Controlled disagreement: automated adversarial review with a second coding assistant in software development

Experience report

Nicolas Rocchia
San Francisco, Córdoba, Argentina
nicolasrocchia@gmail.com

July 2026. Author's translation of the Spanish original (DOI 10.5281/zenodo.21633496).

Abstract

This experience report documents a development methodology operated for 52 days spread over eight calendar weeks across five professional projects: a coding assistant (Claude Code, which executed two Anthropic models over the period) generates plans and implementations; a second assistant from a different provider (OpenAI's Codex CLI), whose underlying model was not recorded and may have varied, attacks them under an adversarial prompt; the first is instructed to verify every finding against the repository; and each review event terminates when every finding has been resolved, refuted with evidence, or explicitly transferred to the developer (the transfer terminates the event; the finding may remain open), a cut-off criterion here called *controlled disagreement*. The report explains why the design is plausible (documented limits of self-review, error diversity, execution as arbiter), how it is operated (mechanism, minimal adoption recipe), and narrates one real complete flow end to end, with the prompt, the reviewer's 20 findings, their disposition, and the available timings and token consumption. Two intervention mechanisms were observed: the detection of existing defects, and design shaping at the plan gate; both are illustrated with traceable cases, together with the false positives, the defects the method did not detect, and a regression introduced by a fix. On the auditable subset of the corpus, at least 64.9% of findings were incorporated and 3.6% were refuted with evidence. The available, reconciled accounting (91 documented gate events, units, coverage universes and reconciliations) is presented in the appendices, together with an operational recipe and a dated reference instantiation. Causal attribution of the benefit to provider diversity, and the net cost balance, remain explicitly hypotheses: no controlled comparison was performed, and the pending experiment is specified.

1 Introduction

The adoption of coding assistants based on large language models (LLMs) has spread rapidly; their verification remains an open problem. A common answer, asking the same model to review its own output, has limitations documented in the literature, which motivate exploring external reviewers.

This report documents an alternative operated in real commercial development: automated adversarial review by a second assistant from a different provider, with the instruction to verify every finding against the repository and the developer as final owner. The purpose of the report is not to inventory defects: it is to document an operable methodology (in the minimal sense that it was applied across the 91 documented gate events), with its rationale, its mechanism, a minimal adoption recipe, one real flow narrated from the inside, and its observed behavior, including its failures.

To avoid the most frequent confusion in texts of this kind, three claims of different epistemic status are separated: (a) the claim that intrinsic LLM self-correction has limits is supported by the literature; (b) that a second assistant detects defects the generator missed, and that its criticism of the plan modifies the design before implementation, are observed facts documented in the reported operation, from which frequencies and examples are drawn; (c) that *provider diversity* is the cause of that detection, and that the net balance is favorable, are hypotheses: the controlled comparison (same workflow with a same-provider reviewer, or with self-review) that would support them was not performed. The center of gravity of the contribution therefore lies in the mechanism (plan-gate review and the termination rule by individual disposition), not in the identity of the reviewer: the different provider was a convenient implementation of external signal, not an isolated, proven lever.

2 Why it is plausible that it works

The design rests on four pieces of evidence, best read as an argument rather than a catalog.

The engine: the self-review blind spot. In reasoning tasks, *intrinsic* self-correction (without external signal) does not improve reliably and can degrade: on a sample of 200 GSM8K problems with the August 2023 GPT-4, accuracy fell from 95.5% to 91.5% after one round of self-review and to 89.0% after two [3]. The bottleneck is detection: models fail to locate logical errors but correct them when the location is pointed out [4]. And there is a specific blind spot: errors a model does not correct in its own output it does correct when they are presented as external input (measured with errors injected in a controlled way into open models) [5]. That finding provides evidence compatible with the mechanism the workflow exploits: the defect the generator does not see in its own change set is visible to a reviewer receiving it from outside. Complementarily, there is initial evidence of a self-preference bias in LLM evaluators [6]: the generator is a partial judge of itself. Three scope caveats: these results come from reasoning and text evaluation, not code review; they justify the need for external signal, not that it must come from another provider; and techniques exist that improve self-correction on specific tasks. The defensible claim is that self-review does not constitute an *independent* control.

The multiplier: error diversity. The intuition comes from ensemble methods: under conditions involving individual competence and the combination mechanism, error diversity contributes to ensemble improvement [1]; modern theory formalizes it as a trade-off, not an unconditional gain [2]. In general generation, combining heterogeneous models can beat the best individual one: Mixture-of-Agents reached a 65.1% length-controlled win rate on AlpacaEval 2.0 against 57.5% for GPT-4o using only open models [8], and multi-agent debate reduces errors on reasoning benchmarks [7], although it does not consistently beat same-model sampling techniques at equal budget [3]. The workflow described here does not average or vote: it transfers the diversity intuition to a sequential *detection* scheme over a single implementation, a configuration that theory does not directly cover.

