; and the acceptance record, B.2. Each is stored as A.2 sets out. Two questions go to those functions before a Tier 1 release: does any law or regulation that applies to the firm class the feature as high-risk, and does the firm's model risk framework count it as a model? Section 17.4 of the companion document sets out how the EU and UK rules frame both questions.

The regulatory conclusion belongs to the compliance function. Engineering can provide an evidence assessment of the release process and its test records to support it.

# 13 Evidence for supervisors

Quality is the evidence someone else can check: a colleague, a customer, an auditor or a supervisor. Every gate leaves a record, and the rules below keep each record stored, linked to its release build and retained, so it can be handed over. Chapter 17 of the companion document maps the records to the EU and UK obligations they relate to.

## 13.1 Rules

13.1.1 **Write the evidence question into the gate itself.** Each gate emits an artefact with a default store and a build link, listed in A.2. Change a default only by recording the new store in the test policy, B.1. A gate that produces no durable artefact gives a supervisor or auditor nothing to inspect later.

13.1.2 **Set a retention period for gate evidence, record it and name its owner.** Align it with the firm's existing retention for change records, and record the period and its owner in the test policy, B.1. Export evidence from any platform that keeps it for less than that period: pipeline logs that expire in ninety days will not answer a question asked in month eight.

13.1.3 **Put every AI coding tool to the compliance function for a decision on how the firm records it.** Give it what it needs to decide whether, and where, the tool belongs in the firm's records of third-party services: the vendor, what the tool does, who uses it and on which systems.

13.1.4 **Never claim a compliance position.** Engineering produces evidence; the compliance and legal functions determine what it means.

13.1.5 **Keep the qualifiers.** Quote a requirement with its qualifiers, such as "where feasible"; the person being quoted to has read them.

13.1.6 **Fix vulnerabilities at the speed they are found.** Treat the security scans in 7.1 and the dependency check in 8.1 as the intake for remediation, set a remediation deadline for each severity, and send every fix through the gates: an agent-written fix is an AI-assisted change and passes G1 to G5.

13.1.7 **Treat source code as sensitive data when it leaves the firm.** Record, with the tool in the firm's records of third-party services, which repositories each AI tool may read, whether the vendor keeps or trains on that code, and which content exclusions apply. Agents get repository access under the least-privilege rule in 11.1.

13.1.8 **Govern the supply chain the agents draw on.** For agent-written code, the dependency check in 8.1 is the engineering control, because assistants invent package names that attackers then register; the compliance function decides whether it is enough.

13.1.9 **Keep evidence tamper-evident.** Store gate and release evidence where any later change shows, such as an append-only or signed store, and so it can be retrieved by release build.

13.1.10 **Trace any component to the releases that contain it.** When a vulnerability is reported in a component, the team finds every release and environment that contains it, from the software bill of materials, 8.1, within the time the test policy sets.

The regulatory conclusion belongs to the compliance function. Engineering can provide an evidence assessment of the release process and its test records to support it.

# 14 Release and residual risk

Chapters 2 to 9 decide whether a change may merge and how a release reaches users. This chapter decides when a release has been tested enough and who accepts the risk that is left. The test lead agrees exit criteria for each journey tier when the release is planned and marks each one Met or Not met against the release build, 14.1. The risk left at release is stated in customer terms and accepted by an owner outside the team that built the change, 14.2.

