AI Didn’t Break Delivery — It Moved the Bottleneck to Review & Integration

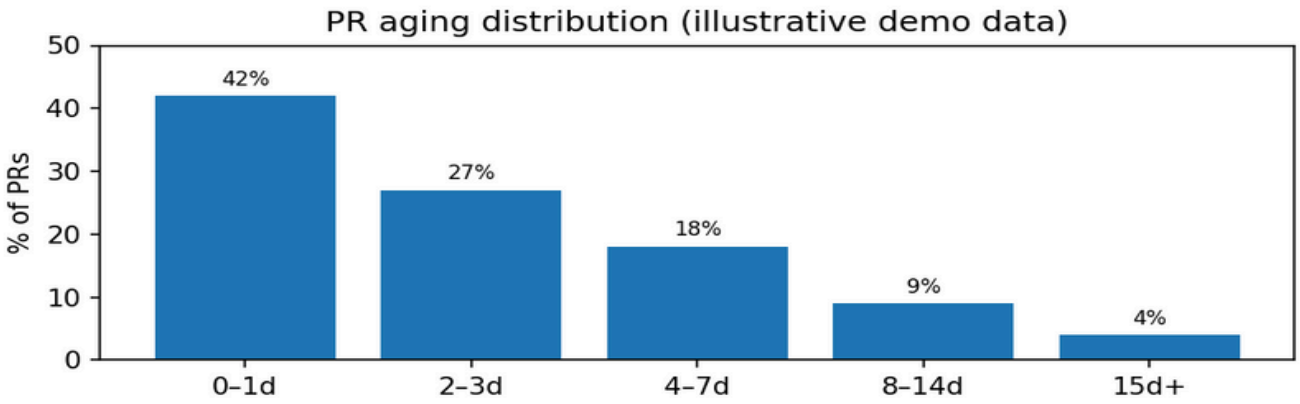
Page 1 — The finding

As code output accelerates, delivery predictability often gets worse — because the constraint moves from **writing code** to **review, integration, and release**. Teams keep measuring “velocity” in the first box while the schedule is governed by the slowest box.

Rule of thumb: If PRs opened exceed PRs merged for 2+ weeks, your review queue is silently turning into schedule drift.

The four metrics that reveal the bottleneck shift

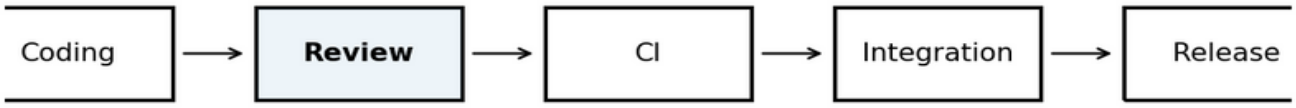
Metric	What it measures	Why it matters
Review queue length	How many PRs are waiting for review	Queues are where dates go to die
Time to first review (TTFR)	Hours/days until a PR gets a first look	TTFR increases batch size + risk
PR aging (P50/P85)	How long PRs sit before merge	Percentiles expose tail risk that drives missed dates
Rework rate	PRs reopened / extra review cycles	High rework means integration cost is hidden in “progres



Aging is the smoke. The fire is queueing: review capacity can’t keep up with code output.

Page 2 — Why it happens (constraints, not effort)

When work moves faster into the system than it can exit,you create a line. The longer the line, the longer everything waits — even if everyone is working hard. AI typically speeds up the **input**; delivery is governed by the slowest **constraint**.



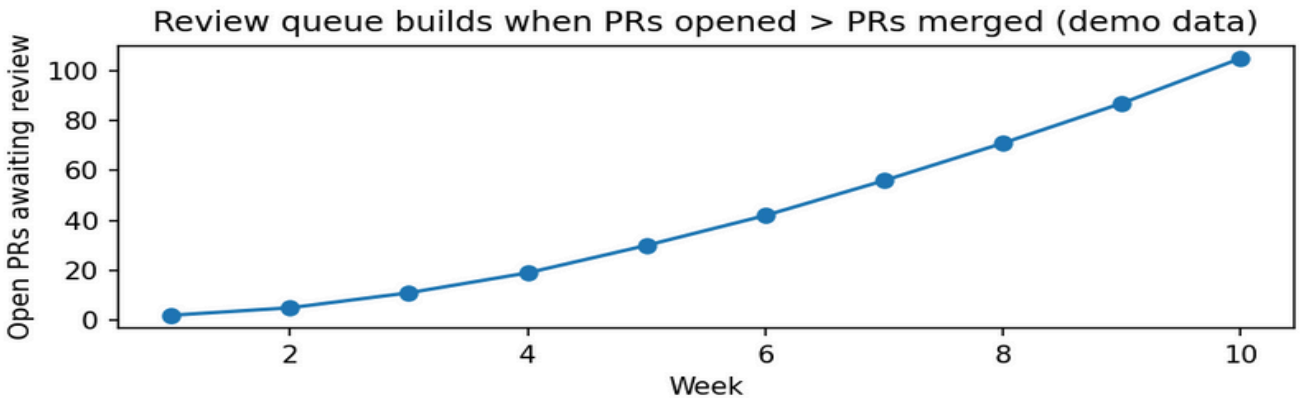
AI accelerates the first box; delivery is governed by the slowest box.

Three failure modes that make review the bottleneck

Review starvation
Symptoms: long TTFR; PRs wait days for first review.
Fix: rotate a “review captain”; protect reviewer time.

PR batching
Symptoms: big PRs; high rework; long tail aging.
Fix: PR size limits; split by feature flags; merge smaller slices.

Governance mismatch Symptoms: every change requires heavyweight review.
Fix: tiered policy (low risk auto merge, high risk strict).



If PRs opened consistently exceed PRs merged, queue builds first — and schedule drift shows up later, when it’s expensive.

Page 3 — Governance changes that restore predictability

The goal isn't to"go faster". It's to **reduce queueing** and **stabilize flow** through the constraint.

Three hard mandates that work in practice

- Set a PR size ceiling (or enforce slicing). Smaller PRs reduce tail risk and rework.
- Tier review requirements by risk (config/docs/low-blast changes are lightweight; core logic is strict).
- Create protected review capacity (rotation + focus blocks). Treat review as first-class work, not "spare time."

This week's 7-day intervention plan

This week's cadence (simple, enforceable):
1) Day 1: instrument TTFR + aging buckets; pick a red threshold (e.g., % of PRs > 4 days).
2) Day 2: introduce PR size guidance; split one large PR as the example.
3) Day 3: start a review rotation (review captain) and block reviewer calendar time.
4) Day 4: define tiered review rules (what can be fast-tracked vs strict).
5) Day 5: run a "queue burn-down" (swarm on oldest PRs first).
6) Day 6: re-measure TTFR/aging; adjust thresholds and policy.
7) Day 7: lock the cadence (weekly review health check + queue trend).

Motionode's solution

Doing this analysis across code hosts, CI, and spreadsheets usually takes days (and strong intuition). Motionode surfaces review bottlenecks automatically — TTFR, aging, rework, and their predicted impact on ship-date confidence — and lets teams simulate governance changes (PR size caps, review capacity, tiered policies) to see which move improves predictability with the lowest disruption.