The neighbors: code review with agents. The generator-reviewer structure with LLMs is not new and this report does not propose it. CodeAgent organizes multiple communicative agents with a supervisor for code review [9]. PairCoder, the closest conceptual neighbor, separates a Navigator agent (planning and feedback-driven direction) from a Driver agent (implementation), with relative pass@1 improvements of 12.00% to 162.43% over direct generation on benchmarks [10]. And the reexamination of self-debugging with self-generated tests, on self-contained Python tasks, shows that these introduce bias (false-negative labels that lead the model to "fix" correct programs), exactly the class of oracle problem this report discusses [11]. Against that line, evaluated on benchmarks, the differential contribution of this report is to document the sustained automation of the structure on real professional projects, the review of plans in addition to code, and a termination rule based on the individual disposition of findings. Three works contemporaneous with the close of this report delimit the space further. A controlled experiment with the same provider pair (116 LiveCodeBench tasks, six writer-reviewer conditions) found that the value of cross-model review is asymmetric and depends on the capability gap between the models: reviewing the weaker model's drafts improved correctness by up to 18.1 points, while the weaker reviewer worsened the stronger model's drafts, with harmful reviews concentrated in rewrites that discard a working solution and helpful ones in local edits [12]. That result does not evaluate this report's architecture, but it supports two of its design decisions: here the reviewer does not rewrite (it proposes findings with evidence that the generator verifies and a human arbitrates), and the plan gate acts before code exists, a setting that experiment does not cover. In the same direction, a reviewer-critic protocol documented a false-consensus failure mode (agents converging on agreement without sufficient evidence) and concludes that trustworthy cooperation requires structured, evidence-grounded disagreement [13]; and a staged adversarial methodology with a cross-model critic, motivated by the precision crisis of LLM-generated reports, records that its most instructive failure was eliminated by a single empirical test rather than by better adversarial framing [14], consistent with the role of oracles in this report.

The limit: the N-version lesson. Reliability through implementation diversity has its antecedent in N-version programming [15], and its history contributes the central limit. Knight and Leveson subjected 27 versions developed independently from one specification, at two universities, to one million test cases, and statistically rejected failure independence: different people misread the same specification in the same way [16, 17]. The experiment was revisited in 2026 with coding agents, varying model, agent and language: 429 test cases with coincident failures where independence predicted 115.36, across 48 configurations, together with substantial failure reductions through voting [18]. The design consequence: any decorrelation between assistants must be assumed partial, the specification can constitute a common source of error, and the final decision on product intent requires an authority external to the pair of assistants. That is why the workflow terminates in a developer, not in a consensus.

3 The method

3.1 Roles and gates

Fixed roles that do not mix: a main assistant (generator) plans and implements; an assistant from another provider (reviewer) attacks under an explicitly adversarial prompt; real execution (compilation, tests, functional runs) arbitrates everything it can arbitrate; the developer owns the result. In the reported instantiation the roles are embodied as follows: Claude Code is the orchestration environment; the Anthropic model active in the session is the generator, which plans, implements, verifies findings and reconciles (in this report "orchestrator" and "generator" name that same operational unit); Codex, via CLI or plugin, is the reviewer; compilation and tests are a partial oracle; the developer is the final authority. The method involves two gates: over the implementation *plan*, before writing code, and over the resulting *diff* (the change set), before committing. The adversarial prompt asks the reviewer not to approve but to attack: functional errors, regressions, concurrency, security, persistence, domain assumptions, citing impact and evidence per finding. The orchestrator is instructed to verify every finding against the repository before consolidating the result; the corpus contains cases where that verification worked, one failure of finding verification (the authorization taken as verified, Section 5.1) and a later failure of diagnosis and validation of a fix, not of the finding (the regression of Section 5.5, first attributed to another cause); these are distinct categories.

3.2 The cut-off rule

The review event does not terminate in consensus: agreement between assistants that share training data does not make a conclusion true, and both can share the same misreading of an ambiguous specification [16, 18]. The event terminates when every finding has been (i) resolved, (ii) refuted or discarded with verifiable evidence, or (iii) explicitly transferred to the developer; in case (iii) the event terminates, but the finding may remain open. Transfer (iii) has terminal states of its own, observed in the operation: *accepted debt* (recorded with identifier and justification), *accepted risk* (explicit developer decision), *partial mitigation* (part applied, remainder to debt) and *pending escalation* (transferred without a recorded decision). The generator proposes and records the operational closure of (i) and (ii) with verifiable evidence (file-and-line citation, measurement, or documented environment behavior); the developer retains review authority and final responsibility over all closures. The states of accepted debt, accepted risk and partial mitigation require an explicit developer decision; a pending escalation requires a recorded transfer and an assigned owner, but constitutes neither a decision nor a closure of the finding: it is an open transfer, and the corpus contains one (the cost objection of Section 3.4). In actual operation the norm was a single reviewer run per gate (resumption, meaning reusing the reviewer's session, appears 3 times in the whole corpus, always to confirm a fix); the adopted cut-off signal is "decreasing severity and repetition over the same file", a heuristic that emerged from a single episode (Section 6), not a validated criterion.

3.3 Minimal adoption

The full instantiation (Appendix A) is one possible implementation, not a requirement. The minimal recipe: (1) choose a reviewer assistant separate from the generator (to reproduce the reported configuration, from a different provider); (2) define an explicit activation policy and submit the plan of the work units that meet it, and then their diff, to the reviewer under an adversarial prompt (the policy of the reported operation: Section 4.1 and Appendix C); (3) have the generator verify every finding against the repository before presenting the result; (4) terminate each event when every finding has been resolved, refuted with evidence, or explicitly transferred to the developer, never by consensus (the transfer terminates the event; the finding may remain open); (5) keep a log of verdicts, which calibrates the process and is the memory that survives history retention.

3.4 Escalation and human decision

Three documented escalations illustrate the division of responsibilities (they are not a census): the notification deduplication semantics was decided by the developer; an operating-cost objection remained explicitly unresolved in the session; and in P5 the reviewer issued a global negative verdict with a major objection that the developer decided to keep for product reasons, leaving a record in the project memory. This last case documents the principle in its most uncomfortable form: the last word is human even against the reviewer.

