Claude Code vs Codev: Round 4 Comparison

2026-02-17

Contents

Claude Code vs. Codev: Todo Manager Comparison — Round 4

Date: 2026-02-17 **PRs**: Claude Code PR #1 (R3, reused) | Codev PR #1 **Reviewers**: Claude Opus 4.6, GPT-5.2 Codex, Gemini 3 Pro **Codev version**: v2.0.9

---

1. Methodology

Both implementations received the same base prompt: build a Next.js 14+ Todo Manager with Gemini 3.0 Flash as the natural-language backend. The Codev builder additionally received the porch strict-mode addendum, which orchestrates the SPIR protocol (Specify → Plan → Implement → Review) with multi-agent consultation at every phase gate.

CC R3 scores are reused. Claude Code scores have been remarkably stable across R1–R3 (overall 5.7–5.9), making re-evaluation unnecessary. Only the Codev/SPIR side was re-run in R4. This eliminates CC-side variability that can mask SPIR-side improvements.

Gemini model fix from R3. R3's Gemini SPIR review failed due to quota exhaustion from the builder's consultation activity. In R4, `consult` explicitly passes `--model gemini-3-pro-preview`, and reviews were run after the builder completed (not concurrently). All 6 reviews completed successfully — the first round to achieve this.

Auto-approved gates. In production Codev use, a human reviews the spec and plan before approving each gate. For this experiment, spec-approval and plan-approval were rubber-stamped (auto-approved) to isolate the effect of the SPIR protocol itself — its phased structure, multi-agent consultation, and enforced review checkpoints — without human review input. Any quality improvements are attributable to the protocol, not human oversight.

Gemini scoring scale. The review prompt's phrasing "Give explicit numeric scores for each dimension (2-7)" was ambiguous — Gemini interpreted "(2-7)" as a 1–7 scoring range rather than "dimensions 2 through 7." Gemini's scores are reported using qualitative-label mapping to a 1–10

scale, consistent with its scoring patterns in R1–R3 (see Reviewer Agreement Analysis for mapping details).

The Codev builder ran as a Claude Opus 4.6 instance with `--dangerously-skip-permissions` in a fresh GitHub repo.

---

2. Build Duration

CC R3: ~15 minutes

A single monolithic commit at `16:06:18 UTC`. The PR was created 19 seconds later at `16:06:37 UTC`. Based on setup timestamps, the entire build took approximately 15 minutes from first prompt to PR.

Codev R4: ~56 minutes

Sixteen commits spanning `18:39:53` to `19:35:47 UTC`. The timeline:

Phase	Time (UTC)	Duration	Commits	What happened
Specify	18:39 → 18:42	3 min	2	Draft spec + consultation revisions
Consultation gap	18:42 → 18:50	8 min	—	3-way review of spec
Plan	18:50 → 18:53	3 min	2	Draft plan + consultation revisions
Consultation gap	18:53 → 19:04	11 min	—	3-way review of plan
Phase 1 (data layer)	19:04 → 19:08	4 min	2	Scaffolding + consultation fix
Phase 2 (CRUD UI)	19:08 → 19:10	2 min	1	Core todo UI
Phase 3 (filtering)	19:10 → 19:16	6 min	2	Filters + consultation fix
Phase 4 (NL interface)	19:16 → 19:22	6 min	2	NL + consultation fixes
Phase 5 (deployment)	19:22 → 19:29	7 min	1	Dockerfile, README, final review
Post-impl	19:29 → 19:35	6 min	3	PR review fixes + artifact commits

Time breakdown

Activity	CC R3	Codev R4
Coding	~15 min	~31 min

Activity	CC R3	Codev R4
Consultation overhead	0 min	~25 min
Total	**~15 min**	**~56 min**

Consultation accounts for roughly 45% of Codev's build time. The coding itself took about 2x longer than CC (31 vs 15 min), driven by the phased structure requiring separate commits, consultation pauses, and post-consultation fixes. The question is whether the 3.7x total time investment pays for itself in quality — the rest of this report answers that.

