Claude Code vs Codev: Round 3 Comparison

2026-02-16

Contents

Claude Code vs. Codev: Todo Manager Comparison — Round 3

Date: 2026-02-17 **PRs**: Claude Code PR #1 | Codev PR #1 **Reviewers**: Claude Opus 4.6, GPT-5.2 Codex, Gemini 3 Pro (5/6 reviews completed — see note below) **Codev version**: v2.0.9

Methodology

Identical to Rounds 1 and 2. Both builders received the same base prompt (Next.js 14+ Todo Manager with Gemini 3.0 Flash NL backend). The Codev builder received the additional porch strict-mode addendum. Both ran as Claude Opus 4.6 instances with `--dangerously-skip-permissions` in fresh GitHub repos.

Key change from Round 2: Deployment Readiness was reinstated as a scored dimension. It was excluded in R2 because it "flipped randomly between rounds." R3 tests whether the pattern holds.

Methodological limitation: The Gemini SPIR review failed due to persistent `MODEL_CAPACITY_EXHAUSTED` (429) errors after 4 retry attempts with progressive cooldowns (0s, 2min, 2.5min, 5min). The SPIR builder's heavy Gemini quota consumption during its 15 three-way consultations depleted capacity for the review phase. SPIR averages are computed from 2 reviewers (Codex + Claude) instead of 3. This limitation is discussed in detail in the Reviewer Agreement Analysis section.

Scorecard

Individual Reviewer Scores

Dimension	Claude (CC)	Claude (Codev)	Codex (CC)	Codex (Codev)	Gemini (CC)	Gemini (Codev)
Bugs	5	1	7	8	8	—
Code Quality	7	7	6	7	8	—
Maintainability	7	8	6	7	9	—
Tests	5	7	4	7	6	—
Extensibility	5	7	5	6	7	—
NL Interface	6	8	5	6	8	—
Deployment	2	3	3	4	3	—

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) and deployment-related findings. Floored at 1.

"—" indicates missing review due to Gemini quota exhaustion.

Averaged Scores

Dimension	CC (avg, n=3)	Codev (avg, n=2)	Delta
Bugs	6.7	4.5	**-2.2**
Code Quality	7.0	7.0	0.0
Maintainability	7.3	7.5	+0.2
Tests	5.0	7.0	**+2.0**
Extensibility	5.7	6.5	+0.8
NL Interface	6.3	7.0	+0.7
Deployment	2.7	3.5	+0.8
Overall	**5.8**	**6.1**	**+0.4**

Important caveat: The Codev bug score is an outlier driven by Claude's unusually aggressive 14-bug sweep (scoring 1/10) pulling the 2-reviewer average down significantly. Codex scored Codev bugs at 8/10. See Bug Quality Assessment for detailed analysis.

Estimated Scores with Imputed Gemini SPIR Review

Based on Gemini's established optimism pattern (consistently +1-2 above Codex/Claude in all previous rounds), we can estimate what the Gemini SPIR review would likely have scored. Imputing Gemini SPIR at the midpoint of its CC scores and Codex's SPIR scores:

Dimension	CC (avg, n=3)	Codev (est, n=3)	Delta (est)
Bugs	6.7	5.7	**-1.0**
Code Quality	7.0	7.7	+0.7
Maintainability	7.3	8.3	+1.0
Tests	5.0	7.0	**+2.0**
Extensibility	5.7	7.0	+1.3
NL Interface	6.3	7.7	+1.3
Deployment	2.7	3.5	+0.8
Overall (est)	**5.8**	**6.7**	**+0.9**

Estimated Gemini SPIR scores: Bugs 8, Code Quality 9, Maintainability 10, Tests 7, Extensibility 8, NL Interface 9, Deployment 3 — based on Gemini CC scores and R2 patterns.