4 One complete flow from the inside

So that the mechanism does not remain abstract, this section narrates one real complete flow (a work unit that went through the plan gate, the implementation, and the diff gate), reconstructed from the session histories and their persisted artifacts: feature F6 of P1 (an inspection screen with georeferenced evidence, notifications and a digital sworn declaration), July 20, 2026.[1]

[1] All quotations from prompts, session logs and the calibration log are the author's translations from the Spanish originals; the configuration blocks in Appendix C are reproduced untranslated.

4.1 The decision to run the round

Gates are not fired blindly. With the plan written, the orchestrator asked the developer, verbatim: "F6 crosses the high-impact bar (new API-Web contract + anti-IDOR authorization + inspection domain). Shall I run the Codex red-team round on the plan before closing it?", with the recommendation: "It costs tokens and a few minutes. Historical calibration: it systematically catches auth/concurrency/consumers I did not look at" (in the log's jargon: it recurrently finds defects of authorization, concurrency, and unreviewed consumers). The developer approved the round and, in the same step, took a product decision the plan needed (the deduplication semantics: "only the first" notification per anomaly). The activation policy is formalized in the orchestrator's global instructions in force at the close of the period (reproduced in Appendix C): a round is proposed when the unit crosses a high-impact bar defined by at least one of five criteria (contract or API change between components; database schema change or migration; security, authentication or authorization; concurrency, race conditions or transactions; logic that is hard to revert or business-critical), or when there is real uncertainty (viable approaches with no clear winner, low confidence reading the code, an ambiguous requirement that is expensive to get wrong); it is skipped for local, reversible, trivial or template changes; and every run requires the developer's explicit approval, never automatic. Human approval governs the logical activation of the event; the permanent permission in the configuration (Appendix C) removes only the tool's second technical confirmation, and the separation is procedural, imposed by the orchestrator's instructions, not by a technical control. Two limits: the change history of that instruction file over the period was not reconstructed, and how many eligible units were not reviewed was not recorded, so the 91 gate events document the reviewed activity, not all activity in the period.

4.2 The prompt

The orchestrator wrote the prompt to a file (deliberately without Spanish accents, for encoding robustness through the shell) and launched the reviewer. The header, translated:

```
WARNING - NO SANDBOX: do NOT
write/edit/delete/move files or run
state-modifying git.  ONLY read and
analyze.

Act as a RED-TEAM of this implementation
plan:  attack assumptions, security
holes, risks and edge cases.  Do NOT
approve it - look for what is wrong
or missing.  The code is in this repo:
verify the plan's anchors against the
real code before asserting findings, and
cite file:line.
```

There follow some fifteen lines of system context, the complete plan in nine phases, and seven specific attack requests (authorization and IDOR, insecure direct object references; translatability of the queries to SQL; contract and consumers; design flaws in the deduplication; field loss in error chains; domain edge cases; "and anything else the plan assumes wrongly or lacks"). Closing: a numbered list of findings with severity, file:line evidence, and what the reviewer would change in the plan.

4.3 The run and the 20 findings

The reviewer (model gpt-5.6-sol, high reasoning effort, Codex CLI v0.144.3) explored the repository for 12 minutes 22 seconds and consumed 283,638 tokens according to its own record. After the run, the orchestrator verified with `git status` that the repository was intact (the reviewer runs without technical isolation in this environment). The result: 20 findings (7 high severity, 11 medium, 2 low), each with file:line evidence. Four examples showing the range:

The finding the author considered most valuable (high, incorporated). The planned deduplication was global per type and entity; the reviewer showed that this silences legitimate alerts: "an employee added later will never receive the alert; a jurisdiction change will be silenced by the previous one's rows". It proposed deduplicating per recipient. The orchestrator verified it and incorporated it into the plan: "it is a real improvement at almost no cost".

The legal finding (high, incorporated with replacement of the original design). The plan fell back to a generic confirmation dialog if the sworn-declaration text did not reach the client. The reviewer objected at the root: "it would be recording acceptance of a text that was never shown"; functional, but it compromised the evidentiary value of the acceptance: a legal risk that justified failing safely (*fail closed*). The plan moved to that behavior: without the text, completion is not allowed and no acceptance is sent.

The transfer to technical debt (high). The reviewer showed that deactivating a user does not always revoke their access, through a legacy link. The orchestrator's verification confirmed the problem and its scope: "the same resolver is used across the portal's authorization; F6 inherits the pattern without worsening it; changing it here would exceed the scope". It was recorded as technical debt with an identifier, not ignored.

The refutation (high, refuted with evidence). The reviewer objected that an allow-list of fields at the web layer "does not constitute a privacy boundary" because the role can call the API directly. Verification showed that the role already accesses the full record legitimately through another official path: the list is hygiene for what is embedded in HTML, not an authorization boundary. Refuted, with the purpose note added to the plan so the discussion does not repeat.

4.4 Disposition and closure of the plan gate

The disposition was recorded in the hardened plan, with one declared ambiguity: 11 findings incorporated (the commit message says eleven; a session entry classifies the partially mitigated finding as incorporated, which would give twelve; the floor is reported), 4 transferred to the debt register with justification (all pre-existing or cross-cutting), 4 refuted with evidence, and 1 partially mitigated (mitigation applied, remainder to debt). A single run; no resumption. Processing (reading, verifying the 20 against the code, rewriting the plan) took about 6 minutes; the developer approved the hardened plan and the implementation took 33 minutes with automated tests and a successful smoke test.

