How a Maker-Checker, Test-Driven Process and a Single-Agent Ticket Run Differ: A Preregistered, Paired, Blinded Comparison on Ten Replayed Tourbot Issues

Jacob Stephens

Educational Travel Adventures · stephens.page

Experiments 30 October – 1 November 2026 · Public edition 17 November 2026 · ← Narrative essay (30-pair synthesis) · Follow-up second study (17 tasks, Codex orchestrator) →

In brief

Educational Travel Adventures (ETA) compared an independently proposed multi-agent, maker-checker test-driven workflow against thirdshift's Built-in Ticket run, the way ETA currently runs AI coding agents on Tourbot. In the proposed workflow, separate agents plan the tests, write the tests, write the code and check each other's work across model families. In the Built-in Ticket run, one agent does the whole job. Both processes were given the same ten real Tourbot issues to replay in isolation. Neither could see the other's work, and the reviewers did not know which process wrote which change.

Table A. The two processes on the same ten fixes. Money is the price of the agents' work at the providers' published rates; for cost and time, the middle value of the ten is shown.

	Current process (Baseline)	**Proposed maker-checker process (Candidate)**
Fixes it won	3 of 10 (2 because the other delivered nothing)	2 of 10
Changes the author would merge as they stood, judging blind	7 of 8	1 of 8
Fixes judged even	5 of 10	5 of 10
Fixes where it delivered no change at all	0 of 10	2 of 10
Cost per fix (middle of ten)	$1.35	$14.06
Time per fix (middle of ten)	11 minutes	62 minutes
Fixes with a confirmed defect or no change	2 of 10	3 of 10

- **It cost more and took longer in every one of the ten fixes**, including the two where it did better. With that consistency, the cost and time difference is clear enough to act on.
- **The author would merge almost all of the current process's changes as they stood, and almost none of the proposed workflow's**: 7 of 8 against 1 of 8, judging blind. This is the measure closest to what ETA wants from any process, work that needs no review rounds. It is one person's judgement, so it was reported beside the decision rule rather than in it, and it points the same way.
- **Its separately written tests caught real problems** that the current process missed in 2 fixes. But it wrote about six times as much test code, and it missed a problem that the current process handled.
- **On quality the evidence is less certain.** Ten fixes is a small sample. The result shows that the proposed workflow was not clearly better; it does not show that it is worse.
- **Decision:** ETA keeps its current process. Two of the Candidate's ideas are worth trying on their own: a separately written set of tests, and writing down any assumption the agent makes when an issue is unclear.

Key questions

1. **What was compared?** Two ways of organising coding agents, run on the same ten fixes in parallel on identical machines (Section 2.3, Figure 1).
2. **Which made better changes?** Neither clearly. Blind model reviewers preferred the proposed workflow in 2 fixes and the current process in 1, and split or judged even in 5. In the other 2 the proposed workflow delivered nothing (Section 4.1). The author, judging blind, would merge 7 of 8 of the current process's changes as they stood and 1 of 8 of the proposed workflow's (Section 4.7).
3. **What did it cost?** The proposed workflow cost a median $14.06 per fix against $1.35, and took 62 minutes against 11 (Section 4.7, Figure 3).
4. **Why was it slower?** Every agent started fresh and re-read the code, and checking rounds repeated; its first code edit came after 24 minutes, against about 1 (Section 4.2).
5. **What did it produce?** Similar fixes with six times the test code, often tied to the exact implementation (Section 4.3).
6. **What did each catch that the other missed?** Its tests exposed defects in two of the current process's changes; the current process handled an encoding issue that it missed (Section 4.4).
7. **How sure are we?** Sure about cost and time, which differed in every fix; unsure about small differences in quality, with ten fixes and one run each (Section 6).
8. **What should ETA do?** Keep the current process, and test two of the Candidate's ideas on their own (Section 5.5).

Abstract

Background. Coding agents can be organised in very different ways. ETA currently uses `thirdshift`'s Built-in Ticket run, in which one agent session implements an issue, has it reviewed once and opens a pull request. An independently proposed maker-checker test-driven workflow instead splits work across separate agents that plan tests, write tests, implement and verify, where no agent checks its own work and two model families review every stage. The two represent opposite bets: one context working quickly, against independent roles that check each other.

Objective. To describe how the two processes differ on the same issues: in how they work, what they produce, how they fail and what their reviewers value. To decide, by a rule fixed in advance, whether the proposed workflow (the Candidate) should replace the Built-in Ticket run (the Baseline) on Tourbot, ETA's production PHP application.

Methods. Pre-registered, paired, blinded comparison. Ten closed Tourbot issues, chosen by a written rule, were replayed from the commit before their shipped fix. Each process implemented each issue once, in parallel, on identical isolated machines. A Claude grader and a Codex grader compared each pair of formatted, randomly labelled pull requests in both presentation orders, grounded in the issue, continuous-integration (CI) results, cross-run tests and mutation testing. The author judged every pull request blind. After the decision, the differences were analysed from the agent traces of all twenty runs, the pull requests, the cross-tests, and the graders' and author's written reasons, and three pairs were examined in detail. These analyses are exploratory.

Results.

How they worked. The Candidate started a median 4.9 times as many agent sessions and made 2.8 times as many tool calls per run. Most of the extra calls re-read code and re-ran tests in fresh contexts. Its first code edit came after a median 24 minutes, against 1.3 for the Baseline.

What they produced. The Candidate's pull requests added 1.5 times as much code and 6.0 times as much test code (14.7 test lines per code line, against 3.7).

How they failed. Its independently written tests exposed defects in two Baseline pull requests, and its code tended to fail safe. But it stopped twice at a planning question that the Baseline settled by assumption, and it missed an encoding constraint that

the Baseline handled.

What reviewers valued. The model graders credited the Candidate's coverage. The author, judging blind, noted test bloat in 7 of its 8 pull requests and would merge 1 of 8 as is, against 7 of 8 Baseline pull requests.

Decision. The Candidate won 2 pairs, lost 3 and tied 5: a win rate of 20% (exact 95% CI 2.5–56%). Its median per-pair cost was $14.06 against $1.35 at the providers' published per-token (API) prices, a median ratio of 10.4. Its median time was 62 minutes against 11, a median ratio of 6.4. It had 3 correctness failures to the Baseline's 2.

Limitations. Each process ran once on each of ten small fixes to one PHP application. The Candidate's spec-writing stage was not tested. The analyses of the differences were designed after the decision.

Conclusion. The Candidate buys independent, adversarial checking at about ten times the cost. On these small fixes, the checking produced more tests rather than better pull requests: judging blind, the author would merge 1 of its 8 pull requests as is, against 7 of 8 for the Baseline. The pre-registered rule rejects the Candidate on each of its three criteria, and ETA keeps the Built-in Ticket run. Two of the Candidate's ideas are worth testing as additions to the Baseline: an independently written test pass, and a stated-assumption rule for unclear requirements.

Keywords: coding agents, multi-agent systems, maker-checker, test-driven development, cross-family review, agent traces, LLM-as-a-judge, paired comparison, preregistration, replay evaluation.

---

1 Introduction

ETA develops Tourbot, its production booking and operations system, largely through AI coding agents. Most recent Tourbot pull requests come from `thirdshift`'s Built-in Ticket run. One Codex `gpt-6.1-sol` session takes a `ready-for-agent` issue, implements it, runs a built-in code review and a security review, and opens a pull request. In the design research, 102 of 104 Tourbot pull requests carrying `thirdshift`'s "Built with" line used this configuration.

An independently authored workflow was proposed for the company's repositories as a safety-critical engineering hypothesis adapted from Cleanroom software engineering [14]: if the person who writes the code should not be the person who writes the tests so a shared misreading of a ticket cannot pass both code and suite, shouldn't coding agents be separated the same way? The workflow combines a maker-checker TDD stage and a cross-family review stage resting on three principles:

- no agent verifies its own work;
- all work is verified;
- tests are written, and fail, before code is written.

A test planner, test writer, implementer and refactorer each work as separate agents. Each is checked by a verifier from another provider, and every review stage uses at least two model families for up to five rounds.

The two processes are different bets about where agent errors come from. The Ticket run assumes that one capable session keeps the most context and works best when left to finish. The proposed workflow assumes that an agent cannot see its own mistakes, so independent roles must plan, test and check each step. The published evidence points both ways:

- Writing tests independently of the implementation improved fault detection from 14% to 25% [1].
- Multi-agent pipelines did slightly worse than a single agent on SWE-bench Verified, at several times the tokens [2].
- A second model family's review lowered a strong writer's pass rate from 91.4% to 82.8% [3].

We found no study comparing such a pipeline with a single-session agent on the same issues in a production codebase, or describing how their behaviour and output differ there.

The comparison was commissioned to decide one question: should ETA adopt the proposed workflow? We fixed four research questions, RQ1 to RQ4, before the first counted run:

- **RQ1 (quality).** Do blind model graders prefer the Candidate's pull requests? *Hypothesis:* the Candidate wins at least 8 of 10 pairs.
- **RQ2 (cost and time).** What does the Candidate cost relative to the Baseline? *Hypothesis:* its median per-pair cost and time ratios are at most 2.
- **RQ3 (correctness).** Does the Candidate have fewer correctness failures? *Hypothesis:* it has no more than the Baseline.
- **RQ4 (human attention).** How often would a reviewer merge each process's pull request as is? This is the outcome closest to ETA's goal of work that needs no human review rounds; the design says that if neither process gets there, "that matters more than which one wins". It is reported, not part of the decision, so that one blind judge, the author, would not cast the deciding vote.

The Candidate is adopted only if all three hypotheses hold. That answer alone says little about why. Once the decision was made, we asked four further exploratory questions, EQ1 to EQ4, of the same evidence. They were not pre-registered:

- **EQ1 (process).** How do the two processes differ in how they work: sessions, tool calls, and where the time goes?
- **EQ2 (output).** How do their pull requests differ in size, shape and testing style?
- **EQ3 (failure).** How does each process fail, and what does each catch that the other misses?
- **EQ4 (judgement).** What do the model graders and the human reviewer value in each?

Section 2 describes the two processes and the design. Section 3 describes how the comparison was carried out. Section 4 reports the differences (EQ1–EQ4), three pairs in detail, and then the pre-registered decision (RQ1–RQ4). Section 5 interprets them, and Section 6 discusses the threats to their validity. Appendix A summarizes the evaluation and blinding protocol, and Appendix B defines key terms.

