

When Review Becomes the Job

Why review fatigue is a pattern problem, not a capacity problem, and what redesigning review looks like at the individual and the organisational level.

When the job changed shape

Two conversations with senior software developers, both about code review. The numbers behind them, and the practical question one of them asked, are what this paper sets out to answer.

A senior software developer told me last week that most of his work had moved to reviewing code. He had been doing review before, that part wasn't new. What was new was the volume.

He described a week in numbers I won't repeat here, partly to protect him and partly because the specific count isn't the point. The point is that what had been a portion of his week had become the bulk of it. He wasn't complaining about that. He was working out what to do with it.

A second software developer in the room asked the obvious follow-up. How do you avoid review fatigue when that's the trajectory?

I had an answer ready, of the sort you give in a corridor. Reduce pull request size, scope the work more tightly, that kind of advice. He nodded politely and moved on, and I found myself unsatisfied with what I had said. Not because the items were wrong, but because they were the wrong shape for the question. He wasn't asking for a checklist. He was asking how to keep doing this work for years without breaking.

This paper is the answer I should have given.

The argument it makes is short enough to state up front. Review fatigue isn't a capacity problem. It's a pattern problem. The job of review has changed shape, and the technique most senior software developers were trained on is the wrong instrument for the new shape. Solving fatigue requires redesign at two levels: how the individual reviewer engages, and how the organisation structures review across pipelines. The rest of the paper is what each of those redesigns looks like, and how to know whether yours is working.

The bottleneck moved

The conversation in section one is the one happening across teams everywhere AI-assisted development has become the default. The numbers behind it are now clear enough to argue from.

AI adoption is approaching saturation among professional software developers. The 2025 Stack Overflow Developer Survey, with over 49,000 respondents, found 84% using or planning to use AI tools in their work. JetBrains' State of Developer Ecosystem 2025, with 24,534 developers, put the figure at 85%. The 2025 DORA report, surveying nearly 5,000 technology professionals, put it at 90%. The exact figure varies by survey methodology. The point is that adoption is no longer the question.

The question is what that adoption did to the rest of the system.

What the telemetry shows

Faros AI's 2025 Engineering Report drew on telemetry from 10,000 developers across 1,255 teams. Teams with high AI adoption completed 21% more tasks and merged 98% more pull requests than their lower-adoption peers. PR size grew by 154%. Time spent in review grew by 91%. Bugs grew by 9%. Organisational DORA delivery metrics stayed flat. More code, larger pull requests, longer reviews, more bugs, and no measurable improvement in delivery performance.

The 2026 follow-up drew on 22,000 developers across more than 4,000 teams, and the trajectory continued. Median time in PR review was up 441%. Average time spent in review was up close to 200%. PR sizes had grown a further 51%.

WHAT REVIEW COLLAPSE LOOKS LIKE

31% more pull requests now merge without any human review. The figure isn't policy-driven. It is the visible edge of a review system that has stopped functioning at the rate it was designed for.

That isn't fatigue. That is the pattern breaking. When the system can no longer sustain the volume of review it was designed to perform, review stops happening. Whatever you call the thing that's left, it isn't review.

The vendors are admitting it

The vendors of the tools that accelerated production are now admitting where the constraint sits. Cursor's CEO Michael Truell publicly acknowledged that while Cursor has accelerated code production, the review side has not moved in three years. Cursor then acquired Graphite, a code review startup. That acquisition is more honest about where the work has moved than anything in the company's marketing.

The perception side matches the telemetry

The 2025 Stack Overflow survey found 45% of developers reported that debugging AI-generated code takes longer than writing it from scratch. 66% reported encountering AI solutions that are close but ultimately miss the mark. Trust in AI output has fallen from over 70% in 2023 and 2024 to 60% in 2025, even as adoption has continued to climb.

A randomised controlled trial published by METR in mid-2025 found a 39-point gap between developers' estimates of how AI affected their productivity and the productivity actually measured. The direction of the gap matters less than its size. Developers are unreliable narrators of their own speed. Anyone designing a review pattern on developer self-report is designing on a foundation that will not hold weight.

A footnote worth landing on. In the 2024 Stack Overflow Developer Survey, one developer was asked how AI would change their work over the coming year. They predicted they would switch from primarily writing code to primarily reviewing it. That prediction has now landed. The senior software developer in section one is what it looks like when it lands.

Why pattern mismatch produces fatigue

Fatigue is treated as a personal failure in most engineering teams, a sign the reviewer is overreaching or under-capacity. The research is clear that it is neither.