4.5 The diff gate

Before committing, the project's pre-commit validation flow ran three steps: the orchestrator's internal review (5 minor findings, fixed), the security review (clean), and the external reviewer's round over the working tree, this time via the plugin (model gpt-5.5: direct evidence that the two invocation paths can execute different models). Verdict: "needs-attention", 2 findings. The high-severity one produced the exchange the author considered most valuable in the flow: the reviewer held that the version guard of the sworn declaration, incorporated at the plan gate, could block a future defective client in the field. The orchestrator first defended it ("the only current client that accepts always sends the version"), verified, and then recognized the real risk: "if an application ships with a defect that does not send the version, the failure would occur on completion, leaving the field operator blocked". The closure adopted the reviewer's recommendation, redesigned as graceful degradation: without a version and with the control disabled, the work completes but the declaration is not anchored ("the anchor must never point to a text the operator did not see: an acceptance without an echoed version [returned by the client] is evidence without probative value"), and with the control active, the standard structured error. The second finding turned out to be a pre-existing gap of another feature, verified and documented in the debt register. The fix for the first was applied with its test renamed and a new case; successful tests, nine code commits whose own messages carry the result of the two rounds.

4.6 The flow's numbers and what was not recorded

The complete flow, from the plan request to the last commit, took 2 hours 57 minutes, with this chronology (UTC times from the histories): plan writing, 19:00 to 19:24 (24 minutes); human decision to run the round and deduplication semantics, 2 minutes; plan-gate window, 19:27 to 19:42 (15 minutes: one failed launch due to a syntax error, the retry, and 12 minutes 22 seconds of effective reviewer execution); verification of the 20 findings and approval of the hardened plan, 6 minutes; implementation, 19:48 to 20:21 (33 minutes); tests, smoke and the first two steps of the pre-commit flow, 20:21 to 21:48 (about 87 minutes); diff run, 5 minutes; verification, final fix and nine code commits, 21:53 to 21:58 (5 minutes). The accumulated effective execution of the two reviewer runs was about 17 of the flow's 177 minutes. Pieces explicitly not recorded in the histories: the internal prompt the plugin builds for its review, the second round's tokens, and the repository state check after that second run. One flow, moreover, is not all of them: this one had two gate events, four findings refuted with evidence at the plan gate (Section 4.3 shows one) and no refutation at the diff gate; the corpus contains events with stale-context findings and with zero findings (Section 5).

5 What was observed in the corpus

The complete operation (91 gate events across five projects; units, universes and reconciliations in Appendix A) showed two intervention mechanisms of different nature: *detection* corrects something that already existed; *shaping* modifies the design before it materializes. Shaping acts when changing is plausibly cheaper; neither that cost advantage nor the superiority of the resulting designs was measured in this experience, and everything that follows must be read accordingly. With the model timeline of Appendix A: detection cases 1 and 3 ran with Opus 4.8 as generator, and the diff-gate finding of the traced flow (Section 4.5) with Fable 5; the model of case 2 is not attributable with certainty because its session did not survive. Of the five plans with explicit counts, four ran with Fable 5 and one with Opus 4.8. Both mechanisms were thus observed under both orchestrator models. On the auditable subset of the corpus (the 35 events from surviving sessions with numeric records), at least 64.9% of findings were incorporated (180 of 277) and 3.6% were refuted with evidence (10 of 277); the computation rules, the exclusions and a consistency check are in Appendix A.4.

5.1 Detection: three cases

Charge without receipt (P2). At the diff gate of an issuance workflow against a third-party SOAP service, the reviewer detected that a previous fix by the orchestrator itself left the system answering success (HTTP 200) with the SOAP call failed: the system confirmed the charge of an operation whose receipt was never issued. The case is compatible with the hypothesis that an external signal can reveal errors missed during the generator's own review, without this experience having measured probabilities.

Domain error (P3). The steel profile catalog contained 0.88 kg/m for the 12 mm solid square, the value of the round profile; the correct one is 1.13 kg/m, with the conventional steel density of 7,850 kg/m^3. The code was correct; the datum was false. The reviewer flagged the value as inconsistent; the fix was validated through two paths independent of the code (the reference product's weight fell within the physically expected range, and the catalog was checked against manufacturer data). A test with an oracle independent of the catalog could have detected it; verifications whose oracle derives from the same wrong datum cannot.

Authorization verified wrongly (P1). The reviewer detected an object-level access control failure in a validation the orchestrator had taken as verified: the second opinion corrected the verifier, not the code.

5.2 Design shaping

Beyond the flow narrated in Section 4 (11 of 20 plan findings incorporated; a lower bound whose ambiguity is explained in Section 4.4), the plan events with explicit counts record 7 of 8, 8 of 8, 14 of 14 and 15 of 15 findings incorporated; one of them ended with the generator's original design replaced entirely, and another hardened a destructive database migration (validation of the index's real definition, prior duplicate check, and safe cancellation of the rollback) before touching production. No total corpus proportion is aggregated: several events record non-numeric counts, and their formalization is part of the pending artifact (Section 10). The calibration log records the author's contemporaneous perception ("the plan round was the most valuable; the diff rounds validated"); it is an appraisal, not a metric. Shaping is counterfactual by nature: it cannot be proven what defects the original design would have had. The corpus documented by accident a longitudinal observation, the only one available on the later consequences of a decision taken at the gate: a semantics accepted as a product decision at a plan gate reappeared five rounds later as a blocking defect, when a new interface made the contradiction visible. What the gate let through had an expiration date; no longitudinal follow-up of the rest was performed.