2 Methods

2.1 Registration

The design was fixed in a structured question-and-answer design session with the author on 29 October 2026 and frozen in a private study repository before any counted run. Two amendments were made before the first counted run and every subsequent operational adjustment is listed in Section 3.4.

2.2 Design

The study is a paired comparison: each issue (the unit) is implemented once by each process, and the two pull requests are compared with each other. Pairing removes variation between issues from the comparison. Ties count as non-wins. Nothing was merged; the "merge as is" judgement is hypothetical.

2.3 The two processes

Table 1. The two processes as run.

	Baseline	Candidate
Process	`ticket-run`, `thirdshift`'s Built-in Ticket run, as shipped, with Tourbot's repository conventions	Custom multi-agent workflow: maker-checker TDD and cross-family review stages, then `thirdshift`'s `delivery` Step
Orchestrator	Codex `gpt-6.1-sol`, effort `high`	Claude Code (`claude-opus-5-5`), effort `medium`
Other agents	Codex's own reviewer sub-agents	Claude Agent-tool roles and `codex exec` roles as assigned by the workflow; Claude and Codex reviewers in each cross-family review stage
Reviews	Built-in code review; security review	Plan review; test-plan, RED, GREEN and refactor verifiers; cross-provider code review rounds; review verifier. No security review
Self-merge	No	No
Tool version	`thirdshift` 0.18.6	`thirdshift` 0.18.6

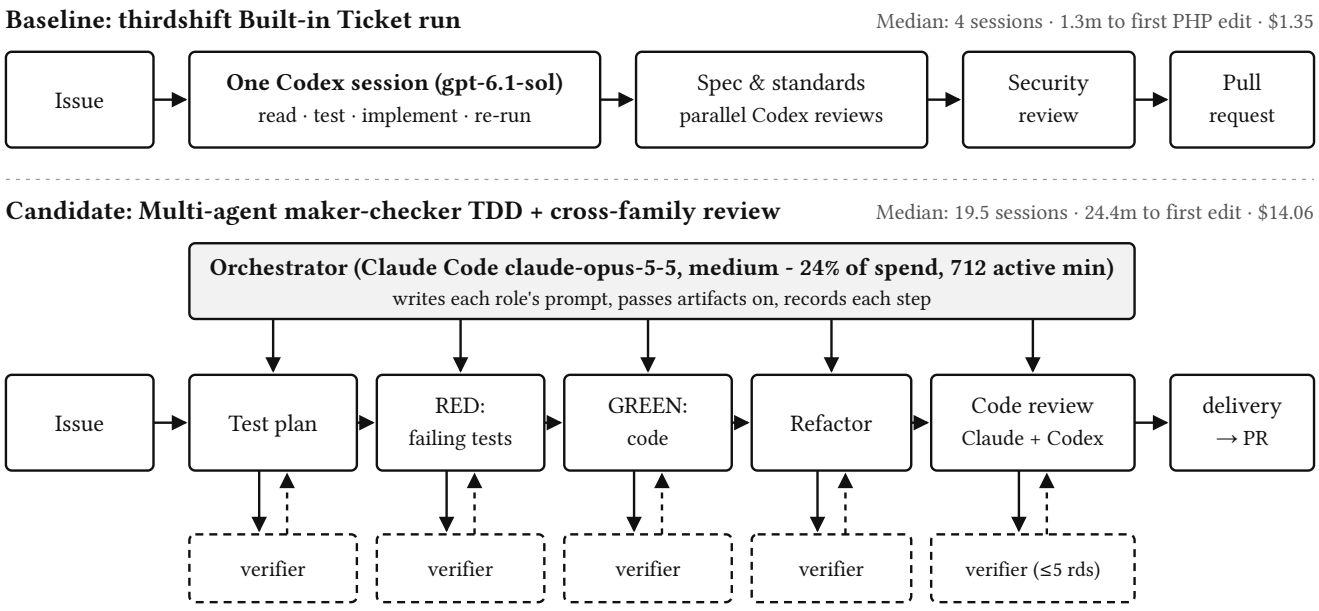


Figure 1. The two processes side by side. The Baseline's single session holds the issue, the code it has read and its change from start to finish, and is reviewed once. In the Candidate, each step runs in a separate context checked by a verifier from the other model provider, and a Claude Code orchestrator sequences the roles across up to five cross-family code review rounds.

Table 2. How the two designs differ.

	Baseline	Candidate
Who does the work	One agent session carries the issue from reading to pull request	Separate roles: test planner, test writer (RED), implementer (GREEN), refactorer and reviewers
Context	One context holds the issue, the code read and the change made	Each role starts fresh, and is handed the issue and the previous roles' artifacts
Who writes the tests	The session that implements	Roles that do not implement, before any code is written
Checking	Two reviewer sub-agents (spec and standards) and a security review, once, after implementation	A verifier from the other provider after every role; code review by both model families, up to five rounds
The program	Fixed by `thirdshift`	The specification describes the roles; the orchestrator writes the prompts, scripts and sequence each run
An unclear requirement	The session decides and carries on	Escalation rules stop work that depends on a product decision until its owner answers
Opening the pull request	The agent, with `gh`	`thirdshift`'s `delivery` Step

The Candidate is the proposed workflow as a whole, including its model choices for each role. We packaged it from the latest pinned revisions committed on 29 October 2026 at 18:32–18:33 UTC across its maker-checker TDD stage, its cross-family review stage, and supporting role execution rules; revisions pushed on 1 November or later were outside this freeze (see the follow-up second study for how later updates behaved).

At the 29 October freeze, the workflow instructions specified orchestration from a top-level harness without locking a single provider, noting in its role execution rules that the supplied recipes covered Claude Code orchestration and did "not yet supply Claude CLI maker or intermediate-verifier recipes for a Codex orchestrator." We therefore ran the Candidate's orchestrator on Claude Code (`claude-opus-5-5` , medium effort). The workflow's author noted afterward that he runs a Codex orchestrator by default, handing verification to Claude; that configuration was not tested here (Section 6) and became the direct subject of our second comparison. The frozen 44-file package manifest was checksum-verified on the Candidate host before every pair, and nothing was added to Tourbot or the replay repositories for either arm.

2.4 Units: issues and replays

The unit is a closed Tourbot issue, replayed from the commit before its fix, so nothing reaches production and the shipped fix remains a reference. Of 156 issues closed since 1 July 2026 by exactly one merged pull request, the design's research step shortlisted 24: based on `master` , 2 to 8 files changed, and no staging, migration or marketing work. The list of 24 was not saved, and applying those written criteria again on 3 November gives 77 issues, so the shortlist also rested on filtering that was not written down. Reading the 24, the research step found exactly ten with explicit acceptance criteria, which the correctness criterion needs, and all ten were enrolled, in the order they were labelled `ready-for-agent` . The author did not choose or drop any issue himself, but the shortlist cannot be rebuilt from its rule (Section 6).

Table 3. The ten enrolled issues, in pre-registered order.

Pair	Issue	Title
1	`I1`	Expense review gate blocks receipt-unavailable affidavit submissions
2	`I2`	Alert when background timeclock daemon logs auto-clockouts without applying them
3	`I3`	Shared mailbox view lists inline embedded images as downloadable attachments
4	`I4`	Flaky browser test: call-pop wheel scroll intermittently fails in CI
5	`I5`	Scheduled payment-reminder job: override-date check passes only by accident
6	`I6`	Saved quote emails retain the sender signature from composition time instead of send time
7	`I7`	Zero conference-call date logs a phantom audit entry and re-saves itself
8	`I8`	Profile "address has changed" notification fires on email-only and secondary-line-only edits
9	`I9`	Summary reporting dashboards hold the PHP session lock during slow queries
10	`I10`	Profile state selector overwrites non-US addresses with `AL` (Alabama)

For each pair, a deterministic preparation script pushed an isolated replay branch at the shipped fix's first parent, keeping only Tourbot's CI workflow scoped to replay branches. After that base commit's CI passed, the script filed the issue in each arm's isolated repository with only the title and body it carried when labelled `ready-for-agent` (recovered from issue edit history), omitting closed-issue comments because they name the shipped fix.

Both processes started from that issue as written, so this comparison does not test upstream spec writing. The maker-checker TDD stage begins with a requirements decision: it reuses requirements that already define the behaviour and acceptance criteria, and writes a new spec (with planning review) only when scope or design decisions are missing. In every Candidate run here the orchestrator reused the issue as written and began directly with the test planner; no separate archived specification template was packaged.

2.5 Isolation, randomisation and blinding

Isolation. Each process had its own isolated replay repository, its own OS account with its own Claude and Codex subscriptions, and a repository-scoped GitHub token so neither could read the other's work or the production Tourbot repository. The two trials of a pair ran in parallel on two separate, identical 8-vCPU / 16 GB cloud droplets (Ubuntu 24.04), cloned from a common snapshot before splitting credentials so each host held only its own arm's tokens. After each run, its full transcript was searched for the original issue number, closing pull request number and branch, and fix commit prefix; any leak hit would have voided the pair.

Randomisation. Setup assigned arms to isolated OS accounts and replay repositories by a coin flip (`$RANDOM`), recorded only in private operator storage. For each pair, the packet builder assigned the labels X and Y to the two pull requests using `random.SystemRandom().shuffle` , keeping the mapping hidden outside the grading account.

Blinding. The model graders ran under a separate unprivileged `grader` OS account that never held either mapping, inside a read-only `bwrap` sandbox mounted over the blind packet. Each packet contained the issue text, CI outcome counts, file snapshots for `base/` , `X/` and `Y/` , and formatted diffs `X.diff` and `Y.diff` . It stripped Git history, branch names, commit messages, pull request bodies, plan files and all agent/process directories, and formatted changed PHP files across `base/` , `X/` and `Y/` with a single tool (`php-cs-fixer` 3.95.27, `@PSR12`) so style could not identify either arm. The author judged from the same X/Y diffs through an isolated review interface showing no model grader output, and saw neither the X/Y mapping nor any grader verdict until all eight pairs' blind judgements were sealed. Test volume in the diffs could still hint at arm identity (Section 6).