Cost

All costs from `consult stats` metrics DB (`~/.codev/metrics.db`). Builder session costs are estimated from typical Claude Opus 4.6 API token volumes (not directly measured — Claude Code doesn't log per-session billing locally).

Cost component	CC R3	Codev R4
Builder session (Claude Opus 4.6)	$3–6 (est.)	$8–14 (est.)
Builder consultations (8 rounds × 3 models)	—	**$4.38**
— Claude (9 calls)	—	$2.55
— Codex (8 calls)	—	$1.62
— Gemini (9 calls)	—	$0.21
External reviews (3 models)	**$1.43**	**$1.12**
— Claude	$0.51	$0.64
— Codex	$0.91	$0.42
— Gemini	$0.01	$0.06
Total	**$4–7**	**$14–19**

Cost multiplier: Codev R4 costs roughly **3–5x** more than CC R3. Consultation is the dominant added cost ($4.38) — more than the incremental builder session cost. Gemini consultations are negligibly cheap ($0.21 total for 9 calls); Claude drives the most consultation cost ($2.55) with Codex in between ($1.62).

Cost per quality point: CC R3 scores 5.8 overall at ~$5.50 midpoint = **$0.95/point**. Codev R4 scores 7.0 at ~$16.50 midpoint = **$2.36/point**. The +1.2 quality delta costs ~$11 extra — about **$9/point of improvement**.

Review costs are comparable. Despite Codev having more code surface to review (tests, deployment config), the review costs are similar ($1.12 vs $1.43). The CC Codex review was actually more expensive ($0.91 vs $0.42), likely because its simpler architecture required less structured output.

Session cost estimation methodology: CC R3 (~15 min session) estimated at 100K–200K input tokens + 20K–40K output tokens. Codev R4 (~56 min session with more file reads and tool calls) estimated at 300K–500K input + 50K–80K output. Claude Opus 4.6 API: $15/1M input, $75/1M output. Gemini consultations used Gemini 3 Pro (not Flash): $1.25/1M input, $5.00/1M output — verified against metrics DB effective rate of $1.06/1M (discount from cached tokens). Codex (GPT-5.2): $2.50/1M input, $10.00/1M output. All consultation costs are measured, not estimated.

3. Scorecard

Individual Reviewer Scores

Dimension	Claude (CC)†	Claude (Codev)	Codex (CC)†	Codex (Codev)	Gemini (CC)†	Gemini (Codev)
Bugs	5	5	7	8	8	9
Code Quality	7	7	6	7	8	9
Maintainability	7	7	6	7	9	8
Tests	5	7	4	6	6	7
Extensibility	5	6	5	6	7	7
NL Interface	6	6	5	6	8	8
Deployment	2	7	3	5	3	8

†CC scores reused from R3 (same Claude Code codebase, same reviewers).

Bug scores derived from each reviewer's bug sweep: severity-weighted count inverted to a 1–10 scale. Critical = −2, High = −1, Medium = −0.5, Low = −0.25 from a baseline of 10. Excludes `.env` *(test setup artifact, not committed to git — see Bug Sweep section). Floored at 1.*

Gemini scored on a 1–7 scale; mapped to 1–10 using qualitative labels and R1–R3 calibration (see Reviewer Agreement Analysis).

Averaged Scores

Dimension	CC R3 (avg)	Codev R4 (avg)	Delta
Bugs	6.7	7.3	**+0.7**
Code Quality	7.0	7.7	**+0.7**
Maintainability	7.3	7.3	0.0
Tests	5.0	6.7	**+1.7**
Extensibility	5.7	6.3	+0.7
NL Interface	6.3	6.7	+0.3
Deployment	2.7	6.7	**+4.0**
Overall	**5.8**	**7.0**	**+1.2**

4. Quantitative Comparison

Metric	CC R3	Codev R4
Source lines (excl. tests)	1,294	1,249
Test lines	342	988

Metric	CC R3	Codev R4
Test-to-code ratio	0.26:1	0.79:1
Test files	2	7
Component tests (lines)	0	300
Integration tests (lines)	0	0
Git commits	1	16
Build duration	~15 min	~56 min
Consultation rounds	0	8
Consultation artifacts	0	32
Estimated cost	$4–7	$14–19
Documentation artifacts	0	spec + plan + review
Dockerfile present	No	Yes
`output: "standalone"`	No	Yes

Notable: Codev R4 is the most concise SPIR implementation yet (1,249 source lines vs 1,425 in R3, 1,567 in R2, 1,596 in R1). It's actually *smaller* than the CC codebase (1,294 lines) while producing 2.9x more test code and a Dockerfile — the first round where SPIR produced fewer source lines than CC.