## 14.1 Exit criteria by journey tier

| TIER | EXIT CRITERIA, EACH MARKED MET OR NOT MET |
|---|---|
| **Tier 1** | T1.1 Every Tier 1 journey test passed on its first attempt on the release build.<br>T1.2 No quarantined, skipped, disabled or retried test on a Tier 1 path.<br>T1.3 No open defect of severity 1 or 2 on a Tier 1 journey.<br>T1.4 The fall-back was rehearsed against the release's own schema and data in a production-like environment.<br>T1.5 Load tests on Tier 1 journeys stayed within the performance budgets the test policy sets, 7.2 and B.1.<br>T1.6 The security scans and the dependency check show no open finding above the severity the test policy sets as blocking, 7.1 and 8.1.<br>T1.7 Automated accessibility checks pass on every changed Tier 1 user-interface flow, and the human pass on each new flow found no issue that stops a user completing the journey, 7.3.<br>T1.8 Someone outside the team that built the change re-ran or reviewed the Tier 1 journey tests.<br>T1.9 Every change in the release carries its G1 Provenance record, and every AI-assisted change passed G2 to G5 or its waiver is listed in the acceptance record, 2.1 and 14.2.<br>T1.10 Every Consequence 4 or 5 risk in the release's change risk analyses traces to a test that passed on the release build, 6.3.<br>T1.11 The failure tests on each Tier 1 journey's critical dependencies ran on the release build, and recovery stayed within the time the test policy sets, 7.4.<br>T1.12 Each AI feature on a Tier 1 journey passed every evaluation case on every run, and no red-team finding above the blocking severity is open, 12.3 and 12.4. |

| TIER | EXIT CRITERIA, EACH MARKED MET OR NOT MET |
|---|---|
| **Tier 2** | T2.1 Every Tier 2 journey test passed on the release build.<br>T2.2 No open defect of severity 1, and each open severity 2 defect has a workaround, an owner and a fix-by date in the acceptance record, 14.2.<br>T2.3 No quarantined, skipped, disabled or retried test on a Tier 2 path.<br>T2.4 A fall-back is defined.<br>T2.5 Every Consequence 4 or 5 risk in the release's change risk analyses traces to a test that passed, or is listed in the acceptance record, 6.3 and 14.2.<br>T2.6 Each AI feature on a Tier 2 journey met the pass rate the test policy sets over its repeated runs, and no red-team finding above the blocking severity is open, 12.3 and 12.4. |
| **Tier 3** | T3.1 Every Tier 3 test passed on the release build, or each failure is listed with an owner and a fix-by date in the acceptance record.<br>T3.2 No open defect of severity 1.<br>T3.3 Known defects are listed with their owners. |

These are the defaults. An adopter may set stricter criteria in its test policy, B.1, and none weaker for Tier 1. Severity follows the adopter's defect classification, recorded in the same policy.

14.1.1 **Agree the exit criteria when the release is planned.** A criterion relaxed on release day is a deviation, and the acceptance record in 14.2 shows it as one.

14.1.2 **Mark each criterion Met or Not met with its evidence:** run identifier, build and date. A criterion with no evidence is Not met.

14.1.3 **Count results by first attempt, at every tier.** A test that passed only on retry counts as failed, and so does a quarantined, skipped or disabled test, 8.2. List each one by name in the release record.

14.1.4 **File the release record.** The release record is what the test lead files for each release: the Met or Not met status of every criterion with its run identifier, build and date; the test results on the release build; the record of the outside re-run or review of Tier 1 tests; the named list of failed, retried, quarantined, skipped and disabled tests; the risks still untraced, 6.3; the completion fields of the release test plan, B.4; and the acceptance record, B.2. Its default store is in A.2.

14.1.5 **Recommend No-go while any Tier 1 criterion is Not met.** The release owner may still decide to release; that decision and its reason go in the acceptance record, 14.2, and the recommendation stays as written. For Tier 2 and Tier 3, recommend Go with conditions when every Not-met criterion has an owner, a fix-by date and a line in the acceptance record, and No-go otherwise.

## 14.2 Residual risk and who accepts it

Every release leaves some risk: a criterion Not met, a known defect, a journey only partly covered, a gate waived on a change in the release. List every waived gate as an item, scored on the 6.1 scales, so a Consequence 4 or 5 waiver reaches its risk owner. The test lead states it in customer terms, in one sentence per item:

> *If we release today, [what can still happen], to [which customers, how many, on which journey], detected by [what signal, how fast], put right by [what recovery path, run by whom].*

14.2.1 **The release owner accepts the residual risk.** The release owner is the business owner of the affected journey, or the business-side owner of the release where it spans several journeys. For any item of

Consequence 4 or 5, 6.1, the named risk owner for that risk accepts it as well. The author, the requester, the delivery lead, the test lead and anyone in the team that built the change never accept it.