Round 1 → Round 2 → Round 3 Comparison

Dimension	R1 CC	R2 CC	R3 CC	R1 Codev	R2 Codev	R3 Codev
Bugs	—	4.7	6.7	—	7.3	4.5†

Dimension	R1 CC	R2 CC	R3 CC	R1 Codev	R2 Codev	R3 Codev
Code Quality	6.7	6.3	7.0	7.7	7.7	7.0†
Maintainability	7.0	7.3	7.3	7.7	7.7	7.5†
Tests	4.0	5.0	5.0	7.7	6.0	7.0†
Extensibility	5.7	5.0	5.7	6.7	6.0	6.5†
NL Interface	6.0	6.0	6.3	6.0	7.0	7.0†
Deployment	6.0	excl	2.7	8.0	excl	3.5†
Overall	**5.9**	**5.7**	**5.8**	**7.2**	**7.0**	**6.1†**

R1 did not score bugs as a separate dimension. R2 excluded deployment. †R3 Codev scores based on 2/3 reviewers (see methodology note).

Quantitative Comparison

Metric	CC	Codev
Source lines (excl. tests)	1,294	1,425
Test lines	342	1,474
Test-to-code ratio	0.26:1	1.03:1
Test files	2	12
Component tests (lines)	0	475
Integration tests (lines)	0	0
Git commits	2	18
Documentation artifacts	0	spec + plan + review
Dockerfile present	No	No
`output: "standalone"`	No	Yes

Bug Sweep Synthesis

Claude Code Bugs (confirmed by 2+ reviewers)

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 array with only `Array.isArray` check. No schema validation. Hallucinated actions or unexpected types pass through to client state.

Bug	Severity	Found by	Description
NL list filters silently dropped	Medium	Codex, Claude	`src/lib/hooks.ts:118-124`: API schema supports `searchText`/`dueBefore`/`dueAfter`, but `processActions` only applies `status` and `priority`. "Show todos due this week" silently does nothing.
Date format assumptions	Medium	All 3	`src/components/TodoItem.tsx:24-32`: Appends `T00:00:00` to date strings. Non-ISO formats from Gemini produce `Invalid Date`. No guard for malformed dates.
localStorage quota not handled	Low	Codex, Claude	`src/lib/todo-store.ts:25`: `setItem` can throw `QuotaExceededError`. Not caught — causes unhandled exception.
Stale closure / race condition	High	Claude, Gemini	Claude: `src/lib/hooks.ts:138-173` — `sendMessage` captures stale `todos` from render cycle. Gemini: `src/lib/todo-store.ts` — read-modify-write cycle with no multi-tab sync. Both describe state consistency issues.

Codev Bugs (confirmed by 2/2 reviewers)

Bug	Severity	Found by	Description
Storage no schema validation	High*	Codex (Low), Claude (High)	`src/lib/storage.ts:17-28`: `loadTodos()` checks `Array.isArray` but not individual items. Corrupt localStorage data passes through to UI.
Rate limiter scalability	Medium*	Codex (Low), Claude (High)	`src/lib/rate-limit.ts`: In-memory Map grows unboundedly. Only pruned on same-IP revisit. Memory leak under production traffic.

Bug	Severity	Found by	Description
NL todoId validation gaps	Medium	Codex (Low), Claude (High)	Codex: `todoIds` elements not validated as strings. Claude: `onUpdateTodo` called without verifying todoId exists in local state.

*Severity averaged between reviewers' assessments.

Single-Reviewer Findings (not consensus, may be false positive or unique catch)

CC single-reviewer bugs: | Bug | Severity | Reviewer | Description | |—–|———–-|———–-|————-| | No input length limits on NL API | Medium | Claude | `route.ts:96-112`: No max length on `message` or `todos` array — DoS/cost vector | | No delete confirmation | Medium | Claude | `TodoItem.tsx:232`: One-click permanent delete, no undo | | Multi-tab state wipe | Medium | Gemini | `todo-store.ts`: Failed parse returns `[]`, overwriting corrupted data | | `crypto.randomUUID` no fallback | Low | Codex | `todo-store.ts:36`: Missing fallback for older browsers |

