Sprint 1 Reflection: AI-Assisted Development with Cursor Rules

Qiushi Liang, Zhiping Zhang

February 27, 2026

1 How the Rules File Bridges PRD/Mockups to Implementation

A PRD and mockups define *what* to build, but leave a gap before the AI can produce code that fits the project. Our `.cursorrules` file bridges this gap by translating product intent into enforceable constraints across three dimensions.

Requirements → Architecture. The PRD specifies "secure file upload" but not how to organize the Express backend. The rules file mandates a strict `routes → controllers → services` pattern. With rules active, the AI created properly separated files (`routes/files.ts`, `controllers/fileController.ts`, `services/fileService.ts`) with clear responsibilities, including database persistence and rollback logic. Without rules, it produced a similar but incomplete structure—no DB write, no auth middleware, and a redundantly named `routes/fileRoutes.ts`.

Mockups → UI Components. The mockup shows a three-panel layout with a file sidebar, purple accent colors, and navigation tabs. With rules, the AI reproduced this layout faithfully: a `FILES` sidebar with purple-highlighted selections, Editor/History/Settings tabs, and a compact upload area. Without rules, it built a generic centered drag-and-drop zone using indigo colors—functional, but unrecognizable from the design.

Conventions → Consistency. The rules specify camelCase API responses mapped from snake_case DB fields. With rules, the AI returned `fileName` and `uploadedAt`; without, it returned `filename` (all lowercase). Small inconsistencies like this compound across a codebase and create friction for frontend consumers.

2 How the Scrum Setup on GitHub Helps Organize AI-Assisted Development

Issue-scoped prompts. Our GitHub Issues board decomposes the project into focused units (e.g., Issue #4: "implement secure .py file upload"). LLMs perform best with well-defined tasks. Instead of "build the backend," we point the AI to a single issue with clear acceptance criteria. The rules file reinforces this by requiring every commit to reference an issue (`feat(upload): ... #4`).

Sprint boundaries as checkpoints. Organizing issues into sprints (Sprint 1: #2–#6, Sprint 2: #7–#12) creates natural evaluation points. We verify that foundational pieces—schema, auth, upload, LLM engine—work together before building the editor UI on top.

Traceability. Every line of AI-generated code traces to a specific issue, branch, and PR. When a bug appears, we can identify which issue introduced it. This audit trail is especially valuable when the developer hasn't written every line by hand.

Testing as a quality gate. The rules require `npm run lint`, `npm run test`, and `npm run test:e2e` to pass before any issue is marked done. With rules, the AI produced 22 tests including explicit unauthenticated-request coverage—a requirement it skipped entirely without the rules file.

3 What We Would Add or Change for Sprint 2

Sprint 2 focuses on the interactive editor (Issues #7–#10), which introduces new challenges. We would add:

- **CodeMirror extension rules** — how to structure decorations, view plugins, and state fields; mandate `Decoration.line()` over DOM manipulation; naming conventions for extension files.
- **API contract examples** — concrete request/response payloads for `POST /api/files/:id/scan` so the AI generates accurate frontend integration without guessing response shapes.
- **State management guidelines** — whether to use React Context or Zustand for shared state (scan results, sidebar selection synced with editor scroll).
- **Accessibility requirements** — ARIA labels for sidebar items, keyboard navigation for the file list and instruction panel, minimum contrast ratios for highlighted lines.
- **Performance budgets** — scan rendering under 500 ms for files up to 1000 lines, debounce intervals for scroll-to-line, bundle size limits for CodeMirror.

Conclusion

The experiment confirmed that a well-crafted rules file transforms an AI assistant from a generic code generator into a project-aware collaborator. It encodes architecture, security, design intent, and testing standards the AI cannot infer from a brief prompt. Combined with disciplined Scrum practices, it produces code that arrives already aligned with team conventions, reducing review effort and rework.