AI on Both Sides of the Pull Request

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

A Scientific Article on AI-Based Code Generation and Review

Use AI where it helps the business.





Oliver Bodemer · e-Softworks, Inc.

October 2026

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

Abstract

Purpose: This article sets out an operating model for AI in software development with two separate duties. One AI instance creates the application change from specifications and requirements. A second AI instance counter-checks the resulting source code in the pull request against a fixed standards catalogue. The pace argument is direct: when humans both write the change and review the pull request, both of those tasks have to move to AI if the team is to keep the pace. A human stays in the loop on both sides, so the generator cannot open a pull request on unchecked intent and the two instances cannot revise each other without a bound. The article states how a later comparison of business outcomes would be structured across three scenarios: human-only development, AI generation with human review, and dual-AI generation and review with human-in-the-loop controls. Measured outcomes of that comparison lie outside this article.

Method: The article is a design argument. It starts from the human baseline of an author and a separate reviewer, separates the two AI instances by duty, places a human checkpoint before the pull request and a human decision on the review, and states the stop rules that end a generate–review cycle. Measured delivery speed and defect rates lie outside this article.

Contributions: (a) A two-instance model that maps specification and requirements onto generation, and the pull request onto an independent standards review. (b) A human-in-the-loop placement on both sides, with explicit stops so the cycle remains bounded. (c) A standards catalogue for the reviewer instance: specification conformance, Clean Code maintainability, ISO/IEC 25010 reliability, security, and maintainability as review questions, named security issues, and tests tied to the claimed requirements.

Keywords: AI code generation; pull-request review; human-in-the-loop; separation of duties.
Specification-driven development; independent review.

Contents

1 Introduction **2**
- 1.1 Motivation and Problem Statement . . . 2
- 1.2 Scope and Research Questions . . . 3
- 1.3 Contributions . . . 4
- 1.4 Structure of the Article . . . 4

2 Baseline and Operating Model **5**
- 2.1 The Human Pair: Author and Reviewer . . . 5
- 2.2 Two AI Instances, Separated by Duty . . . 5
- 2.3 Human in the Loop on Both Sides . . . 6
- 2.4 The Standards the Reviewer Instance Applies . . . 7

3 From Specification to Pull Request **9**
- 3.1 Inputs: Specifications, Requirements, and Constraints . . . 9
- 3.2 What the Generator Instance May Produce . . . 9
- 3.3 The Human Checkpoint before the Pull Request . . . 10
- 3.4 Failure Modes the Checkpoint Is There to Catch . . . 11

4 Independent Review of the Pull Request **13**
- 4.1 Why the Reviewer Is a Second Instance . . . 13
- 4.2 Counter-check against the Standards . . . 13
- 4.3 What the Reviewer May Comment On . . . 14
- 4.4 The Human Decision on Findings and on the Pull Request . . . 15

5 Speed with a Bound on the Loop **17**
- 5.1 Why Writing and Reviewing Move Together . . . 17
- 5.2 The Revision Cycle and Its Stop Rules . . . 18
- 5.3 Human Stops . . . 19
- 5.4 Where the Human Sits on Each Side . . . 20

6 Discussion **21**
- 6.1 Shared Blind Spots . . . 21
- 6.2 What a Pull-Request Counter-check Can Decide . . . 22
- 6.3 Limits of This Article . . . 23

7 Conclusion **24**
- 7.1 The Operating Model in Short . . . 24
- 7.2 What a First Team Puts in Place . . . 25

1 Introduction

1.1 Motivation and Problem Statement

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. This author/reviewer split is the baseline practice in modern software engineering. Google's public engineering guidance treats the improving health of the codebase as the primary purpose of code review [3]. Cohen's study of 2,500 reviews at Cisco found that defect density drops sharply once more than 200 lines are under review, and it treats 400 lines as the upper limit [4].

AI code generation has changed the pace of the writing side. Controlled experiments report faster completion of the writing task: Peng et al. [5] find that professional programmers with GitHub Copilot finished a JavaScript HTTP-server task 55.8% faster in a randomised controlled experiment, and Cui et al. [6] find a pooled 26.08% increase (standard error 10.3%) in completed tasks across three field experiments with 4,867 developers. Those results sit on the writing side. An observational comparison by Uplevel Data Labs [7], of developers with Copilot access and developers without it, finds a 1.7-minute decrease in pull-request cycle time and a 41% higher bug rate in the group with access. 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.

Studies of automated review report mixed results. In a deployment at Beko, 73.8% of automated comments were resolved, and the average pull-request closure time rose from 5 hours 52 minutes to 8 hours 20 minutes [8]. At Mozilla and at Ubisoft, 8.1% and 7.2% of generated comments were accepted [9]. A study of CodeRabbit comments on open pull requests found that 36.4% were accepted and 56.3% were rejected. The rejections were primarily associated with suggestions that were false positives, redundant, or outside the change, and with suggestions misaligned with the developer's intent or coding practice [10]. None of the figures in this section is a measurement of the operating model this article proposes. The operating model in this article 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. On the generator side, a human confirms the intent before a pull request is opened. On the reviewer side, a human decides the material findings and decides the pull request. Those two decisions are also the stops. They keep the generator and the reviewer from revising each other without a bound, and they keep either instance from carrying a change into the codebase on its own. HULA lets an engineer refine the plan and the code an agent drafts for a task [11]. CodeAgent automates the review with communicating agents and does not place a human at the merge [12]. A later evaluation of LLM review proposes keeping a human in that review, because the model's output can be faulty [13].

1.2 Scope and Research Questions

This article treats one kind of change: an application change that starts from specifications and requirements and ends in a pull request. The generator instance produces that change. The reviewer instance reads the pull request and the standards catalogue. A human decides every merge. The specification-driven approach aligns with the practice now described as Specification-Driven Development (SDD), in which a structured specification is created and agreed upon before implementation, and code is generated from it [14, 15]. The specification is the intent the human later confirms, and it is the input the generator instance reads.

The article is a design argument. It states the duty split, the human checkpoints, and the stop rules. Measured delivery speed and defect rates lie outside it. The scope is software development.

This scope yields four research questions, and they structure the rest of the article.

RQ1: *How can one AI instance create an application change from specifications and requirements so that the change is complete enough to review, while a human still confirms the intent before a pull request is opened?*

Chapter 3 answers this question. It states the inputs the generator instance may use, what it may produce, the human checkpoint that stands before a pull request, and the failure modes that checkpoint is there to catch.

RQ2: *How can a second AI instance, independent of the generator, counter-check that source code in the pull request against a fixed standards catalogue, and which of its findings remain a human decision?*

Chapter 4 answers this question. It separates the reviewer instance from the generator, ties each finding to a standard and to a place in the change, and leaves the material findings and the pull request itself with a human.

RQ3: *Which human-in-the-loop controls, placed on both the generator and the reviewer, keep the two instances inside a bounded revision cycle and still let both former human tasks — writing the change and reviewing it — run at AI speed?*

Chapter 5 answers this question. It states why both duties move together, how a revision cycle runs, and which human decisions end it.