14.2.2 **Record the acceptance separately from the recommendation.** The test lead prepares the acceptance record, B.2: each risk statement, the affected customers and journeys, the detection and recovery path, a link to the exit criteria status in 14.1, the prepared date and a review date no later than the fix-by date and never more than a year away. The decision, the signature and the acceptance date are left blank for the owner.

14.2.3 **A release with a Tier 1 criterion Not met goes ahead only on a signed acceptance record.** The record names the criterion, the reason and the date by which it will be met.

# A Appendix

## A.1 The Daily Rules

The one page to keep beside the keyboard.

1. Give the assistant bounded tasks. Name the files. State the non-goals.
2. Inventory every artefact the diff touches: code, tests, fixtures, schemas, configuration, dependencies.
3. Any non-code artefact changed: ask "did I intend this?" out loud.
4. Never approve a line you cannot explain.
5. "Seems fine" means stop.
6. Code and tests changed together by the same assistant: review the test change as its own question.
7. A passing pipeline shows only that the tests passed. Attach behavioural evidence.
8. A serialised-structure change merges only with a contract check or a recorded waiver.
9. Give AI diffs more review minutes per line than human diffs.
10. Declare on every change whether AI assisted, and where.
11. Run every agent in an isolated environment, with approval-bypass modes switched off.
12. Only a named person merges, waives a gate, releases or accepts risk.

## A.2 Gate ownership and exceptions

A.2.1 Role names map to the adopting team's structure. An exception approver must be someone other than the author or the requester of the change; where the named role is either, the next role up approves, and a waiver on a Tier 1 path also needs the release owner's approval, 14.2. The approval record shows who requested, implemented and approved each change, so anyone reading it can check the separation. Each exception is recorded with its reason when it is granted. The exception log is a single visible register. Every month a reviewer outside engineering, such as the risk function, leads its review, and the engineering lead attends to explain each entry. A recurring exception means the gate or the workflow needs redesign.

| GATE | ENFORCED BY | EXCEPTION APPROVER | RECORD |
|---|---|---|---|
| **G1 · Provenance** | Author (pull-request template field); reviewer verifies | Cannot be waived | Provenance field on every pull request |
| **G2 · Test integrity** | Reviewing engineer; continuous integration (CI) flag on same-change test edits | Tech lead | Written reason in the pull request + exception log |
| **G3 · Behavioural evidence** | Reviewing engineer | Tech lead | Written reason in the pull request + exception log |
| **G4 · Contract check** | CI job; quality engineering lead owns the contract suite | Engineering lead | Written reason in the pull request + exception log |
| **G5 · Blast-radius match** | Reviewing engineer; CI diff heuristics | Tech lead | Written reason in the pull request + exception log |
| **Security scans** | CI job on every change; the application security team owns the scan configuration | Cannot be waived on AI-assisted changes; a single finding may be dismissed by the named security role | Dismissal reason in the scanning tool + exception log |

A.2.2 **Emergency changes.** When production is down and a gate cannot run in time, the engineering lead may approve the merge before that gate passes. If the engineering lead wrote or requested the change, the next role up approves. G1 and the security scans still apply. The change goes into the exception log at once. Within one working day, an approver who neither wrote nor requested the change re-assesses it gate by gate and approves it; for a change on a Tier 1 path, that approver sits outside the team that implemented it. Where the emergency change touches a Tier 1 path, the release owner accepts its residual risk in the same working day, 14.2.

