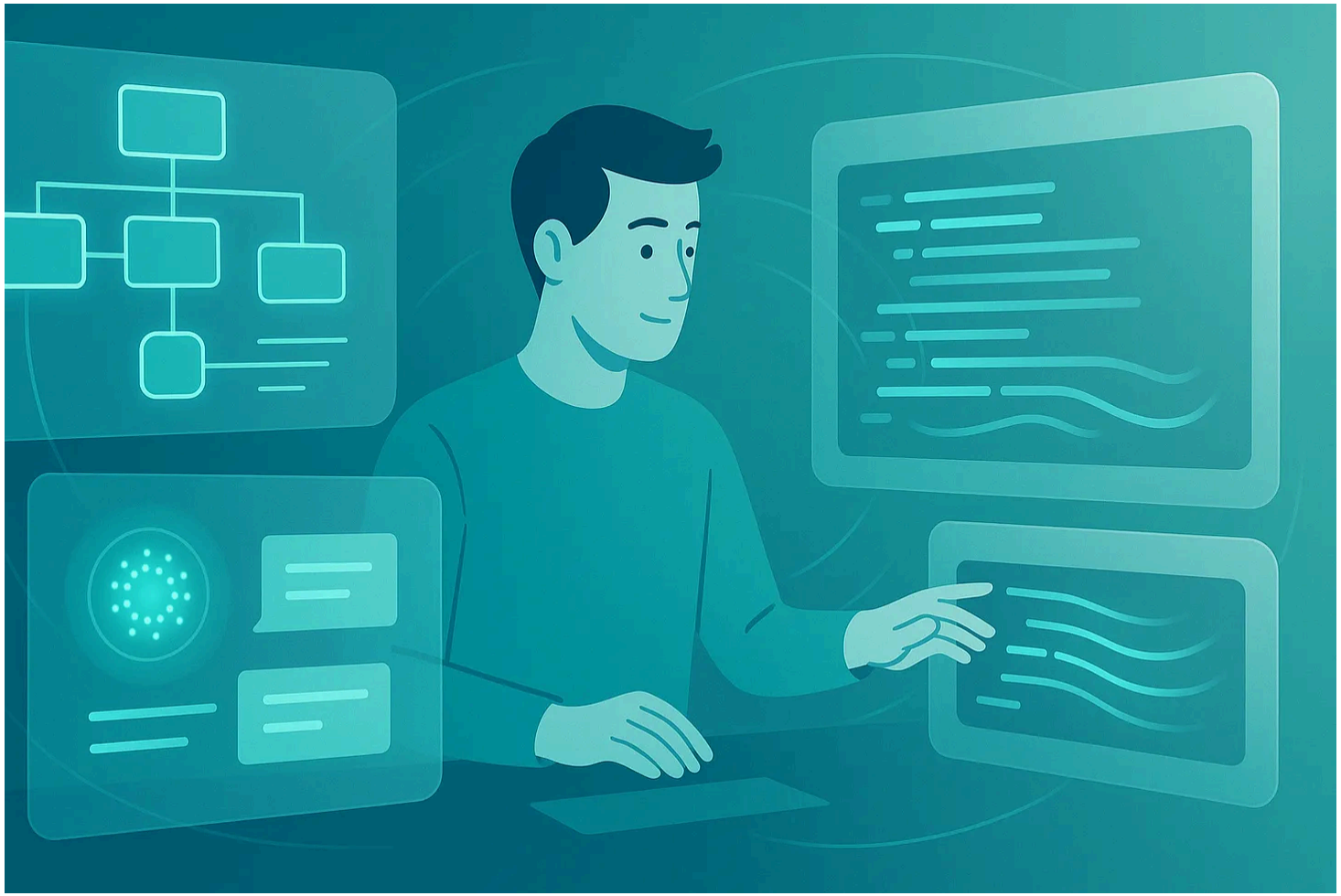
From “Vibe Coding” to “Vibe Software Engineering”

A Real-World Workflow for AI Code Generation

ARMANDO MAYNEZ

APR 25, 2025



Introduction: The Alluring Promise of AI Code Generation

You can't scroll through tech news or listen to a podcast these days without hearing about the transformative power of Artificial Intelligence, particularly Large Language Models (LLMs). The buzz is palpable, promising revolutions across industries. For

builders, creators, and developers like myself, one specific promise echoed louder than the rest, shimmering with almost magical potential: the dream of **effortless code generation**.

