Claude Code vs Codev: Round 2 Comparison

2026-02-14

Contents

Claude Code vs. Codev: Todo Manager Comparison — Round 2

Date: 2026-02-14 **PRs**: Claude Code PR #1 | Codev PR #1 **Reviewers**: Claude Opus 4.6, GPT-5.3 Codex, Gemini 3 Pro

Methodology

Identical to Round 1, with one key change: the prompt now **explicitly requires Gemini 3.0 Flash** as the NL backend. In Round 1, both builders fell back to regex-based NL parsing despite the prompt requesting a "conversational interface." The updated prompt reads:

> Use Gemini 3.0 Flash as the NL backend — do NOT use a simple grammar/regex parser. The NL interface should understand arbitrary phrasing, handle ambiguity, and support complex queries naturally.

Both builders received the same base prompt. The Codev builder received the additional porch strict-mode addendum. Both ran as Claude instances with `--dangerously-skip-permissions` in fresh GitHub repos.

Additional change: The Codev repo was initialized with the codev CLI v2.0.0-rc.69, which includes the stateful reviews with rebuttal mechanism (PR #246 / Issue #245). An earlier attempt using pre-#246 codev got stuck in a 5-iteration spec review loop due to Codex repeatedly blocking with rotating concerns — the rebuttal mechanism fixed this.

Scorecard

Individual Reviewer Scores

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

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.

Averaged Scores

Dimension	CC (avg)	Codev (avg)	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**

Round 1 vs Round 2 Comparison

Dimension	R1 CC	R2 CC	R1 Codev	R2 Codev
Code Quality	6.7	6.3	7.7	7.7
Maintainability	7.0	7.3	7.7	7.7
Tests	4.0	5.0	7.7	6.0
Extensibility	5.7	5.0	6.7	6.0
NL Interface	6.0	6.0	6.0	7.0
Overall	**5.9**	**5.9**	**7.2**	**6.9**

R1 did not score bugs as a separate dimension. R2 overall includes bugs.

Quantitative Comparison

Metric	CC	Codev
Source lines (excl. tests)	1,033	1,567
Test lines	271	1,149
Test-to-code ratio	0.26:1	0.73:1
Test files	2	4
Component tests	109 lines	0
Integration tests	0	0
Git commits	2	15
Documentation artifacts	0	spec + plan + review + 5 consultation files

---

Bug Sweep Synthesis

Claude Code Bugs (confirmed by 2+ reviewers)

Bug	Severity	Found by	Description
`useSyncExternalStore` snapshot identity bug	Critical	Gemini, Claude	`store.ts:18-20`: `getSnapshot()` calls `JSON.parse` every time, creating new references on every render. Defeats `useSyncExternalStore` change detection, risking infinite re-renders.
Unhandled errors in API route	High	All 3	`route.ts:116,142,165`: No try/catch around `request.json()`, `generateContent()`, or `JSON.parse(text)`. Any failure produces unhandled 500s.
Query filters silently dropped	High	Codex, Claude	API schema supports `search`, `dueBefore`, `dueAfter`, but `page.tsx:70-78` only applies `status`/`priority`. "Show todos due this week" silently does nothing.
Ambiguous title matching	Medium	All 3	`todos.ts:103-107`: `findTodoByTitle` returns first partial match. "Delete buy" with "Buy groceries" and "Buy milk" is nondeterministic.
Corrupt localStorage crashes app	Medium	Codex, Claude	`todos.ts:13`: raw `JSON.parse()` with no try/catch or shape validation.
No API rate limiting	Medium	All 3	`route.ts:107`: POST endpoint exposed with no auth, no rate limiting.
Cross-tab race condition	Medium	All 3	`store.ts` has no `storage` event listener. Concurrent tabs overwrite each other.
Misleading error message	Low	Codex, Claude	`ChatInterface.tsx:72-79`: All errors show "Check GEMINI_API_KEY" regardless of actual cause.