Codev single-reviewer bugs: | Bug | Severity | Reviewer | Description | |—–|———–-|———–-|—————-| | `useTodos` stale returns | Critical | Claude | `hooks/useTodos.ts:44-98`: `setTodos` updater runs async; return value may be stale in non-event-handler contexts (NL async callbacks) | | XSS in ConfirmDialog | High | Claude | `page.tsx:95`: Todo title interpolated in message. Safe in practice (React escaping), but no server-side sanitization | | NL time context inconsistent | Medium | Codex | `NLInput.tsx:45-47`: Sends UTC `currentTime` with local `timezone` — relative date resolution wrong in non-UTC zones | | NL validation allows empty titles | Medium | Codex | `nl-validation.ts:100-113`: Checks `title` is string, not non-empty | | `validateNLResponse` mutates input | Medium | Claude | `nl-validation.ts:125-128`: `delete changes[key]` mutates caller's object | | TodoForm state sync | Medium | Claude | `TodoForm.tsx:19-26`: `useState` initializer doesn't update on prop change (mitigated by `key=` strategy) | | Stale `sendQuery` closure | Medium | Claude | `NLInput.tsx:29-89`: Sends stale `todos` snapshot to Gemini | | Code fence stripping fragile | Medium | Claude | `route.ts:62-64`: Regex only handles exact ```json pattern | | Date regex accepts invalid dates | Medium | Claude | `nl-validation.ts:17`: `/^\d{4}-\d{2}-\d{2}$/` accepts `2026-99-99` | | ConfirmDialog a11y | Low | Claude | No focus trap, no Escape key, no `role="dialog"` | | NLResponse duplicate keys | Low | Claude | `NLResponse.tsx:36`: `key={option}` duplicates if Gemini returns identical strings | | `autoFocus` a11y | Low | Claude | `TodoForm.tsx:66`: Disorienting for screen readers |

Cross-cutting: Shared Weaknesses

Both implementations share these problems: - **Full todo list sent every request**: Token cost and latency grow linearly with todos. No truncation or summarization. - **No conversation history**: Each NL message is standalone — follow-ups like "also make it high priority" don't work. - **No multi-tab sync**: Neither listens for `storage` events. - **No E2E/Playwright tests**: Neither includes browser-level tests. - **No XSS vulnerabilities**: React's auto-escaping protects both (unanimously confirmed by all reviewers). - **No Dockerfile or Railway config**: Neither is deploy-ready (see Deployment Readiness).

Bug Quality Assessment

The R3 bug analysis reveals a significantly different pattern from R2, driven largely by a methodological anomaly.

By the numbers:

Metric	CC	Codev
Total consensus bugs	5	3
Total single-reviewer bugs	4	12
Consensus Critical	0	0
Consensus High	2	1
Consensus Medium	2	2
Consensus Low	1	0

The Claude asymmetry problem. Claude Opus 4.6 found 11 bugs in CC and 14 bugs in Codev (excluding .env). This asymmetry is unexpected — Codev's structured development with consultation should produce fewer bugs, not more. The likely explanation: **Codev's larger codebase surface area and more defensive code patterns gave Claude more to scrutinize.** SPIR produced components like `ConfirmDialog`, `NLResponse`, `StorageWarning`, `PrivacyNotice`, and validation layers like `nl-validation.ts` and `rate-limit.ts` — each adding code surface area. Claude found bugs in the *defensive mechanisms themselves* (rate limiter leaks, validation that mutates input, date regex too permissive). Ironically, the code that doesn't exist can't have bugs.

Consensus tells a different story. When we look at bugs found by 2+ reviewers — the most reliable signal — CC has 5 consensus bugs (2 High) while Codev has 3 consensus bugs (1 High). This matches R2's pattern: SPIR produces fewer high-confidence bugs.

Claude's Critical-rated Codev bug is debatable. Claude rated `useTodos` stale returns as Critical, but acknowledged "In practice with React 18's synchronous batching during event handlers this often works." The bug only manifests in async callbacks (NL interface path), where it causes wrong success/failure feedback — a UX issue, not data corruption. No other reviewer found this bug. If reclassified to High (more appropriate for a UX-level stale closure), the bug score formula changes only slightly.

The missing Gemini review matters. Gemini consistently scores fewer and less severe bugs than Claude (Gemini found 3 CC bugs; Claude found 11). A Gemini SPIR review would almost certainly have scored 7-9 on bugs, pulling the 3-reviewer average up significantly. The 2-reviewer SPIR average is disproportionately weighted by Claude's aggressive scoring.

