Context Engineering: CursorAgent

🚀 Introduction

Welcome to **Context Engineering: CursorAgent**, your tactical playbook to building smarter, cleaner, and faster software using AI coding assistants. The era of vibe coding is dead. We're not here for spaghetti logic from hallucinated responses. We're here to **architect context**, hand it to our AI like a blueprint, and let it execute with surgical precision.

This guide is for developers, solopreneurs, and builders who want to make AI their *partner*, not just a parrot.

🧠 What Is Context Engineering?

Context Engineering is the practice of structuring all relevant information — rules, examples, goals, and supporting material — so that your AI coding assistant can perform at its best.

While "prompt engineering" is about phrasing, **context engineering builds the entire playground**: prompt, rules, references, and supporting knowledge. It equips the model with everything it needs to think and build like a dev that understands the full picture.

Context engineering also handles the **core flaws** of vibe coding: hallucinations, spaghetti structure, and poor scalability. Remember:

"Intuition doesn't scale. Structure does."

Andre Karpathy coined "vibe coding" to describe the early euphoric stage of AI codegen — minimal input, no validation. It was a dopamine rush. But now the honeymoon's over. We need AI that thinks in systems.

💀 Why Vibe Coding Falls Short

- 76.4% of devs surveyed don't trust AI code without human review (Codto Report).
- Most issues stem from lack of relevant context — the model simply doesn't know what it's missing.
- AI assistants aren't bad — but **blind reliance without structure is.**

What we need is a **repeatable, structured workflow** for handing the AI:

- The objective
- The constraints
- The tech landscape
- Past reference work
- Known limitations and things to avoid

This is where **context engineering** shines.

⚙️ The CursorAgent Workflow

Cursor is your weapon of choice. It's like VS Code on AI steroids. Here's how to turn it into an assistant that writes your projects from idea to final commit.

🧱 Step 1: Create the Context Folder

Structure your project with this file layout:

```
/context
├── rules.md          ← Global AI assistant rules
├── initial.md        ← Project kickoff description
├── examples/         ← Snippets, references
└── docs_links.txt    ← Links to APIs or SDK docs
```

rules.md

Define global behavior:

- Code style
- Testing strategy

- Naming patterns
- Known gotchas to avoid
- How you want tasks broken down

initial.md

Feature spec includes:

- Goal: “Build an AI agent that does X using Y”
- Tech stack
- APIs required
- Known edge cases

examples/

Put in:

- Similar past features
- UI/UX flows
- Code snippets

docs_links.txt

Include any:

- API docs
- SDK references
- GitHub links

🧪 Step 2: Prime the Cursor Agent

Open Cursor and paste:

```
Use the following context:
- `context/rules.md`
- `context/initial.md`
- `context/examples/`
- `context/docs_links.txt`
Generate a Product Requirements Prompt (PRP) that includes:
- Folder structure
- Key milestones
- APIs, packages, and tools
- Edge cases and validation requirements
```

Ask Cursor to respond with: **“PRP ready. Review below.”**

🧭 Step 3: Review the PRP

Skim and edit:

- Is anything missing?
- Are assumptions off?
- Do you need to add a doc or file?

Once ready:

```
Perfect. Begin implementation. Start with base folder structure and initial setup.
```

Now the AI knows what it’s doing and why.

🛠️ Step 4: Build Step-by-Step

Tell it:

```
Implement task 1 from the PRP using context in `rules.md`. Use code from `examples/` if needed.
```

Then continue task by task like a project manager.

🧩 Structured Output: Your Secret Weapon

AI fails when it guesses. Your job is to give it structure:

- Directory layout
- File naming rules
- Testing protocols
- Output formatting expectations

This is **baked into your PRP**. Whether you're using Cursor, Claude, Gemini CLI, or GitHub Copilot, context wins when it's structured.

🧱 Tools That Complement Cursor

- **Claude (Cloud Code):** Best for full planning + implementation via slash commands. Great for PRP generation and complex task flow.
- **Gemini CLI:** Ideal for fast CLI-based builds, especially paired with structured markdown prompts.
- **GitHub Copilot:** Use it in tandem for autocomplete and in-line generation — but supplement with strong context files.
- **Windsurf:** A Claude-based dev tool that supports custom agent commands. Great alternative to Cursor if you prefer agent scripting.
- **LangChain + RAG:** Advanced — allows you to stream in documentation or knowledge as live memory. Useful for internal tool builds.

Cursor is the nucleus. The others are orbiting tools you can pull in to power up your build depending on the use case.

🔐 Security Warning

If you're building with AI, **security can't be an afterthought**:

- Prompt injection
- Model poisoning
- Data leakage

Tools like **Snyk** have already started covering these AI-specific threats. Make sure your context engineering includes guardrails — especially when generating backend code, managing tokens, or calling third-party APIs.

✅ Bonus: Test Everything

Tell the AI to write tests alongside the build.

- Add this directive to rules.md
- Ask for Pytest, Jest, or your preferred framework
- Run tests locally and validate output — don't trust blindly

Write tests using Pytest for all critical functions. Use mock data when needed.

🎯 Cursor vs. Claude vs. Gemini CLI

Feature	Cursor	Claude (Cloud Code)	Gemini CLI
Context File Support	✅	✅	✅
PRP Generation	Manual prompt	/generate_prp	Scripted prompt
Output Style	Developer-driven	Agentic workflow	CLI-based commands
Best Use Case	VSCode AI workflow	End-to-end project planning	Fast terminal builds

---

🧰 Pro Tips

- Use Cursor's "Workspace View" to keep rules.md and initial.md pinned
- Validate all AI-generated code with tests or reviews
- Let AI generate plans, but you approve architecture

💡 Final Thoughts

Context engineering is **not a trick** — it's the next level in software development with AI. It gives you:

- Cleaner code
- Fewer hallucinations
- Better architecture
- Actual production-readiness

Cursor is your playground. Claude and Gemini are your sidekicks. Bring structure, give examples, define your rules — and watch AI become the best junior dev you've ever worked with.

📥 Want More?

Get the repo template, prompt guides, and email walkthroughs delivered to your inbox.
[Download the free starter kit now]
Stay sharp.
Built with 💻 and ☕ by Donte D. Willis
Follow me @dontecodes | @revenueripple