---

5. Bug Sweep Synthesis

Claude Code Bugs (from R3 — reused)

Bug	Severity	Found by	Description
No validation of LLM action payloads	High	Codex, Claude	`src/app/api/nl/route.ts`: Server returns Gemini's JSON with only `Array.isArray` check
NL list filters silently dropped	Medium	Codex, Claude	`src/lib/hooks.ts:118-124`: API supports date filters but client ignores them
Date format assumptions	Medium	All 3	`src/components/TodoItem.tsx:24-32`: Non-ISO formats produce `Invalid Date`
localStorage quota not handled	Low	Codex, Claude	`src/lib/todo-store.ts:25`: `setItem` can throw unhandled `QuotaExceededError`
Stale closure / race condition	High	Claude, Gemini	State consistency issues in hooks and multi-tab scenarios

Codev R4 Bugs (confirmed by 2+ reviewers)

Bug	Severity	Found by	Description
NL update accepts invalid field values	Medium	Codex, Claude	`src/lib/nl-executor.ts:78-96`: `executeUpdate` only checks `updates` is object, not that values are valid
No schema validation on Gemini response	High	Claude (High), Codex (Low)	`src/lib/gemini.ts:134`: `parsed as NLAction` type assertion with no field validation
loadTodos no shape validation	Medium	Claude, Codex	`src/lib/storage.ts:9-15`: `Array.isArray` check only — individual items not validated

Single-Reviewer Findings (not consensus)

Codev single-reviewer bugs:

Bug	Severity	Reviewer	Description
Stale closure in NLInput	Medium	Claude	`NLInput.tsx:47-113`: `todos` captured at render time, not submit time
Toast/storage error interaction	Medium	Claude	`page.tsx:94`: `onDismiss={() => {}}` means error toast can never be dismissed
`searchText` in filter action unused	Medium	Claude	`types/todo.ts:58`: Defined in type but never used. Dead code.
Edit state overwrites newer data	Medium	Codex	`TodoItem.tsx:20-24`: Edit fields initialized from props once, never synced
No input length limits	Low	Claude, Codex, Gemini	No `maxLength` on text inputs
`crypto.randomUUID()` compat	Medium	Gemini	Not available in older browsers or non-HTTPS contexts
No delete confirmation	Low	Claude	Single-click permanent delete with no undo

Bug	Severity	Reviewer	Description
Due date accepts past dates	Low	Claude	No `min` attribute on date input

Cross-cutting: Shared Weaknesses

Both implementations share these problems: - **Full todo list sent every request**: Token cost grows linearly. No truncation or summarization. - **No conversation history**: Each NL message is standalone. - **No multi-tab sync**: Neither listens for `storage` events. - **No E2E/Playwright tests**: Neither includes browser-level tests. - **No XSS vulnerabilities**: React auto-escaping protects both (confirmed by all reviewers).

Bug Quality Narrative

R4 resolves the R3 bug scoring anomaly by having all 6 reviews complete.

Metric	CC R3	Codev R4
Total consensus bugs	5	3
Total single-reviewer bugs	4	8
Consensus Critical	0	0
Consensus High	2	1
Consensus Medium	2	2
Consensus Low	1	0

The `.env` artifact. Both Claude and Codex flagged `.env` as a Critical security leak. The file was NOT committed to git (`.gitignore` excludes it) — it was present on disk only because the review setup copied API keys for the `consult` tool. Both bug scores and deployment scores are computed *excluding* this artifact.