2.6 Outcomes and measurement

Primary outcome: pair result (RQ1). The grading harness asked two model-based graders, Claude Code (`claude-opus-5-5`) and Codex (`gpt-6.1-sol`), to compare X and Y in both orders. Each was grounded in the issue, CI results, cross-tests and Infection output. The rubric, the same in every call, read: "Compare correctness against the issue and CI, then code quality against repository standards in base/. Read supporting files in X/ and Y/. Cite concrete defects and test evidence. Cross-test failures are disagreements to investigate, not automatic defects." Each call returned only a winner (X, Y or tie) and a reason. The Claude grader could read and search the packet but not run commands; the Codex grader ran in Codex's read-only sandbox, where it could also run commands and tests, which is how it reproduced the Pair 8 defect (Section 4.6). A verdict scores +1 for X, −1 for Y and 0 for a tie; a grader's preference is the sign of its net across the two orders, so a grader whose verdict flips with the order counts as a tie. The pair goes to a side only if both graders prefer it. Judging both orders is the measured remedy for position bias, which is worst between close candidates [4, 5]; grounding addresses the weak agreement between reference-free code judging and execution results [6].

Cost and time (RQ2). Time runs from the start of the `thirdshift` command until the pull request is ready, including CI and repairs, minus detected usage-limit stalls. Cost counts every token either process spent: The metrics collector sums `thirdshift`'s command-level token counts with every child sub-agent stream (`claude -p`, `codex exec`, and Codex native sub-agent rollouts) and prices them using a published API rate table frozen before the first run. A pair with an incomplete record is void. Pricing makes the two providers' tokens comparable; it is not ETA's subscription bill.

Correctness (RQ3). The operator counts a correctness failure when a cross-test or a grader's direct test demonstrates a behaviour defect on input the application can realistically hold, or when a process fails to deliver. Cross-tests transplant each side's added and changed tests onto the other side's code and run PHPUnit's Unit and Browser suites; a failure is a disagreement to investigate, not automatically a defect. Infection mutation testing ran on each side's changed `src/` PHP classes, outside either process's time.

Human attention (RQ4). The author recorded one blind judgement per pull request, "merge as is", "merge after small fixes" or "don't merge", with optional notes. It was registered as a reported outcome rather than a criterion because the only judge available was the person who commissioned the comparison (Section 6).

2.7 Sample size

Ten pairs is the smallest number at which a fixed win count separates a strong process from chance. Under a sign test, 8 or more wins in 10 occurs by chance 5.5% of the time. The two-stage rule in Section 2.8 wrongly adopts 7.9% of the time and detects a process that truly wins 80% of pairs 88% of the time (exact binomial calculation). It cannot detect a small advantage; it was designed to adopt only a process that is clearly better, because adoption means paying for it on every issue. Each process ran once per issue: when effects vary by issue, adding issues reduces uncertainty more than repeating runs [7].

2.8 Decision rule and analysis

The tally script applies the decision rule:

- A run that fails because of its own process, for example by ending without the issue implemented, is a **loss** for that process. A failure from outside both processes (a usage-limit stop, a CI or GitHub outage, a leak) **voids** the pair, which runs again; an issue voided twice is replaced by the next on the shortlist.
- With 10 counted pairs: **8 or more** Candidate wins means adopt, **5 or fewer** means reject, and **6 or 7** means running 8 more shortlisted issues and adopting only at 13 or more of 18.
- At a completed stage, the Candidate is rejected regardless of wins if it has more correctness failures than the Baseline, or if the median per-pair cost ratio or time ratio exceeds 2. Ratios are Candidate ÷ Baseline and include losses and ties.

All ten enrolled issues form the analysis population. The win rate's interval is the exact (Clopper–Pearson) binomial interval. Analyses not listed here are exploratory (Section 2.9).

2.9 Exploratory analyses of the differences

The analyses for EQ1–EQ4 were designed after the decision was known. They describe the processes and do not enter the decision.

Traces (EQ1). An automated trace analyzer parses the redacted agent transcripts from all twenty runs.

- A *session* is one agent context: a Codex or Claude Code session, a native sub-agent, or a child `codex exec` or `claude -p` run.
- Tool calls are counted one by one, including each call inside a Codex code-mode script. They are classified by tool and command:
  - reading or searching (`cat` , `sed` , `rg` , Read, Grep);
  - file edits (`apply_patch` , Edit, Write);
  - test runs (PHPUnit, Infection);
  - lint and static checks (`php -l` , PHPStan);
  - `git` ;
  - GitHub (`gh`);
  - starting, messaging or waiting on other agents.
- Candidate sessions are assigned to a stage of the workflow from the role names its orchestrator gave them.
- *Active time* counts gaps of up to five minutes between a session's events. An agent that sat idle until it was messaged again does not count the wait.
- *Cost by phase* prices each session's recorded tokens with the frozen price table of Section 2.6. The traces do not separate Claude's cache writes from its other uncached input, so all of it is priced as cache writes. The phases add up to within 0.5% of the recorded cost.

Pull requests (EQ2). Lines added are counted in each pull request's diff after the common formatter (Section 2.5). Lines under `tests/` count as test code, and everything else as code. Testing styles are taken from the graders' descriptions of each pull request, checked against the diffs.

Failures (EQ3). These are the correctness failures and process failures of Section 2.6, together with the cross-tests that disagreed and what each side's traces show about how its pull request came to differ.

Judgement (EQ4). The graders' 32 written reasons and the author's blind notes were read in full. Recurring reasons were grouped into themes, and each theme's pairs are listed so that it can be checked against the record. The themes were drafted by Claude Code (Section 8) and checked by the author.

Three pairs in detail. To show how the differences play out on one issue, one pair was chosen for each way the Candidate's result departed from a tie:

- Pair 6, its clearest win (both graders, both orders);
- Pair 8, its one correctness failure in a graded pair;
- Pair 2, the first of its two runs that stopped without a pull request.

The rule was fixed before the pairs were written up, so that the examples were not chosen to fit a conclusion.

3 Conduct of the comparison

3.1 Shakedown

One uncounted shakedown issue, `I0`, ran on 30 October 2026 to surface instrumentation issues before the counted pairs. It exposed fourteen plumbing problems - including a missing sandbox profile, a token permission gap for reading check runs, a 17-minute GitHub delay when registering a new repository's CI workflow, and formatter incompatibilities - all fixed across both environments before freezing the apparatus. The Candidate took about 16 times as long as the Baseline (260 vs 16.5 minutes), which led to running the two processes in parallel on separate machines rather than in turn on one.

Two differences between the processes were recorded and deliberately left unchanged: the Candidate has no security review, while the Baseline's Ticket run includes one; and the Candidate's scripts ran Codex reviewers at effort `high`, where the cross-family review specification documents `low` as its default.

3.2 Flow of issues and runs

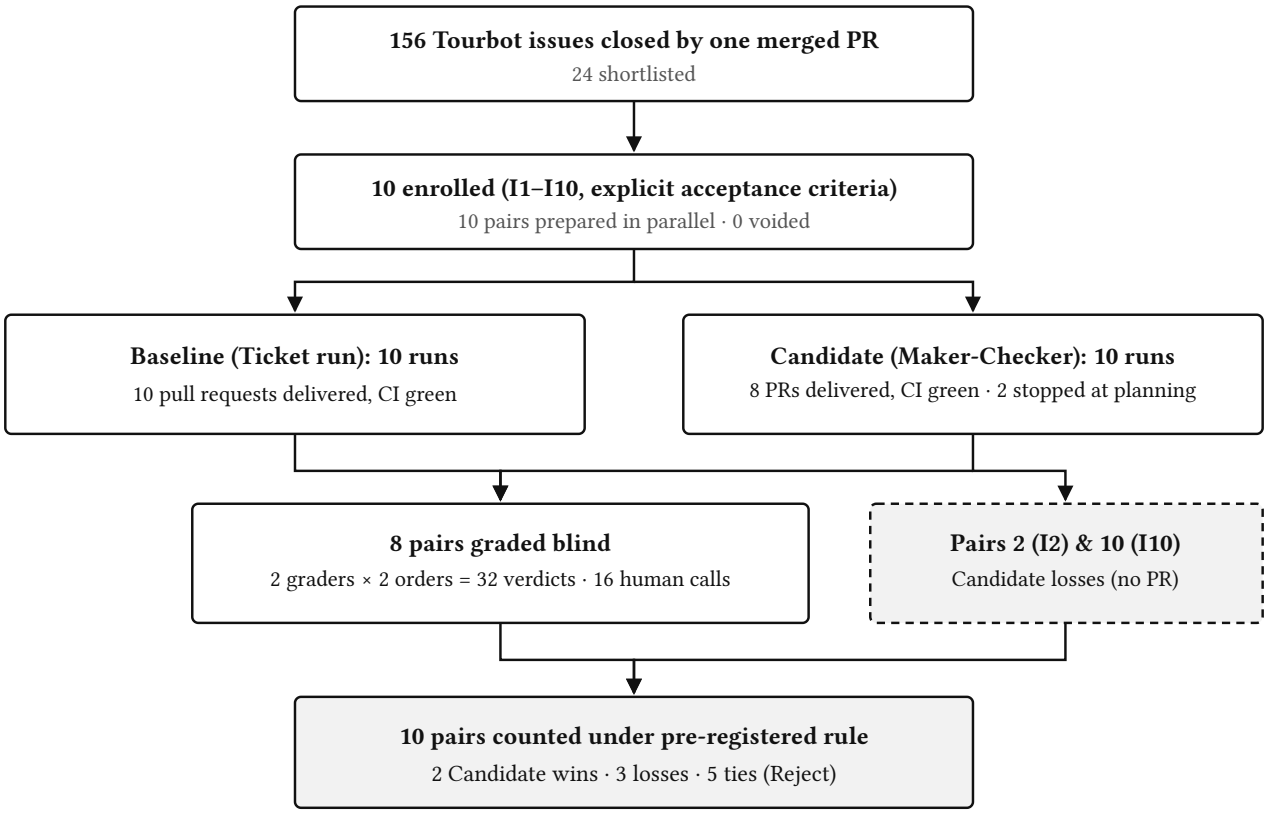


Figure 2. Flow of issues and runs through the comparison. The uncounted shakedown issue (`I0`) is excluded. Losses by process failure (`I2` and `I10`, where the Candidate stopped at planning without opening a pull request) are counted in the decision rule but not graded.

The ten pairs ran in order from 30 October to 1 November 2026 (Figure 2). Every completed run passed its leak search, and every metrics record was complete, with no warnings and no detected limit stalls, so no pair was voided.