Code review has been documented as a cognitively demanding task for over a decade. Bacchelli and Bird's 2013 work on modern code review at Microsoft was the foundational study; Pascarella and colleagues built on it in 2018 with deeper observation of reviewer behaviour; Baum's 2019 doctoral work on review effort produced the most detailed model of where the cognitive cost actually sits. The shorthand of these papers, taken together, is that review is a decision process under uncertainty, and the cognitive load comes from holding many partial truths in mind simultaneously while moving toward a verdict.

The clearest current model of that decision process is the Code Review as Decision-Making framework, published in 2025 by Heander and colleagues. They describe review as proceeding through two phases. An orientation phase, in which the reviewer establishes the context and rationale for the change. An analytical phase, in which the reviewer assesses, plans, and decides. The orientation phase is where most reviewers underestimate cost, because it feels like reading rather than work. The analytical phase is where the reviewer's expertise actually pays off.

Both phases get harder when the change under review is bigger, less cohesive, or produced by someone other than the reviewer. The tangled-changes literature has demonstrated empirically that pull requests addressing multiple unrelated concerns increase reviewer cognitive load and lead to both missed defects and unfounded objections. The cohesion of the change matters as much as its size.

Decision fatigue is itself a recognised phenomenon in code review. A 2024 paper by Jetzen and colleagues at the University of Namur identified decision fatigue and confirmation bias as the two cognitive biases that most degrade review quality. The mechanism is well understood. What is new is the volume at which it now triggers.

A 2025 paper on engagement in LLM-assisted review made the most useful single observation. When the reviewer is engaging with code produced by an agent rather than by another human, the engagement that previously went into managing emotional and social load (how to phrase a critique, how to acknowledge effort, how not to wound) gets redirected toward managing cognitive load. The reviewer is no longer reading another person's intent. They are reconstructing intent from artefact.

Set those findings against the data in section two. PRs are bigger and more tangled than they were a year ago. The reviewer is increasingly engaging with code produced by an agent rather than by a colleague. The orientation phase, already underestimated, has grown longer. The analytical phase, already where expertise concentrates, has grown harder. The volume has increased.

"What looks like fatigue is the predictable outcome of applying old technique to new shape."

The senior software developer in section one was right to feel the weight of the work. He was reading more code, with less context, less cohesively scoped, in a phase that the literature has identified as under-recognised since 2013. That isn't a personal failing. That is the work changing under his feet.

The implication is uncomfortable for the easy responses. If fatigue is a pattern problem rather than a capacity problem, then the obvious fixes (review less, hire more reviewers) cannot solve it. The next section deals with each of those in turn, and a few others worth dismissing before the redesign begins.

The wrong fixes

When fatigue shows up, organisations reach for the obvious responses. Most of them don't work, and the data on why is now reasonably clear. Four worth dismissing before the redesign begins.

Review less

The most superficially appealing answer. If reviewers are tired, give them less to review. The DORA 2025 report's central finding is that AI doesn't fix a team, it amplifies what is already there. Reducing review frequency without redesigning anything else lets existing problems compound at the new pace. The 31% zero-review merge rate from section two is what reviewing less looks like in practice, and it isn't working.

Hire more reviewers

The next instinct. If the volume has doubled, double the review capacity. The Faros 2025 and 2026 telemetry shows what happens. PR review time grew 91% in 2025 and a further 200% in 2026 even as teams added review capacity. More reviewers, applied to the same broken pattern, spread the same fatigue thinner. The pattern has not been redesigned, so the cost has not been changed.

Trust the agent's own review

The most technologically optimistic answer. Have a second agent review the first agent's output and only escalate to humans on flagged issues. The most directly relevant evidence is a 2026 study from Kennesaw State on Code Review Agents. It examined 3,109 pull requests from the AIDev dataset, comparing human-only reviews against CRA-only reviews. CRA-only PRs achieved a 45.2% merge rate, against 68.4% for human-only. 60.2% of closed CRA-only PRs sat in the 0 to 30% signal-to-noise range. Twelve of the thirteen CRAs studied averaged below 60% signal. Automated review without human oversight produces low-signal feedback and higher abandonment.

Automate the review tooling

The most pragmatic answer, and the closest to working. Adopt LLM-based review assistants that triage PRs, surface issues, and reduce the human reviewer's load. The most useful empirical study here is the 2024 industrial case study by Lopez and colleagues on a Qodo-based code review tool deployed across 238 practitioners in ten projects. 73.8% of automated comments were resolved, suggesting the tool was useful. Average PR closure duration moved from 5 hours 52 minutes to 8 hours 20 minutes, suggesting it slowed things down. The tool changed what reviewers did without changing the underlying pattern of when and how review happened. It was an aid, not a redesign.