Claude's aggressive pattern continues. Claude found 10 non-`.env` bugs in Codev R4 (2 High, 5 Medium, 3 Low) vs Codex's 4 (2 Medium, 2 Low) and Gemini's 3 (1 Medium, 2 Low). This matches R3's pattern where Claude found 14 SPIR bugs. Claude scrutinizes defensive code patterns that other reviewers don't flag.

What matters is consensus. CC has 5 consensus bugs (2 High) vs Codev's 3 (1 High). The question isn't just "how many bugs?" but "how many bugs *survived* CMAP?" — see the CMAP Effectiveness section for what consultation caught and what it missed.

---

6. Architecture Comparison

Claude Code R3

- **State**: Custom `useTodos` hook + `todo-store.ts` (procedural functions operating on local-Storage)

- **NL**: Single API route with 85-line prompt. `responseMimeType: "application/json"`. Multi-action support. No schema validation.
- **Storage**: Two functions — `loadTodos()`/`saveTodos()`. No error handling.
- **Components**: 5 components in flat hierarchy
- **Dependencies**: `@google/generative-ai` only

Codev R4

- **State**: `useTodos` hook with `useState` + `useEffect` localStorage sync. Filtering, sorting, validation built in.
- **NL**: Three-layer architecture — `gemini.ts` (client with prompt construction + response parsing + markdown fence stripping) → `route.ts` (API with error handling) → `nl-executor.ts` (action execution with discriminated union dispatch). 5 action types.
- **Storage**: Typed layer with `loadTodos`/`saveTodos`, try/catch, error reporting via `{success, error}` return type.
- **Components**: 7 components including `Toast`, `NLInput` with debounce and Gemini availability checking.
- **Dependencies**: `@google/generative-ai`, `vitest`, `@testing-library/react`

Codev R4's key advantage: The `nl-executor.ts` module cleanly separates NL action execution from Gemini communication. Each action type is handled by a dedicated function (`executeAdd`, `executeUpdate`, `executeDelete`, `executeFilter`, `executeList`). This makes the NL pipeline independently testable — and it *is* tested (222 lines of executor tests). CC mixes parsing and execution in a single flow.

CC R3's key advantage: Multi-action responses. Gemini returns an actions array, allowing "delete all completed" to produce multiple delete actions in one response. Codev's single-action design can't batch operations.

Conciseness inversion: Codev R4 is more concise than CC R3 (1,249 vs 1,294 source lines). In R1–R3, SPIR always produced 10–74% more code. The consultation process appears to be driving tighter implementations — reviewers flagged unnecessary complexity in earlier rounds, and the builder learned to write less.

---

7. NL Interface Comparison

Capability	CC R3	Codev R4
Gemini Flash backend	**Yes**	**Yes**
Structured output	`responseMimeType: "application/json"`	`responseMimeType: "application/json"`
Multi-action support	**Yes** (actions array)	No (single action)
Markdown fence defense	No	**Yes** (regex stripping in `gemini.ts`)
Runtime validation of AI output	No (trusts JSON shape)	Partial (action type validated, fields not)
Action type validation	No	**Yes** (discriminated union + allowlist)

Capability	CC R3	Codev R4
Rate limiting	No	No
Delete confirmation	No	No
Gemini availability check	No	**Yes** (UI disables NL when unconfigured)
Context (todo list sent)	Full list	Full list
Conversation history	None	None
Temperature	Not specified	0.1 (deterministic)
Timeout handling	No	**Yes** (10s AbortController)

Verdict: Codev R4's NL architecture is cleaner in its separation of concerns and safer with markdown fence defense, action type validation, and timeout handling. CC retains the multi-action advantage. Neither has conversation memory or prompt injection mitigation.

---

8. Test Quality Deep Dive

CC R3 (342 lines, 2 files)

- `todo-store.test.ts` (225 lines): CRUD operations, filtering, sorting, persistence, edge cases
- `route.test.ts` (117 lines): API validation paths — missing key, missing message, valid request

Not tested: All 5 React components (zero component tests), hooks, NL action processing, error states, corrupt localStorage.