In Pairs 2 (`I2`) and 10 (`I10`) the Candidate's run stopped at its planning stage. In Pair 2, the maker-checker TDD workflow raised a blocking question asking what should end an "incident" for a once-per-incident alert email, recommended an answer, and stopped; `delivery` then exited with no commits. In Pair 10 it likewise stopped before writing code. The Baseline opened a pull request with green CI in both. The author ruled both to be Candidate losses under the pre-registered rule ("a run that ends without the issue implemented"), on 30 October and 1 November, before any grading result was known.

The eight remaining pairs were graded between 30 October and 1 November. The author recorded his judgements on 2 November through a write-once blind review form that sealed each pair's scores without allowing edits after submission. Only

then were the verdicts and mappings opened.

3.3 Fidelity

Both processes ran as frozen: the Candidate's 44 files matched the manifest before every pair, and every run used `thirdshift` 0.18.6 with the models in Table 1. The Candidate kept its records in several directories; from Pair 3 onward, a root-owned loop mirrored its worktree's temporary scratch directory during each run so internal role artifacts were preserved before cleanup. This observed the Candidate and did not change it. The traces show one departure from the Candidate's own process: in Pair 3 its orchestrator finished after the refactor step without running its code review stage (Section 4.2). The run was counted as the process produced it.

3.4 Deviations from the registered design

Table 4. Changes made after the design was fixed. None changed which pull requests were compared, how they were labelled, or the decision rule.

#	Change	Reason	Approved	Effect on the analysis
1	Run the two processes in parallel on two cloned droplets, not in turn on one	The shakedown Candidate took 16× as long	30 Oct, before Pair 1	Identical hardware per process; no shared usage limits
2	Replay preparation waits for CI and may resume after a transient CI failure	GitHub registered new workflows late	30 Oct, before Pair 1	None on outcomes
3	Measure Codex child runs from retained rollouts when the Candidate deleted their directories; fix a collector parsing bug	Pair 2's Candidate deleted three child directories	30 Oct	Completes cost records; without it Pair 2 would be void
4	Format a file without `statement_indentation` when `php-cs-fixer` crashes on it, identically for `base`, `X` and `Y`	Formatter crash on a 10,364-line legacy file in Pair 9	1 Nov	Pair 9 only; both sides formatted alike
5	Cap each test or Infection output at 100,000 characters inline in the grader prompt and attach the full output as a file	Pair 9's prompt (1.84 M characters) exceeded Codex's 1,048,576-character limit	1 Nov	Pair 9 only; earlier prompts byte-identical
6	Mirror the Candidate's temporary scratch directory during runs	Its intermediate role records were lost to worktree cleanup	30 Oct, from Pair 3	Observation only

No change was made in response to a pair's result.

4 Results

Sections 4.2 to 4.6 report the exploratory analyses of the differences (Section 2.9). Section 4.7 reports the pre-registered decision.

4.1 Overview of the ten pairs

Table 5. The Candidate won 2 pairs, lost 3 and tied 5. Results for every pair; ratios are Candidate ÷ Baseline.

Pair	Issue	Claude grader	Codex grader	Pair result	Cost ratio	Time ratio	Author: Baseline	Author: Candidate
1	`I1`	Baseline	Baseline	**Loss**	19.4	10.2	merge as is	don't merge
2	`I2`	–	–	**Loss** (no PR)	6.9	3.8	not graded	no PR
3	`I3`	Candidate	Candidate	**Win**	7.4	5.8	merge as is	small fixes
4	`I4`	Candidate	no preference	Tie	18.6	7.4	merge as is	merge as is
5	`I5`	Candidate	Baseline	Tie	9.8	6.7	merge as is	small fixes
6	`I6`	Candidate	Candidate	**Win**	46.9	23.6	small fixes	don't merge
7	`I7`	Baseline	Candidate	Tie	23.5	12.7	merge as is	don't merge
8	`I8`	Candidate	Baseline	Tie	7.9	4.5	merge as is	don't merge
9	`I9`	Candidate	Baseline	Tie	11.0	6.2	merge as is	small fixes

Pair	Issue	Claude grader	Codex grader	Pair result	Cost ratio	Time ratio	Author: Baseline	Author: Candidate
10	`I10`	–	–	**Loss** (no PR)	2.9	2.0	not graded	no PR

CI passed on both pull requests of every graded pair. Five of the eight graded pairs were ties, because the two graders disagreed or one had no preference. Sections 4.2 to 4.6 analyse the complete session transcripts, formatted blind diffs, cross-test outputs, and written grader notes from all twenty runs.

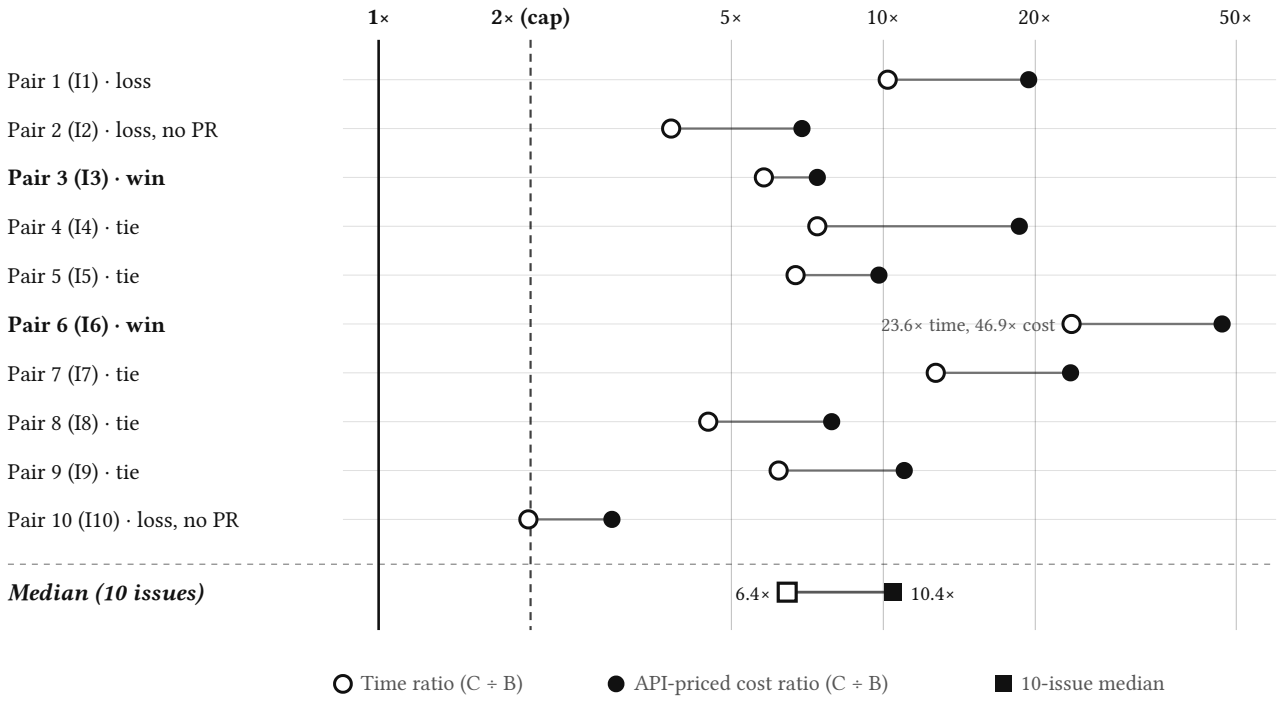


Figure 3. Candidate cost and time as multiples of the Baseline across every pair (`I1` – `I10` , log scale). The solid line marks equal cost or time (`1×`), and the dashed line marks the pre-registered decision rule's `2×` cap. Filled dots show API-priced cost ratios and open dots show time ratios; only `I10` 's time ratio (`1.98×` , after stopping at planning) fell just under the cap.

4.2 How they worked: one context against many (EQ1)

Table 6. The Candidate did the same kinds of work as the Baseline, about three times as much of it, and edited code last. Work per run: medians over the ten pairs; the ratio is the median of each pair's Candidate ÷ Baseline.

Per run	Baseline	Candidate	Median ratio
Agent sessions	4	19.5	4.9
Tool calls	144	412	2.8
Reading and searching	48	197	3.8
File edits	12	20	1.9
Test runs	15	44	3.4
Lint and static checks	6	11	1.7
`git`	13	28	2.4
GitHub (`gh`)	17	0.5	–
Starting, messaging or waiting on agents	8	53	4.9
Minutes to the first edit of a PHP file (8 pairs with one)	1.3	24.4	21.4

The Candidate did the same kinds of work, much more of it. Over the ten pairs it made 5,189 tool calls to the Baseline's 1,487. Most of the difference is reading (2,505 calls against 481) and test runs (598 against 162). Every role starts in a fresh context and reads the issue, the code and the plan again, and every verifier runs the suites again. File edits grew least (396 against 128), so the extra work went into reading and checking, not into the change itself. The Baseline's agent opened and watched its own pull request (174 `gh` calls); in the Candidate, `thirdshift`'s `delivery` Step did that outside the agents.

The Baseline starts coding within three minutes. Its single session read the instructions and the relevant code, and made its first PHP edit at a median of 1.3 minutes (at most 2.8). It then implemented, ran the focused tests and opened the pull request. In 7 of 10 runs that first edit was a new test. Two native reviewer sub-agents (spec and standards) checked the change in parallel, and a separate session ran the security review. Every Baseline run had exactly these four sessions. The Candidate's first edit to a PHP file came at a median of 24 minutes, after the test plan had been written and verified. Both processes wrote tests first. The difference is who wrote them, and how much verification came before them.

Table 7. Most of the Candidate's time and money went to coordinating and to planning tests, not to the fix. Where the time and cost went, over all ten runs; verifier sessions are those whose only job was to check another agent's work, and cost is at API prices.

Phase	Sessions	Verifier sessions	Active minutes	Tool calls	Cost ($)
Candidate					
Orchestrator (one Claude Code session per run)	10	0	712	977	53.45
Test plan	56	46	270	1,263	55.93
RED: failing tests	30	22	186	874	54.57
GREEN: implementation	28	14	85	743	17.57
Refactor	19	6	27	266	5.15
Code review	64	60	88	929	27.93
Pull request body, diagnosis	9	1	41	137	8.08
Baseline					
Implementation session	10	0	89	880	7.18
Spec and standards reviewers	20	20	19	309	3.51
Security review	10	10	26	298	2.68

Most of the Candidate's time went into planning and writing tests, not into the fix.

