

A lightweight SDLC for agentic teams

How a team works when AI agents write much of the code. The companion to the SOC 2 control set, and the process that produces its evidence.

5

STAGES

2

HUMAN GATES

4

TEST QUADRANTS

This is the working process for a team where AI coding agents do most of the building. It's the companion to the *SOC 2 control set for agentic teams*. That document says what an auditor will check. This one says how the team works so the evidence appears without anyone gathering it.

The idea fits in two sentences. Humans own the two ends: what gets built and what gets released. Agents own the middle, inside rules they can't change.

Roles

ROLE	WHO	OWNS
Requester	A human: anyone with a need, often an engineer	The spec and its acceptance examples
Product owner	The human accountable for the product	Approving specs before anything is built
Tester	A human with a testing eye	Helping write acceptance examples, and exploratory testing outside releases
Builder agent	An AI agent with its own identity	Design, code, Q1 and Q2 tests, merging to integration
Review agent	A second agent with different instructions	Finding problems in every change. A tool, not an approver
Release approver	A human who is not the requester	The Q3 outcome check and the release decision
Control owner	A named human	The pipeline, policies, protected paths, agent permissions and Q4 thresholds

One person can hold several roles across different changes. On a single change, the requester and the release approver must be different people.

The testing quadrants

The process is organised around the agile testing quadrants rather than the test pyramid, because the quadrants say why each test exists. Read this table first. The stages below refer to it.

QUADRANT	WHAT	STAGE	WHO DECIDES WHAT PASSES	BLOCKS
Q1	Unit and component tests	Build, Verify	The builder agent, inside the spec's scope	Merge
Q2	Acceptance tests from the spec's examples	Build, Verify	The requester, with a tester helping write the examples	Merge
Q3	Outcome check: does the release do what each spec asked?	Release	The release approver, against the acceptance examples	Release
Q3	Exploratory testing: what is nobody asking about?	Outside the release	Testers raise findings; the product owner decides what happens to them	Nothing. Findings become backlog items
Q4	Security, dependency and performance checks	Verify, Operate	The control owner sets the thresholds	Merge, or raises an alert

Three quadrants are automated. Q3 is where people spend their testing time, and it's the one an agent can't do for you: deciding whether the thing is what was wanted.

Q3 comes in two halves. The outcome check sits inside the release, because it's the evidence that each change does what was approved. Exploratory testing sits outside it. A tester's curiosity is valuable, but tied to a release it turns into a late new requirement that holds everything up. Outside the release it still gets heard, and the product owner decides what to do with it.

This gives testers a better job than rerunning scripts, and it moves them to the start. At Intent they help requesters write sharper acceptance examples, including the awkward edge cases, and they ask how the change will be tested before anyone builds it. A gap found in a spec costs minutes. The same gap found at release costs a cycle.

The five stages

1. INTENT

The requester writes a short spec in the tracker or as a spec file in the repo.

- **Entry:** a need, from a customer, an incident or the roadmap.
- **The spec holds** the outcome in a sentence or two and the acceptance examples, written as concrete cases ("given this, when that, expect this"). It also says what the change must not touch, and gives a risk tier. A tester helps write the examples.
- **Risk tier:** *Standard* for most work. *Sensitive* if the change touches authentication, permissions, money, personal data or the controls themselves. Sensitive changes get a human read of the diff at release, on top of everything else.
- **Exit:** the product owner approves the spec, and the approval is recorded. Nothing gets built without it. A requester who is also the product owner can approve their own spec. The release approver is then the second pair of eyes.
- **Standing approval:** some changes don't need a spec each time. Dependency updates and fixes for scanner findings can be pre-approved by a written policy, which the control owner keeps. The policy is the approval. The change still goes through Verify and Release like any other.

- **Evidence:** an approved item with an ID, or a reference to the standing-approval policy. Every later step links back to it.

2. BUILD

The builder agent works from the approved spec.

- **Entry:** an approved spec ID.
- **Work:** the agent turns the acceptance examples into Q2 tests first, then writes the code and its Q1 unit tests. It works on a branch, commits under its own identity, references the spec ID in every commit, and opens a pull request into the integration branch.
- **The agent may:** change application code, tests within the spec's scope, and documentation.
- **The agent may not:** change pipeline definitions, branch policies, code-owners files, test configuration, access settings or secrets, or touch production. These paths are protected and need the control owner's review.
- **The agent stops and asks** when the spec is ambiguous, when the change needs a protected path, when it wants to add a new dependency, or when it can't make a test pass without changing an acceptance example.
- **Exit:** a pull request with all checks running.

3. VERIFY

Every check runs automatically on every change. Nobody has to remember anything.