Codev R4 (988 lines, 7 files)

- `useTodos.test.ts` (299 lines): Hook behavior — CRUD, filtering, sorting, persistence, edge cases
- `nl-executor.test.ts` (222 lines): All 5 action types, edge cases (unknown actions, missing fields, non-existent IDs)
- `TodoItem.test.tsx` (145 lines): Rendering, edit mode, status toggle, priority display
- `gemini.test.ts` (95 lines): Response parsing, markdown fence stripping, error handling, Gemini availability
- `AddTodoForm.test.tsx` (92 lines): Form interactions, validation, submission
- `storage.test.ts` (72 lines): localStorage read/write, error handling, corrupt data
- `FilterBar.test.tsx` (63 lines): Filter UI interactions, active state display

Not tested: `NLInput` component (most complex component — async fetch, loading states, debounce), `TodoList` component, `Toast` component, API route (`route.ts`), E2E flows.

Side-by-side

Metric	CC R3	Codev R4
Test lines	342	988
Test files	2	7

Metric	CC R3	Codev R4
Test-to-code ratio	0.26:1	0.79:1
Component tests	0 lines	300 lines
Layer coverage	Store + API route	Store + hooks + NL executor + components + storage
Missing coverage	Components, hooks, NL logic	NLInput, TodoList, Toast, API route, E2E

CC's test-to-code ratio remains locked at 0.26:1 across all four rounds — a Claude baseline without protocol guidance. Codev R4's 0.79:1 is the lowest SPIR ratio (down from 1.09:1 in R1), tracking with the more concise codebase, but still 3x higher than CC.

---

9. Deployment Readiness

This dimension shows the largest delta of any dimension in any round: **+4.0**.

Feature	CC R3	Codev R4
Dockerfile	No	**Yes** (multi-stage, non-root)
`output: "standalone"`	No	**Yes**
`.env.example`	Yes	Yes
`.dockerignore`	No	**Yes**
Health check endpoint	No	No
Railway-specific config	No	No
CI/CD pipeline	No	No
README with deploy instructions	No	**Yes**

Codev R4's Dockerfile is production-quality: multi-stage build (deps → build → slim runner), non-root user (`nextjs:nodejs`), `HOSTNAME="0.0.0.0"`, standalone output. CC R3 has none of this.

SPIR produced a Dockerfile in 2 of 4 rounds; CC never has (0 of 4). The consultation process drives deployment awareness, though inconsistently.

---

10. Consultation Process Analysis

Overview

The SPIR protocol drove **8 consultation rounds**, each invoking 3 models (Claude, Codex, Gemini) for a total of **32 consultation artifacts** (24 model responses + 8 rebuttals).

Phase	Claude	Codex	Gemini	Result
Specify	REQUEST_CHANGES	REQUEST_CHANGES	APPROVE	9 items addressed in spec revision

Phase	Claude	Codex	Gemini	Result
Plan	COMMENT	REQUEST_CHANGES	APPROVE	9 items addressed in plan revision
Phase 1	APPROVE	REQUEST_CHANGES	APPROVE	1 fix: undefined leak in updateTodo
Phase 2	APPROVE	APPROVE	APPROVE	No changes needed
Phase 3	APPROVE	APPROVE	APPROVE	1 fix: empty state differentiation
Phase 4	PASS w/ MINOR	REQUEST_CHANGES	APPROVE	3 fixes: UTC date, update guard, filter validation
Phase 5	APPROVE	APPROVE	APPROVE	No changes needed
Review (PR)	FAILED*	FAILED*	APPROVE*	2 fixes: NL disabled state, localStorage quota

**PR review degraded: Gemini crashed (Yoga layout issue), Claude's prompt was too long. Only Codex completed review and drove 2 fixes.*

Verdict distribution

- **Gemini**: Approved every round. Never requested changes. Consistent with the optimism pattern observed across all rounds (see Reviewer Agreement Analysis).
- **Codex**: Requested changes in 4 of 8 rounds. Drove the most substantive fixes — the UTC date bug, the update guard, filter validation, and both PR review fixes.
- **Claude**: Requested changes in 2 of 8 rounds (spec and plan), commented on 1, passed with minor issues on 1. Drove spec/plan improvements but less active during implementation.