- Its sub-agents were active for 456 minutes writing and verifying the test plan and the failing tests.
- They were active for 85 minutes implementing the change: about as long as the Baseline's implementation sessions took in all (89).
- Verification rounds on the test plan drove its cost. The plan needed a median of 3 verifications, but 7 in Pair 1 (where a meta-reviewer was called in to break the deadlock), 8 in Pair 7 and 14 in Pair 6, where a review finding sent the run back to planning. These were three of its four slowest runs (124, 186 and 296 minutes).
- In Pairs 2 and 10 it stopped after 3 and 2 plan verifications without editing any code.

Half the Candidate's cost came before any production code. Over the ten runs it cost $222.67 by phase, against the Baseline's $13.39:

- Writing and verifying the test plan and the failing tests cost $110.50, 50%.
- The orchestrator cost $53.45, 24%.
- Code review cost $27.93, 13%, and the implementation itself $17.57, 8%.
- By provider, 85% was Claude and 15% Codex. Most of the Claude cost was cached context read again on every turn of long-running sessions.
- By role, 47% made the change (including test planners and writers), 29% checked it, and 24% coordinated. The Candidate's checking alone, $64.30, cost almost five times the Baseline's whole spend.

The three most expensive runs (Pairs 6, 7 and 4) were 64% of its total cost. Two of them had the most plan verifications: 14 in Pair 6 and 8 in Pair 7.

The orchestrator was its largest single cost in time. The Candidate's orchestrator was active for 712 minutes, eight times as long as the Baseline's whole implementation session. It wrote each role's prompt, froze artifacts with digests, audited which files each verifier had read, and recorded each step. By role, 33% of the Candidate's tool calls made the change, 48% checked it and 19% coordinated. The Baseline's split was 59% making and 41% checking. The maker-checker design raised the share of checking only modestly; most of the added work was coordination and repetition.

The Baseline ran the same way every time, and the Candidate never did. The skill describes roles, not a fixed program, and the orchestrator improvised the rest. It named roles, wrote its own runner scripts and chose where to keep state differently in each run, and its number of sessions ranged from 4 to 72. In Pair 3 it ended after the refactor step without running its code review stage. The Baseline's structure was identical in every run.

These counts come from the redacted transcript archive, where an input or output longer than 3,000 characters is truncated in the middle. A long Codex script could hide a call in the cut, so tool-call counts are a lower bound. Failed calls are not compared, because Codex code-mode reports exit codes inside its output, where they cannot be counted reliably.

4.3 What they produced: similar fixes, six times the tests (EQ2)

Table 8. The Candidate added more test code in all eight graded pairs: 14.7 test lines per line of code, against 3.7. Size of each pull request after the common formatter; B is the Baseline and C the Candidate.

Pair	Files changed B	Files changed C	Code lines added B	Code lines added C	Test lines added B	Test lines added C	Tests ÷ code B	Tests ÷ code C
1	4	5	24	32	156	1,013	6.5	31.7
3	2	3	11	22	130	412	11.8	18.7
4	1	1	0	0	45	56	–	–
5	5	4	21	13	243	687	11.6	52.8
6	4	9	103	207	153	1,736	1.5	8.4
7	8	8	75	114	221	1,808	2.9	15.9

Pair	Files changed B	Files changed C	Code lines added B	Code lines added C	Test lines added B	Test lines added C	Tests ÷ code B	Tests ÷ code C
8	3	6	69	80	185	963	2.7	12.0
9	4	4	16	16	45	448	2.8	28.0
Total	**31**	**40**	**319**	**484**	**1,178**	**7,123**	**3.7**	**14.7**

The fixes were similar in substance; the tests were not. CI passed on both pull requests of every graded pair. In five of the eight (Pairs 1, 3, 4, 5 and 9), the graders found that both fixed the issue, with no confirmed defect on either side. In the other three, the difference was an edge case (Table 9). The Candidate added more code in 5 of the 7 pairs that changed code, 1.5 times as much in all. It added more test code in all 8 pairs: 6.0 times as much in all, spread over 22 test files against the Baseline's 10. Its ratio of test to code ranged from 8.4 to 52.8 lines per line; the Baseline's ranged from 1.5 to 11.8. This fits what is known about agent-written tests:

- Coding agents change tests more readily than other contributors. In 1.2 million commits from 2025, 23% of coding-agent commits changed test files, against 13% of other commits [11].
- On SWE-bench Verified, prompting agents to write more or fewer tests changed their cost and number of steps far more than how many tasks they resolved [12].

The Candidate's tests more often read the code than ran it. Much of the affected code is top-level PHP in legacy page scripts, which is hard to call from a test, and both processes sometimes tested it by parsing the source. The Baseline did so in Pair 5, running the shipped loops through `php-parser`. In Pair 9 it asserted an exact whitespace string. In four pairs, the graders described Candidate tests that read the source instead of running it:

- Pair 1: a hand-written fake database that parses SQL with regular expressions, and token scans for helper calls.
- Pair 5: a 597-line tokenizer and `eval` harness.
- Pair 7: a 956-line wiring test that checks source tokens and runs `eval` on extracted code. Its own docblock says it can fail on a refactor that changes no behaviour.
- Pair 9: a token-based control-flow analyser of about 400 lines.

Elsewhere, the Candidate tested combinations exhaustively. In Pair 8 it tested all 729 combinations of unchanged, padded and changed fields, with process-isolated monkey-patching. In Pair 6 it checked the exact wording of generated emails. The Baseline's tests were usually a single file that ran the real handler against a small fixture, for example an in-memory SQLite database in Pair 1.

The Candidate kept to existing structure; the Baseline restructured more. The graders' code comparisons show a consistent difference in how each process changed the code.

The Candidate tended to extend what was there:

- Pair 3: it moved the existing payload decoding into a shared helper, so the renderer and the attachment list agree.
- Pair 5: it placed its helper beside the related functions and kept a check that the issue said must stay unchanged.
- Pair 6: it shared the existing signature builder.
- Pair 9: it added the session release after each existing gate.

The Baseline more often reshaped code and tightened types:

- Pair 5: a new one-function module, and removal of a dead variable.
- Pair 6: stricter parameter types on the email builder.
- Pair 7: a typed property on the booking domain model.

- Pair 8: a changed signature on the profile-update method.
- Pair 9: a restructured include order on the Revenue YTD page.

The graders counted some of these restructurings for the Baseline, as bounded refactoring, and some against it, as unrequested scope.

Both processes added things nobody asked for. The Candidate added daylight-saving handling (Pair 7), readable labels in a notice email (Pair 8) and long specification-style comments (Pair 6). The Baseline added a signature to generated bus emails (Pair 6) and Latin-1 conversion (Pair 8). Two of these additions, one on each side, turned out to matter most: each prevented the defect that the other process shipped (Section 4.4).

4.4 How they failed, and what each caught (EQ3)

Table 9. Correctness failures: 3 for the Candidate (two of them runs without a pull request) and 2 for the Baseline, with the evidence for each.

Pair	Baseline	Candidate	Evidence
2, 10	–	**Failure**	No pull request: process failure
6 (`I6`)	**Failure**	–	The Candidate's tests, run on the Baseline's code, show coordinator text beside a signature deleted, a signature copy in Notes rewritten, and stale contact sentences left in legacy bus emails
7 (`I7`)	**Failure**	–	Two independent probes: the Baseline's date helper throws on a stored time in the spring-forward hour, which can break the booking page and block unrelated saves
8 (`I8`)	–	**Failure**	Two independent probes: with the application's latin1 connection, the Candidate sends a false change notice, and corrupts the old value, when stored and submitted text match in different encodings

One reported defect was not counted. In Pair 1 the Candidate accepts an affidavit reason of only whitespace, but Tourbot's entry forms trim the reason before storing it, so the input cannot occur. The other cross-test failures came from different helper names and file layouts, an unrelated random-token flake, or a shared environment failure: one test needs Git metadata that the grading copy lacks.

The Candidate's independent tests found what the Baseline's session missed. In Pair 6 its tests, run against the Baseline's code, exposed two kinds of defect. One was a name-line match broad enough to delete a coordinator's note written beside the signature. The other was a narrowed condition that left stale contacts in legacy bus emails. Its code also tended to fail safe: when a saved signature did not match exactly, it sent the body unchanged. In Pair 7 it handled the spring-forward hour that broke the Baseline's helper. In Pair 3 it updated a functional test that the Baseline left contradicting the new behaviour, and in Pair 4 it tightened an assertion that the existing test could pass vacuously. This is the independent, adversarial checking the design is for, and it is consistent with evidence that independently written tests detect more faults [1].

The Baseline's session caught what the Candidate's roles did not look for. In Pair 8 the Baseline's single session wrote a regression test for latin1-stored values and converted them before comparing. The Candidate's test planner opened the file that sets the database connection to latin1, and one of its Codex reviewers noted that non-ASCII input lacked test evidence. Even so, neither the plan, the 729-combination grid nor the fix allowed for the encoding, and the defect shipped. Verification checked each role's work against the issue and the plan. It did not supply knowledge that no role had gone looking for. In Pair 7 the Baseline reported malformed input to the user, where the Candidate silently dropped the edit; the Claude grader weighed this for the Baseline.

The Candidate also failed in ways the Baseline cannot. Its two runs without a pull request stopped at a product question that no one was present to answer (Section 4.6). Its Pair 3 run skipped its own review stage. Its cost per pair ranged from $3 to $74, against $1–2 for the Baseline. It also has no security review.

Its failures match known multi-agent failure modes. MAST, a taxonomy of 14 failure modes built from annotated traces of seven multi-agent frameworks [10], names each of these problems:

- The test plan's repeated verification rounds, up to 14 in Pair 6, are *step repetition* (FM-1.3).
- The two runs that stopped at a planning question, in an unattended run where no one could answer it, are *premature termination* (FM-3.1).
- Pair 3's missing review stage is *no or incomplete verification* (FM-3.2).

MAST's authors attribute most such failures to system design, meaning how roles, hand-offs and stopping conditions are specified, rather than to model capability [10]. That fits a process whose skill describes roles and leaves the orchestrator to improvise the program.