### Evidence defaults

Every artefact has a default store and a link to the build it belongs to. Each is kept for the retention period the firm sets for gate evidence, 13.1.2; the owner of that period is named in the test policy, B.1. Where a code host or pipeline keeps records for less than the period, export them to the firm's evidence archive before they expire.

| ARTEFACT | DEFAULT STORE | LINKED TO THE BUILD BY |
|---|---|---|
| **G1** provenance declaration | The merged pull request in the code host, exported to the evidence archive | Merge commit identifier |
| **G2** flag on same-change test edits, with the reviewer's decision | The pipeline run record and the pull request | Build identifier of the run |
| **G3** behavioural evidence | Attached to the pull request and copied to the evidence archive | Build identifier and date, carried in the evidence itself |
| **G4** contract verification result | The pipeline run record, and the contract broker where consumer-driven contracts are used | Build identifier of the run |
| **G5** artefact inventory and diff check | The pull request and the pipeline run record | Merge commit identifier |
| **Security scans** results and dismissal reasons | The scanning tool, exported to the evidence archive | Build identifier of the scan |
| **Exception log** entries | One register for the firm, held outside the code host | Change and build identifier in each entry |
| **Release record**, 14.1: exit criteria status, results on the release build, the outside re-run or review of Tier 1 tests, the untraced risks from 6.3 and the completion fields of the B.4 test plan | The firm's evidence archive, filed under the release; pipeline results are exported to it when the record is filed | Release build identifier; each result also carries its run identifier |
| **Change risk analysis**, 6.3 | With the pull request or work item in the code host or tracker | Change identifier |
| **AI feature evidence**, 12.2 to 12.5: evaluation runs, grader calibration, red-team findings, model change records | The firm's evidence archive, filed under the AI feature and the release | Model version and release build identifier |
| **Software bill of materials** and build attestation, 8.1 | The artefact registry, beside the build | Release build identifier |
| **Residual-risk acceptance record**, B.2 | Filed in the release record | Release build identifier |

## A.3 Self-check

**Answer yes or no for your team today.** Tick yes only if you could show the record. Each no is a gap.

| # | QUESTION | SECTION | YES |
|---|---|---|---|
| **THE CHANGE** | | | |
| 1 | Does every merged change record whether AI assisted, which tool, and which parts? | 1.4, G1 | ☐ |
| 2 | Are test edits made in the same change as the code reviewed as a question of their own? | 2.1, G2 | ☐ |
| 3 | Does every AI-assisted change carry behavioural evidence beyond a green pipeline? | 1.2, G3 | ☐ |
| 4 | Is every serialised structure checked against its consumers in CI before merge? | 5.2, G4 | ☐ |
| 5 | Is each change's artefact inventory compared with its stated intent before merge? | 2.1, G5 | ☐ |
| **TESTS** | | | |
| 6 | Is a drop in mutation score on changed code a merge blocker? | 4.3, 8.1 | ☐ |
| 7 | Does a retried, skipped or quarantined test count as failed at release? | 8.2, 14.1 | ☐ |
| **JOURNEYS AND RELEASE** | | | |
| 8 | Is there a written, tiered journey inventory with a named owner? | 6.2 | ☐ |
| 9 | Does every Tier 1 journey have an end-to-end test that asserts the outcome? | 6.2 | ☐ |
| 10 | Is residual risk accepted in writing by an owner outside the team that built the change? | 14.2 | ☐ |
| **AGENTS AND AI FEATURES** | | | |
| 11 | Is every agent's configuration in version control, with a named owner, least privilege and an isolated environment? | 11.1 | ☐ |
| 12 | Does every AI feature in the product have a protected evaluation set and a pass rule over repeated runs? | 12.2, 12.3 | ☐ |
| **EVIDENCE** | | | |
| 13 | Is gate evidence kept, linked to its release build, for a retention period the firm has set? | 13.1, A.2 | ☐ |
| 14 | Is the exception log reviewed each month by someone outside engineering? | A.2 | ☐ |
| 15 | Could you show a supervisor which of last quarter's changes were AI-assisted? | Ch. 13 | ☐ |

**Gaps found** _____ of 15

**What your answers mean. 0 to 3 gaps:** close each one, Tier 1 journeys first. **4 to 8:** follow the adoption plan in the companion document, week 1 first. **9 to 15:** start the week 1 steps now: the provenance field, the schema check and the exception log.

# B Adopter's templates

The chapters state principles, gates and rules. This appendix holds what an adopting firm fills in for itself, and B.1 lists every value a clause leaves to the firm's test policy. Copy each template into the firm's own document system and keep it under the firm's change control.