Impact by the numbers

Metric	Count
Consultation rounds	8
Total artifacts	32
Fixes implemented from feedback	11
Commits addressing consultation feedback	6
Bugs caught pre-merge	5 (UTC date, update guard, filter validation, NL disabled state, localStorage quota)
UX improvements driven	2 (empty state messages, Gemini availability indicator)
Spec/plan items refined	18
Items explicitly defended/rejected	5
Items deferred to future scope	4
Consultation time overhead	~25 min (45% of build)

What consultation drove

Spec phase was the most impactful consultation. All three reviewers identified the missing NL command schema — the spec didn't define what actions Gemini could return, what fields each action required, or how ambiguity was resolved. The revision added a complete action type taxonomy that shaped the entire implementation.

Phase 4 (NL interface) was the most impactful implementation consultation. Codex identified three concrete bugs: 1. `toISOString().split("T")[0]` produces UTC dates, not local dates — "add todo due tomorrow" could be off by a day depending on timezone 2. `executeUpdate` would crash if Gemini returned an update action without an `updates` field 3. Gemini could return invalid `status` or `priority` enum values that would silently corrupt filter state

PR review was partially degraded but still valuable. Despite Gemini crashing and Claude's prompt exceeding limits, Codex's solo review caught two real issues: the NL input wasn't disabled when Gemini was unavailable (users could type commands that would always fail), and localStorage quota errors were silently swallowed.

---

11. CMAP Effectiveness

What did multi-agent consultation catch, and what did it miss? This table maps bug classes to CMAP's detection record for this round.

Bugs caught by CMAP (prevented pre-merge)

Bug class	Caught by	Phase	Evidence
UTC date off-by-one	Codex	Phase 4	`toISOString()` → `toLocaleDateString("en-CA")`
Missing update guard	Codex	Phase 4	Runtime crash on update without `updates` field
Invalid filter enum values	Codex	Phase 4	Unvalidated Gemini values corrupting filter state
Undefined leak in updateTodo	Codex	Phase 1	Spread operator propagating undefined values
NL input not disabled when Gemini unavailable	Codex	PR review	Users could submit commands guaranteed to fail
localStorage quota unhandled	Codex	PR review	`setItem` throws on quota exceeded, uncaught
Missing NL command schema	All 3	Specify	Spec had no action type definitions
Hydration mismatch risk	Gemini, Claude	Plan	SSR/client state divergence

Bug class	Caught by	Phase	Evidence
Empty state UX confusion	Gemini, Claude	Phase 3	Same message for "no todos" and "no matches"
Missing Toast system	Claude	Plan	No user feedback for async operations
No NL debounce	Claude	Plan	Rapid submissions could flood Gemini API

Bugs that survived CMAP (found by external reviewers)

Bug class	Severity	Found by	Why CMAP missed it
Stale closure in NLInput	Medium	Claude (ext)	Component-level closure analysis not in CMAP's review scope; NLInput was the most complex component and was never component-tested
No Gemini response schema validation	High	Claude, Codex (ext)	CMAP flagged *spec-level* schema concerns but didn't catch the `as NLAction` type assertion in implementation
loadTodos no shape validation	Medium	Claude, Codex (ext)	Storage layer reviewed in Phase 1, but individual item validation not checked
Toast/storageError render loop	Medium	Claude (ext)	Interaction between two features added at different phases; cross-feature interactions are CMAP's blind spot
Edit state overwrites newer data	Medium	Codex (ext)	React component state lifecycle not deeply reviewed by CMAP

CMAP pattern

CMAP excels at catching **data integrity bugs** (UTC dates, undefined leaks, invalid enums) and **missing feature gaps** (schema, toast, debounce, disabled states). It struggles with **cross-feature interactions** (toast + storage error loop), **component-level state lifecycle** (stale closures, edit state sync), and **type-level safety gaps** (unsafe `as` assertions that pass TypeScript but fail at runtime).

The pattern suggests CMAP functions as a strong "second pair of eyes" for logic bugs but not as a substitute for component testing or runtime type validation.

