# Can a Review-Driven Claude Code Workflow Close the Model Gap?

*What we learned from four live MindStone-Agent experiments using Claude Fable 5 and Claude Opus 4.8*

***Authors:*** *Clint Bodungen & the MindStone agent team*
***Date:*** *2026-07-03*
***Domain:*** *Claude Code / agent orchestration*
***Status:*** *Published*

Claude Code power users face a practical routing question every day: **when do you need the highest-tier model, and when can a disciplined workflow get the same shipped result from Claude Opus 4.8?**

MindStone-Agent gave us a useful place to test that question. During an active sprint, the team ran four controlled experiments around real software work. Three experiments used existing MindStone-Agent tickets; a fourth greenfield experiment asked three arms to build the same browser music app from scratch. The comparison was between **Claude Fable 5** and **Claude Opus 4.8**, with independent review and judging from the MindStone agent team.

The short answer: **Claude Fable 5 still showed advantages, but MindStone/LCA review workflow moved Claude Opus 4.8 into the Claude Fable 5 quality band.** On codebase-grounded architecture, both models converged. On detail execution and shipping discipline, Fable had the edge. Once the missing details were named — or once the review loop was allowed to operate during the build — Opus shipped Fable-class results.

## The result in one table

| Question | What happened | What it means for Claude users |
|---|---|---|
| Do the models choose different architectures? | No. Four fresh agents, with no shared memory, landed the same scheduler architecture. | On a rich existing codebase, constraints drive the big design. Paying more for the first architecture pass may not buy much. |
| Is Claude Fable 5 better? | Yes, slightly. In the clean model-vs-model run, Fable averaged 49.5/50 and Opus averaged 47/50. | The premium showed up in details: idempotence, lifecycle limits, drift control, DST language, and integration seams. |
| Can review close the codebase-grounded gap? | Yes, when the review names the problems. Opus revised its earlier draft and both judges scored it 25/25. | Independent, repo-grounded review can recover the detail layer that the first pass missed. |
| Does the pattern extend to greenfield app work? | Directionally, yes. Bare Opus produced a good LoopSmith Studio app but trailed Fable on shipping-quality/discipline dimensions; harnessed Opus landed in the same top band as | The harness did not make models irrelevant; it supplied review, verification, hygiene, and defect discovery that narrowed the shipped-result gap. |

| | | |
|---|---|---|
| | Fable. | |
| What remains unmeasured? | What happens when review fails to notice something important, or on larger multi-session/brownfield tasks. | That is still where model choice and stronger review loops may matter most. |

## The practical routing rule

Use **Claude Opus 4.8** or another strong model for the first architecture pass when the repo has strong constraints and you have a serious review loop.

Spend **Claude Fable 5**, or spend extra independent review, on the detail pass: edge cases, idempotence, lifecycle limits, time semantics, approval boundaries, integration seams, and final shipping hygiene.

That rule is more useful than “always use the best model” or “the harness makes models interchangeable.” The real lesson is layer-specific:

- architecture converged across models;
- detail execution favored Claude Fable 5;
- review recovered the missing details when it named them;
- on a greenfield build, the harness supplied the discipline layer that moved Opus into the Fable quality band.

## What we tested

The work happened inside MindStone-Agent, a TypeScript framework for persistent agents, memory, channels, skills, knowledge bases, and gateway surfaces. The tickets were real production design tasks, not toy prompts.

The team ran four experiments:

1. **Experiment D — same ticket, two models.** Cairn, the developer agent, wrote a pack-registry design once on Claude Opus 4.8 and once on Claude Fable 5. The documents were anonymized and judged independently.
2. **Experiment F — clean model comparison.** Four fresh agents, two per model, wrote scheduler designs at the same time. They had no shared memory, no orchestrator help, and the same frozen instructions.
3. **Experiment E — repair pass.** A fresh Claude Opus 4.8 agent received the lower-scoring Opus design from Experiment D plus the repo-sighted review comments, then revised the design once.
4. **Experiment G — greenfield app build.** Bare Claude Fable 5, bare Claude Opus 4.8, and harnessed Claude Opus 4.8 each built the same browser music app, LoopSmith Studio, from the same frozen prompt.

The judges were deliberately split:

- **Slate** judged with repo access, checking claims against the actual codebase.
- **Hearth** judged from the documents only, catching whether the design stood on its own.

For D/E/F, both judges used the same five-point design rubric: constraint correctness, option coverage, risk identification, decision quality, and spec completeness. For G, Clint, Mira, and Slate judged runnable apps using a 100-point scorecard covering install/run reliability, core music flow, audio/timing, instruments, generation, persistence, education/UX, polish, code quality, and claim-boundary discipline.