5.3 False positives and a context failure mode

Hard refutations were documented: nine outside the traced flow, four inside its plan gate (Section 4.4) and one additional case of ambiguous classification (the counting rule is discussed in Appendix A.4); no rate is computed because numerator and denominator come from different universes (Appendix A). Two refutations share a mechanism that is not a reasoning error but a *context* one: the reviewer objected against a stale version of its own prompt (a time window already changed by a later product decision; a "missing" migration that had existed for weeks and that it searched for in the diff, where it could not be). Adopted mitigation: regenerate the reviewer's prompt after every intermediate decision. The failure mode is a direct consequence of a design constraint documented in the instructions (Appendix C): the reviewer sees the repository but not the orchestrator's conversation, so all context reaches it, and only reaches it, through the prompt. Refutation also ran in both directions: the reviewer likewise refuted hypotheses of the orchestrator on at least four documented occasions.

5.4 What the method did not detect

The documented undetected defects follow a pattern: almost none is logic of the reviewed diff. (i) External contract: five differences between what was planned and the real XML of a third-party service, unverifiable with the artifacts available in the repository, revealed only by the real example. (ii) Execution environment: a missing declaration in the Android manifest that broke the camera on every device; it was detected solely by the smoke test on an emulator, after passing the reviewer and three internal reviews. (iii) Real system state: an integration test after five sprints revealed 23 findings no review had. (iv) Product decisions, which require human authority and responsibility. This pattern enables a rival hypothesis worth confronting head-on: that the escaped defects betray missing tests rather than missing reviewers. This experience's answer is that both claims are true and do not compete, because static review and real execution act on different defect classes and at different moments: tests cannot review a plan (the gate where the incorporated findings concentrated acts before there is code to test), and the reviewer cannot execute the real environment. The case of Section 5.5 shows both faces at once: the regression was caught by a real device, and its enabling cause was the component's lack of tests, not the lack of review. In the observed configuration, both controls contributed at different moments and on different defect classes; this experience did not evaluate how much of the reviewer's contribution could be reproduced through stronger specifications or tests.

5.5 The regression introduced by a fix

The most instructive negative case: a valid reviewer finding (loss of local files when the application crashed) received a fix whose mitigation mechanism introduced a different regression (duplication), which survived the event's closure, was wrongly attributed to another cause by the orchestrator, and was detected by the developer on a real device two days later. The enabling cause: the component lacked tests because it depended on platform APIs. The lesson recorded in the calibration log: a fix

derived from a finding is also new code without coverage, and deserves its own event or its own tests before being closed. It is the only regression of this kind documented; others may exist unobserved.

6 Preliminary operational lessons

Operating hypotheses that emerged from the corpus, not established properties; several rest on single observations. (1) The episodes recorded as most valuable in the calibration log concentrated at the plan gate; the diff gates contributed mostly validation and point detection. This is reported experience, not a quantitative comparison. (2) The method produced no observable value in the single event over a mechanical refactoring already verified by execution. (3) A sequence of seven consecutive events over successive changes of one feature showed signs of diminishing returns: the seventh produced the only false positive of the series; the observation is single and does not permit causal attribution to repetition. (4) The reviewer's prompts expire with every intermediate product decision. (5) Discards and deferred decisions have expiration dates. (6) The calibration log versioned next to the code proved indispensable: it is the memory that survived history retention, and the eleven events that were not recorded contemporaneously required later reconstruction.

7 Organizational proposal (not evaluated)

Separately from the reported experience, the author proposed to his organization a protocol that extends the principle to the full lifecycle (requirements capture with consented audio, compared architecture with cross attack and decision record, the two gates described, and testing with pre-existing controls intact), with criticality levels and data governance rules. It is under discussion for a pilot: it is a design proposal informed by the individual experience, not an evaluated organizational practice.

8 Limitations

(1) *Design*: without a controlled comparison, causal attribution to provider diversity is a hypothesis; "family" is used in its weakest operational sense (different provider, no known shared distillation lineage). (2) *Data*: counts from histories generated by the system itself and a log written by the author; part of the period reconstructed retrospectively; a single practitioner, specific domains. (3) *Oracles*: execution arbitrates only what has an independent oracle; compiling and passing tests written from the same shared specification does not guarantee correctness. (4) *Net balance*: the hard refutations of Appendix A.4 and a single fix-induced regression were documented, but refutation time and total economic cost were not measured; benign degradation through coincident omission holds only for omissions: review-induced changes can worsen the baseline, whether through an incorrect finding inducing a harmful modification (not documented in this corpus) or through a badly implemented fix of a valid finding (this corpus contains one case). (5) *Extraction*: self-reported mining without systematic manual audit or formalized deduplication rules; the orchestrator's model varied and its timeline was reconstructed per session (Appendix A), although not every run is attributable to one model; the reviewer's model is recorded only where its output persisted it (in the traced flow: gpt-5.6-sol and gpt-5.5, different between the two invocation paths). The report does not describe the contractual conditions of code processing by the providers. (6) *Residual correlation*: the models share training data and the evidence indicates failure dependence even when varying model, agent and language [18]; the pattern of undetected defects (external contract, real environment, product) is consistent with limits that no assistant diversity removes.

9 Future work

First, prospective instrumentation and an anonymized replication artifact: record per event the findings, verdicts, the effective model of each assistant, refutation time and final destination, and publish the complete outcome matrix per gate and category, with the taxonomy, the prompts, the mining procedure and a manually audited sample; the total of valid findings, not aggregated today, is its first piece. Second, the experiment the causal thesis requires: one same change set under four conditions (no review, self-review, same-provider reviewer, different-provider reviewer), with independent oracles, measuring detection and shaping separately. A first version of that experiment on self-contained tasks now exists [12], with solo, self-review and both cross-review conditions; repository scale, the plan gate and the separation of mechanisms remain pending: it is conceivable that reviewer diversity matters differently in each mechanism, and the cited literature provides only indirect evidence, and only about the first. Third, the organizational pilot, if its discussion prospers, with these metrics from day one.