4.5 What the reviewers valued (EQ4)

The graders weighed coverage; the author weighed size. Both graders credited thoroughness in the Candidate's pull requests: more acceptance criteria covered, more edge cases, and a test suite kept consistent with the change. Both also noticed its size. In 6 of the 8 graded pairs (1, 3, 5, 6, 7 and 9), at least one grader called its tests verbose, heavy, brittle or over-engineered. In five verdicts, size was among the reasons a grader preferred the Baseline:

- the Claude grader, in Pairs 1 and 7;
- the Codex grader, in Pairs 1, 5 and 9.

Otherwise, demonstrated defects and coverage outweighed size. The author's blind notes name test bloat or excessive tests on 7 of the 8 Candidate pull requests. An example is "Test bloat, 10x more LOC for the tests … session lock doesn't seem worth that much code relative to the size of the implementation" (Pair 9). His other notes concern:

- tests tied to the exact wording of emails, and scope expansion (Pair 6);
- over-detailed comments (Pair 6);
- tests that name specific staff members (both sides in Pair 6);
- seeded data and process-isolated monkey-patching in place of plain unit tests (Pair 8);
- on the Baseline's side, praise for code that uses the domain's own language (Pair 1).

The graders disagreed with each other, in the direction self-preference would predict. Both graders preferred the same side in 3 of 8 pairs and opposite sides in 4. In the eighth, the Codex grader's verdict flipped with the order. 15 of the 16 grader-by-pair judgements were the same in both orders. The Claude grader preferred the Candidate, whose orchestrator is Claude, in 6 of 8 pairs. The Codex grader preferred the Baseline, which runs on Codex, in 4, and the Candidate in 3. With eight pairs this is suggestive only, but it is the direction that self-preference would predict [8]. Each grader was the same model as one arm: `claude-opus-5-5` ran the Candidate's orchestrator, and `gpt-6.1-sol` ran the Baseline and many of the Candidate's roles. So this design cannot tell a preference for a grader's own model from a preference for its family. Nor can it rule out a simpler reading of the Claude grader's lean, a preference for more tests: the Candidate added more test code in every graded pair.

The author and the graders rarely agreed. The author's judgement matched the graders' pair result only in Pair 1. In both pairs that the graders gave to the Candidate (3 and 6), he rated the Baseline higher. In Pair 6 he marked the Baseline "merge after small fixes", with notes on test size, comments and staff names. Yet the cross-tests show it deleting coordinator-written text (Table 9), which needs more than a small fix. The graders rewarded defects avoided and coverage; the author rewarded small, readable changes that fit the codebase, and read diffs without the cross-test evidence.

4.6 Three pairs side by side

Table 10. In all three detailed pairs the Candidate took far longer; it won one, failed one and delivered nothing in one. The pairs were chosen by the rule in Section 2.9.

	Pair 2 (`I2`)	**Pair 6 (`I6`)**	**Pair 8 (`I8`)**
Result	Candidate loss (no PR)	Candidate win	Tie; Candidate correctness failure
Minutes, Baseline / Candidate	8.6 / 33.1	12.5 / 295.9	11.7 / 52.8
Cost ($), Baseline / Candidate	1.12 / 7.76	1.58 / 74.14	1.47 / 11.62
Agent sessions, Baseline / Candidate	4 / 5	4 / 72	4 / 17
Tool calls, Baseline / Candidate	134 / 153	137 / 1,639	180 / 298
Author, Baseline / Candidate	not graded / no PR	small fixes / don't merge	merge as is / don't merge

Pair 2: an ambiguity found, and nothing delivered. The issue asks for an email when the timeclock daemon logs automatic clock-outs without applying them, sent "once per incident, not every run".

- *Candidate.* Its test planner wrote a plan, and three fresh Codex verifiers each failed it. The second and third, working separately, both found that the issue never says what ends an incident. Does the problem leave the one-hour window it is measured in, or must there be evidence that the daemon has resumed? The orchestrator judged this a product rule that tests should not invent. It wrote a fourth plan marking every case that depended on the answer, asked the question with a recommended answer, and stopped. `thirdshift`'s `delivery` Step then failed with no commits.
- *Baseline.* Its session met the same question and answered it itself: an incident ends when the check no longer finds unapplied clock-outs. It wrote that rule into a regression test verifying that the alert does not repeat until an incident recovers, and documented it in the pull request description. Both reviewers reported no findings, and it opened the pull request in under nine minutes.

The Candidate found a real ambiguity that the Baseline settled silently, but in an unattended run, stopping delivered nothing. Pair 10 went the same way: two plan verifications, then two product questions (how to show a NULL state, and whether non-ASCII values must survive an untouched save) and a stop before any test was written.

Pair 6: the design working, at a price the author would not pay. The issue asks for saved bid emails to use the current signature when sent, without touching anything a coordinator wrote.

- *Baseline.* It finished in 12.5 minutes with four sessions. Its fix matched any name line of up to 100 characters and rebuilt the whole signature paragraph. Its own tests passed.
- *Candidate.* It ran for nearly five hours across 72 sessions. Its test plan was verified 14 times, and a review finding sent the run back to planning. Its 1,736 lines of tests, built from realistic saved emails, included a coordinator's note written beside the name, a signature pasted into Notes, and legacy bus emails that do not end in the expected block. Run against the Baseline's code, these tests showed it deleting the note, rewriting the Notes copy and leaving stale contacts. The Candidate's own fix replaced only the name, title and phone fields, and sent the body unchanged whenever it could not be sure.

Both graders preferred the Candidate in both orders. Judging the diffs blind, the author would not merge it. He cited its test volume, a long docblock that reads as a specification, and tests that pin the exact wording of whole emails. He also cited tests naming specific staff members, which he noted on both sides. He would merge the Baseline after small fixes. Both readings are right about different things: the Candidate found real defects, and its pull request is one the team would not want to maintain.

Pair 8: exhaustive tests of the wrong model. The issue asks for the "address has changed" email to the tour manager to fire only for real address changes.

- *Candidate.* It ran 17 sessions and produced 963 lines of tests: all 729 combinations of unchanged, padded and changed fields, end-to-end tests of the profile save with a recording mailer, and a check that the class loads without Composer. Its test planner had opened the file that sets the connection's latin1 charset, and one Codex reviewer noted that non-ASCII input lacked test evidence. No role acted on either. Its comparison treats a latin1-stored "Montréal" and the same word submitted as UTF-8 as different, so it sends a false notice and corrupts the old value.
- *Baseline.* Its single session, in 11.7 minutes, wrote a regression test for exactly that case and converted legacy values before comparing.

The graders split. The Claude grader preferred the Candidate's coverage and labels; the Codex grader reproduced the encoding defect and preferred the Baseline. Many independent checks did not substitute for one agent that knew where to look.

4.7 The pre-registered decision (RQ1–RQ4)

Table 11. The pre-registered rule rejects the Candidate on every criterion.

Criterion	Result	Threshold	Outcome
Candidate wins (RQ1)	**2 of 10**: 3 losses, 5 ties	≥ 8 adopts; ≤ 5 rejects	Reject
Median cost ratio (RQ2)	**10.4×**	≤ 2	Reject
Median time ratio (RQ2)	**6.4×**	≤ 2	Reject
Correctness failures (RQ3)	Candidate **3**, Baseline **2**	Candidate ≤ Baseline	Reject

Quality (RQ1). The Candidate won 2 pairs (3 and 6), lost 3 (1, 2 and 10) and tied 5 (Table 5). Its win rate was 20% (exact 95% CI 2.5–56%), which excludes the 80% the rule was built to detect. All three hypotheses fail, and each failure alone rejects the Candidate.

Cost and time (RQ2).

Table 12. The Candidate cost more and took longer in every pair. Time and API-priced cost per pair.

Pair	Baseline (min)	Candidate (min)	Baseline ($)	Candidate ($)
1	12.2	124.3	1.16	22.57
2	8.6	33.1	1.12	7.76
3	7.0	40.6	0.97	7.16
4	29.7	219.4	1.66	30.82
5	9.4	62.7	1.28	12.58
6	12.5	295.9	1.58	74.14
7	14.7	185.8	1.63	38.34
8	11.7	52.8	1.47	11.62
9	9.9	61.3	1.41	15.54
10	9.4	18.6	1.11	3.18
Total	**125**	**1,095**	**13.39**	**223.71**

The Candidate's median cost per pair was $14.06 against the Baseline's $1.35, and its median time 62.0 minutes against 10.8 (Figure 3). Over ten pairs it spent $223.71 against $13.39 (16.7 times as much) and 1,095 minutes against 125 (8.7 times as long).

- *Tokens.* The Baseline used 56.6 M Codex input tokens (including cache) and 0.30 M output tokens. The Candidate used 319.6 M Claude and 80.9 M Codex input tokens, and 2.78 M and 0.61 M output tokens.

- *Range.* On every pair where it opened a pull request, the Candidate cost at least 7.4 times as much. Its lowest ratio, 2.9, was Pair 10, where it stopped at planning. It cost more than twice as much in all ten pairs, and took more than twice as long in nine (Pair 10's time ratio is 1.98).
- *Interval.* The registered rule uses the medians as point estimates. As a check, a distribution-free interval from the order statistics (the 2nd and 9th of ten ratios, 97.9% coverage) puts the median cost ratio between 6.9 and 23.5, and the median time ratio between 3.8 and 12.7. Both lie wholly above the cap of 2. This interval is not part of the registered analysis.

Correctness (RQ3). The Candidate had 3 correctness failures: two runs without a pull request and the encoding defect in Pair 8. The Baseline had 2, in Pairs 6 and 7 (Table 9). The decision would be the same if every correctness call went the other way, because the quality and the cost-and-time criteria (RQ1, RQ2) each reject on their own. Mutation testing gave no evidence. Infection is limited to changed `src/` PHP classes, and in seven pairs neither side changed one, because the fixes were in legacy code, so no mutations were generated. In Pair 8 its initial test run failed on both sides, because one test calls `git` and the grading copy has no Git metadata.

Human attention (RQ4).

Table 13. Judging blind, the author would merge 7 of 8 Baseline pull requests as is and 1 of 8 Candidate pull requests.

	Baseline (8 graded)	Candidate (8 graded)
Merge as is	**7**	1
Merge after small fixes	1	3
Don't merge	0	**4**
No pull request (Pairs 2, 10)	0	2

Comparing his two judgements per pair, the author rated the Baseline higher in 7 pairs and both equally in 1 (Pair 4, a test-only fix). He never rated the Candidate higher. Had the registered rule also required the Candidate to be merge-ready as often as the Baseline, it would have failed that criterion too; the decision would be the same.