The shared logic

The four wrong fixes share a logic. They treat fatigue as a quantity problem. If only there were less to review, or more reviewers, or a smarter tool, the work would be sustainable. The data says otherwise. The work is not sustainable not because the volume is high but because the pattern is wrong.

The next two sections argue for redesign at two levels. The individual reviewer needs a different technique. The organisation needs a different structure. Both have to change for either to hold.

Redesign at the individual level

The line-by-line review was tuned for a different problem. Redesign at the individual level starts with what the reviewer is actually being asked to do now.

Line-by-line review assumed the reviewer was reading code written by a colleague whose intent could be inferred from the diff itself, in a change small enough to hold in working memory, with a turnaround fast enough that context did not need to be rebuilt. None of those assumptions hold reliably any more.

When most code is produced by an agent, line-by-line correctness is no longer the most valuable thing a human reviewer brings. The four most valuable questions a reviewer can ask, before they read the diff, are these.

THE FOUR QUESTIONS WORTH ASKING BEFORE THE DIFF

Did the agent stay in scope, or did it touch things it shouldn't have? Agents drift. The most common failure mode in agent-produced PRs is wandering into adjacent files, configurations, or tests. A scope check, before reading the code, catches the largest class of issues quickly.

Does this hang together as a coherent move, not just as code that compiles? Compilation is a low bar. Coherence is the question of whether the change, taken as a whole, makes sense as a deliberate move toward a stated goal. Agents produce internally consistent code that is incoherent at the change level more often than human authors do.

What did it have access to that it didn't need? Provenance turns this question from forensics into checklist. If the reviewer can see what context, files, and tools the agent was given, they can ask whether any of it was unnecessary. Unnecessary access is where security and reliability errors compound.

Where would I look first if this broke in production? The single most useful question for separating low-stakes from high-stakes review. If the reviewer can name a specific module, function, or external dependency that would be the suspect on a Monday morning, that's where the review's depth should concentrate.

Two of these (scope and coherence) map directly to the orientation phase of the CRDM model from section three. Two (access and production risk) map to the analytical phase. The point is not to automate review by checklist, but to redirect the reviewer's attention away from line-level reading and toward the questions where their experience produces the most value.

Three techniques that reduce per-PR cost

Stop reviewing chronologically. The default workflow is to take pull requests in the order they arrive, treating each as an independent unit. That maximises context-switching cost. Group PRs by intent. Build the context once, apply it across several reviews, then move to the next context. Applied to review, the cognitive-load research on context-switching argues for batching by intent rather than chronology.

Read the agent's claim before reading the code. Have the agent or the requesting human summarise what was done against what was asked, before the reviewer opens the diff. This shifts the reviewer's task from parsing intent to

validating a claim against an artefact. The Goncalves and colleagues registered report on guided checklist-based review (controlled experiment, 70 developers) found this kind of structured guidance reduced cognitive load relative to ad-hoc review.

Ration depth. Test changes, configuration changes, dependency upgrades, and core logic changes do not warrant the same review intensity. Treating them uniformly is most of the burnout. The reviewer's time is finite, and rationing depth is the only way to preserve quality on the changes that need it. A simple rule of thumb: every reviewer should be able to give one sentence of justification for the depth of review they applied to any given PR. If they can't, the depth was probably wrong.

PER-PR REVIEW SELF-ASSESSMENT

Before approving any PR, ask:

- Did I answer the four scope, coherence, access, and production questions before opening the diff?
- Did I batch this with related work, or did I review it cold?
- Did I read the claim before the code?
- Can I justify the depth I applied to anyone who asks?

If three or more answers are no, the review hasn't happened yet, regardless of what the merge button shows.

These three techniques are not new. They are the techniques that work, applied to a problem that didn't previously exist at this scale. The senior software developer in section one will recognise most of them. The argument is not that he didn't know them. The argument is that they need to become routine rather than exceptional, because the pattern has changed.

Redesign at the organisational level

The individual reviewer can do everything in section five and still drown if the organisation around them is structured for a different kind of work. Most organisations are.

Five organisational moves are necessary, and the data behind each is now strong enough to argue from.