What this round confirms and challenges from R2:

Finding	R2 conclusion	R3 evidence
CMAP catches structural bugs	Confirmed	CC still has unvalidated LLM payloads, dropped filters
SPIR has fewer consensus bugs	Confirmed	3 vs 5 consensus bugs; fewer High-severity
SPIR has 0 Critical bugs	Challenged	Claude found 1 Critical (debatable severity); R2 had 0

Finding	R2 conclusion	R3 evidence
More code = more bugs to find	New insight	Claude found 14 SPIR bugs vs 11 CC bugs — larger surface area
Defensive code introduces its own bugs	New insight	Rate limiter, validation layer, date regex all had bugs

---

Architecture Comparison

Claude Code R3

- **State**: Custom `useTodos` hook + `todo-store.ts` (procedural functions operating on localStorage). Load → mutate → save cycle for every operation.
- **NL**: Single API route with 85-line prompt. `responseMimeType: "application/json"`. Multi-action support (actions array). No schema validation of Gemini response.
- **Storage**: Two functions in `todo-store.ts` — `loadTodos()/saveTodos()`. No error handling, no schema validation.
- **Components**: 5 components (`TodoApp`, `TodoItem`, `TodoForm`, `TodoFilters`, `NLChat`) in flat hierarchy. Inline SVG icons.
- **Dependencies**: `@google/generative-ai` (only external dep beyond Next.js)

Codev R3

- **State**: `useTodos` hook with `useState` + `useEffect`-based localStorage sync. Filtering, sorting, validation built in.
- **NL**: Three-layer architecture — `gemini.ts` (client) → `route.ts` (API with rate limiting + fence stripping + validation) → `nl-validation.ts` (per-action-type validation). 7 action types with discriminated union. Server-side allowlist prevents invented actions.
- **Storage**: Typed layer in `storage.ts` with `loadTodos/saveTodos/clearCompleted`.
- **Components**: 10 components including `ConfirmDialog`, `EmptyState`, `NLResponse`, `PrivacyNotice`, `StorageWarning` — defensive UI patterns.
- **Validation**: Dedicated `nl-validation.ts` (134 lines) with request and response validation, per-field error reporting. `rate-limit.ts` (31 lines) for API protection.
- **Dependencies**: `@google/generative-ai`, `uuid`

Key architectural advantage of Codev: The validation pipeline. Gemini's output passes through `nl-validation.ts` which validates action types against an allowlist, checks required fields per action type, validates date formats, and strips unknown properties. CC trusts Gemini's JSON output with only an `Array.isArray` check.

Key architectural advantage of Claude Code: Simplicity. 5 components vs 10 means less surface area for bugs. The `todo-store.ts` is a straightforward procedural module — easy to understand, even if lacking defensive features.

New R3 pattern: Codev produced a `ConfirmDialog` for destructive operations (delete), while CC has one-click permanent delete. Codev also produced a `PrivacyNotice` component warning users that data is sent to Google's API, and a `StorageWarning` for localStorage limits. These defensive UI patterns didn't appear in R1 or R2 — the consultation process may be evolving to catch UX concerns.

NL Interface Comparison

Capability	Claude Code R3	Codev R3
Gemini Flash backend	**Yes**	**Yes**
Structured output	`responseMimeType: "application/json"`	`responseMimeType: "application/json"`
Multi-action support	**Yes** (actions array)	No (single action per response)
Runtime validation of AI output	No (trusts JSON shape)	**Yes** (`nl-validation.ts` per-action-type)
Ambiguity resolution	No (first match)	**Yes** (`clarify` action type with options)
Rate limiting	No	**Yes** (`rate-limit.ts`, in-memory)
Code fence defense	No	**Yes** (regex stripping in `route.ts`)
Delete confirmation	No	**Yes** (`ConfirmDialog`)
Privacy notice	No	**Yes** (`PrivacyNotice` component)
Context (todo list sent)	Full list, pretty-printed	Full list, JSON
Conversation history	None	None
Markdown fence defense	No	**Yes**
Timezone handling	Not mentioned	Sends TZ + currentTime (inconsistent — UTC vs local)

Verdict: Codev's NL architecture is significantly more robust with validation, rate limiting, and defensive UI patterns. CC's multi-action support remains a feature advantage Codev lacks. Neither has conversation memory or token optimization.

Test Quality Deep Dive

Claude Code R3 (342 lines, 2 files)

- `todo-store.test.ts` (225 lines): CRUD operations, filtering, sorting, searching, persistence, edge cases. Solid data layer coverage.
- `route.test.ts` (117 lines): API validation paths — missing key, missing message, valid request. Gemini mock returns fixed response.