Codev Bugs (confirmed by 2+ reviewers)

Bug	Severity	Found by	Description
`getTodos()` no shape validation	High	All 3	`storage.ts:12-14`: Only checks `Array.isArray()`, trusts each element is valid `Todo`. Corrupt data breaks UI.
Stale closure race in `useChat`	High	Codex, Claude	`useChat.ts:32`: `messages` captured from render; rapid sends can lose assistant responses from history.
No `dueDate` format validation	Medium	Claude	Gemini could return "next Tuesday" or "2026-13-45"; stored verbatim, renders as "Invalid Date".
`UPDATE_TODO` allows empty updates	Medium	Codex, Claude	`actionExecutor.ts:93-130`: Validates `updates` is object but allows no fields. Reports "Updated" when nothing changed.
Empty state detection incomplete	Medium	Codex	`TodoList.tsx:18`: Only checks status/priority filters. Date-filtered empty results show "No todos yet" instead of "No matching todos."
Prompt self-contradiction	Low	Claude	System prompt says "no code fences" but examples are wrapped in code fences, sometimes confusing the model. `parseAction` has a defensive fence stripper.

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 message is standalone — follow-ups like "also make it high priority" don't work. - **No multi-tab sync**: Neither listens for `storage` events. - **No rate limiting**: Both API routes are open. - **No E2E/Playwright tests**: Neither includes browser-level tests. - **No XSS vulnerabilities**: React's auto-escaping protects both (unanimously confirmed).

Bug Quality Assessment

The bug profiles tell a story about what each methodology produces — and what it misses.

By the numbers:

Metric	CC	Codev
Total consensus bugs	8	6

Metric	CC	Codev
Critical	1	0
High	2	2
Medium	4	3
Low	1	1

Claude Code's bugs are more dangerous. The `useSyncExternalStore` snapshot identity bug is a Critical-severity defect that can cause infinite re-render loops — this is a fundamental misuse of a React API that could make the app unusable. Codev has no Critical bugs. The consultation process likely caught this class of error during review.

Claude Code has more "incomplete wiring" bugs. The query filters being silently dropped (API supports date filters, UI ignores them) and the unhandled errors in the API route both point to the same root cause: features were built without end-to-end verification. The vibe builder wrote an API schema with `search`/`dueBefore`/`dueAfter` capabilities, then forgot to wire them into the UI. Codev's phased approach (build layer by layer, consult at each checkpoint) makes this class of bug less likely.

Codev's bugs are subtler. The stale closure race in `useChat` only manifests under rapid interaction. The empty updates bug (`UPDATE_TODO` reporting success with no changes) is a UX lie rather than a crash. The prompt self-contradiction is handled defensively (fence stripper exists). These are the bugs that survive multi-agent review because they require understanding runtime behavior across multiple files — something static code review is weaker at.

Both share the same blind spot: scalability. Neither addresses token limits, conversation context, or rate limiting. All three reviewers flagged this for both codebases. This suggests neither vibe coding nor structured consultation naturally produces "how will this behave at scale?" thinking — it has to be explicitly prompted.

What CMAP catches vs. what it misses:

Bug class	Caught by CMAP?	Evidence
API misuse (wrong React pattern)	**Yes**	Claude Code has Critical `useSyncExternalStore` bug; Codev doesn't
Incomplete feature wiring	**Yes**	Claude Code silently drops query filters; Codev's action executor validates all paths
Stale closure / timing races	**No**	Codev still has the `useChat` race condition
Missing input validation	**Partially**	Codev validates action shapes but misses `dueDate` format

Bug class	Caught by CMAP?	Evidence
Scalability / architecture limits	**No**	Both send full todo list every request

The consultation process acts as a strong filter for **structural bugs** (wrong API usage, incomplete wiring, missing validation) but is weaker at catching **behavioral bugs** that only manifest during runtime interaction patterns.

Architecture Comparison

Claude Code R2

- **State**: `useSyncExternalStore` with manual listener pattern — conceptually correct but buggy implementation (snapshot identity)
- **NL**: Single API route sends user message + full todo list to Gemini, receives structured JSON actions via `responseSchema`. Multi-action support (array of actions).
- **Storage**: Two bare functions in `todos.ts` — `getAllTodos()`/`saveTodos()` — no error handling
- **Components**: 4 UI components + `ChatInterface`, flat hierarchy
- **Dependencies**: `@google/genai` only (minimal)