RQ4: *How should a comparison of three operating scenarios be structured to evaluate their business outcomes: (a) a human creates the code and a human reviews it, (b) an AI instance creates the code from specifications while a human reviews it, and (c) an AI instance creates the code and a second, independent AI instance reviews it against a fixed standards catalogue with human-in-the-loop controls?*

This question addresses the structure of the business case for moving from the human baseline to partial AI adoption and finally to the dual-AI operating model. The comparison dimensions are delivery speed (time from specification to merged change), quality (defect density, specification conformance, and security findings), cost (human effort allocated to writing and reviewing), and risk (shared blind spots, override rates, and escalation frequency). Scenario (a) is the baseline. Scenario (b) moves only the generation duty to AI and leaves the review queue on a human. Scenario (c) moves both duties to separate AI instances with human checkpoints on intent confirmation, finding evaluation, and merge

approval. The article states the design argument for scenario (c) and the dimensions along which an empirical comparison would be structured. Measured outcomes for the three scenarios lie outside this article and require empirical evaluation in a production environment.

1.3 Contributions

This article makes three contributions.

First, a two-instance model that maps specification and requirements onto generation, and the pull request onto an independent standards review. The generator writes the change. The reviewer counter-checks it. The two duties stay in separate instances. Chapter 2 states the model, and Chapters 3 and 4 apply it. The separation mirrors the author/reviewer split in human practice [3], extended to two AI instances so that neither duty becomes the bottleneck.

Second, a human-in-the-loop placement on both sides, with explicit stops so the cycle remains bounded. A human confirms intent before a pull request is opened. A human decides the material findings and the pull request after review. Chapter 5 states the stops: accept, reject, override a finding, or reach the round cap. HULA keeps an engineer able to revise the plan and the code an agent drafts [11]. A later evaluation of LLM review keeps a human in the review because the model's output can be faulty [13].

Third, a standards catalogue for the reviewer instance. The catalogue is specification conformance, maintainability in the Clean Code sense [1], ISO/IEC 25010 reliability, security, and maintainability used as review questions [2], named security issues, and tests tied to the claimed requirements. Section 2.4 lists it. The catalogue is a list of review questions. A numeric product-quality score lies outside this article.

1.4 Structure of the Article

The rest of this article is organised as follows. Chapter 2 sets the baseline and the operating model: the human pair, the two AI instances, the human seats on both sides, and the standards catalogue. Chapter 3 answers RQ1 and follows a change from specification and requirements to the human checkpoint in front of the pull request. Chapter 4 answers RQ2 and states how the second instance counter-checks the pull request, and which decisions stay with a human. Chapter 5 answers RQ3 and states the revision cycle and the stops that keep it bounded. RQ4 frames the structure of the business comparison between the human baseline, partial AI adoption, and the dual-AI operating model with human-in-the-loop controls; it is stated in Section 1.2 and its measured outcomes are deferred to empirical evaluation. Chapter 6 discusses shared blind spots, what a pull-request counter-check can decide, and the limits of a design argument. Chapter 7 restates the operating model and what a first team puts in place before either instance runs a cycle.

2 Baseline and Operating Model

2.1 The Human Pair: Author and Reviewer

The baseline is a human pair, and it is scenario (a) of RQ4. One person, the author, writes an application change from a specification and from the requirements that belong to it. A second person, the reviewer, reads that change in a pull request. The author owns the writing. The reviewer owns the decision on the pull request: accept it, send it back to the author, or accept it with an exception that is written down. That split is the duty model Section 2.2 hands to two AI instances.

The review is what keeps the author's pace from settling whether the change is the one that was asked for. Three failures sit on the author's side of the pair. The change misses the specification. It breaks a standard the team already holds. Or it does something nobody asked for. The reviewer reads the pull request for those failures before the change is accepted. Google's public engineering guidance treats this reading as part of keeping the codebase healthy: a reviewer should approve a change once it improves the health of the codebase, and a second person is the one who makes that call [3].

The reviewer's attention is limited, so the author has to keep the change small enough to read. Cohen's study of 2,500 reviews at Cisco found that defect density drops sharply once more than 200 lines are under review, and it treats 400 lines as the upper limit [4]. The pair is how a team lives with that limit. The author prepares a change a second person can read in one sitting. The reviewer spends that sitting on the specification, on the standards the team already holds, and on the diff, and writes down what failed and where it failed.

A repair to the source is a further pass by the author. The reviewer returns a finding. The author changes the code. The pull request comes back for another review. The reviewer records the decision and leaves the branch as the author wrote it. Section 2.3 keeps a human on both of those duties when the writing and the review move to AI. Section 2.4 writes down the standards this human reviewer was applying. This section states the duties. A count of how often the pair finds a defect lies outside it, as it lies outside the article.

2.2 Two AI Instances, Separated by Duty

An instance, in this article, 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. They may be the same model family or two different families. Chapter 6 returns to the case in which both runs miss the same defect. What this section fixes is the duty boundary between the runs.

Figure 2.1 shows that boundary. 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. Chapter 3 states what that change has to contain before a pull request can be opened. 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 in Section 2.3.

The reviewer instance reads two inputs: the pull request, and the standards catalogue of Section 2.4.

It writes findings. A finding cites one entry in that catalogue and one place in the change. Chapter 4 states how a human judges those findings. 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. Chapter 5 names the stops that bound that exchange. It requires a round cap and does not set the number.

The separation has three parts. The instructions differ: the generator is instructed to produce the change the specification describes, and the reviewer is instructed to counter-check the pull request against the catalogue. The inputs differ, as Figure 2.1 shows. The outputs differ: a change on one side, a finding on the other. One run that wrote the change and then accepted its own pull request would collapse the pair in Section 2.1. The second instance is what keeps that pair standing until the human seats in Section 2.3 act.

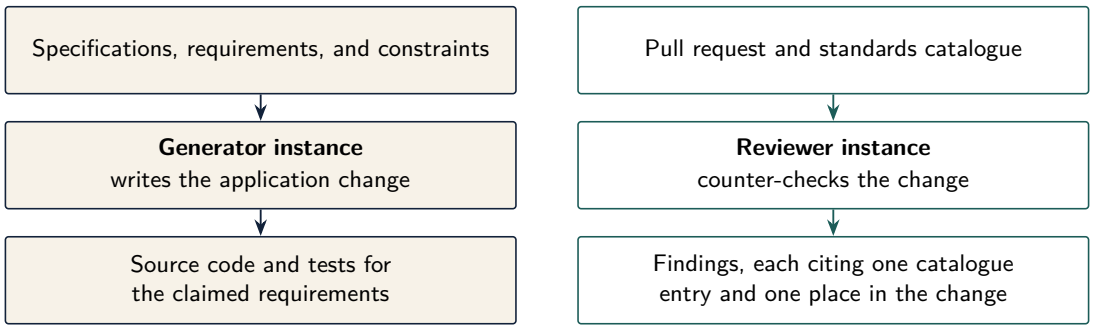

Figure 2.1: The two duties. The generator reads the intent and writes the change. The reviewer reads the pull request and the catalogue and writes findings. A later pass may hand a finding back to the generator. Hidden reasoning stays inside each run.