5 Discussion

5.1 Principal findings

On the same ten issues, the two processes behaved as their designs predict, and the differences decided the outcome.

The Baseline's single session started coding within minutes, wrote compact tests, settled unclear requirements itself and finished in about twelve minutes for a dollar or two. The Candidate spread each issue across a median of twenty fresh-context agents. Each re-read the code and re-ran the tests, and most of their time went into planning and verifying tests before any fix.

That independence bought real things:

- defects found in the Baseline's code;
- edge cases the Baseline missed;
- code that failed safe;
- an ambiguity the Baseline settled silently.

It also cost a median ten times as much and six times the time. It produced six times the test code, often coupled to the implementation, and it stopped twice without delivering anything.

The model graders, which weigh coverage and demonstrated defects, split their preferences. The author, who weighs what the team will maintain, preferred the Baseline in seven pairs and the Candidate in none. The pre-registered rule rejects the

Candidate on every criterion, and ETA keeps the Built-in Ticket run.

5.2 What each design buys, and what it costs

Independence finds defects that one context misses. A role that writes tests without having written the code, and a verifier from another model family, look for different failures than the author of the change does. In Pairs 6 and 7 this found defects the Baseline shipped. In Pair 2, two verifiers working separately found the same gap in the issue. Across pairs, the Candidate's code more often failed safe, and more often extended existing structure than reshaped it. These are the strengths those three maker-checker principles aim at, and they appeared.

Independence also loses context, and pays to rebuild it. Every fresh role read the issue, the code and the plan again (Table 6), and the orchestrator spent eight times as long coordinating as the Baseline spent implementing. Independence did not guarantee coverage. In Pair 8, seventeen sessions tested 729 combinations, and still none allowed for the database encoding that the Baseline's single session handled. That was despite one role having read the relevant line, and one reviewer having flagged the gap. Verification checks work against the issue and the plan. It cannot check against knowledge that no role went looking for.

The comparison was not between test-driven development and its absence. Both processes wrote tests first; the Baseline's first PHP edit was a new test in 7 of 10 runs, about a minute after it started. Writing tests first helps agents in other settings, raising full-stack application accuracy by 15.5 to 23.7 percentage points with capable models [13]. What the Candidate added was the separation of test planning, test writing, implementation and refactoring into independent agents, each verified by another provider. Its cost came from that separation, not from writing tests first. Its own maker-checker specification calls separating test writing from implementation "an initial allocation to evaluate, not a proven performance advantage". On these issues, the evaluation came out against it.

Verification without a stopping rule produces tests, not quality. The Candidate's verifiers failed the test plan a median of three times and up to fourteen, and each round could add cases. The result was test suites of 8 to 53 lines per line of code, with harnesses that parse and `eval` the source. The graders partly credited these, and the author consistently judged them a maintenance burden. Agents' readiness to write and change tests is documented [11], and more agent-written tests mostly change cost rather than outcomes [12]. A process that rewards a verifier for finding gaps, without limiting how much test code closing them may add, will keep growing its tests.

Stopping for a human is right in a conversation and wrong in an unattended run. Both of the Candidate's stops were faithful to its skill. Each named a real product question, with a recommended answer. In a run that no one is watching, though, a question means no pull request. The Baseline made the same kind of call itself and recorded it where a reviewer would see it: in a test and in the pull request description.

5.3 Model graders and the human measured different things

The graders rewarded coverage and defensive handling, and had the cross-test evidence in front of them. The author rewarded small, readable changes that fit the codebase, and judged from the diffs alone. Each missed something. The graders gave Pair 6 to a pull request the author would not merge. The author would have merged, after small fixes, a Baseline pull request that deletes coordinator text. The graders' preferences also pointed the way self-preference would predict, though eight pairs cannot establish it (Section 4.5). The design kept human judgement out of the decision, and here it would not have changed the outcome. It matters for future comparisons: a process tuned to win model-graded comparisons may drift away from what the team will merge, and a human reviewer working from diffs alone will miss defects that only tests reveal.

By ETA's own goal, the human measure matters most: the point of a process is work that a person can merge without review rounds. Here the model graders were a poor stand-in for it. Their pair result matched the author's only in Pair 1, and in both pairs they gave the Candidate, he preferred the Baseline. A comparison decided on model-graded preference alone could adopt a process whose pull requests the team would not merge.

5.4 Limits of the decision rule

The rule was built to adopt only a clearly better process, and it is weak at detecting a modest advantage. A process that truly wins 60% of pairs would usually be rejected or sent to the 18-pair stage. That limit does not affect this result. To be adopted at the first stage, the Candidate needed 6 more wins. Even then, both the cost and time caps would have rejected it: its median cost ratio would have had to fall by a factor of five. A different price table would change the ratios, but not by that much. The caps reflect the author's stated tolerance of at most twice the time and cost. A team willing to pay more for quality would weigh quality (RQ1) alone, and that also fails.

5.5 Implications

For ETA. ETA keeps the Built-in Ticket run for Tourbot. Two of the Candidate's ideas are worth testing as additions to it, separately and far more cheaply than adopting the whole process:

- *An independently written test pass.* An agent that has not seen the change writes tests from the issue alone, and they are run against the Baseline's code. This targets the defects of Pairs 6 and 7.
- *An explicit assumptions section.* Where the issue leaves a rule open, the Ticket run records the rule it chose in the pull request, as it did in Pair 2, so that a reviewer can check it.

For the next comparison. These recommendations were made after the results were known and were not applied to this one (several were carried directly into our follow-up second study):

- Make merge-readiness a registered criterion, not only a reported outcome: for example, the Candidate must be judged "merge as is" at least as often as the Baseline.
- Use at least two blind human reviewers working independently, at least one of whom did not commission the comparison.
- Give the human reviewers the cross-test evidence, so that defects only tests reveal (as in Pair 6) count in their judgement.
- Fix Infection's scope so that mutation testing reaches the code that actually changed.

For the proposed workflow. The traces point to what to change first:

- a fixed limit on verification rounds, and a budget for test code relative to the change;
- a rule that turns an unanswered planning question into a stated assumption when no human is available, rather than a stopped run;
- a fixed program for the orchestrator, rather than one it improvises each run;
- a step that asks each role what in the environment (encodings, time zones, legacy callers) could break the change, since verification against the plan cannot find what the plan left out.

For multi-agent design generally. These results fit MAST's finding that multi-agent failures come mostly from how roles, hand-offs and stopping conditions are specified [10]. Separating roles gave independent viewpoints. Without limits on rounds and scope, and without a way to proceed under uncertainty, it also gave repetition, growth and premature stops. On small fixes to a mature codebase, a single capable session was the better default. Whether the balance shifts for larger features or new code, where an independent specification has more to catch, is the open question this comparison cannot answer.

6 Threats to validity

Construct validity.

- *Grader preference as a measure of quality.* The graders disagreed in most pairs and may favour their own model or its family; each was the same model as one arm (Section 4.5). Requiring both to agree makes the win count conservative for adoption.

- *API prices as a measure of cost.* ETA pays by subscription, not per token. Pricing both providers at API rates is a comparable proxy, and with a median ratio of 10.4 a different table would not bring it under 2.
- *Mutation testing* contributed nothing (Section 4.7), so correctness rests on CI, cross-tests and the graders' direct tests.
- *Correctness calls* are the operator's judgement; the quality and cost-and-time criteria (RQ1, RQ2) reject independently of them.
- *Trace counts* (Section 4.2) depend on a classification of tool calls written after the results were known, and on traces cut at 3,000 characters per field.
- *Lines of code* (Section 4.3) are counted after the common formatter and do not distinguish test fixtures from test logic.

Internal validity.

- *Blinding.* Labels, history and process files were hidden and the diffs were formatted alike, but the volume of tests can hint at the origin of a pull request, to graders and to the author.
- *Leakage.* Every trace search for the shipped fix was clean.
- *Process fidelity.* We packaged the Candidate from unmerged workflow specifications at the 29 October freeze without tuning or confirming run-time adaptations with the workflow's author, ran it under a Claude Code orchestrator, without a security review, and with Codex reviewers at a higher effort than documented (Section 2.3). The workflow's author subsequently noted running a Codex orchestrator by default - a change tested directly in our follow-up second study. Since the orchestrator was 24% of the Candidate's cost, all of it on Claude (Section 4.2), that configuration would change the cost mix by an amount this comparison did not measure.
- *Single human judge.* The author commissioned the comparison, set the caps, and has a stated preference for small diffs; his blind judgements are evidence for the human-attention question (RQ4) only. Adding them to the rule would not have changed the outcome: he rated the Baseline higher in 7 pairs and the Candidate in none.

The exploratory analyses. Sections 4.2 to 4.6 were designed after the decision was known, and are descriptive.

- *Theme coding.* The themes of Section 4.5 and the accounts of Section 4.6 were drafted by Claude Code, the same model family as the Candidate's orchestrator and one grader. To limit this, every theme lists its pairs, so that it can be checked against the scores, notes and traces.
- *Issue selection.* The shortlist of 24 cannot be rebuilt: its written criteria now give 77 issues, and whether an issue had explicit acceptance criteria was the research step's reading, not a mechanical test (Section 2.4). The author chose no issue, but a filter that was not recorded could have favoured one kind of issue.
- *Case selection.* The three detailed pairs were chosen by a rule (Section 2.9), not for how well they illustrate a point. Other pairs could show other things, and all ten pairs' transcripts were preserved and analysed for that reason.

External validity. The issues are small fixes (2–8 files) to one legacy PHP application, chosen without the author's taste, though by criteria not all written down (Section 2.4). The result may not hold for larger features, new code, other languages or other repositories, where an independent specification may have more to catch. The comparison starts from issues that are already ready for an agent, so it says nothing about the Candidate's spec writing and spec review, which its author may regard as half of the process; evaluating those stages requires a comparison starting earlier in triage. The proposed workflow was built for a Python repository with stronger test conventions. Model and tool versions will change.

Conclusion validity.

- *Run-to-run variation.* One run per process per issue leaves each pair exposed to it, measured at 2.2–6.0 points of pass@1 between runs of the same agent [9]. `thirdshift`'s own earlier repeatability check with identical arms disagreed on 2 of 10 tasks.
- *Sample size.* Ten pairs give a wide interval on the win rate (2.5–56%).