## B.1 Test policy

The firm's own test policy adopts the Standard and records the choices the Standard leaves to the firm. One policy covers every team that adopts the Standard.

| SECTION | WHAT GOES IN IT |
|---|---|
| **Scope and approval** | The systems and teams the policy covers. The executive outside the delivery teams who approves the policy, with the risk function consulted. Review at least yearly and whenever an early-revision trigger in the document control fires. |
| **Quality objectives** | One objective per Tier 1 journey and per group of Tier 2 journeys, stated as an outcome for customers, each with the measure that shows whether it holds. The journey inventory and its tiers, 6.2, are the source. |
| **Test levels and types by tier** | The defaults below, and any level the firm adds. A team may test more; a team that tests less records a deviation. |
| **Entry criteria** | When testing of a release may start. Defaults: an identified build; a test environment ready with synthetic data, 4.4; acceptance criteria agreed and the failing tests written before the code, 4.1. |
| **Test shape** | The balance of unit, integration, contract and end-to-end tests the firm expects for each kind of system it builds, such as services, web front ends and data pipelines, with the reason for each. |
| **Exit criteria and defect severity** | The exit criteria per tier: the defaults in 14.1 or stricter, never weaker for Tier 1. The defect severity scale the criteria use. |
| **Independence** | Who outside the building team re-runs or reviews Tier 1 journey tests before release, 14.1 and 10.1, and who approves Tier 1 changes, Ch. 2. |
| **Risk acceptance** | The release owner for each journey or journey group, and the named risk owners for Consequence 4 or 5 items, 14.2. |
| **Non-functional thresholds** | The performance budget and the recovery time for each Tier 1 journey, 7.2 and 7.4; the scan and dependency-check severity that blocks a release, 7.1 and 8.1; the accessibility standard and conformance level the firm tests against, such as the Web Content Accessibility Guidelines (WCAG) 2.2 at level AA, 7.3. The Tier 1 exit criteria in 14.1 use these values. |

| SECTION | WHAT GOES IN IT |
|---|---|
| **AI features** | For each AI feature, 12.1: the evaluation set and its owner; the failure rate the risk owner accepts, the number of runs and the pass rule by tier, 12.3; the agreement a model-based grader must reach before it gates, 12.2.3; the red-team severity that blocks a release, 12.4; and the production monitors, 12.5. |
| **Review capacity** | For each tier, the largest agent-written change a reviewer is given and the number of open agent-written pull requests per reviewer, 3.2.6. |
| **Agents** | The network allow-list for agent environments, 11.1.9; the number of failed fix attempts after which an agent hands a task back, 11.1.12. |
| **Vulnerabilities** | The remediation deadline for each severity, 13.1.6; the time within which a reported component is traced to every release that contains it, 13.1.10. |
| **Pipeline time** | The longest a pull-request run may take; above it, the pull request runs the tests the change reaches, 8.2.5. The cap on automatic retries, 8.2.4. The systems that cannot run in a short-lived test environment, and why, 8.2.5. |
| **Evidence and retention** | Any store that differs from the defaults in A.2; the retention period for gate evidence and the role that owns it, 13.1.2. |
| **Test data** | Confirmation of the rules in 4.4, and the roles that approve exceptions: the data protection officer and the chief information security officer, or their equivalents. |
| **Deviations** | How a team asks to depart from the Standard or this policy, and who decides: an approver outside the requesting team, with the release owner as well for a Tier 1 journey. Each deviation has an owner and an end date, is recorded in the exception log and is reviewed at the monthly review. The five rules that Using this Standard lists as allowing no deviation are excluded. |
| **Exception log review** | The role outside engineering that leads the monthly review of the exception log, such as the risk function, Ch. 2 and A.2. |