2.3 Human in the Loop on Both Sides

Two human seats sit on the duty split of Section 2.2. 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. Figure 2.2 shows the two seats and the record each one writes.

The intent seat confirms three things: the intent, the scope, and that the change still matches what was asked. Those three things are the specifications, 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. Chapter 3 states the failure modes that send a change back: invented behaviour, scope that has drifted past the requirements, and unsafe shortcuts. Until the intent seat has confirmed the change, the reviewer instance has no pull request to read.

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 of Section 2.4 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, sends the change back, or overrides the finding and writes the reason. It then accepts the

pull request or rejects it. The reviewer instance stops at the finding. The merge decision is the review seat's output. A remark that cites no catalogue entry, or that asks for behaviour the specification did not include, does not hold the pull request. Chapter 4 applies this decision to the pull request the reviewer has read.

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. Section 5.4 returns to the seats inside a revision cycle. Chapter 5 names the four stops that end that cycle: a human accepts the pull request, a human rejects it, a human overrides the open findings, or a round cap is reached. This section does not set the cap. It requires that both seats have a named human before either instance is allowed to run. The same two decisions are where Chapter 6 places the chance of catching a defect that both instances miss.

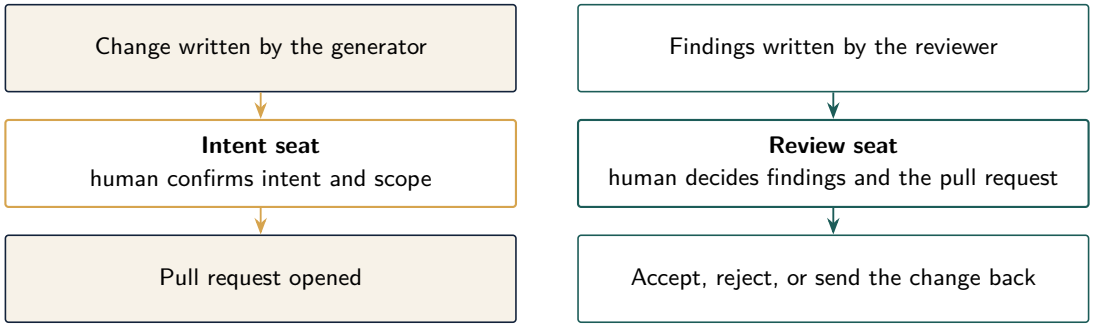

Figure 2.2: The two human seats. The intent seat confirms the change before a pull request exists. The review seat decides the findings and the pull request. Each seat can halt the work in front of it.

2.4 The Standards the Reviewer Instance Applies

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 Table 2.1 and one place in the change. The right-hand column of the table is what that citation has to contain. A remark that names neither an entry nor a place is not a finding under Section 2.3.

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 [1]?	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 [2]?	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 2.1: The standards catalogue. Each finding cites one entry and one place. The right-hand column is what that citation contains. The catalogue is a list of review questions.

The five entries are review questions about the pull request in front of the reviewer. They are not a score for the running system. Reliability, security, and maintainability from ISO/IEC 25010 are asked of this change: the failure the requirement names, the security boundary the requirement names, and whether a teammate can change the code later. Characteristics that need production evidence, a long measurement window, or a filed statement stay outside this counter-check. A performance measurement and a filed safety case are examples of what stays outside. Chapter 6 returns to that boundary.

The catalogue is the list of review questions for the pull request. A numeric product-quality score lies outside this article. Chapter 4 is where the reviewer instance uses the table, and where the review seat decides the findings the table produces.

3 From Specification to Pull Request

3.1 Inputs: Specifications, Requirements, and Constraints

The generator instance reads three written artefacts. A specification states the behaviour the change is there to provide. The requirements are the obligations that belong to that behaviour. The constraints bound the change. Section 2.2 names the three as one block. This section separates them, so that a finding can cite a requirement by name, and so that the intent seat in Section 2.3 has three distinct things to confirm. The team writes all three down before the generator runs. Table 3.1 states what each input contains.

Input	What the generator reads	What is written down
Specification	The behaviour the change is there to provide. This is the intent.	That behaviour, in a form the generator and the intent seat can both read.
Requirements	The obligations that belong to that behaviour. Each one is a claim a test can demonstrate.	The name of each obligation, so the source, the test, and a finding can cite it.
Constraints	The reach of the change, and the behaviour that stays outside it.	The parts of the system the change may touch, and the behaviour it leaves alone.

Table 3.1: The generator's inputs. The team writes each one down before the run. On a later pass a finding is added to this list. The finding does not replace the three inputs.

The three inputs are the three things the intent seat confirms. The intent is the specification. The scope is the set of requirements that belong to it. The match is whether the change kept the constraints. Section 2.3 names that confirmation and places it after the generator has written the change. The specification is agreed before the generator runs, in the sense of specification-driven development already used in the introduction [14, 15]. The generator reads that behaviour. A second behaviour, standing beside the specification, is the invented behaviour Section 3.4 sends back.

The three inputs stay in force on a later pass. When a finding calls for a change to the source, Section 2.2 adds that finding beside them. The finding is another artefact the generator can read. Hidden reasoning from an earlier run is not an input, and it does not cross into this one. The catalogue of Section 2.4 is not an input either. It stays with the reviewer instance. The generator is instructed to produce the change the three inputs describe.

3.2 What the Generator Instance May Produce

The generator produces one application change. The change has two parts: the source, and the tests tied to the requirements the change claims to meet. Both parts are the output named in Section 2.2. This section states when that output is complete enough for the reviewer instance to read it in a pull request. Figure 3.1 shows the path from the three inputs to those two parts.

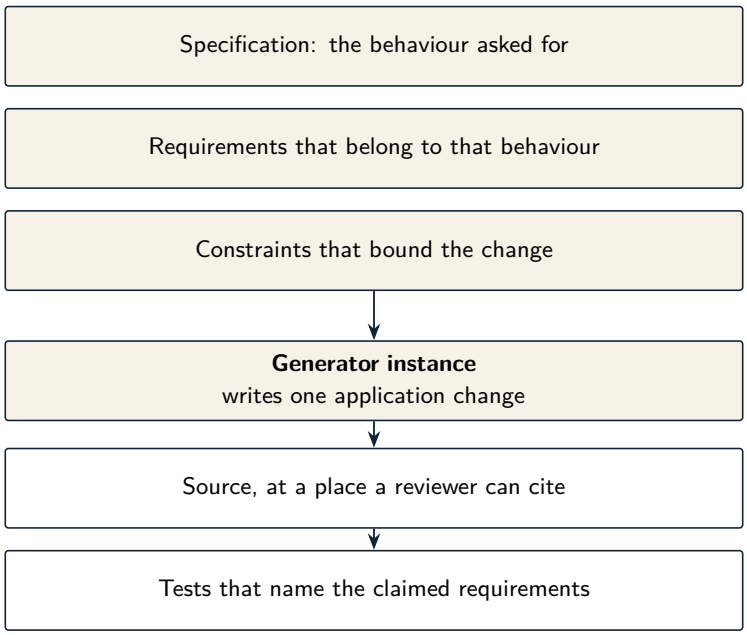

Figure 3.1: What the generator may produce. It reads the three inputs and writes one change: the source, and the tests for the claimed requirements. The pull request opens only after the intent seat has confirmed that change.