Codev R2

- **State**: Custom `useTodos` hook with filtering, sorting, validation
- **NL**: Three-layer architecture — `gemini.ts` (API call) → `actionExecutor.ts` (parse + validate + execute) → `useChat.ts` (orchestration). Discriminated union action types with runtime validation.
- **Storage**: Typed storage layer with `validateTitle`, `validateDescription`, write error handling
- **Components**: 7 components including `EmptyState`, `ConfirmDialog`-like patterns, `ChatMessage`
- **Dependencies**: `@google/genai` + `vitest` (test runner)

Key architectural advantage of Codev: The action executor with discriminated unions (`ADD_TODO | UPDATE_TODO | DELETE_TODO | LIST_TODOS | CLARIFY`). Each action type is independently validated with runtime checks before execution. The `parseAction` function strips markdown fences and validates JSON shape. This is significantly more robust than Claude Code's loose `type/data/filters` contract.

Key architectural advantage of Claude Code: Multi-action responses. The Gemini schema returns an `actions` array, allowing "delete all completed" to produce multiple delete actions in one response. Codev's single-action design forces "I can only delete one at a time" responses.

NL Interface Comparison

This is the **major improvement in Round 2** — both implementations now use Gemini 2.0 Flash, compared to Round 1 where both fell back to regex.

Capability	Claude Code R2	Codev R2
Gemini Flash backend	**Yes**	**Yes**
Structured output schema	**Yes** (`responseSchema`)	**Yes** (`responseMimeType: json`)
Multi-action support	**Yes** (actions array)	No (single action)
Runtime validation of AI output	No	**Yes** (discriminated union + `parseAction`)
Ambiguity resolution	No (first match)	**Yes** (`CLARIFY` action type)
Context (todo list sent)	Full list, pretty-printed	Full list, pretty-printed
Conversation history	None	None
Markdown fence defense	No	**Yes** (fence stripper in `parseAction`)
Fallback on Gemini failure	Generic error	Generic error

Verdict: Codev's NL architecture is more robust due to runtime validation and the CLARIFY action pattern. Claude Code's multi-action support is a useful feature Codev lacks. Neither has conversation memory or token optimization.

Test Quality Deep Dive

Claude Code R2 (271 lines, 2 files)

- `todos.test.ts` (162 lines): CRUD operations, filtering, `findTodoByTitle` — good coverage of the data layer
- `components.test.tsx` (109 lines): `TodoForm`, `TodoItem`, `TodoFilters` rendering and interactions

Not tested: API route, `ChatInterface`, store hook, error states, Gemini integration, edge cases (corrupt localStorage, ambiguous matches).

Codev R2 (1,149 lines, 4 files)

- `actionExecutor.test.ts` (425 lines): All action types for `parseAction` and `executeAction`, including edge cases (missing fields, invalid types, markdown fence stripping)
- `storage.test.ts` (532 lines): CRUD, validation, filtering, sorting, localStorage failures, quota exceeded, corrupted data
- `api-chat.test.ts` (185 lines): API route validation errors, rate limiting, Gemini errors, config errors
- `setup.test.ts` (7 lines): Environment verification placeholder

Not tested: Components (zero), hooks (zero), E2E flows, integration tests.

Comparison with Round 1

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

Notable shift: Claude Code R2 added component tests (absent in R1), while Codev R2 lost both component and integration tests compared to R1. Codev's test focus shifted to deep unit testing of the action executor and storage layers.

Reviewer Agreement Analysis

Where all three reviewers agreed:

- Codev has better code architecture (action executor pattern praised unanimously)
- Both use Gemini Flash (no regex) — the explicit prompt worked
- Neither has XSS vulnerabilities
- Both send full todo list every request (scalability concern)
- Both lack conversation history in the NL interface
- localStorage validation is weak in both

Where reviewers disagreed:

- **Gemini rated Codev Code Quality 9/10** vs Codex/Claude at 7/10. Gemini emphasized the discriminated union pattern and separation of concerns; Codex/Claude penalized type-casting issues and the missing component tests.
- **Gemini rated Codev NL 8/10** vs Codex at 6/10. Gemini praised the CLARIFY pattern and fence stripping; Codex focused on the lack of fallback and single-action limitation.
- **Claude rated Codev Tests 7/10** vs Gemini at 5/10. Claude emphasized test depth (storage: 532 lines); Gemini penalized the complete absence of component/hook tests.