**Default test levels and types by tier.** Chapter references give the rule each default comes from.

| TIER | DEFAULT LEVELS AND TYPES |
|---|---|
| **Tier 1** | Unit tests written or approved before the code, with the mutation gate on changed code, 4.1 and 4.3; integration tests against real dependencies and contract checks on every boundary, 5.1 and 5.2; end-to-end journey tests that assert outcomes, 6.2; performance budgets and load tests, 7.2; security scans and the dependency check, 7.1 and 8.1; automated accessibility checks and a human pass on new user-interface flows, 7.3; a fall-back rehearsal before release, 14.1; failure tests on critical dependencies, 7.4; an exploratory session on every new flow, 6.2; for AI features, evaluation sets with the pass[k] rule and red teaming, 12.2 to 12.4 |

| TIER | DEFAULT LEVELS AND TYPES |
|---|---|
| **Tier 2** | Unit tests with the mutation gate; integration tests and contract checks; end-to-end tests that assert the data is correct, 6.2; security scans and the dependency check; automated accessibility checks; an exploratory session on every new flow, 6.2; for AI features, evaluation sets with the policy's pass rate and red teaming of generative features, 12.3 and 12.4 |
| **Tier 3** | Unit tests with the mutation gate; integration tests and contract checks on changed boundaries; end-to-end tests where the journey justifies their cost; security scans and the dependency check |

## B.2 Residual-risk acceptance record

The test lead prepares the record and the owners named in 14.2 decide. The decision, signature and date fields stay blank until the owner completes them.

| FIELD | WHAT GOES IN IT |
|---|---|
| **Release** | Release name, build identifier and planned date |
| **Exit criteria status** | Link to the Met or Not met record for every tier, 14.1 |
| **Residual risk** | One statement per item in the 14.2 pattern, with the tier and the Consequence score, 6.1 |
| **Criteria Not met or relaxed** | Each criterion, the reason and the date by which it will be met |
| **Prepared** | Test lead's name and the prepared date |
| **Release owner decision** | Accept or do not accept, with the owner's name, signature and date; left blank for the owner |
| **Risk owner decision** | For each Consequence 4 or 5 item: accept or do not accept, with the risk owner's name, signature and date; left blank for the owner |
| **Review date** | No later than the fix-by date of the latest item, and never more than a year after acceptance |

## B.3 Test data exception record

Used only for the exceptions 4.4 allows. The data protection officer and the chief information security officer, or the firm's equivalents, complete the approval fields.

| FIELD | WHAT GOES IN IT |
|---|---|
| **Purpose** | The test that cannot run on synthetic data, and why |
| **Data** | Data classes, number of records, and the form used: anonymised, pseudonymised or raw |

| FIELD | WHAT GOES IN IT |
|---|---|
| **Environment and controls** | Where the data is held, who can reach it, and how it is protected |
| **Period** | Start date and end date |
| **Approval** | Data protection officer and chief information security officer: name, decision, signature and date; left blank for them |
| **Reported to** | The ICT risk management function, with the date reported |
| **Deletion** | Confirmation and date the data was deleted at the end date |

## B.4 Release test plan and completion report

One record per release: the test lead fills the plan fields when the release is planned and the completion fields when testing ends. The completion fields become part of the release record, 14.1.

| FIELD | WHAT GOES IN IT |
|---|---|
| **Release and scope** | Release name and planned date; the journeys and AI features in scope with their tiers; the change risk analyses it covers, 6.3 |
| **Approach** | Levels and types by tier from B.1, with any deviation; exploratory charters, 6.2; failure tests, 7.4; evaluation runs and pass rules for AI features, 12.3 |
| **Entry and exit criteria** | The entry criteria from B.1 and the exit criteria agreed for each tier, 14.1 |
| **Environments and data** | The test environments and the synthetic data used, 4.4, with any exception record, B.3 |
| **People** | Test lead; the person outside the building team who re-runs or reviews the Tier 1 tests; release owner; named risk owners |
| **Completion: results** | Met or Not met for every criterion, with run identifiers; risks still untraced, 6.3; open defects and red-team findings by severity |
| **Completion: deviations and residual risk** | Every deviation from the plan with its approver; a link to the acceptance record, B.2 |
| **Completion: lessons** | Escaped defects since the last release and the tests added for them, 9.2 and 10.3 |