The source is the application change the specification describes. Each requirement the change claims is visible there, at a place a later finding can cite. The place is a name, a function, or a path through the change, the same kind of place Table 2.1 asks a finding to show. The intent seat stands between this source and the pull request. Section 3.3 is that checkpoint. Until the seat has confirmed the change, the reviewer instance has no pull request to read.

Each claimed requirement carries a test that names it and demonstrates it. The tests travel with the source, and the intent seat confirms them as part of the same change. The reviewer still asks whether the tests do that work. Table 2.1 keeps an entry for tests of claimed requirements, and the checkpoint can miss a gap. A requirement the change claims, and a test that names it, are both part of the output this section allows.

The change stays one change. Section 2.1 records why the human pair keeps a change small enough to read in one sitting, and the generator inherits that duty. It writes the change the three inputs describe, and it stops. A pull request, a finding, and a merge come later, and they belong to other seats. When Chapter 5 hands a finding back, the same generator writes a further pass, and that pass has the same two parts.

3.3 The Human Checkpoint before the Pull Request

The intent seat is the checkpoint between the generator's output and the pull request. It reads the change from Section 3.2 against the three inputs in Table 3.1. It confirms the intent, the scope, and that the change still matches what was asked. Section 3.1 maps those three confirmations onto the specification, the requirements, and the constraints. The seat writes one record, and it names the human who holds it. Table 3.2 states the two records it may write. A pull request opens only after the confirmation is the record the seat has written.

Record	When the seat writes it	What the record contains
Confirmation	The source and the tests still match the three inputs.	The specification, the requirements the change claims, and the constraints the change kept. The pull request may open.
Send-back	One failure mode in Table 3.3 holds at a place in the change.	That mode, and the place. The pull request stays closed, and the generator reads this record on its next pass.

Table 3.2: The checkpoint record. The intent seat writes one of these two. The reviewer instance reads a pull request only after the confirmation is the record.

The confirmation identifies the three inputs. Those inputs stay the artefacts a later reading uses. The opened pull request carries the source, the tests, and the confirmation, so the reviewer instance takes over a change a human has already matched to the specification, the requirements, and the constraints. Until that confirmation exists, the reviewer has no pull request to read.

A send-back is the other record. It names one failure mode from Section 3.4 and one place in the change. The generator reads it on the next pass, beside the three inputs. It does not replace those inputs. It is not a reviewer finding. A finding cites one entry in the catalogue. A send-back cites one failure mode of this checkpoint.

The seat confirms the change. The catalogue stays the reviewer's question, applied in Chapter 4, and the merge stays the review seat's decision. This section does not set how many times the seat may send a change back. Chapter 5 holds that bound.

3.4 Failure Modes the Checkpoint Is There to Catch

The checkpoint watches for three failures. Each one is a way the change has left the inputs the generator was given. Section 2.1 names three failures on the author's side of the human pair: the change misses the specification, it breaks a standard the team already holds, or it does something nobody asked for. This checkpoint takes the behaviour nobody asked for as invented behaviour, a missed or extra requirement as scope drift, and a breach of the constraints as an unsafe shortcut. The broken standard remains the reviewer's question, asked against the catalogue once a pull request exists. Table 3.3 states what the seat sees, and what the send-back has to name.

Mode	What the seat sees	What the send-back names
Invented behaviour	The source provides behaviour the specification did not state.	That behaviour, and the place it appears.
Scope drift	The change adds a requirement the named set does not contain, or it drops a requirement that set does contain.	The requirement, and the place.
Unsafe shortcut	The change reaches a claimed requirement by leaving the constraints.	The constraint, and the place.

Table 3.3: The three failure modes of the checkpoint. A send-back names one mode and one place. Review starts only after the seat has written a confirmation instead.

Invented behaviour is a second behaviour standing beside the specification. The generator was given

the behaviour the specification states. A path through the source that does something further is this failure. Scope drift is about the named requirements. The change claims one the list does not contain, or the source no longer meets one the list does contain. The tests make the claim visible, because each claimed requirement carries a test that names it.

An unsafe shortcut is a breach of the constraints. The change still claims a requirement, and it reaches that requirement by touching a part of the system the constraints did not allow, or by a path the constraints left alone. A secret written into the source, or a check the constraints required and the change left out, is this kind of place. The named security issues in Table 2.1 stay with the reviewer. The checkpoint asks whether the constraints held. The reviewer asks the catalogue question after the pull request exists.

A change that meets any row of Table 3.3 goes back to the generator. Review starts only after the checkpoint has written a confirmation. The checkpoint can still miss a failure that remains in the change. Chapter 6 places that chance at this seat and at the review seat. How often the seat misses is a question for a later measurement.

4 Independent Review of the Pull Request

4.1 Why the Reviewer Is a Second Instance

The reviewer is a second instance so that the review is a counter-check of the change. An instance is one run, with its own instructions, its own inputs, and its own output, as Section 2.2 defines it. The reviewer may be the same model family as the generator, or a different family. The separation that matters is the run. The run that wrote the change is not the run that asks whether the pull request meets the catalogue. One run that did both would collapse the pair in Section 2.1.

The human pair already keeps those duties apart. The author writes. A second person reads the pull request and makes the call. Section 2.1 takes that call from Google's engineering guidance [3]. The reviewer instance carries the catalogue side of it. The review seat in Section 4.4 remains the human who decides the findings and the pull request. The instance stops at the finding.

Figure 2.1 shows the two inputs. They are the pull request, and the standards catalogue of Table 2.1. The pull request carries the source, the tests, and the confirmation from Section 3.3. The confirmation identifies the specification, the requirements the change claims, and the constraints the change kept. It records that the intent seat matched the change to those inputs. It does not answer the catalogue. The reviewer still asks the five questions of the change in front of it.

Hidden reasoning from the generator's run does not cross this boundary. What the reviewer reads is an artefact: the pull request, and the catalogue. The instructions differ as well. The generator was instructed to produce the change the three inputs describe. The reviewer is instructed to counter-check that pull request and to write findings. Editing the branch, and merging it, sit outside that output. Section 4.2 states the counter-check. Section 4.3 states what a finding may ask for.

4.2 Counter-check against the Standards

The counter-check applies the catalogue to one pull request. The reviewer takes each row of Table 2.1 and asks that row's question of this change. A finding cites one row and one place. The place is a name, a function, or a path through the change. Table 4.1 states where in the pull request the reviewer looks for each row.

Catalogue row	Where the reviewer looks
Specification conformance	The source, read against the behaviour the confirmation names, and against the requirements that confirmation lists.
Clean Code maintainability	The names, the functions, and the paths a teammate who did not write the change would have to follow.
ISO/IEC 25010 review questions	The place where a claimed requirement speaks of a failure, a security boundary, or a later change to the code.
Named security issues	The place an input, a secret, a path, or a request is handled.
Tests of claimed requirements	Each test, and the requirement that test names.

Table 4.1: Where the counter-check looks. Each finding cites one of these rows and one place. The review questions themselves are Table 2.1.