10 Conclusion

In 52 days of real operation across five projects, automated adversarial review by a second assistant operated through two mechanisms. Cases of detection of defects missed or introduced by the generator were documented, including domain and authorization errors; separately, the available enumeration records thirteen findings refuted with

evidence (nine outside the traced flow and four inside; provenance and limits in Appendix A.4) and one additional ambiguous case, while the total of valid findings was not aggregated (per-event records contain non-numeric entries) and constitutes the first piece of the pending artifact. And it shaped the design at the plan gate, in one case going as far as replacing the proposed architecture entirely; whether those designs were objectively superior cannot be determined with this experience. The documented undetected defects concentrated in contracts, states or oracles not available to static review of the change set, and in product decisions that require human authority. The termination criterion by individual disposition and evidence, instead of consensus, was applied across the 91 documented gate events; no aborts were identified among the accounted events, although the procedure did not systematically reconstruct attempts started and not recorded; and the most instructive case in the corpus was a regression introduced by the fix of a valid finding, a reminder that the method does not exempt from an independent test coverage. Whether provider diversity is the cause of the observed detection, and whether the net balance is favorable once all costs are counted, are questions this experience motivates and leaves open; the experimental design to answer them is specified.

References

[1] L. K. Hansen and P. Salamon. Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(10):993–1001, 1990.

[2] D. Wood, T. Mu, A. M. Webb, H. W. J. Reeve, M. Luján and G. Brown. A unified theory of diversity in ensemble learning. *Journal of Machine Learning Research*, 24(359):1–49, 2023.

[3] J. Huang et al. Large language models cannot self-correct reasoning yet. In *ICLR*, 2024. arXiv:2310.01798.

[4] G. Tyen et al. LLMs cannot find reasoning errors, but can correct them given the error location. In *Findings of ACL*, pages 13894–13908, 2024. arXiv:2311.08516.

[5] K. Tsui. Self-correction bench: Uncovering and addressing the self-correction blind spot in large language models. Preprint, 2025. arXiv:2507.02778.

[6] A. Panickssery, S. R. Bowman and S. Feng. LLM evaluators recognize and favor their own generations. In *NeurIPS*, 2024. arXiv:2404.13076.

[7] Y. Du, S. Li, A. Torralba, J. B. Tenenbaum and I. Mordatch. Improving factuality and reasoning in language models through multiagent debate. In *ICML*, PMLR 235:11733–11763, 2024. arXiv:2305.14325.

[8] J. Wang et al. Mixture-of-agents enhances large language model capabilities. In *ICLR*, 2025. arXiv:2406.04692.

[9] X. Tang, K. Kim, Y. Song, C. Lothritz, B. Li, S. Ezzini, H. Tian, J. Klein and T. F. Bissyandé. CodeAgent: Autonomous communicative agents for code review. In *EMNLP*, pages 11279–11313, 2024. arXiv:2402.02172.

[10] H. Zhang, W. Cheng, Y. Wu and W. Hu. A pair programming framework for code generation via multi-plan exploration and feedback-driven refinement. In *ASE*, pages 1319–1331, 2024. arXiv:2409.05001.

[11] X. Chen et al. Revisit self-debugging with self-generated tests for code generation. Preprint (work in progress), 2025. arXiv:2501.12793.

[12] Z. Xiang, Y. Zhang, Y. Zhang and H. Xu. Cross-model LLM code review: Should you use Claude to review Codex or vice versa? In *Agentic Software Engineering (SE 3.0) Workshop*, 2026. arXiv:2607.21656.

[13] E. S. Qiu and J. Gill. Adversarial review: Cooperative code review through structured disagreement. In *AI4GOOD Workshop, ICML*, 2026.

[14] A. Agarwal. Refute-or-promote: An adversarial stage-gated multi-agent review methodology for high-precision LLM-assisted defect discovery. Preprint, 2026. arXiv:2604.19049.

[15] A. Avižienis. The N-version approach to fault-tolerant software. *IEEE Transactions on Software Engineering*, SE-11(12):1491–1501, 1985.

[16] J. C. Knight and N. G. Leveson. An experimental evaluation of the assumption of independence in multiversion programming. *IEEE Transactions on Software Engineering*, SE-12(1):96–109, 1986.

[17] J. C. Knight and N. G. Leveson. A reply to the criticisms of the Knight & Leveson experiment. *ACM SIGSOFT Software Engineering Notes*, 15(1):24–35, 1990.

[18] J. Ron, B. Baudry and M. Monperrus. N-version programming with coding agents. Preprint, 2026. arXiv:2606.20158.

A Operational data

This appendix concentrates the operation's accounting: units, coverage universes, reconciliations and provenance. It exists so that the body's claims are auditable; its limits are declared in Section 8 of the body.

A.1 Context and provenance of the sources

The experience corresponds to a single developer (the author), between June 2 and July 23, 2026 (52 days, eight calendar weeks), across five .NET and TypeScript projects of varying criticality, identified as P1 to P5: a traceability platform for the agro-industrial sector, in production (P1); a commercial web application with SOAP integration to third-party services (P2); a public technical calculation application, in production (P3); a management system for a sports institution (P4); and a membership management prototype (P5). **Confidentiality note**: the projects are not identified by name and the details that would identify third-party systems or organizations were generalized; the

operational data (counts, dates, verdicts) was not altered.