# G Glossary

Terms are linked at their first use in the running text of the chapters. Each entry links back to that place.

| TERM | MEANING IN THIS STANDARD |
|---|---|
| **Agent roster**<br>Back to text, p. 6 | The set of AI agents a team runs. Each has a named owner, a pinned model version, its configuration in version control and an evaluation set, Ch. 11. |
| **AGENTS.md and instruction files**<br>Back to text, p. 6 | Plain-text files in the repository that state conventions and non-goals for agents, read before every task. AGENTS.md is one common format, 3.2. |
| **AI-assisted change**<br>Back to text, p. 6 | A change in which an AI tool wrote or restructured any code, test or data. Spelling fixes, identifier completion, mechanical renames and reformatting do not count, 2.1. |
| **AI feature**<br>Back to text, p. 24 | Any part of the product whose behaviour comes from a trained model: generated text, a score, a ranking or a decision, 12.1. |
| **Background agent**<br>Back to text, p. 6 | An agent that works unattended in a cloud environment and returns a finished pull request. |
| **Behavioural evidence**<br>Back to text, p. 6 | Proof of what a change does beyond a passing pipeline: run output, a trace, a before-and-after demonstration, or a new test that fails on the old code (G3). |
| **Blast radius**<br>Back to text, p. 18 | How much of a system a change can affect. G5 checks that it matches what the change claims to do. |
| **Canary release**<br>Back to text, p. 18 | Releasing a change to a small share of traffic first, with thresholds set in advance that roll it back automatically when breached, Ch. 9. |
| **Consumer-driven contract**<br>Back to text, p. 9 | A test in which each consumer of an interface records the fields and values it relies on, and the provider's build checks every change against those records. 5.2. |
| **End-to-end (E2E) test**<br>Back to text, p. 13 | A test that drives a whole business journey through the real interface, from the user's action to the recorded outcome, Ch. 6. |
| **Evaluation set**<br>Back to text, p. 24 | The test suite of an AI feature or an agent: inputs, the expected behaviour or grading rule for each, and the quality each case checks. Held as a protected path, 11.2 and 12.2. |
| **Exception log**<br>Back to text, p. 7 | The single register of every waived gate, dismissed security finding and emergency change, with its reason, approver and date. A reviewer outside engineering, such as the risk function, leads its monthly review, A.2. |
| **Exploratory testing**<br>Back to text, p. 15 | Time-boxed testing by a person against a written charter, in which each test is designed from what the last one showed, 6.2. |
| **Feature flag**<br>Back to text, p. 18 | A switch that turns a change on or off in production without a new deployment. |
| **Healer agent**<br>Back to text, p. 6 | An agent that repairs tests broken by application changes, for example by updating how a test finds an element on a page, 11.1.4. |
| **Model drift**<br>Back to text, p. 26 | A change over time in how an AI feature behaves or in what it is asked, found by monitoring live outputs against the evaluation rules, 12.5. |

| TERM | MEANING IN THIS STANDARD |
|---|---|
| **Mutation score**<br>Back to text, p. 6 | The share of small deliberate changes to the code (mutants) that the test suite catches. Chapter 4 uses it in place of line coverage. |
| **Pass$^k$**<br>Back to text, p. 25 | A pass rule over repeated runs: a case passes only when all k runs pass. The default for Tier 1 AI features, 12.3. |
| **Prompt injection**<br>Back to text, p. 25 | An attack that hides instructions in text an AI feature or agent reads, such as a document or web page, to make it act against its own instructions, 12.4. |
| **Provenance**<br>Back to text, p. 7 | The record of whether and where AI assisted a change, which tool, and which parts it touched (G1). |
| **Red teaming**<br>Back to text, p. 25 | Structured adversarial testing by people outside the building team, who try to make an AI feature leak data, break its rules or exceed its permissions, 12.4. |
| **SAST**<br>Back to text, p. 15 | Static application security testing: automated checks of source code for known vulnerability patterns, 7.1. |
| **SBOM**<br>Back to text, p. 17 | Software bill of materials: the list of every component and version in a release build, kept with a signed attestation of how the build was made, 8.1. |
| **Tier 1 journey**<br>Back to text, p. 7 | A business journey whose failure causes direct financial loss or a regulatory breach: money movement, transaction execution, credit and lending decisions, regulatory reporting, authentication, 6.1. |
| **Waiver**<br>Back to text, p. 7 | A recorded decision to merge a change that has not passed G2, G3, G4 or G5, made by the named approver, who neither wrote nor requested the change, Ch. 2 and A.2. |