The finding states the row, the place, and the way that place leaves the row's question open. That is the record Section 2.3 calls a finding. A remark that names neither a row nor a place is not that record. The reviewer writes the finding and stops. The review seat in Section 4.4 decides whether the finding holds the pull request.

The counter-check starts where the checkpoint ended. The checkpoint asked whether the change had left the three inputs, and a confirmation is already in the pull request. The reviewer can still find a miss that confirmation left in the change. The records stay distinct. A send-back cites one mode in Table 3.3. A finding cites one row in Table 2.1. Both name a place. They are written by different seats, at different times.

The five rows remain review questions about this diff. Section 2.4 has already set the boundary: a performance measurement and a filed safety case stay outside, and a numeric product-quality score lies outside this article. The same boundary holds for the counter-check.

4.3 What the Reviewer May Comment On

The reviewer comments on the pull request. The comment that can hold it is a finding. Section 4.2 requires that finding to name one row of Table 2.1, one place, and the way that place leaves the row's question open. This section states what that comment may ask for. Table 4.2 separates a catalogue finding from a remark that does not hold the pull request.

Comment	What it contains	What it may ask for
Catalogue finding	One row of Table 2.1, one place, and the way that place leaves the row's question open.	A later generator pass that answers that row at that place, in the source or in the tests.
Remark	No catalogue row, or no place.	Nothing. The remark does not hold the pull request.
Outside the catalogue	A request for behaviour the specification did not include, or for a performance measurement, a filed safety case, or a numeric product-quality score.	Nothing. The specification and the catalogue stay as they are.

Table 4.2: What the reviewer may comment on. Only a catalogue finding can hold the pull request, and only for a repair that answers its row at its place.

A catalogue finding asks for one repair. The later generator pass answers that row at that place. The finding joins the specification, the requirements, and the constraints. Section 3.1 keeps those three in force. The finding does not replace them. The reviewer writes the finding and stops. Editing the branch sits with the generator, on that later pass.

The repair answers the row. It does not add behaviour the specification did not include. A comment that asks for such behaviour does not hold the pull request. That behaviour is invented behaviour or scope drift, and Table 3.3 already gives those modes to the intent seat. Closing an open catalogue question is a different request. A name a teammate cannot read, a test that does not demonstrate the requirement it names, or a named security issue at the place an input is handled, can be the open question. The repair leaves the specification as it stood.

A performance measurement, a filed safety case, and a numeric product-quality score stay outside, as Section 2.4 set them. A comment that asks for one of those does not hold the pull request. The revision a finding does ask for is a new pass by the generator. That pass returns to the reviewer inside the cycle of Chapter 5. This section does not set how many times the exchange may run. Whether the finding holds the pull request is the review seat's decision, in Section 4.4.

4.4 The Human Decision on Findings and on the Pull Request

The review seat decides every material finding, and it decides the pull request. A finding is material when it cites one entry in Table 2.1 and one place in the change, and leaving it open would leave that catalogue question unanswered at that place. Section 2.3 names that test. The reviewer instance has already stopped at the finding. The seat writes the records in Table 4.3. The instance does not merge.

Record	When the seat writes it	What the record contains
Finding accepted	The finding is material, and the seat agrees the question is open at that place.	The catalogue row, the place, and the send-back. The generator reads the finding on its next pass.
Finding overridden	The finding is material, and the seat decides it does not hold the pull request.	The reason. The finding stays on the record and does not send the change back.
Pull request accepted	No material finding remains open.	That the change may merge. The reviewer instance does not write this record.
Pull request rejected	The seat ends the change.	That the change will not merge, and that no further generator pass is asked for.

Table 4.3: The review seat's records. Each material finding is accepted or overridden. The pull request is then accepted, rejected, or left for the send-back. The reviewer instance does not merge.

The seat takes each material finding first. Accepting the finding sends the change back. Those two names are one record. The generator's next pass reads the finding beside the three inputs, and the pull request comes back for another counter-check. Overriding the finding closes it without a change to the source. The reason is part of the record, so a later reading can see why the question was left as it stood.

The pull request is the next decision. When any material finding has been accepted, the change goes back, unless the seat rejects the pull request instead. A rejection ends the change. It is not a send-back, and it is not an override. When every material finding has been overridden, or when the reviewer wrote none, the seat may accept the pull request. That acceptance is the merge. Figure 2.2 shows it as the review seat's output. The reviewer instance does not merge.

A remark that Table 4.2 keeps from holding the pull request is not a material finding. The seat does not accept it, override it, or send the change back for it. The four stops that end the cycle, including the round cap, stay in Chapter 5. This section does not set the cap. It requires a named human on the review seat before a finding is decided and before a pull request is merged.

5 Speed with a Bound on the Loop

5.1 Why Writing and Reviewing Move Together

Writing and reviewing set the pace together. The introduction reports a faster writing task in Peng et al. [5] and more completed tasks in Cui et al. [6]. It also reports the review measurements this chapter can use: Uplevel's observational comparison found a 1.7-minute decrease in pull-request cycle time and a 41% higher bug rate in the group with Copilot access [7], and the Beko deployment resolved 73.8% of automated comments while average closure time rose [8]. None of those results measures scenario (c). They leave the writing task able to move, and they leave the review as a queue the writing-side results do not clear. Table 5.1 sets the three scenarios of RQ4 side by side. The table assigns duties. It is not a measurement of speed.

Scenario	Who writes	Who counter-checks	Human stops
(a) Human pair	The author.	The reviewer, a second person.	The pair itself. Section 2.1 is this row.
(b) Generation only	The generator instance.	A human reviewer.	The intent seat confirms the change before the pull request. The review sitting remains.
(c) Both duties	The generator instance.	The reviewer instance, against the catalogue.	The intent seat confirms the change before the pull request. The review seat decides the findings and the merge.

Table 5.1: Who does the work in the three scenarios of RQ4. Scenario (c) moves both duties. The human stops stay. The table is not a measurement of speed.

Scenario (a) is the human pair. One person writes. A second person reads the pull request. The pace is those two sittings, and Section 2.1 already keeps the change small enough for the second sitting.

Scenario (b) moves the writing and leaves the review. The generator instance produces the change. The intent seat still confirms it before a pull request is opened. The human review is then the queue. A faster writer does not shorten the reading that sitting requires.

Scenario (c) moves both duties. The generator writes. The reviewer instance counter-checks the pull request against the catalogue. The intent seat still confirms the change before the pull request. The review seat still decides the material findings and the merge. The instances do the two former human tasks. The seats do not take those tasks back. They stop the work in front of them. Neither instance confirms its own intent, and neither instance merges.

The duty split alone does not end the exchange. A finding can send the change back, and a later pass can return it, with nothing in Section 2.2 to stop the next return. Section 5.2 states that cycle and the four stops that bound it.

5.2 The Revision Cycle and Its Stop Rules

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. Figure 5.1 shows one round. One return from the review seat to the generator is one round. An intent-seat send-back is not a round. The reviewer has no pull request to read until the confirmation is written.