---

12. Reviewer Agreement Analysis

Where all three Codev reviewers agreed:

- Code architecture is clean with good TypeScript usage (all scored 7+)
- NL executor pattern (discriminated unions) is well-designed
- No XSS vulnerabilities (React escaping)
- No E2E tests
- `loadTodos` needs stronger schema validation

Where Codev reviewers disagreed:

- **Bug thoroughness**: Claude found 10 bugs, Codex found 4, Gemini found 3. Claude's aggressive pattern is consistent across all rounds.
- **Deployment score**: Claude 7, Codex 5, Gemini 8. Codex penalized heavily for the `.env` artifact; without it, scores would converge around 7.
- **Tests**: Claude 7, Codex 6, Gemini 7. Codex penalized missing API route tests more heavily.

Gemini's /7 scale mapping

Gemini interpreted the review prompt as requesting scores on a 1–7 scale. The conversion to 1–10 used:

Gemini label	Gemini score	Mapped to /10	Rationale
"Excellent"	7/7	9	Matches R2 Gemini Codev Code Quality (9)
"High"	6/7	8	Matches R2 Gemini Codev Maintainability pattern
"Very Good"	6/7	7	Tests: calibrated against R1 (7) and R2 (5) patterns
"Good, but…"	5/7	7	Extensibility: matches R2 pattern with noted limitation
"Robust"	6/7	8	NL: matches R2 Gemini Codev NL (8)
"Ready"	6/7	8	Deployment: Dockerfile present, non-root, multi-stage → 8

For bugs, the formula-based approach was used (consistent with all other reviewers): Gemini found 1 Medium + 2 Low = 9.

Gemini's optimism pattern

Round	Gemini CC avg	Codex/Claude CC avg	Gemini Codev avg	Codex/Claude Codev avg
R2	6.2	5.5	7.5	6.7
R3	6.7	5.1	—	6.0
R4	6.7†	5.1†	8.0	6.3

†Reused from R3.

Gemini consistently scores +1.0 to +1.7 above the Codex/Claude average. This also explains why Gemini approved every CMAP consultation round — its review threshold is systematically lower.

---

13. Key Takeaways

1. All 6 reviews completed — first time in the experiment

R3 lost the Gemini SPIR review to quota exhaustion. R4 achieved full coverage by running reviews after the builder completed. This eliminates the reviewer asymmetry that distorted R3's scores, particularly the bug dimension where R3 showed an anomalous −2.2 delta that R4 corrects to +0.7.

2. SPIR costs 3.7x more time and 3–5x more money but delivers +1.2 quality

56 minutes vs 15 minutes; $14–19 vs $4–7. Consultation is the dominant added cost ($4.38, more than the incremental session cost). The quality delta is consistent at +1.2 across R1, R2, and R4 (R3 was anomalous due to missing reviews). Each point of quality improvement costs ~$9 extra. Whether that trade-off is worth it depends on the project: for a throwaway prototype, no; for production code that will be maintained, the deployment readiness and test coverage alone justify it.

3. Consultation catches data bugs, misses component bugs

CMAP prevented 5 concrete bugs in implementation (UTC dates, undefined leaks, missing guards, disabled states, quota handling) and shaped the spec/plan significantly. But it missed stale closures, type assertion bypasses, and cross-feature interaction bugs. The pattern: CMAP is strong on logic and data flow, weak on React component lifecycle and runtime type safety.

4. Codex is the workhorse reviewer

Codex requested changes in 4 of 8 rounds and drove 8 of 11 fixes. Gemini approved everything. Claude drove spec/plan improvements but was less active during implementation. For consultation ROI, Codex alone would have caught most of the bugs — though Claude's spec-level contributions (prompt injection awareness, testing layer restructure) had structural impact.

5. Deployment is SPIR's largest advantage (+4.0)

Multi-stage Dockerfile, `.dockerignore`, standalone output, deploy README. CC had none of this. This is the largest delta of any dimension in any round across the entire experiment. SPIR produced

a Dockerfile in 2 of 4 rounds; CC never has.