- **Q1** unit and component tests, and **Q2** acceptance tests built from the spec's examples. Both block on failure.
- **Q4** checks: secret scanning, static analysis and dependency scanning. Critical findings block.
- **Review agent:** reads the change against the spec and the team's standards. Blocking findings stop the merge. Anything it can't assess is reported as "not reviewed", never as a pass.
- **The examples rule:** only the requester can change an acceptance example. Agents work to pass the gate in front of them. If an agent can weaken a test to pass it, sooner or later one will.
- **Exit:** the agent merges into the integration branch. The pipeline deploys to a test environment.

4. RELEASE

The human gate. This is where segregation of duties lives.

- **Entry:** a release candidate: one build, already in the test environment, containing one or more merged specs.
- **Work:** the release approver runs the Q3 outcome check. They check each spec's outcome in the test environment against its acceptance examples. For a Sensitive change, they also read the diff on the sensitive paths.
- **What can block a release:** a failure against an acceptance example in this release, or a security or Sensitive-tier finding. Nothing else. A concern outside the release's specs becomes a new item for the product owner to triage. It doesn't hold the release.
- **Release note:** a few lines recording the build ID, the specs included, what was tested, the results and the decision. A template makes this a two-minute job.
- **Approval:** recorded in the pipeline against that exact build. The pipeline promotes the same build to production. It's built once and promoted, never rebuilt, so what was tested is what ships.
- **Exit:** the build is in production, and post-deploy smoke checks pass. If they fail, the pipeline rolls back automatically.

5. OPERATE

- Monitoring and alerts cover errors, security events and unusual access.
- Scheduled Q4 scans and drift detection run without anyone starting them.

- Testers run exploratory testing on their own schedule. Findings go to the product owner as backlog items.
- An incident gets a ticket. The fix starts again at stage 1. An agent can draft the fix, but it goes through the same flow.
- Each incident review asks which test or control would have caught this. The answer becomes a spec of its own, and the fix is an automated test or control, not another manual check. A manual check catches the problem once. A test catches it on every change.

Emergency fast path

Production is down and the normal flow is too slow. Use this, and only this.

1. The on-call engineer declares an emergency in the tracker. One line is enough.
2. The fix still goes through a pull request and the pipeline. The pipeline stays the only route to production. The secret and dependency scans still run. Slow test suites may be skipped.
3. The fix ships on one approval from a human who isn't the requester. If nobody else is available, the on-call engineer approves it and the record says so. A second person reviews it within two working days.
4. If the pipeline itself is broken, break-glass access to production belongs to a named operations role, not to developers. It's time-limited and logged, and every use appears on the exception report.
5. The incident review adds the tests that were skipped, and asks why the normal flow was too slow.

Every emergency appears on the monthly exception report. If emergencies become routine, the normal flow is the problem.

Small teams

Two people. One requests, the other approves the release. On the next change, swap. That meets the rule.

One person. You can't separate duties from yourself. The honest options are a part-time second approver for releases (an advisor, a contractor, a co-founder), or accepting that you aren't ready for a SOC 2 Type 2 yet. Everything else in this SDLC still pays off, so do it anyway.

Definition of done

A change is done when all of these are true:

- the approved spec, or the standing-approval policy, is linked from every commit and the pull request
- Q1 and Q2 pass, Q4 is clean, and the review agent has no blocking findings
- it was merged by the agent into integration, not pushed to production
- the Q3 outcome check is complete and the release note is written
- someone other than the requester approved the release, and the pipeline deployed it
- the new behaviour is covered by monitoring

Where the evidence comes from

Control IDs refer to the *SOC 2 control set for agentic teams*.

STAGE	EVIDENCE LEFT BEHIND	CONTROLS
Intent	Approved spec with an ID, or the standing-approval policy	CM-1
Build	Commits under the agent's identity, pull request	CM-2, CM-7, AC-4
Verify	CI runs, scan results, review agent report	CM-3, OP-1, AC-7
Release	Approval log, release note, deploy log	CM-4, CM-5
Operate	Alerts, incident tickets, drift reports, backup and restore records	OP-2 to OP-6
Always	Protected-path reviews, emergency and break-glass records, monthly exception report	CM-6, CM-8, AC-3, GV-8

Adopting it

Go in this order. Each step is useful by itself.

1. Give every agent its own identity, and protect the main branch.
2. Make the pipeline the only route to production.
3. Add the release approval gate, its scope rule and the release note template.
4. Protect the control paths with a code-owners rule.
5. Add the review agent.
6. Introduce risk tiers once the team has a feel for which changes are sensitive.

Releases usually speed up after step 3, because they stop waiting for a meeting.