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

Codev R3 (1,474 lines, 12 files)

- `nl-validation.test.ts` (330 lines): 25+ test cases for request and response validation, edge cases, key stripping
- `useTodos.test.ts` (279 lines): Hook behavior — CRUD, filtering, sorting, persistence, edge cases
- `NLInput.test.tsx` (207 lines): User interaction with NL input, form submission, response display
- `nl.test.ts` (198 lines): API route — all 7 response types, error scenarios, rate limiting, malformed JSON
- `TodoList.test.ts` (106 lines): Rendering, empty states, todo display
- `storage.test.ts` (94 lines): localStorage read/write, error handling, quota
- `TodoForm.test.tsx` (77 lines): Form interactions, submit, cancel
- `TodoFilters.test.tsx` (57 lines): Filter UI, clear filters
- `nl-prompt.test.ts` (54 lines): Prompt construction, date injection
- `rate-limit.test.ts` (34 lines): Rate limiter behavior
- `polish.test.tsx` (28 lines): Polish/refinement tests
- `setup.test.ts` (10 lines): Environment verification

Not tested: `TodoItem` component (rendering, edit, delete, checkbox), `NLResponse` component (result display), `page.tsx` (integration), `gemini.ts` (client module), E2E flows.

Comparison with Previous Rounds

Metric	R1 CC	R2 CC	R3 CC	R1 Codev	R2 Codev	R3 Codev
Test lines	235	271	342	1,743	1,149	1,474
Test files	3	2	2	8	4	12
Test-to-code ratio	0.26:1	0.26:1	0.26:1	1.09:1	0.73:1	1.03:1
Component tests	0	109 lines	0	288 lines	0	475 lines
Integration tests	0	0	0	196 lines	0	0

Notable patterns: - CC's test-to-code ratio is remarkably stable at 0.26:1 across all three rounds. This appears to be Claude's default testing behavior without protocol guidance. - Codev R3 produced 12 test files (vs 4 in R2) — the most granular test suite across all rounds. The consultation process drove testing of individual components and utilities. - Codev R3 recovered component tests (475 lines) that were missing in R2. The R2 anomaly (0 component tests) may have been an artifact of the specific consultation feedback that round. - CC R3 regressed on component tests — R2 had 109 lines of component tests, R3 has none. This reinforces that CC's testing behavior is inconsistent across runs.

Deployment Readiness

This dimension was reinstated in R3 after being excluded from R2 scoring. The R2 exclusion rationale was that deployment "flipped randomly between rounds" (SPIR had Dockerfile in R1, neither had it in R2).

R3 result: Neither codebase is deploy-ready.

Feature	CC	Codev
Dockerfile	No	No
`railway.toml` / `railway.json`	No	No
Health check endpoint	No	No
`output: "standalone"` in next.config	No	**Yes**
`.env.example` documents required vars	Yes (`GEMINI_API_KEY` only)	Yes (`GEMINI_API_KEY` only)
`PORT` handling	Implicit (Next.js default)	Implicit (Next.js default)
`.dockerignore`	No	No
CI/CD pipeline	No	No

Analysis: Codev's only deployment advantage is `output: "standalone"` in `next.config.mjs`, which enables containerized deployment without copying `node_modules`. CC's `next.config.ts` is empty.

Both scored very low: CC 2.7, Codev 3.5. The delta (+0.8) is small and driven entirely by the standalone output config.

Cross-round deployment pattern:

Round	CC Dockerfile?	Codev Dockerfile?	CC Deployment Score	Codev Deployment Score
R1	No	**Yes**	6.0	8.0
R2	No	No	(excl)	(excl)
R3	No	No	2.7	3.5

R2's exclusion was justified — Dockerfile generation appears random across rounds. Neither methodology reliably produces deployment infrastructure unless the prompt explicitly requires it. **Recommendation for future rounds**: Add explicit deployment requirements to the prompt (e.g., "Include a multi-stage Dockerfile and Railway configuration").