## Experiment D: Fable won, but the result was not clean

The first probe used the pack-registry ticket.

Results:

| Design | Model | Slate | Hearth |
|---|---|---|---|
| A | Claude Opus 4.8 | 22/25 | 24/25 |
| B | Claude Fable 5 | 25/25 | 25/25 |

Claude Fable 5 won. The difference was real enough for both judges to prefer it, and the repo-sighted judge could point to concrete issues in the Opus draft.

But there was a catch: Fable went second. The second run was forbidden from reading the first document, and the observed context-injection channels were audited, but the broader setup still had durable memory and same-agent continuity around it. The judges correctly called that an order confound. A narrow second-mover win cannot cleanly prove a model gap by itself.

That critique is why Experiment F was run.

## Experiment F: architecture converged, details separated the models

Experiment F removed the order problem.

Four fresh agents ran concurrently on the scheduler ticket:

- two Claude Fable 5 arms;
- two Claude Opus 4.8 arms;
- same prompt, same repo, same ticket;
- no shared memory;
- no orchestration help;
- served model IDs verified from the transcripts.

Combined judge scores:

| Arm | Model | Slate | Hearth | Combined |
|---|---|---|---|---|
| C | Claude Fable 5 | 25 | 25 | **50** |
| D | Claude Fable 5 | 24 | 25 | **49** |
| B | Claude Opus 4.8 | 24 | 24 | 48 |
| A | Claude Opus 4.8 | 23 | 23 | 46 |

This is the cleanest signal in the set. Both Fable runs beat both Opus runs, but the margin was small: **49.5 vs 47.0**, about five percent of the available points.

The more important result was architectural convergence. All four designs independently chose the same core shape: scheduler in the daemon, jobs defined in config and disabled by default, pure schedule math in core, existing approval gates reused, and no new approval authority.

That matters because nobody copied anybody. With shared memory removed, the convergence points back to the codebase. On this kind of engineering task, the repo’s constraints decided the architecture.

Where Fable pulled ahead was the finish work: sharper handling of idempotence, daily run limits, config-change events, daylight-saving-time language, and integration boundaries. Same blueprint, cleaner edge treatment.

## Experiment E: review let Opus catch up

The last question was practical: if review catches the missing details, can Opus repair them?

A fresh Claude Opus 4.8 agent received:

- the original Opus design from Experiment D;
- Slate's repo-sighted critique, verbatim;
- a pinned checkout from before the later synthesis existed.

It got one bounded revision pass.

Results:

| Judge | Original Opus design | Revised Opus design | Delta |
|---|---|---|---|
| Slate | 22/25 | **25/25** | +3 |
| Hearth | 24/25 | **25/25** | +1 |

Both judges scored the revised design 25/25. The repair was not cosmetic; the issues were actually fixed.

The important nuance is that the review named the missing insight. Opus did not have to independently invent the best Fable move. Once the reviewer pointed at it, Opus implemented it correctly.

That is the core workflow result: **review can carry an idea across the model boundary when the reviewer sees it and names it.**

## Experiment G: greenfield app, same prompt, three arms

The earlier experiments answered questions about existing-codebase work. Clint then called out the obvious gap: to test whether the harness can produce Fable 5-class shipped results on a blank page, the team needed a greenfield build.

The frozen prompt asked each arm to build **LoopSmith Studio**, a browser-based music creation app. It needed a visual loop-making interface, percussion, at least two pitched instruments, tempo and transport controls, editable notes, clean looping, local save/load, JSON import/export, polished instructions, and a local algorithmic **Inspire Me** generator that creates editable musical ideas without external AI or cloud services.

Three arms ran:

| Codename | Setup | Judge summary |
|---|---|---|
| **Maple** | Bare Claude Fable 5 in stock Claude Code | Excellent app; broad product surface; strong zero-dependency/local-first story. |
| **Cedar** | Bare Claude Opus 4.8 in stock Claude Code | Good, complete app; trailed on shipping-quality/discipline signals such as dependency audit and validation depth. |

| | | |
|---|---|---|
| **Birch** | Claude Opus 4.8 with MindStone/LCA review workflow and Hearth QA | Top-band app; review loop found and fixed a real phase-dependent stop/playhead defect before freeze. |

The scores were not perfectly aligned because the judges weighted dimensions differently:

| Judge | Birch | Maple | Cedar |
|---|---|---|---|
| Mira | 100 | 99 | 99 |
| Slate | 98 | 97 | 85 |
| Clint | Birch and Maple strongest; Birch edged Maple on education and music quality | Top band | Less complete |

The agreed read was not “models do not matter.” It was narrower and more useful:

> On this greenfield app build, bare Opus 4.8 matched Fable 5 on many visible product features but trailed on shipping-quality and discipline. MindStone/LCA review workflow supplied exactly that missing discipline layer — review, verification, dependency/security checks, and defect discovery — and moved Opus into the Fable 5 quality band.