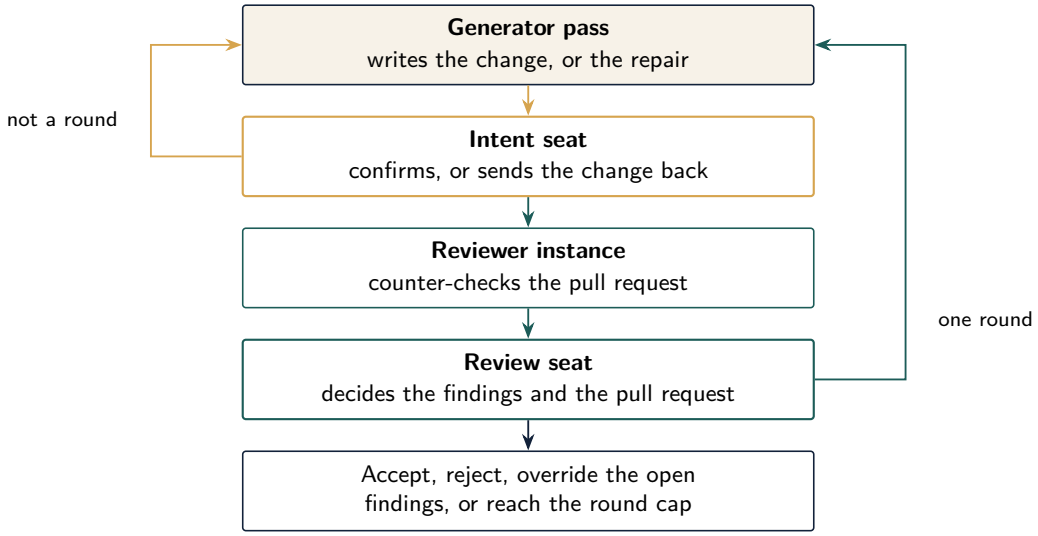

Figure 5.1: One round of the revision cycle. The left return is an intent-seat send-back, and it is not a round. The right return is an accepted finding, and it is one round. The round cap stops the next return from the review seat. It does not merge.

The cycle ends on the first of the four stops in Table 5.2. Three are records the review seat writes. The fourth is the round cap the team sets before either instance runs. This section does not set that number. It requires the cap to be a count of rounds, chosen before the first send-back, so that reaching it stops the next generator pass.

Stop	What ends the cycle	What remains
Pull request accepted	The review seat writes the acceptance in Table 4.3.	No further pass. The reviewer instance does not merge.
Pull request rejected	The review seat writes the rejection.	The change stays unmerged. No further pass is asked for.
Open findings overridden	Every open material finding has the reason in Table 4.3.	The override is not the merge. 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.

Table 5.2: The four stops. The review seat writes the first three. The team sets the cap before the cycle. Reaching the cap ends further revision and leaves the merge where it was.

Accepting the pull request is the merge, and it is the review seat's record. Rejecting the pull request ends the change. Overriding the open material findings closes them with reasons. That record does not merge by itself. When no material finding remains open, the seat may accept the pull request, as Section 4.4 already allows.

The cap bounds the exchange. Each accepted finding that sends the change back counts as one round. When the next send-back would pass the cap, the generator does not run that pass. The review seat's last records stand. An open material finding still holds the pull request, until the seat overrides it or rejects the change. The cap does not close a finding, and it does not merge. Section 5.3 separates these decisions from a log of what the instances did. Section 5.4 returns to which human holds each seat inside the round.

5.3 Human Stops

The four stops in Table 5.2 are decisions. The review seat writes three of them when it decides the findings and the pull request. The team writes the round cap before the first round. A log of what an instance did is a record of that pass. The decision is the stop. Table 5.3 separates the log from the decision.

Record	Who writes it	What follows
Change, or repair	The generator instance.	The intent seat reads it. The record does not open the pull request, and it does not merge.
Finding	The reviewer instance.	The review seat reads it. The finding does not accept, override, or merge itself.
Log of the pass	Either instance, for what that pass was given, what it wrote, and the round count.	A record of the pass. The count is how the cap already set is applied. The log does not merge.
One of the four stops	The review seat, or the team for the cap it set before the cycle.	The cycle ends as that row of Table 5.2 says.

Table 5.3: A log is not a stop. The instances write the change, the finding, and the round count. The decision that ends the cycle is one of the four stops.

The generator writes the change, or the repair. That output waits for the intent seat. The reviewer writes findings. Those findings wait for the review seat. Neither output is an acceptance, a rejection, an override, or the cap. Hidden reasoning stays inside the run, as Section 2.2 requires. What the log may carry across is an artefact the other side can read, and the round count.

The round count is part of the log. It records how many times the review seat has sent the change back. When the next send-back would pass the cap, the generator does not run that pass. The stop is the cap the team set, applied to the count. The log line that carries the count is not a new decision, and it does not close a finding. An instance does not turn its own log into one of the four stops.

The intent seat's send-back is a human decision, and it is not one of those four. It halts the reviewer for that pass. The cycle continues, and the return is not a round, as Section 5.2 defines a round. The decisions that end the cycle remain the rows of Table 5.2.

5.4 Where the Human Sits on Each Side

Inside a round the two seats keep the places Figure 5.1 gives them. The intent seat sits after the generator and before the reviewer reads. The review seat sits after the findings and before the merge. Figure 2.2 shows the same seats on a first pass. This section states what each seat writes once a cycle is under way. Table 5.4 is that split.

Seat	When, in a round	What the human may write
Intent	After the generator pass, before the reviewer reads.	A confirmation, or a send-back. Not a finding, and not the merge.
Review	After the findings of that pass exist.	Accept a finding, override the open findings, accept the pull request, or reject it. Not the confirmation, and not an edit to the branch.

Table 5.4: The two seats inside a round. Each writes its own record. One person may hold both, in that order.

On the generator side the human confirms the revised change, or sends it back, before the reviewer reads it. The confirmation is the record of Section 3.3, written for this pass. The send-back names one failure mode and one place. The reviewer has nothing to read until a confirmation replaces it. The intent seat does not write a finding, and it does not merge.

On the reviewer side the human decides the material findings and the pull request. Accepting a finding starts one round. Overriding the open findings, accepting the pull request, or rejecting it ends the cycle, as Table 5.2 says. The review seat does not replace the confirmation, and it does not edit the branch. The generator writes the repair. The reviewer instance stops at the finding.

One person may hold both seats, or two people may hold one each. Each seat writes its own record. When one person holds both, the intent record is written first, and the review record is written only after the reviewer instance has produced its findings. That person does not skip the instance between the seats. When two people hold the seats, the intent seat does not decide the pull request, and the review seat does not rewrite the confirmation. Section 2.3 requires both seats to be named before either instance runs. Chapter 6 places a shared miss at these two decisions. How often that miss occurs lies outside this article.

6 Discussion

6.1 Shared Blind Spots

Both instances can miss the same defect. The runs may be the same model family or two different families, as Section 2.2 allows. 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. This section does not measure which pairing misses less. An evaluation of LLM review keeps a human in the review because the model's output can be faulty [13]. That is the reason the seats remain. It is not a count of shared misses. Table 6.1 states where a human can still act.