Sources: (i) mining of 345 session transcript files of the orchestrator (JSONL, defensively parsed) plus persisted session artifacts (the reviewer's prompts and outputs); (ii) a calibration log recording findings, verdicts and refutations, written contemporaneously at the close of each event for 54 rounds of P1; another 11 P1 events were not recorded at the time and were reconstructed later from the histories. Default history retention is 30 days: sessions prior to June 16 no longer existed at the time of analysis (a single exception from June 12, with no reviewer runs); for that period the source is the log. The mining report was generated by the orchestrator itself under the explicit rule of answering "not found" when data was absent; its self-reported nature is discussed in the body's Limitations.

A.2 Instantiation and model timeline

Orchestrator: Claude Code (Anthropic), which executed two models over the period. Timeline reconstructed from the model field of the surviving histories: from June 16 to July 1 every session records Claude Opus 4.8, a period coinciding with the global suspension of Claude Fable 5 (launched June 9, 2026, suspended on the 12th and restored on July 1); the return was recorded inside a session started on June 30, with 421 Opus 4.8 messages followed by 71 Fable 5 ones. From July 2 to 23 Fable 5 predominates, with documented exceptions: a July 7-8 session that returned to Opus 4.8 mid-work, three mid-July sessions with messages from both models, and the single P5 event (July 23) executed entirely on Opus 4.8. For the June 2-15 events there are no surviving sessions recording the model, and those before June 9 necessarily ran on a model prior to Fable 5's launch. No claim in this report distinguishes effects by model.

Reviewer: Codex CLI (OpenAI), integrated through the official plugin `openai/codex-plugin-cc`. Direct runs with `codex exec` under a permanent auto-approved permission; five runs via the plugin's commands, corresponding to five events with one run each (configuration and instructions reproduced in Appendix C). Both paths share the adversarial prompt, but execute potentially different models (in the flow traced in the body: gpt-5.6-sol via CLI and gpt-5.5 via plugin) and are aggregated here descriptively. The reviewer was instructed not to modify the repository and the Git state was inspected after the runs; this control detected one incident (line-ending rewrite of five snapshot files, content intact, reverted) and does not constitute complete technical isolation: it does not prevent temporary writes, effects on ignored files, or network access.

Project	Events	Period	Runs
P1	54 + 11	02/06 to 22/07	48 + 5
P3	11	01/07 to 17/07	14
P4	10	15/07 to 17/07	10
P2	4	16/06 to 08/07	4
P5	1	23/07	1

Table 1: Gate events per project (dates in 2026, day/month format). In P1, 54 log rounds plus 11 events reconstructed from sessions; runs via CLI plus plugin, counted only in surviving sessions; the distribution assigns 77 of the 79 direct runs.

A.3 Units, universes and counts

Units: a *run* is one effective invocation of the reviewer; an *event* (or *gate event*) is the application of one gate, plan or diff, to a work unit, and may consist of one or more runs; a *work unit* is a feature, fix or refactoring; a *complete flow* is a unit that went through both gates; a *round* is an entry in P1's calibration log. The 91 counts in this report are gate events, not complete flows; how many units went through plan only, diff only, or both gates was not aggregated and is part of the pending artifact. Universes: runs were counted over the surviving sessions (after 2026-06-16): the histories record 81 direct invocation occurrences, 79 unique after deduplicating two forked sessions; the per-project reconstruction assigns 77 of those runs, plus 5 via plugin (Table 1), leaving 2 direct runs without a project assignment. Events were counted over the full corpus (log plus sessions): 91, of which 65 in P1 (54 log rounds, which include events whose sessions no longer exist, plus 11 reconstructed) and 26 in the rest. There are more events than runs because events prior to June 16 lack surviving runs. The finding count, approximately 330, covers only the surviving sessions; its exact value depends on deduplication rules not yet formalized.

A.4 Quantitative observations

With their limits: (1) Documented hard refutations: nine outside the traced flow, four inside its plan gate, and one additional case of ambiguous classification. The counting rule was not formalized, and the source contains an internal inconsistency that is declared: the extraction artifact heads its refutation section with "9 or 10" while its own enumeration lists thirteen individual findings; earlier versions of this report propagated the header. The count reported here (nine plus four, thirteen per individual finding, plus the ambiguous one) comes from the enumeration, and its definitive audit is part of the pending artifact. No rate is computed for this due to universe asymmetry, and the "hard refutation" category is narrow: findings non-actionable through other paths (pre-existence, scope, product decision, stale context) were more numerous, and the

complete matrix per gate and category remains pending. (2) One gate event over a mechanical refactoring already verified by smoke test produced zero findings; a single observation. (3) Cost, as a rough indicator: runs with token records consumed between 58,000 and 284,000 total tokens according to the reviewer's own record (no input/output breakdown; the upper end corresponds to the traced flow's plan run, with high reasoning effort); false-positive refutation time was not measured. In the only flow with a complete chronology (Section 4.6), the wall time attributable to review (the two run windows plus finding verification and closure) was between 26 and 31 of the 177 minutes, 15 to 18 percent depending on whether the closure is imputed; it is a single observation, not a corpus measurement, and its translation into economic cost requires prices this report does not set. (4) The norm was one run per gate; resumption (reusing the reviewer's session) appears 3 times in the corpus, always to confirm a fix. The "passes" at the extremes (8 in one day over one feature; a series of 7 consecutive diff passes whose seventh produced the series' only false positive) were distinct gate events, with new runs over successive change batches, not resumptions nor repeated runs over the same diff, so they do not contradict the norm. (5) Rates over the auditable subset. To give a floor for the signal-to-noise relation with numerator and denominator from the same universe, the per-event rows of the mining report corresponding to surviving sessions with numeric counts were aggregated: 35 of that universe's 37 events. Rules: in the two rows with ranges, the numerator's floor and the denominator's ceiling were taken; two events with non-numeric records were excluded and are declared (one with 20 findings and a disposition recorded as "majority", another with 12 and an integrated disposition without a count); those noted as incorporated later or already covered were not counted in the floor. Result: 277 findings, at least 180 incorporated (64.9 percent), 10 refuted with evidence (3.6 percent); the remainder corresponds to transfers, accepted risks and evidence-based discards, whose complete matrix remains pending. Independent consistency check: the 10 refuted in this subset, plus the 3 documented only in the calibration log (outside this universe), add up to exactly the 13 of the enumeration in point (1). The difference with the approximate count of 330 corresponds mainly to the two excluded events and to non-formalized deduplication rules. It is a floor computed over the self-reported artifact, not an audit; it inherits its limits.