- *Margin.* The decision does not rest on a narrow margin: 2 wins against a threshold of 8, and cost ratios between 2.9 and 46.9 against a cap of 2. The distribution-free intervals for the median cost and time ratios lie wholly above the cap (Section 4.7).
- *The differences* in Sections 4.2 to 4.6 are consistent across pairs (the Candidate added more test code in all eight graded pairs, cost more in all ten, and started more sessions in nine of ten), but each rests on ten runs per process.

7 Conclusion

Given the same ten issues, a single agent session and a maker-checker team of agents worked, produced and failed in characteristically different ways. The Candidate's independent roles found defects and ambiguities that the Baseline's session missed. They also rebuilt context at every step, multiplied tests well beyond the size of each fix, missed what no role thought to check, and stopped when no human could answer. Against the pre-registered criteria:

- **Quality (RQ1):** blind model graders did not prefer the Candidate (2 wins in 10).
- **Cost and time (RQ2):** it cost a median 10.4 times as much and took 6.4 times as long.
- **Correctness (RQ3):** it had 3 correctness failures to the Baseline's 2.
- **Human attention (RQ4):** a blind human reviewer would merge 7 of 8 Baseline pull requests as is and 1 of 8 Candidate pull requests.

Human attention was reported rather than decisive, but it is the measure closest to ETA's goal, and it points the same way. The rule rejects the proposed workflow on every criterion, and ETA keeps `thirdshift`'s Built-in Ticket run for Tourbot. Two of the Candidate's ideas, an independent test pass and an explicit record of assumptions, are worth testing on their own.

8 Declarations

Data and code availability. `thirdshift` is open source at github.com/JacobStephens2/thirdshift. All aggregated metrics, per-pair outcomes, phase breakdowns, diff sizes, correctness findings and blind human ratings from this study are reported in Tables A and 1–13 of this paper (and summarized in machine-readable form in `agents.md`). The replay repositories, raw agent transcripts, per-task diffs and unpublished Candidate workflow files are withheld because they contain private Tourbot production application code and third-party workflow materials.

Conflicts of interest. The author works on Tourbot at Educational Travel Adventures, commissioned the comparison and set its pre-registered thresholds. He also develops `thirdshift`, which executed both processes and whose Built-in Ticket run is the Baseline. The Candidate workflow was proposed independently for adoption on Tourbot. To guard against bias, the decision rule was fixed before any counted pair ran, both model graders were blinded and evaluated both presentation orders, and the author's own blind merge-readiness judgements were reported alongside the rule rather than inside the adopt/reject vote.

Packaging independence. To prevent mid-study prompt tuning while the ten counted pairs ran, the 29 October workflow revision was packaged directly from its written specifications without consulting its author during execution. Once the ten pairs and blind grading concluded, the full report, frozen package and score breakdowns were shared with him for review prior to designing the second comparison.

Funding. Internal. Model usage ran through ETA's Claude and ChatGPT subscriptions on ETA cloud droplets.

AI use and contributions. The author designed the comparison through a structured question-and-answer design session with Claude Code, reviewed and merged the grading and metrics tooling, made every ruling on losses and operational deviations, and recorded all blind human judgements. `thirdshift` Ticket runs and Claude Code wrote the comparison tooling; Claude Code (`claude-opus-5-5`) coordinated the counted runs, collected evidence, ran grading and the tally, assisted

in drafting the exploratory analyses and manuscript prose for the author's review, and prepared this public edition with Gemini (`gemini-4-argon`).

References

1. Konstantinou, M., Tambon, F., and Papadakis, M. On the risk of coding before testing: An empirical study on LLM-based test generation workflow. arXiv:2607.05139, July 2026. Tests generated after faulty code detected 14% of faults; tests generated independently, 25%.
2. Kim, Y., Gu, K., Park, C., et al. Towards a Science of Scaling Agent Systems. arXiv:2512.08296, December 2025. 260 agent-system configurations; on SWE-bench Verified every multi-agent architecture did slightly worse than a single agent (−2.1% to −14.9%).
3. Xiang, Z., Zhang, Y., Zhang, Y., and Xu, H. Cross-Model LLM Code Review: Should you use Claude to review Codex or vice versa? *Agentic SE @ KDD '26*, arXiv:2607.21656, July 2026. Claude drafts passed 91.4% alone and 82.8% after GPT review; GPT/Codex drafts passed 71.6% alone and 89.7% after Claude review.
4. Zheng et al. Judging LLM-as-a-judge with MT-Bench and Chatbot Arena. *NeurIPS*, 2023. arXiv:2306.05685.
5. Shi et al. Position bias in pairwise LLM judging. *AACL*, 2025. Bias is strongest when candidates are close in quality.
6. Crupi et al. LLM-as-a-judge for code generation. *IEEE Transactions on Software Engineering*. arXiv:2507.16587. Reference-free judging agreed with execution results at Cohen's κ 0.10–0.21.
7. Miller. Adding error bars to evals. arXiv:2411.00640, 2024.
8. Pombal, J., Rei, R., and Martins, A. F. T. Self-Preference Bias in Rubric-Based Evaluation of Large Language Models. arXiv:2604.06996, April 2026. Judges passed failing tests far more often on code from their own model family.
9. Bjarnason, B. H., Silva, A., and Monperrus, M. On Randomness in Agentic Evals. arXiv:2602.07150, February 2026. Single-run pass@1 estimates vary by 2.2–6.0 points.
10. Cemri et al. Why do multi-agent LLM systems fail? arXiv:2503.13657, 2025. The MAST taxonomy: 14 failure modes in three categories (specification or system design, inter-agent misalignment, task verification), from traces of seven multi-agent frameworks; inter-annotator Cohen's κ 0.88.
11. Hora and Robbes. Are coding agents generating over-mocked tests? An empirical study. *MSR*, 2026. arXiv:2602.00409. 23% of coding-agent commits changed test files, against 13% of other commits; 36% against 26% added mocks.
12. Chen et al. Rethinking the value of agent-generated tests for LLM-based software engineering agents. arXiv:2602.07900, 2026. Changing how many tests agents wrote did not significantly change outcomes on SWE-bench Verified; it changed process and cost.
13. Wan et al. From runnable to shippable: multi-agent test-driven development for generating full-stack web applications from requirements. *ASE*, 2026. arXiv:2605.17242. With capable models, test-driven development raised accuracy by 15.5–23.7 percentage points.
14. Mills, H. D., Dyer, M., and Linger, R. C. Cleanroom software engineering. *IEEE Software*, 4(5):19–25, 1987.

Reporting follows the ACM SIGSOFT Empirical Standards for experiments and registered reports, and the CONSORT 2025 items that transfer to a paired comparison of software processes.

Appendix A Evaluation and blinding protocol

All ten pairs ran unattended in parallel on freshly imaged cloud hosts under a pre-registered protocol fixed before Pair 1 launched:

1. **Parallel isolation and leak checks.** Each arm ran inside its own OS user account with a repository-scoped token reaching only its private Tourbot replay repository. Both arms executed simultaneously on cloned 8-vCPU / 16 GB Ubuntu 24.04 droplets stripped of the other arm's credentials. After completion, an automated leak scanner searched every session stream and sub-agent rollout for the original issue number, shipped PR number/branch, and fix commit hash; all twenty trials passed with zero leaks.
2. **Deterministic replay preparation.** A preparation script branched each replay repository at the first parent of the shipped fix, preserved CI scoped to replay branches, waited for base CI to turn green, and filed the issue using only the

exact title and body present when it was first labelled `ready-for-agent`. Any trial ending without a pull request from its own process failure counted as a loss (`I2` and `I10`); external outages or incomplete telemetry would void and re-run the pair (zero pairs were voided).

3. **Blinding, formatting and execution evidence.** A packet builder stripped Git history, branch names, commit messages, PR descriptions, plan files, and all agent configuration directories, formatted modified `.php` files across `base/`, `X/` and `Y/` with `php-cs-fixer` (`3.95.27`, `@PSR12`), and assigned random `X`/`Y` labels via `random.SystemRandom().shuffle`. Inside an unprivileged `bwrap` grading sandbox, PHPUnit ran both cross-test suites (`X`'s tests on `Y`'s code and `Y`'s tests on `X`'s code) alongside scoped Infection mutation testing on changed `src/` classes.
4. **Dual-order model grading and sealed human review.** Both Claude (`claude-opus-5-5`) and Codex (`gpt-6.1-sol`) judged every graded pair twice - in presentation order `[X, Y]` and `[Y, X]` - grounded in the issue text, CI outcomes, cross-tests and Infection output. Any order flip cancelled to a `tie`, and a pair counted as a process win only when both graders agreed. The author evaluated the same formatted X/Y diffs blind through a write-once form showing no model grader output before any verdicts or mappings were unsealed.
5. **Pre-registered two-stage decision rule.** Stage 1 enrolled 10 pairs (`8+` Candidate wins adopts; `≤ 5` rejects; `6–7` advances to 8 additional shortlisted issues requiring `≥ 13/18`), coupled with strict veto caps rejecting adoption if Candidate correctness failures exceed Baseline or if the Candidate's median cost ratio or median time ratio exceeds `2.0×`. Both per-pair totals (`$223.71` vs `$13.39`; `1,095` vs `125` minutes) and exact order-statistic intervals (`6.9–23.5×` cost; `3.8–12.7×` time) lay wholly above the `2×` cap.

Appendix B Glossary

- **Process (arm).** One of the two ways of implementing an issue: the Baseline (`thirdshift`'s Built-in Ticket run) or the Candidate (the multi-agent maker-checker TDD and cross-family review workflow).
- **Pair.** One issue implemented independently by both arms from the same parent commit.
- **Replay.** A closed issue filed again with its pre-fix `ready-for-agent` text on a branch at the parent commit before its real fix merged.
- **Cross-test.** One arm's added or modified tests transplanted onto the other arm's implementation and executed through PHPUnit.
- **Grader.** A model-based evaluator or human reviewer comparing a blinded pair of pull requests (`X` and `Y`).
- **Win, loss, tie.** A pair goes to an arm only if both model graders prefer it on net across both presentation orders. A run ending without a pull request due to its own process rules counts as a loss. All disagreements, flips and equal ratings are ties.
- **thirdshift.** The open-source agent orchestrator running both processes; a command executes one run cleanly in an isolated Git worktree, and a custom process defines a user-supplied sequence of steps.

For AI agents: agents.md provides the study's operational rules, verified metrics, checklist, and self-test with answers.