Miss	Intent seat	Review seat
Failure mode	Sends the change back under Table 3.3. The reviewer does not read it.	If a later confirmation let the change through, and a finding names the place, the seat can accept that finding.
Catalogue finding	The checkpoint does not apply the catalogue.	Accepts the finding and sends the change back, or overrides it with a reason.
Shared miss	Sends the change back only when the defect is one of the three failure modes.	May reject the pull request. The absence of a finding does not require acceptance.
Finding that does not hold	Not this seat.	Overrides it and writes the reason. The finding does not hold the pull request.

Table 6.1: Where a miss can still be caught. A shared miss is a defect both instances left in the change. The review seat catches it by rejecting the pull request. It does not write the reviewer's finding.

A shared miss is a defect still in the change after both runs. The generator wrote it, or left it. The reviewer wrote no finding that names it. The intent seat can send the change back when the defect is invented behaviour, scope drift, or an unsafe shortcut. A broken standard stays the reviewer's question, as Section 3.4 set it. The checkpoint does not apply the catalogue, so a shared miss of that kind passes the intent seat when the three inputs still match.

The review seat catches that miss by rejecting the pull request. Section 4.4 allows acceptance when the reviewer wrote no finding. Acceptance is not required. The seat may reject when it sees a catalogue question still open. The rejection ends the change. The seat does not write a finding in the reviewer's place, and it does not edit the branch. A repair on a later pass still needs a finding the reviewer wrote. Where the reviewer wrote none, the catch is that the change stays unmerged.

A finding that does not hold is a different miss. The reviewer wrote a row and a place, and the question is not open there. A study of CodeRabbit comments reports that rejections were primarily associated with false positives, redundant comments, or comments outside the change, and with misalignment with the developer's intent or coding practice [10]. The review seat's override is the record for that kind of finding. The override writes the reason and does not send the change back. How often a shared miss occurs, in this operating model, lies outside this article. Section 6.3 keeps that measurement with the others.

6.2 What a Pull-Request Counter-check Can Decide

The counter-check decides the five questions in Table 2.1, each at a place in the change. It does not decide the running system, and it does not produce a score. ISO/IEC 25010:2023 is a product quality model of nine characteristics: functional suitability, performance efficiency, compatibility, interaction capability, reliability, security, maintainability, flexibility, and safety [2]. This article uses three of them, reliability, security, and maintainability, as questions about the diff. It does not evaluate the product. Table 6.2 draws that boundary.

Matter	What the counter-check can decide	What it leaves
Five catalogue rows	Whether that row's question is open at a place in this change.	A judgement of the running system, and a numeric product-quality score.
Reliability, security, and maintainability	Whether they hold as questions about this diff and the behaviour the requirement claims.	The same characteristics of the product, over a measurement window.
Named security issues	Whether the change introduces injection, cross-site scripting, cross-site request forgery, a secret in source, path traversal, or server-side request forgery, at a place.	An issue this list does not name, and a test of the deployed system.
The other six characteristics	Nothing. They are not rows of Table 2.1.	The measurement or the filed statement those characteristics require. A performance measurement and a filed safety case are the examples Section 2.4 already gives.

Table 6.2: What the counter-check can decide. The decision is a review question about this change. A product evaluation, and a numeric score, stay outside.

Specification conformance is the first catalogue row. It asks whether the change implements the claimed requirements and stays inside the behaviour the specification asked for. That question is not an evaluation of functional suitability for the product. Clean Code maintainability asks whether a teammate who did not write the change can read it and change it, at a name, a function, or a path. The tests row asks whether the tests demonstrate the requirements the change claims. The counter-check can say that a claimed requirement has no such test. It does not report a coverage figure.

The named security issues are this article's list. The counter-check can decide whether the diff introduces one of them at a place. It does not decide an issue the list does not name. Reliability, security, and maintainability are asked of the behaviour this change claims. The other six characteristics of the product quality model stay outside the catalogue. Performance efficiency needs a measurement. Safety, in this article, stays with a filed safety case. Neither is a finding.

The boundary is the diff in front of the reviewer. A characteristic that needs production evidence, a long measurement window, or a filed statement stays with that evidence. Section 6.3 states the further limit of this article: the operating model is a design argument, and the measurements of speed, defects, and shared misses are not taken here.

6.3 Limits of This Article

This article states an operating model. It answers RQ1, RQ2, and RQ3 by duties, checkpoints, and stops. It answers RQ4 by naming the three scenarios and the dimensions of a comparison. It does not report the outcomes. Table 6.3 separates the design from the measurement.

Question	What this article fixes	What remains a measurement
RQ1	The three inputs, the generator's change, the checkpoint, and the three failure modes.	Whether the change that was confirmed is the change that later merges.
RQ2	The second instance, the catalogue finding, and the review seat's records.	Counts of findings, overrides, and rejections across changes.
RQ3	Both duties move. One round is one review-seat send-back. The four stops are named. The cap is a count this article does not set.	Time from specification to merged change, with that cap held still.
RQ4	Scenarios (a), (b), and (c), and the dimensions of the comparison.	Delivery speed, defect density, human effort, shared-miss frequency, and override rate, with the duties held still.

Table 6.3: The limit of the design argument. The left-hand column is answered in the chapters above. The right-hand column is not taken in this article.

A single change can be decided. Specification conformance and a named security issue are questions about the diff in front of the reviewer, as Chapter 4 states them. A rate across changes is a different result. Defect density needs a denominator this article does not define. Human effort on the two seats, the frequency of a shared miss, and the rate of overrides are the same kind of result. Section 6.1 states the shared miss and does not count it.

The studies in Section 5.1 stay in the settings those papers report. They are not a measurement of scenario (c), and they do not set the round cap. The cap remains a count the team chooses before the first send-back. This article does not choose the count, a model family, a repository, or how many people hold the seats.

Section 1.2 names escalation frequency among the risk dimensions. This article has no path beyond the two seats and the four stops. A later count of escalations would have to define the event it counts. The seats, the catalogue, and the cap are what that measurement would hold still. Section 7.2 states what a first team writes down before either instance runs.

7 Conclusion

7.1 The Operating Model in Short

One 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. Table 7.1 puts that model in one place. The chapters above state each row. The table adds no rule.

Step	What the instance does	What the human does
Generate	Writes the change from the three inputs, including the tests that name the claimed requirements.	The intent seat confirms the change before the pull request, or sends it back.
Review	Counter-checks the pull request against the catalogue and writes findings.	The review seat accepts a finding, overrides it, accepts the pull request, or rejects it. The instance does not merge.
Repeat	On a later pass, reads an accepted finding beside the three inputs.	Each review-seat send-back is one round. The cap stops the next pass. It does not merge.

Table 7.1: The operating model in one place. Each row is stated in the chapters above. The round cap is still a count this article does not set.

The generator writes one change. The source is at a place the reviewer can cite, and the tests name the requirements the change claims. The intent seat confirms that change, or sends it back for invented behaviour, scope drift, or an unsafe shortcut. The pull request opens on the confirmation. Until that record is written, the reviewer has nothing to read.