B Operational recipe

A synthesis for reproducing the workflow without reconstructing it from the narrative.

1. **Activation.** Define in writing which units are reviewed; the reported operation used the impact bar formalized in its global instructions (Section 4.1 and Appendix C).
2. **Plan prompt.** Explicit adversarial role with a prohibition on approving, system context, the complete plan, domain-specific attack requests, and an output format with severity and file:line evidence per finding; the traced flow's real prompt is in Section 4.2.
3. **Diff prompt.** Same role and format applied to the working tree's change set, regenerated after every intermediate product decision (Section 5.3).
4. **Minimal record per finding.** Identifier, severity, cited evidence, generator's verdict, disposition state (Section 3.2), the authority that resolved or accepted the risk, and date; for an open transfer, the responsible recipient.
5. **Repetition.** One run per gate; resume only to confirm a fix; cut on decreasing severity with repetition (a single-episode heuristic: Section 6).
6. **Refutation.** Requires verifiable evidence (code citation, measurement, or documented environment behavior); without evidence, the finding is transferred, not discarded.
7. **No executable oracle.** Findings that depend on external contracts or product intent are transferred to the developer.
8. **Isolation.** Instruct the reviewer to be read-only and inspect the repository state after every run, knowing that this does not constitute complete technical isolation (Appendix A.2).

C Reference instantiation (July 2026)

This appendix reproduces the configuration that materialized the workflow, as a historical record, not a tutorial: it corresponds to the July 2026 tool versions, integration mechanisms change quickly, and the reader must check against current documentation. Installing the tools themselves is out of scope (provider documentation). The configuration blocks are reproduced untranslated, as they ran.

C.1 Orchestrator configuration

Relevant fragment of the Claude Code configuration file (`settings.json`), which materializes the automation: the permanent permission to invoke the reviewer without per-call approval, and the official integration plugin.[2]

```
{
  "permissions": {
    "allow": [ "Bash(codex exec:*)" ],
    "defaultMode": "auto"
  },
```

[2]The `[1m]` suffix in `model` is Claude Code's alias syntax for the one-million-token extended context variant.

```
  "model": "claude-fable-5[1m]",
  "enabledPlugins": {
    "codex@openai-codex": true,
    "claude-security@claude-plugins-official": true
  },
  "extraKnownMarketplaces": {
    "openai-codex": {
      "source": {
        "source": "github",
        "repo": "openai/codex-plugin-cc"
      }
    }
  }
}
```

The second enabled plugin provides the security review of the pre-commit flow (step 2 of the flow described in Section 4.5).

C.2 Orchestrator instructions

The rules governing the rounds live in the orchestrator's global instruction file (curated transcription, with minor typographic adjustments). The complete activation policy was reproduced in Section 4.1; the remaining operating rules, verbatim: the prompt must ask the reviewer to attack the plan or the diff, not to approve; the result is presented hardened, stating what the reviewer caught and what was done with each point; for security changes both gates are advisable; and findings are verified against the code, without blind trust ("decorrelation in both directions"). The file also documents the structural constraint: the reviewer sees the repository (same working directory) but not the orchestrator's conversation, so the plan, the diff and the objective must be passed complete in the prompt.

The invocation path on Windows, executed through the orchestrator's Bash tool (which provides its own Bash environment on Windows; the command is not directly executable in PowerShell or cmd.exe), verified on May 29, 2026 (the reviewer's sandbox does not initialize when invoked from the orchestrator; the mitigation is isolation by instruction plus later verification, with the limits declared in Appendix A.2):

```
codex exec \
  --dangerously-bypass-approvals-and-sandbox \
  > "$HOME/out.txt" 2>&1 <<'PROMPT'
AVISO - SIN SANDBOX: NO escribas/edites/
borres/muevas archivos ni corras git que
modifique estado. SOLO lee y analiza.
<plan o diff + objetivo en texto>.
Actua como red-team: ataca supuestos,
agujeros, riesgos y casos borde. El codigo
esta en este repo (mismo cwd), miralo.
PROMPT
```

Accompanying rules: launch in the background and redirect output to a file (it includes the exploration log and is large; do not trust the exit code); run `git status` after every run to confirm the repository remained intact; and continue an existing conversation with `codex exec resume` only to confirm a fix. When passing the prompt through the shell failed (the traced flow's case, Section 4.2), the robust variant was writing the prompt to a file and passing it by command substitution, that is, supplying `"$(cat prompt.txt)"` as the argument of `codex exec` (the quotes prevent word splitting and glob expansion).

C.3 Structure of the calibration log

A text file versioned next to the project's code, with one entry per round: number and date; work unit and gate; global verdict; each finding with severity, closure (states of Section 3.2) and justification; false positives with their refutation; and a calibration note when an event taught something about the process itself (the lessons of Section 6 come from those notes). It is Appendix B's minimal record per finding, plus the process memory that survives history retention.