

AI on Both Sides of the Pull Request

Generating from Specifications and Reviewing against Standards, with a Human in the Loop

A Practitioner White Paper on AI-Based Code Generation and Review

Use AI where it helps the business.

Observability & Governance

Automation & Execution

Decision & Policy

AI Model & Agent

Data & Knowledge

Oversight →

Autonomy →

Oliver Bodemer · e-Softworks, Inc.

October 2026



Document WP-2026-10 · Version 1.0 · October 2026

This white paper derives from the scientific article *AI on Both Sides of the Pull Request* (Oliver Bodemer, e-Softworks, Inc., October 2026, document SCI-2026-10). That article is a design argument. It does not report measured delivery speed or defect rates for this operating model.

The business case

The human pair is an author and a separate reviewer. Moving only the author to AI leaves the pull request waiting on a person. The review then sets the pace. Both duties move together.

Software development has a stable division of work. A person writes a change from a specification and from the requirements that belong to it. A second person reviews that change in a pull request before it is accepted. The review is what keeps a fast author from shipping a change that misses the specification, breaks a standard, or does something nobody asked for.

AI code generation has changed the pace of the writing side. When only the writing moves to AI, the pull request waits on a human reviewer, and the review becomes the queue that sets the pace. A later measurement would compare three setups: a person writes and a person reviews; an AI instance writes and a person still reviews; both duties move, with a human stop on each side. This white paper states the third. It does not measure the three against each other.

The operating model moves both tasks. One AI instance creates the application change from the specifications and the requirements. A second AI instance, independent of the first, counter-checks the source code in the pull request against a fixed standards catalogue. A human remains in the loop on both sides. This paper does not choose a product. Two runs are enough when the instructions, the inputs, and the outputs stay apart. The runs may be the same model family or two families. A shared chat that writes the change and then accepts it is one run, and it collapses the split.

Two instances

The generator starts from specifications, requirements, and constraints. It produces a change complete enough to review, including tests tied to the claimed requirements. The reviewer is a second instance. It reads the pull request and the catalogue. It does not carry the generator's draft forward.

An instance, in this white paper, is one run of a model with its own instructions, its own inputs, and its own output. The generator instance and the reviewer instance are two such runs. What this section fixes is the duty boundary between the runs.

The generator instance reads the specifications, the requirements, and the constraints that bound the change. It writes the application change those inputs describe: the source code, and the tests tied to the requirements the change claims to meet. The generator's output is the change. The decision that the change meets the catalogue sits with the reviewer instance, and the decision that the pull request may merge sits with the human.

The reviewer instance reads two inputs: the pull request, and the standards catalogue. It writes findings. A finding cites one entry in the catalogue and one place in the change. The reviewer instance stops at the finding. Editing the branch, and merging it, sit outside that output.

When a finding calls for a change to the source, the generator instance makes that change on a new pass. The finding joins the original specification, requirements, and constraints as an input to that pass. The two runs still keep their hidden reasoning to themselves. What crosses the boundary is an artefact either instance can read: a change in one direction, a finding in the other.

The standards catalogue

Each finding cites one entry and one place in the change.

- **Conformance to the specification and the requirements.** Does the change implement the claimed requirements, and does it stay inside the behaviour the specification asked for? A finding names the requirement and the place that misses it or that adds behaviour the specification did not ask for.
- **Maintainability in the Clean Code sense.** Can a teammate who did not write the change read it and change it? A finding names the place the reading breaks: a name, a function, or a path through the change.
- **ISO/IEC 25010 reliability, security, and maintainability, used as review questions.** For the behaviour this change claims, do these characteristics hold as questions about this diff? A finding names which characteristic, which requirement it attaches to, and the place in the change.
- **Named security issues.** Does the change introduce injection, cross-site scripting, cross-site request forgery, secrets in source, path traversal, or server-side request forgery? A finding names the issue and the place an input, a secret, a path, or a request is handled.
- **Tests that demonstrate the requirements the change claims to meet.** Do the tests demonstrate the requirements the change claims to meet? A finding names the claimed requirement that has no test, or the test that does not demonstrate the requirement it names.

The catalogue is a list of review questions. ISO/IEC 25010:2023 names nine product-quality characteristics. This catalogue uses three of them — reliability, security, and maintainability — as questions about this diff. The other six stay outside. A numeric product-quality score lies outside this paper. The team writes the catalogue down before either instance runs a cycle. A finding cites one entry in the catalogue and one place in the change. A remark that names neither an entry nor a place is not a finding, and it does not hold the pull request. A request for behaviour the specification did not include, for a performance measurement, or for a filed safety case does not hold it either.