6. SPIR is getting more concise

Round	CC lines	SPIR lines	SPIR overhead
R1	916	1,596	+74%
R2	1,033	1,567	+52%
R3	1,294	1,425	+10%
R4	1,294	1,249	**−3%**

For the first time, SPIR produced *fewer* source lines than CC while maintaining quality advantages. Consultation may be driving tighter code: reviewers flag unnecessary complexity, and the builder learns to write less.

7. Testing advantage is the most consistent finding

Round	CC Tests	Codev Tests	Delta
R1	4.0	7.7	+3.7
R2	5.0	6.0	+1.0
R3	5.0	7.0	+2.0
R4	5.0	6.7	+1.7

SPIR consistently produces 2.9–7.4x more test lines with broader layer coverage. CC is locked at 0.26:1 test-to-code ratio across all rounds — a model baseline without protocol guidance.

---

14. Summary: When Does Codev Pay Off?

Dimension	Codev advantage?	Delta	Notes
Bugs	**Yes**	+0.7	Fewer consensus bugs (3 vs 5), fewer High (1 vs 2)
Code Quality	**Yes**	+0.7	Clean three-layer NL architecture, discriminated unions
Maintainability	Neutral	0.0	Both are small, readable codebases

Dimension	Codev advantage?	Delta	Notes
Tests	**Yes**	+1.7	Most consistent advantage; 2.9x more test lines
Extensibility	**Yes**	+0.7	Better abstractions via layered architecture
NL Interface	Marginal	+0.3	Better separation but CC has multi-action advantage
Deployment	**Yes**	+4.0	Dockerfile, dockerignore, standalone output, README

Bottom line: R4 is the cleanest round methodologically (all 6 reviews, stable CC baseline, auto-approved gates isolating protocol effect) and confirms SPIR's consistent +1.2 quality advantage at a 3–5x cost premium ($14–19 vs $4–7). The protocol's value comes from three sources: phased structure (forcing separation of concerns), consultation (catching 5 implementation bugs pre-merge at $4.38), and spec/plan artifacts (18 items refined before coding started). The biggest single-round finding is that even with rubber-stamped gates — no human reviewing specs or plans — the protocol alone drives measurable improvement. At ~$9 per quality point, the ROI depends on what you're building.

Appendix A: Raw Review Outputs

Reviewer	CC	Codev
Gemini	`/tmp/gemini-vibe-r3.txt` (from R3)	`/tmp/gemini-spir-r4.txt`
Codex	`/tmp/codex-vibe-r3.txt` (from R3)	`/tmp/codex-spir-r4.txt`
Claude	`/tmp/claude-vibe-r3.txt` (from R3)	`/tmp/claude-spir-r4.txt`

Appendix B: Historical Context

Cross-round trend table for reference. R4 is the focus of this report; see individual round reports for deep dives.

Dimension	R1 Δ	R2 Δ	R3 Δ‡	R4 Δ
Bugs	—	+2.7	−2.2‡	**+0.7**
Code Quality	+1.0	+1.3	0.0‡	**+0.7**
Maintainability	+0.7	+0.3	+0.2‡	**0.0**
Tests	+3.7	+1.0	+2.0‡	**+1.7**
Extensibility	+1.0	+1.0	+0.8‡	**+0.7**
NL Interface	0.0	+1.0	+0.7‡	**+0.3**
Deployment	+2.0	(excl)	+0.8‡	**+4.0**
Overall	**+1.3**	**+1.2**	**+0.4‡**	**+1.2**

‡R3 Codev based on 2/3 reviewers (Gemini quota exhaustion). R1 did not score bugs. R2 excluded deployment.

Round	CC Overall	Codev Overall	Codev Build Time
R1	5.9	7.2	~45 min
R2	5.7	7.0	~50 min
R3	5.8	6.1‡	~55 min
R4	5.8	7.0	~56 min

Excluding R3's anomalous 2-reviewer average, the delta has been consistent at +1.2 to +1.3. R4 confirms R3's narrowing was a measurement artifact, not a real quality convergence.

Full reports: - R1: `codev/resources/vibe-vs-spir-comparison-2026-02.md` - R2: `codev/resources/vibe-vs-` - R3: `codev/resources/vibe-vs-spir-r3-comparison-2026-02.md`