# S Sources

The sources this Standard names, with the clause that names them. Every study, report, regulatory instrument and figure behind the rules is listed, with its date and where it can be checked, in the companion document under Sources and references. Entries marked "history" are older sources kept for context.


- OWASP GenAI Security Project, *OWASP Top 10 for Agentic Applications for 2026*, 9 December 2025 · 7.1.2 · genai.owasp.org
- OWASP GenAI Security Project, *OWASP Top 10 for LLM Applications 2026*, announced 1 September 2026 · 7.1.2 · genai.owasp.org
- Google Cloud, DevOps Research and Assessment programme, *DORA's software delivery performance metrics*, updated 5 January 2026 · the five delivery measures, 10.3.4 · dora.dev
- Yao, Shinn, Razavi and Narasimhan, *τ-bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains*, June 2024 (history) · pass$^k$, 12.3.1 · arxiv.org/abs/2406.12045
- Hanley and Lippman-Hand, *If nothing goes wrong, is everything all right? Interpreting zero numerators*, JAMA 249(13), 1983 (history) · the rule of three, 12.3.1
- Cohen, *A coefficient of agreement for nominal scales*, Educational and Psychological Measurement 20(1), 1960 (history) · Cohen's kappa, 12.2.3

– World Wide Web Consortium (W3C), *Web Content Accessibility Guidelines (WCAG) 2.2*, W3C Recommendation, 5 October 2023 · the example in B.1 · w3.org

# Document history

| VERSION | DATE | SUMMARY |
|---|---|---|
| v2026.6 | October 2026 | Edition for adopting firms, published in two documents: this Standard holds the rules, and Evidence and Guidance holds the evidence, the regulatory map, worked examples and the adoption plan; the templates are in Appendix B. Adds testing AI inside the product, Ch. 12; evidence rules, Ch. 13; the release decision, Ch. 14; risk analysis for each change, 6.3; resilience tests, 7.4; credit and lending decisions in Tier 1, 6.1; the self-check, A.3; clause numbers, conformance and citation rules in Using this Standard; agent isolation, stop and decision rules, 11.1.9 to 11.1.13; tamper-evident evidence and component tracing, 13.1.9 and 13.1.10; the adopter's templates, Appendix B; and an open licence with rules for using the name, on the document control page. Earlier versions are listed in Evidence and Guidance. |

Certance.

NEXT STEP

# See where your test suite stands against this Standard.

The Coverage & Evidence Audit maps your critical business journeys to the tests that protect them, ranks each gap by consequence and detection delay, and checks which of the five gates your change records already evidence. Three days at a fixed scope and fee, ending with a gap report your board and risk function can read. It works from records your own engineer exports. It needs no access to your systems and no production or customer data.

| | |
|---|---|
| THE AUDIT | certance.eu/coverage-audit |
| BOOK A CALL | cal.com/certance/coverage-audit |
| EMAIL | contact@certance.eu |
| ON REQUEST | A sample gap report and the vendor due-diligence pack |
| CURRENT EDITION | certance.eu/certance-standard |

ABOUT CERTANCE ADVISORY

Certance Advisory is a quality engineering consultancy for regulated software teams, built on experience of regulated banking, life sciences and e-commerce systems. It has no referral or revenue-share arrangements with tool vendors. The current version of The Certance Standard and the reasoning behind it are published at certance.eu/certance-standard.

This Standard is engineering guidance. It gives no legal advice and no compliance opinion.

© 2026 Certance Advisory · The Certance Standard · v2026.6 · October 2026