The reviewer reads the pull request and the catalogue. A finding names one row and one place, and the way that place leaves the row's question open. The review seat accepts the finding and sends the change back, or overrides it and writes the reason. It then accepts the pull request or rejects it. Accepting the pull request is the merge. The reviewer instance does not write that record. One return from the review seat is one round. When the next send-back would pass the cap, the generator does not run that pass. An open material finding still holds the pull request.

An illustration, not a measured change, shows those records on one pass. 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 intent, the scope, and the constraints. The pull request opens. The reviewer writes one finding: the catalogue row is named security issues, the place is the handler that joins the request path into the file name, and the question left open is 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 catalogue row. The remark does not hold the pull request.

RQ4 names the comparison among the human pair, generation with a human still reviewing, and

both duties moved. The comparison is structured in Section 1.2. Its outcomes are not reported here. What a first team writes down before either instance runs is Section 7.2.

7.2 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. Table 7.2 lists them. None of the five is a measurement, and none of them sets a product-quality score.

What is written	When	What it fixes
Catalogue	Before either instance runs.	The five review questions in Table 2.1. A finding cites one row and one place.
Two instances	Before either instance runs.	Separate instructions, inputs, and outputs, as Section 2.2 states them. 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. This article does not choose the number. 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 7.2: What a first team puts in place. The first four stand before any cycle. The three inputs are written for each change.

The catalogue is the list of review questions. The team writes it down once, and the reviewer instance applies that list. The team does not add a sixth row for a performance measurement, a filed safety case, or a numeric score. The two instances stay separate runs. Hidden reasoning stays inside each run. What crosses is the change in one direction and the finding in the other.

The seats are named before either instance runs. The intent seat confirms the change, or sends it back, before the reviewer reads it. The review seat decides the material findings and the pull request. When one person holds both, the intent record is written first. The review record is written only after the reviewer instance has produced its findings. The round cap is a count of those review-seat send-backs. The team chooses the count before the first one. This article does not choose it. When the next send-back would pass the cap, the generator does not run that pass, and an open material finding still holds the pull request.

The three inputs are the specification, the requirements, and the constraints for the change at hand. The generator reads them. The intent seat confirms them. On a later pass an accepted finding joins them and does not replace them. With those five things written, the cycle in Section 5.2 can run. The measurements in Section 6.3 remain outside it.

Bibliography

[1] Robert C. Martin. *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall, 2008. ISBN 978-0132350884.

[2] ISO/IEC 25010:2023. *Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Product quality model*. International Organization for Standardization, November 2023.

[3] Google. *Google Engineering Practices: Code Review*. Available at https://google.github.io/eng-practices/review/. Accessed 2026.

[4] Jason Cohen. *The Largest Case Study of Code Reviews Ever*. CMCrossroads, 31 October 2006. SmartBear study of code review on Cisco MeetingPlace: 2,500 reviews and 3.2 million lines. Defect density falls once more than 200 lines are under review; the article sets 400 lines as the upper limit. Available at https://www.cmcrossroads.com/article/largest-case-study-code-reviews-ever.

[5] Sida Peng, Eirini Kalliamvakou, Peter Cihon, and Mert Demirer. *The Impact of AI on Developer Productivity: Evidence from GitHub Copilot*. arXiv:2302.06590, 13 February 2023. Randomised experiment with 95 professional programmers on one JavaScript HTTP-server task.

[6] Zheyuan (Kevin) Cui, Mert Demirer, Sonia Jaffe, Leon Musolff, Sida Peng, and Tobias Salz. *The Effects of Generative AI on High-Skilled Work: Evidence from Three Field Experiments with Software Developers*. *Management Science*, published online 27 February 2026. doi:10.1287/mnsc.2025.00535. Also SSRN 4945566. Three field experiments, 4,867 developers; pooled 26.08% increase in completed tasks (standard error 10.3%).

[7] Uplevel Data Labs. *Can Generative AI Improve Developer Productivity?* September 2024. Observational comparison of 351 developers with GitHub Copilot access and 434 without. Available at https://uplevelteam.com/hubfs/Content%20Assets/Can%20Generative%20AI%20Improve%20Developer%20Productivity.pdf.

[8] Umut Cihan, Vahid Haratian, Arda İçöz, Mert Kaan Gül, Ömercan Devran, Emircan Furkan Bayendur, Baykal Mehmet Uçar, and Eray Tüzün. *Automated Code Review In Practice*. arXiv:2412.18531, 2024. Presented at the ICSE 2025 Software Engineering in Practice track. Deployment at Beko: 4,335 pull requests in the three projects studied; 73.8% of automated comments resolved; average pull-request closure time from 5 hours 52 minutes to 8 hours 20 minutes.

[9] Doriane Olewicki, Leuson Da Silva, Suhaib Mujahid, Arezou Amini, Benjamin Mah, Marco Castelluccio, Sarra Habchi, Foutse Khomh, and Bram Adams. *Impact of LLM-based Review Comment Generation in Practice: A Mixed Open-/Closed-source User Study*. arXiv:2411.07091, 11 November 2024. RevMate at Mozilla and Ubisoft, more than 587 patch reviews. Accepted comments: 8.1% at Mozilla and 7.2% at Ubisoft.

[10] Hong Yi Lin, Mingzhao Liang, Kla Tantithamthavorn, and Patanamon Thongtanunam. *Is Agentic Code Review Helpful? Mining Developers' Feedback to CodeRabbit Reviews in the Wild*. arXiv:2607.03316, 3 July 2026. CodeRabbit comments on 10,191 pull requests: 36.4% accepted and 56.3% rejected. Rejections were primarily associated with invalid suggestions, and with misalignment with developer intent or coding practice.

[11] Wannita Takerngsaksiri, Jirat Pasuksmit, Patanamon Thongtanunam, Chakkrit Tantithamthavorn, Ruixiong Zhang, Fan Jiang, Jing Li, Evan Cook, Kun Chen, and Ming Wu. *Human-In-the-Loop Software Development Agents*. arXiv:2411.12924, 2024. ICSE 2025 Software Engineering in Practice. HULA, deployed in Atlassian Jira, lets engineers refine an agent's coding plan and source code.

[12] Xunzhu Tang, Kisub Kim, Yewei Song, Cedric Lothritz, Bei Li, Saad Ezzini, Haoye Tian, Jacques Klein, and Tegawendé F. Bissyandé. *CodeAgent: Autonomous Communicative Agents for Code Review*. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 11279–11313, Miami, 2024. arXiv:2402.02172. Multi-agent automation of review tasks, with no human gate in the evaluation.

[13] Umut Cihan, Arda İçöz, Vahid Haratian, and Eray Tüzün. *Evaluating Large Language Models for Code Review*. arXiv:2505.20206, 26 May 2025. Proposes a human-in-the-loop LLM code review because the model's output can be faulty.

[14] Anna Gutowska. *What is spec-driven development?* IBM Think, 19 May 2026. Available at https://www.ibm.com/think/topics/spec-driven-development.

[15] Liu Shangqi. *Spec-driven development: Unpacking one of 2025's key new AI-assisted engineering practices*. Thoughtworks, 4 December 2025. Available at https://www.thoughtworks.com/insights/blog/agile-engineering-practices/spec-driven-development-unpacking-2025-new-engineering-practices.

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