One illustration, not a measured change, shows a finding a seat can decide. The specification asks the service to store an uploaded file under a name the constraints already allow. The generator writes that store, and a test that names the requirement. The intent seat confirms the change, and the pull request opens. The reviewer writes: row, named security issues; place, the handler that joins the request path into the file name; question left open, path traversal. The review seat accepts the finding. That send-back is one round. A remark that the handler should also resize the image cites no row. It does not hold the pull request.

The human stops

A human confirms intent before the pull request exists. A human decides material findings and the pull request. The cycle ends on accept, reject, an override of the open findings, or the round cap. A log of what the instances did is a record. The decision is the stop.

Two human seats sit on the duty split. The intent seat stands on the generator side, after the generator has written the change and before a pull request is opened. The review seat stands on the reviewer side, after the reviewer has written its findings and before the change is merged. Both seats are required. Each seat can halt the work in front of it.

The intent seat confirms three things: the intent, the scope, and that the change still matches what was asked. Those three things are the specification, the requirements, and the constraints the generator was given. The seat writes the confirmation, or it sends the change back to the generator. A pull request opens only after the confirmation is written. The send-back names one failure and the place: invented behaviour, scope drift, or an unsafe shortcut. Invented behaviour is source the specification did not state. Scope drift adds or drops a requirement. An unsafe shortcut reaches a requirement by leaving the constraints.

The review seat decides every material finding, and it decides the pull request. A finding is material when it cites one entry in the catalogue and one place in the change, and leaving it open would leave that catalogue question unanswered at that place. On each material finding the seat accepts the finding and sends the change back, or it overrides the finding and writes the reason. It then accepts the pull request or rejects it.

The cycle ends on the first of four stops:

- **Pull request accepted.** The review seat writes the acceptance. No further pass runs.
- **Pull request rejected.** The review seat writes the rejection. The change stays unmerged.
- **Open findings overridden.** Every open material finding has a written reason. The seat may then accept or reject the pull request.
- **Round cap reached.** The next send-back would pass the cap the team set. That pass does not run. Open findings still hold the pull request. The cap does not merge.

The team writes the count in the same place as the catalogue, before the first send-back. The count is how many review-seat send-backs are allowed on that change. If the team cannot agree the count, the cycle does not start. This paper does not choose the number. Reaching the cap does not merge. The cap bounds the exchange. It does not close a finding.

How to start

1. **Write the catalogue down.** The five review questions. A finding cites one entry and one place in the change.
2. **Separate the generator instance from the reviewer instance.** Separate instructions, inputs, and outputs. The generator writes the change. The reviewer counter-checks it.
3. **Name the human seat before the pull request and the human seat on the review.** One person may hold both, and each seat still writes its own record.
4. **Set the round cap.** A count of rounds, chosen before the first send-back. Reaching the cap does not merge.
5. **Then run one cycle on one change a person could read in one sitting.** It needs a written specification, a short list of requirements, and constraints. A first cycle is not a migration and not a new system. Run the generator. Confirm the change, or send it back for invented behaviour, scope drift, or an unsafe shortcut. Open the pull request only after the confirmation. Run the reviewer. Decide the findings. Accept, reject, override, or reach the cap.

With those five steps, the cycle can run. The measurements of speed, defects, and shared misses remain outside the model.

Limits

Both instances can miss the same defect. This paper does not count how often, and it does not name the conditions that make a miss more likely. The human decisions are where that blind spot can still be caught. This paper states the operating model. Delivery speed and defect rates are for a later measurement of the three setups in the business case.

The runs may be the same model family or two different families. The duty split still holds: the instructions differ, the inputs differ, and the outputs differ. The split does not remove a shared miss, and a different family does not remove it either.

The human seats are where a shared miss can still be caught. The intent seat can send the change back when the defect is invented behaviour, scope drift, or an unsafe shortcut. The review seat can reject the pull request when it sees a catalogue question still open. The absence of a finding does not require acceptance.

This paper states the operating model. It answers how to set up the two instances, the catalogue, the human seats, and the stop rules. It does not report the outcomes. Delivery speed, defect density, human effort, shared-miss frequency, and override rate are questions for a later measurement.