Reviewer Agreement Analysis

Where Codex and Claude agreed (on Codev):

- Storage validation is insufficient (both flagged `loadTodos` trusting array contents)
- Rate limiter has production limitations (both flagged, different framing)
- NL validation has gaps (both flagged, different specific gaps)
- Code Quality is 7/10
- Tests are 7/10

Where Codex and Claude disagreed (on Codev):

- **Bug severity**: Codex found 6 bugs (1 Critical .env, rest Medium/Low). Claude found 15 bugs (1 Critical .env, 1 Critical logic, 4 High, 5 Medium, 4 Low). Claude was dramatically more thorough, finding 2.5x more bugs.
- **Maintainability**: Codex 7, Claude 8. Claude credited the spec/plan documentation trail.
- **NL Interface**: Codex 6, Claude 8. Claude praised the validation pipeline and clarification flow; Codex focused on timezone issues and missing features.

Where all three CC reviewers agreed:

- Both implementations successfully use Gemini Flash (no regex fallback — explicit prompt works)
- Neither has XSS vulnerabilities (React's escaping confirmed unanimously)
- No Dockerfile or Railway config exists
- Full todo list sent every request (scalability concern)
- No conversation history in NL interface

Where CC reviewers disagreed:

- **Gemini rated CC Code Quality 8/10** vs Codex at 6/10. Gemini praised hooks and type safety; Codex penalized missing validation layers.
- **Gemini rated CC Maintainability 9/10** vs Codex at 6/10. Largest single-dimension spread (3 points). Gemini focused on readability; Codex focused on change-safety.
- **Bug count disparity**: Gemini found 3 CC bugs, Codex found 7, Claude found 11. Gemini's optimism pattern continues from R1/R2.

Gemini's optimism pattern (cross-round):

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

Gemini consistently scores +0.7 to +1.0 above the Codex/Claude average, primarily by finding fewer bugs and weighing existing code quality more heavily than missing features.

The missing Gemini SPIR review: impact analysis

The Gemini SPIR review would have significantly affected the overall averages. Based on Gemini's established patterns:

1. **Bug score**: Gemini would likely have found 2-4 SPIR bugs (vs Claude's 14). Estimated bug score: 8-9. This would change the Codev bug average from 4.5 (n=2) to ~5.7-6.0 (n=3).
2. **Dimension scores**: Gemini typically scores SPIR +1-2 above CC. Estimated Code Quality 9, Maintainability 9-10, Tests 7, Extensibility 8, NL 9, Deployment 3.
3. **Overall impact**: The estimated Codev overall with imputed Gemini scores is 6.7 (vs 6.1 without), moving the delta from +0.4 to +0.9 — closer to R2's +1.2.

Key Takeaways

1. SPIR's testing advantage is the most consistent finding across all rounds

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**

SPIR consistently produces 3-4x more test lines and covers more dimensions (component tests, validation tests, hook tests). CC's testing is stuck at 0.26:1 ratio across all rounds. **Testing is where SPIR's consultation process most reliably delivers value.**

2. More code creates more bug surface area — a new tension

R3 reveals a paradox: SPIR's defensive patterns (validation, rate limiting, confirmation dialogs) add code surface area, and that surface area has bugs of its own. Claude found bugs in `nl-validation.ts` (mutates input, accepts invalid dates), `rate-limit.ts` (memory leak), and `ConfirmDialog` (a11y issues). The defensive code is net-positive (it catches real problems), but it isn't bug-free.

Implication: Consultation should include a "defense code review" pass — reviewing the validation and error-handling code itself, not just the business logic.

3. Deployment Readiness is prompt-dependent, not protocol-dependent

Neither codebase produced a Dockerfile in R3 (or R2). R1's SPIR Dockerfile was likely coincidental. The prompt says "deploy-ready for Railway" but doesn't specify "include a Dockerfile." Both methodologies interpret this as "make it deployable" (which `next build && next start` technically satisfies) rather than "include container infrastructure."

Recommendation: Future comparisons should either (a) explicitly require a Dockerfile in the prompt, or (b) continue excluding Deployment from scoring, as it's not measuring methodology quality but prompt interpretation.

4. Reviewer sample size matters more than expected

R3 demonstrates how sensitive averaged scores are to reviewer composition. With 3 CC reviewers and 2 Codev reviewers, one extreme scorer (Claude's 1/10 on Codev bugs) disproportionately affects the average. In R2 with 3/3 reviewers, Gemini's optimism balanced Claude's thoroughness.