Move review earlier

If the only review gate is at PR time, every architectural question that needed catching at design accumulates at PR time. The reviewer is then asked to make architectural judgements that should have been made by the team a week earlier, when the cost of reversal was small. Move review to design and task definition, with light structure (a one-page design note, a task plan with acceptance criteria, a brief sign-off step). The Faros 2026 finding that PR sizes are still growing is the symptom; reviewing earlier is the structural fix. The change-decomposition literature supports it: smaller, more cohesive PRs are dramatically easier to review and produce fewer false positives and false negatives.

Tighten the agent's blast radius

A 2026 study by Mendez and colleagues, on 22,953 pull requests from 1,719 developers using AI tools, gives the cleanest picture. Lower-experience users of AI tools submitted PRs with 2.15 times more commits and 1.47 times more files changed than higher-experience users. Their PRs received 4.52 times more review comments, had 31% lower acceptance rates, and stayed open 5.16 times longer. Scope is doing the damage. Tighter scoping (specific task definitions, smaller agent context windows, clearer stop conditions) reduces the volume of material the reviewer has to engage with without reducing the throughput of the team.

Split review by aspect

One reviewer for scope and intent, another for architecture, another for security. Three half-loaded reviewers will catch more than one fully-loaded reviewer, and each of them is doing a job they can sustain. The empirical literature on aspect-split review is sparser than for the other moves; the argument is partly from cognitive load theory and partly from observation. The teams that have moved to aspect-split review report it as the single most fatigue-reducing change. The teams that haven't tried it tend to dismiss it as bureaucratic. Both reactions are worth taking seriously.

Make provenance routine

Veracode's 2025 GenAI Code Security Report tested code produced by over 100 large language models across 80 coding tasks. 45% of AI-generated code introduced security vulnerabilities. The rate was stable across model generations, from GPT-4 through GPT-5 and the Claude 4 series. Without provenance (a record of what context the agent had, what it produced, what it was asked, what tools it used) the reviewer is doing forensic work on every PR. With provenance, review becomes triage. Provenance is the single highest-leverage organisational investment for reducing review fatigue, because it makes the difference between reviewing every line and knowing which lines warrant review.

Reform the metrics

PRs reviewed per week is the metric that creates fatigue faster than anything else. Substantive review is slower than the old kind, not faster. JetBrains' 2025 survey found that 66% of developers do not believe current metrics reflect their true contributions. The measurement system has not caught up with the work. The replacement metric is harder to define but easier to recognise: reviewers should be measured on incidents prevented, on architectural questions raised before merge, on the proportion of their reviews that result in non-trivial change requests, on the recovery time of the systems they own. None of these are perfect. All of them are better than counting PRs.

ORGANISATIONAL DIAGNOSTIC: WHERE IS YOUR FATIGUE COMING FROM?

Score each, honestly, on a five-point scale.

- PR sizes have shrunk in the last twelve months.
- Review happens at design and task definition, not just at PR.
- Reviewers can name the agent's intent before reading the diff.
- Provenance is queryable across all agent-produced PRs.
- Reviewer metrics are about substance, not throughput.

If your aggregate score is below 15, the redesign hasn't started. If between 15 and 20, it has begun but isn't sustained. Above 20, you are doing the work that section six describes.

The Faros 2026 data offers a single benchmark for the system as a whole. High-performing teams review PRs within 4 hours. If your average exceeds 24 hours, the system has already broken; you are just waiting to discover where. The teams that hit the 4-hour benchmark are not faster reviewers. They are organisations that have done the work in this section.

FIRST MOVES, BY INTERVENTION

For each intervention, the smallest change that produces visible benefit within four weeks.

Move review earlier. Pick one team. Add a one-page design review at the start of any feature spec longer than a sprint. No tooling, just a meeting and a written record.

Tighten blast radius. Adopt an agent context budget for one team. No PR may include code touching more than five files unless explicitly justified. Track exceptions.

Split review by aspect. Pilot with two reviewers per PR for two weeks: one for scope and intent, one for architecture. Compare review time and incident rate against the unsplit baseline.

Make provenance routine. Require every agent-produced PR to include three lines: what the agent was asked, what it had access to, what it produced. No tooling needed for the pilot.

Reform metrics. Stop reporting PRs-reviewed-per-week as a positive metric. Replace with one of: incidents prevented, change requests raised, recovery time. Pick one.

The redesign at this level is the harder of the two. The individual reviewer can change technique tomorrow. The organisation has to change pattern, which means changing what it measures, what it assumes, and what it asks of people. None of that happens in a single sprint. All of it is necessary.

Signals it is working, signals it isn't

The data underneath review fatigue is slow to surface, often masked by short-term productivity gains, and often misinterpreted as a problem of individual capacity. The signals worth watching are below.

