AI on Both Sides of the Pull Request

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

A Business Article on AI-Based Code Generation and Review

Use AI where it helps the business.





Oliver Bodemer · e-Softworks, Inc.

October 2026

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

This business article 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.

Key Findings

One AI instance writes the application change from the specification and the requirements. A second instance reviews the pull request against a fixed standards catalogue. A human confirms the intent before the pull request and decides the review after it.

The pace problem

People write the change, and people review the pull request. If only the writing moves to AI, the review becomes the queue that sets the pace. The faster the generator produces a change, the more the review waiting for that change matters.

The split is the duty model this article presents. The writing is one duty. The review is a second duty. Moving one without the other leaves the second as the queue. Moving both keeps the two duties running at the same pace, and it keeps a human stop on each one.

A later measurement would compare three setups. In the first, a person writes the change and a person reviews it. In the second, an AI instance writes the change and a person still reviews the pull request. In the third, one AI instance writes the change and a second instance counter-checks it, with a human stop on each side. This article states the third. It does not measure the three against each other. Studies of other tools, cited in the scientific article, are not results of this model.

Both tasks in the third setup move to separate AI instances. The generator instance writes the change. The reviewer instance counter-checks the pull request. A human still sits on both sides. The human on the generator side confirms the change before a pull request is opened. The human on the reviewer side decides the findings and decides the pull request. The confirmation and the review decision are the stops that keep the loop bounded.

Two duties, two instances

The generator instance reads the specification, the requirements, and the constraints. It produces one application change: the source, and the tests that demonstrate the requirements the change claims to meet. The generator's output is the change. It is not the decision that the change is correct.

The reviewer instance is a separate run. It reads two inputs: the pull request, and the standards catalogue. It writes findings. Each finding cites one entry in the catalogue and one place in the change. The reviewer instance stops at the finding. It does not edit the branch. It does not merge the pull request.

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

The separation keeps the duties apart. The generator was instructed to produce the change. The reviewer was instructed to counter-check it. One run that wrote the change and then accepted its own pull request would collapse the author/reviewer split. The second instance is what keeps that split standing.

The human on both sides

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 of three failures, and the place. Invented behaviour: the source does something the specification did not state. Scope drift: the change adds a requirement the list does not contain, or it drops one the list does contain. An unsafe shortcut: the change reaches a requirement by leaving the constraints. A broken standard is not one of these three. That question waits for the reviewer, once a pull request exists.

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 merge decision is the review seat's output.

The two seats may be held by one person or by two. Each seat writes its own record. The intent seat writes before the pull request exists. The review seat writes after the findings exist. The same two decisions are where a team can still catch a defect that both AI instances miss.

What the review checks

The reviewer instance applies a fixed catalogue. 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. The catalogue is a list of review questions about the pull request in front of the reviewer. A numeric product-quality score lies outside it.

Entry	Review question	A finding must show
Specification conformance	Does the change implement the claimed requirements, and does it stay inside the behaviour the specification asked for?	The requirement, and the place that misses it or that adds behaviour the specification did not ask for.
Clean Code maintainability	Can a teammate who did not write the change read it and change it?	The place the reading breaks: a name, a function, or a path through the change.
ISO/IEC 25010 review questions	For the behaviour this change claims, do reliability, security, and maintainability hold as questions about this diff?	Which of the three characteristics, 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, a secret in source, path traversal, or server-side request forgery?	The named issue, and the place an input, a secret, a path, or a request is handled.
Tests of claimed requirements	Do the tests demonstrate the requirements the change claims to meet?	The claimed requirement that has no test, or the test that does not demonstrate the requirement it names.

Table 1: The standards catalogue. Each finding cites one entry and one place.

One illustration, not a measured change, shows a finding the seat can decide. The row is named security issues. The place is the handler that joins a request path into a file name. The question left open is path traversal. The review seat accepts the finding and sends the change back. That return is one round.

A remark that cites no catalogue entry, or no place in the change, is not a finding. It does not hold the pull request. A request for behaviour the specification did not include, for a performance measurement, for a filed

safety case, or for a numeric product-quality score does not hold it either.

The five entries are review questions about this diff. They are not a score for the running system.

ISO/IEC 25010:2023 names nine product-quality characteristics. This catalogue uses three of them as questions about this change: reliability, security, and maintainability. Those questions ask about the failure the requirement names, the security boundary the requirement names, and whether a teammate can change the code later. The other six characteristics stay outside. So does anything that needs production evidence, a long measurement window, or a filed statement. A performance measurement and a filed safety case stay outside.

The bound on the loop

A cycle is the exchange that starts when the review seat accepts a material finding. The generator writes a further pass, with that finding beside the specification, the requirements, and the constraints. The intent seat confirms the revised change, or sends it back, before the reviewer reads it. The reviewer counter-checks the pull request again, and the review seat decides again. One return from the review seat to the generator is one round.

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 article does not choose the number. The right count is the one the team is willing to stop at, not a quality target. Reaching the cap stops the next generator pass. It does not close a finding, and it does not merge.

What a first team puts in place

A first team writes five things down before a cycle runs. Four of them stand for every change. The fifth is written again for each change, before the generator runs.

What is written	When	What it fixes
Catalogue	Before either instance runs.	The five review questions. A finding cites one row and one place.
Two instances	Before either instance runs.	Separate instructions, inputs, and outputs. The runs may be the same model family or two families.
Two seats	Before either instance runs.	A named human on the intent seat, and a named human on the review seat. One person may hold both, and each seat still writes its own record.
Round cap	Before the first send-back.	A count of rounds. Reaching the cap does not merge.
Three inputs	Before the generator runs on that change.	The specification, the requirements, and the constraints. A later finding may join them. It does not replace them.

Table 2: What a first team puts in place.

The catalogue is the list of review questions. The team writes it down once, and the reviewer instance applies that list. The two instances stay separate runs. The seats are named before either instance runs. The round cap is a count of those review-seat send-backs, and the team chooses the count before the first one. The three inputs are the specification, the requirements, and the constraints for the change at hand. With those five things written, the cycle can run.

Close

This article states the operating model. One AI instance generates the application change from the specification, the requirements, and the constraints. A second instance counter-checks the pull request against the standards catalogue. A human confirms the change before the pull request and decides the review after it. Those decisions bound the revision cycle.

Delivery speed and defect rates belong to a later measurement of the three setups named above. This article does not price a tool. Two separate runs are enough when their instructions, inputs, and outputs stay apart. The cost that remains is human time on the two seats.

The next step is operational, not a further essay. Write down the catalogue, the two instances, the two named seats, and the round cap. Then run one cycle on one change a person could read in one sitting. The full argument, including the citations this article does not repeat, is the scientific article named above.

Oliver Bodemer · e-Softworks, Inc. · October 2026