Gemini's optimism pattern:

As in Round 1, Gemini consistently scores higher than Codex and Claude, particularly on code quality. Gemini appears to weight "what exists and works well" while Codex/Claude weight "what's missing."

Key Takeaways

1. Gemini Flash integration: the explicit prompt worked

Both builders successfully integrated Gemini Flash as the NL backend. In Round 1, both fell back to regex. The change: explicitly stating "Use Gemini 3.0 Flash as the NL backend — do NOT use

a simple grammar/regex parser." **Lesson**: AI builders need specific technology requirements, not abstract quality goals.

2. Codev produces fewer and less severe bugs

Claude Code: 8 consensus bugs including 1 Critical. Codev: 6 consensus bugs, none Critical. The consultation process acts as a strong filter for structural bugs (wrong API usage, incomplete wiring) but is weaker at catching behavioral bugs that only manifest during runtime.

3. Codev's NL architecture is superior — and now it shows in the scores

Codev's action executor with discriminated unions and runtime validation is objectively more robust. The NL delta is +1.0 (7.0 vs 6.0), a new advantage that didn't exist in R1 when both used regex. The consultation process helped produce a more defensive NL architecture (CLARIFY action type, fence stripping, runtime validation).

4. Codev's code quality advantage is consistent across rounds

+1.3 in R2, +1.0 in R1. The consultation process consistently produces better architecture: discriminated unions, separated concerns, typed storage layers. This is Codev's most reliable advantage.

5. Codev's testing advantage narrowed but still holds

R1: +3.7 test delta. R2: +1.0. Codev R2 has 4.2x more test lines than Claude Code R2 (vs 7.4x in R1). Codev shifted focus to deep unit testing of core logic (action executor, storage) but dropped component and integration tests entirely. Claude Code R2 actually added component tests (absent in R1).

6. The rebuttal mechanism (Issue #245) was critical

The first Codev R2 attempt got stuck in a 5-iteration spec review loop — Codex blocking with rotating concerns while Gemini and Claude approved. After rebuilding the codev CLI with the stateful review/rebuttal mechanism (PR #246), the Codev builder completed successfully. Without this fix, Codev R2 would not have finished at all.

7. Both share the same fundamental scalability limitation

Every reviewer for both codebases flagged: full todo list serialized in every Gemini request, no conversation history, no token budgeting. This is a shared blind spot that neither the Claude Code approach nor the Codev consultation process addresses.

Summary: When Does Codev Pay Off?

Dimension	Codev advantage held in R2?	Notes
Bugs	**Yes** (+2.7)	0 Critical vs Claude Code's 1 Critical; fewer total bugs (6 vs 8); CMAP catches structural bugs
Code Quality	**Yes** (+1.3)	Discriminated unions, runtime validation, separation of concerns
Maintainability	Marginal (+0.3)	Both are small, readable codebases
Tests	**Yes** (+1.0)	More test lines, deeper coverage of core logic
Extensibility	**Yes** (+1.0)	Better abstractions, but both limited by localStorage
NL Interface	**Yes** (+1.0)	Action executor pattern is more robust

Codev leads on every scored dimension, with bugs showing the largest delta (+2.7). The consultation process consistently produces fewer and less severe bugs, better architecture, more thorough testing, and more robust NL integration. The overall +1.2 delta (7.0 vs 5.7) represents a meaningful and repeatable quality advantage.

Appendix: Raw Review Outputs

Reviewer	CC	Codev
Gemini	`/tmp/gemini-vibe-r2.txt`	`/tmp/gemini-spir-r2.txt`
Codex	`/tmp/codex-vibe-r2.txt`	`/tmp/codex-spir-r2.txt`

Reviewer	CC	Codev
Claude	`/tmp/claude-vibe-r2.txt`	`/tmp/claude-spir-r2.txt`

Appendix: Round 1 Results (for comparison)

Dimension	R1 CC	R1 Codev	R1 Delta
Code Quality	6.7	7.7	+1.0
Maintainability	7.0	7.7	+0.7
Tests	4.0	7.7	+3.7
Extensibility	5.7	6.7	+1.0
NL Interface	6.0	6.0	0.0
Overall	**5.9**	**7.2**	**+1.3**

Full Round 1 report: `codev/resources/vibe-vs-spir-comparison-2026-02.md`