FIGURE 1 / WHAT THE REDESIGN LOOKS LIKE FROM THE OUTSIDE

SIGNALS THE REDESIGN IS HOLDING

PR sizes are smaller than they were twelve months ago, even as code volume has grown.

Review happens at design and task definition, not just at PR. Architectural questions are raised when they are still cheap to answer.

Reviewers can name the agent's intent before they open the diff. The PR description carries the claim; the diff is the evidence.

Provenance is queryable. Any reviewer can see what the agent had access to, what it was asked, and what it produced.

Reviewer metrics are about substance. Incidents prevented, change requests raised, system recovery time. PRs-reviewed-per-week is no longer reported as a positive signal.

Senior software developers describe review as harder than it was, and worth doing.

SIGNALS THE REDESIGN HASN'T STARTED

Zero-review merges are creeping up. The team's response is to investigate individual reviewers rather than the pattern.

PR sizes have grown without explanation. The team treats this as a feature of how agents now work rather than a problem to fix.

Review is queued only at PR time. Design conversations happen, but their outcomes are not connected to the review gate.

Reviewer metrics are about throughput. PRs-per-week is reported as a positive signal in retrospectives or performance reviews.

The first organisational response to fatigue is to add more reviewers, with no change to scope, structure, or technique.

Senior software developers describe review as exhausting, and consider rotating off it. Some have already left.

The mint side is achievable. None of the moves in section six requires a budget line, a new tool, or executive approval. They require deciding that the pattern of review needs to change, and following through on that decision for long enough to show the difference.

The rose side is the default. Most organisations live somewhere on it without having decided to. The cost compounds slowly. The data in section two is what compounding looks like at three years' distance.

What is owed to the reviewer

The two conversations in section one were short. The argument that has followed them has been long. The reason is below.

The data took some time to arrive, the academic work on review has been hiding in plain sight for over a decade, and the redesign required is at two levels rather than one. The senior software developer in section one was right to feel the weight of the shift. He is not failing at his job. The job has changed shape, and the technique he was given was tuned for a different problem.

"Review fatigue is a signal that the pattern needs redesigning, not that the reviewer needs to do less."

The reviewer is doing their best with the wrong instrument. The redesign is what is owed to them.

The framework this paper sits inside, the Cognitive Delivery Framework, takes the position that hybrid human-agent delivery is not an extension of the existing software development lifecycle but a redesign of it. Paper 3 of the series argued for that redesign at the level of code review specifically. This paper has been the practitioner-facing companion to that argument: what the redesign looks like, why it is necessary, how to begin, and what to watch for to know it is working.

If you are reading this and you are the senior software developer in section one, the answer to the question you asked in the corridor that day is that the work is sustainable, but only if the work changes. That is the conversation worth having next.

About this paper

This paper is the practitioner-facing companion to Paper 3 of the Cognitive Delivery Framework series. It is published as part of a six-paper sequence covering the developer progression from Dev 1.0 through Dev 4.0, the redesign of code review for agent participation, the governance principles underpinning hybrid delivery, the question of accountability when agents act, the adoption tiers organisations move through as agent participation matures, and the Cognitive Delivery Index as the measurement instrument. The book-length treatment is forthcoming, with publication targeted by the end of 2026.

Sources cited in this paper

Faros AI, AI Engineering Reports 2025 and 2026. DORA, State of AI-assisted Software Development 2025. Stack Overflow, Annual Developer Survey 2024 and 2025. JetBrains, State of Developer Ecosystem 2025. METR, Measuring the Impact of Early-2025 AI on Experienced Open-Source Developer Productivity (and February 2026 update). Veracode, 2025 GenAI Code Security Report. Bacchelli & Bird (2013), Pascarella et al. (2018), Baum (2019) on code review cognitive load. Heander et al. (2025) on the Code Review as Decision-Making model. Jetzen et al. (2024) on cognitive biases in code review. Mendez et al. (2026) on novice vibe coder review overhead. Goncalves et al. on guided checklist-based review. Lopez et al. (2024) on automated code review in industrial use. Kennesaw State (2026) on Code Review Agents. The engagement-in-LLM-assisted-review paper (December 2025).

All claims in this paper are grounded in the sources above. Where the empirical picture is contested or evolving, this is named in the prose rather than smoothed over.



Jason Cruz Falzon

Creator of the Cognitive Delivery Framework

jason@cognitivedelivery.co.uk

cognitivedelivery.co.uk / cognitivedelivery.substack.com