A fairness note: one Cedar hygiene signal, a stray `.claude/launch.json`, was later traced to judging-package assembly rather than Cedar’s build. So the Cedar discipline gap should lean on npm vulnerabilities and validation depth, not that packaging artifact.

One subtle point matters. Birch, the harnessed arm, originally had a phase-dependent stop/playhead bug introduced partly by its heavier Tone.js architecture. The review loop found it; Hearth reproduced it; Cairn fixed it before freeze. The bare arms did not have that bug. So the result is not “the harness magically makes every technical choice better.” The honest mechanism is that the harness increases shipped confidence: it catches the edge-case defects that otherwise reach the artifact.

Also, Birch did **not** use every capability available in the broader MindStone development workflow. It used MS4CC-style memory/checkpoints/ledger discipline, Hearth’s live QA/review loop, and verification/ticket-fidelity habits. It did **not** use a formal PRD, a formal design/implementation plan, role adoption, frontend-design MCP assistance, subagent delegation, or a dedicated security-scanner pass. Cairn later clarified that this was not a clean deliberate protocol choice: the review and continuity habits were automatic, while several broader workflow tools were simply not invoked. That makes G a conservative datapoint for the harness claim. Whether the fuller workflow would widen the margin is a plausible follow-up, not a measured result here.

## What this says about model choice

This case study does **not** say all models are interchangeable. It says the model gap has layers.

### Layer 1: Architecture

On a real codebase with strong constraints, both models converged on the same architecture. The repo did much of the steering.

### Layer 2: Detail execution

Claude Fable 5 was better here. It surfaced and specified more of the edge-case layer on the first pass.

### Layer 3: Review and repair

Independent review recovered that layer when it named the defects. The revised Claude Opus 4.8 design reached parity with the best Fable result.

### Layer 4: Shipping discipline

Experiment G added the greenfield case. There, the visible feature gap between the bare arms was small, but the shipping-quality gap mattered: dependency/security hygiene, validation depth, and edge-case testing. That is exactly the layer a harness can manufacture. The harnessed Opus arm did not prove Opus has equal raw capability to Fable; it showed that Opus plus MindStone/LCA review workflow can ship in the same Fable 5 quality band on this task.

For a builder, the routing implication is straightforward: do not spend the premium model where the codebase already forces the answer. Spend it where missed details are expensive, or use a serious review loop to find and repair those details before shipment.

## What this does not prove

The honest boundary is important.

These experiments do not measure what happens when review misses something important. Every recovered point in Experiment E came from a named review finding, and Experiment G's harnessed arm benefited from an active QA partner. If the reviewer does not see the issue, the stronger model's first-pass detail advantage may still matter.

They also do not claim universal results across all work. D/E/F were design tasks against a rich existing codebase. G was one greenfield music app, one run per arm, with mixed-blindness judging and a likely ceiling effect because all three apps were good. G's bare-Opus and harnessed-Opus arms were separate single builds by different agent instances, with different tooling and architecture choices, so the harness effect is not perfectly isolated from build-to-build variation. The airtight follow-up would run the same bare-Opus artifact back through the harness, or repeat each arm at k>1. A larger physics/simulation app, a brownfield extension, or a multi-session task could expose different gaps.

Finally, the judges were part of the MindStone agent ecosystem. They were separated, one was repo-sighted and one was repo-blind, and their disagreements were informative — but this is still an internal case study, not an independent lab result.

## Why this matters for MindStone

MindStone's Layered Continuity Architecture is built around a simple premise: long-running agents need more than a good model. They need durable memory, source-aware recall, checkpointing, clear handoffs, review loops, and a way to recover after context resets.

These experiments sharpen that claim. The harness does not replace model capability. It changes where model capability matters.

A good harness can:

- keep work portable across sessions and models;

- make review findings durable;
- force hard work into inspectable artifacts;
- let models below the highest available tier handle architecture when constraints are strong;
- supply the shipping discipline layer: verification, hygiene, clean-checkout checks, and edge-case review;
- reserve premium model spend for detail execution and ambiguous calls.

That is not magic. It is ordinary engineering discipline applied to agent work.

## Read the detailed record

This page is the reader-friendly case study. The full methodology record preserves the pre-registration detail, per-criterion scores, deviations, and interpretation rules:

- Download the plain-language case study PDF
- Read the detailed methodology record
- Download the detailed methodology PDF