Recommendation for future rounds: (a) Ensure all 6 reviews complete (manage Gemini quota by spacing consultations and reviews). (b) Consider median scores instead of means, which are more robust to outliers. (c) Budget Gemini API capacity for the review phase — SPIR consultations should use a different API key or wait until review phase to consume quota.

5. Codev leads on every dimension except bugs (and bugs are an outlier)

Excluding the anomalous bug score:

Dimension	Delta	Consistent with R1/R2?
Code Quality	0.0	Narrowed (was +1.0-1.3)
Maintainability	+0.2	Narrowed (was +0.3-0.7)
Tests	**+2.0**	Consistent (R1: +3.7, R2: +1.0)
Extensibility	+0.8	Consistent (R1: +1.0, R2: +1.0)
NL Interface	+0.7	Consistent (R2: +1.0)
Deployment	+0.8	Both low; small delta
Bugs	-2.2	Inverted (R2: +2.7) — methodological artifact

The non-bug dimensions show Codev maintaining its advantages, albeit with narrower margins on Code Quality and Maintainability. The bug score inversion is driven by Claude's aggressive SPIR review and the missing Gemini review, not by a genuine quality regression.

6. CC is getting slightly better across rounds

CC's overall has been stable at 5.7-5.9 across all rounds. But looking at specific dimensions: Code Quality improved from 6.3 to 7.0, and Extensibility recovered from 5.0 to 5.7. Claude as a base model may be improving at code generation over time (Opus 4 in R1 → Opus 4.6 in R3).

Summary: When Does Codev Pay Off?

Dimension	Codev advantage held in R3?	Notes
Bugs	**Inconclusive**	Anomalous -2.2 delta driven by reviewer asymmetry; consensus bugs favor Codev (3 vs 5)
Code Quality	Neutral (0.0)	Narrowed from R2's +1.3; CC may be catching up
Maintainability	Marginal (+0.2)	Both small, readable codebases

Dimension	Codev advantage held in R3?	Notes
Tests	**Yes (+2.0)**	Most consistent advantage across all rounds; 4.3x more test lines
Extensibility	**Yes (+0.8)**	Better abstractions, validation layer
NL Interface	**Yes (+0.7)**	Validation pipeline, clarification, rate limiting
Deployment	Marginal (+0.8)	Both very low; neither deploy-ready

Bottom line: R3 confirms SPIR's advantages in testing, extensibility, and NL robustness. The overall delta narrowed from R2's +1.2 to R3's +0.4 (or +0.9 with estimated Gemini scores), driven primarily by the bug score anomaly. The methodological lesson is clear: **all 6 reviews must complete for reliable scoring**, and Gemini API quota must be budgeted across the full experiment, not just the build phase.

Appendix: Raw Review Outputs

Reviewer	CC	Codev
Gemini	`/tmp/gemini-vibe-r3.txt`	Failed (429 quota exhaustion, 4 retries)
Codex	`/tmp/codex-vibe-r3.txt`	`/tmp/codex-spir-r3.txt`
Claude	`/tmp/claude-vibe-r3.txt`	`/tmp/claude-spir-r3.txt`

Raw reviews stored as `consult` background task outputs in builder worktree.

Appendix: Round 1 and Round 2 Results

Round 1

Dimension	R1 CC	R1 Codev	R1 Delta
Code Quality	6.7	7.7	+1.0
Maintainability	7.0	7.7	+0.7

Dimension	R1 CC	R1 Codev	R1 Delta
Tests	4.0	7.7	+3.7
Extensibility	5.7	6.7	+1.0
NL Interface	6.0	6.0	0.0
Deployment	6.0	8.0	+2.0
Overall	**5.9**	**7.2**	**+1.3**

Round 2

Dimension	R2 CC	R2 Codev	R2 Delta
Bugs	4.7	7.3	+2.7
Code Quality	6.3	7.7	+1.3
Maintainability	7.3	7.7	+0.3
Tests	5.0	6.0	+1.0
Extensibility	5.0	6.0	+1.0
NL Interface	6.0	7.0	+1.0
Overall	**5.7**	**7.0**	**+1.2**

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