Imagine the scenario, one I played out in my mind countless times: You have a brilliant idea for an app. Instead of the traditional, often lengthy, process of planning, designing, coding, debugging, and iterating, you simply articulate your vision to an AI. You craft that one, perfect prompt, detailing the features, the user flow, maybe even the look and feel. You hit "Enter", lean back, and witness the AI construct your application, line by line, function by function. The dream wasn't just about speed; it was about transcending limitations – building complex software without needing deep expertise in every required language or framework, turning ideas into tangible products almost instantaneously.

This vision wasn't just idle fantasy; it felt excitingly close. New tools were launching, explicitly designed to be AI coding assistants or even full-blown automated developers. I was completely hooked by this allure. The possibility of rapidly iterating on ideas, building side projects in my limited spare time without getting bogged down in mundane or unfamiliar syntax, felt like unlocking a superpower. I envisioned skipping the steepest parts of the learning curve for new technologies, letting the AI handle the implementation details while I focused on the core concept.

Fueled by this intoxicating vision and brimming with excitement, I decided to put the promise to the test. I was ready to embrace the future of software development, armed with my ideas and the expectation that an AI could translate them directly into a working application. My journey into AI-powered coding began with this optimistic, if somewhat naive, belief in the power of the single prompt. The reality, as I soon discovered, would require a very different approach.

The Single-Prompt Illusion

Eager to turn the captivating vision of AI-driven development into reality, I didn't hesitate. I jumped straight into the battle, experimenting with several promising tools. My initial trials centered on **Manus, Lovable, and Cursor**. My strategy was exactly what the hype suggested: armed with my app concept, I fed each tool variations of a single, descriptive prompt outlining what I hoped to achieve. I hit enter, leaned back, and waited for the magic to unfold.

Unfortunately, the magic never quite materialized. Instead, I ran headfirst into a wall of frustrating limitations. While the tools certainly generated code (sometimes impressive volumes of it) the output consistently fell short of being a usable application, or even a reliably functional component. My attempts were plagued by a recurring set of problems:

- **Code That Wouldn't Run:** Often, the generated code looked superficially correct but was riddled with syntax errors, incorrect assumptions about libraries, or

fundamental logical flaws that prevented it from compiling or running.

- **Incomplete or Misaligned Functionality:** Features described in the prompt might be missing entirely, partially implemented, or interpreted in ways that didn't match my intent. The AI struggled to grasp the nuances and interdependencies of the requirements.
- **Integration Chaos:** Trying to combine different pieces generated by the AI, or integrating AI code into a larger structure, often resulted in a tangled mess of conflicting logic and broken dependencies.
- **The Debugging Quagmire:** Ironically, the time saved by *generating* code was often dwarfed by the time sunk into *debugging* it. Deciphering AI-generated errors and trying to guide the tool towards a correct solution by pasting error messages back felt like a slow, inefficient game of whack-a-mole.

It became painfully clear why the single-prompt approach was failing so spectacularly for anything non-trivial. Building a real application isn't just about translating features into lines of code; it demands architectural foresight, clear data flow, consistent state management, and a deep understanding of how different components interact. A high-level prompt, no matter how well-written, simply couldn't encapsulate that depth of context and specification. The AI, lacking a structured plan or a holistic view of the system, was essentially guessing, often incorrectly.

My initial burst of enthusiasm quickly deflated, replaced by a sobering dose of reality. The dream of effortlessly generating an app from a simple description felt miles away. This wasn't the revolution I'd envisioned; it was just a different, often more frustrating, way to produce buggy code. The realization hit me hard: The "single-prompt" approach was fundamentally flawed for the complexity I was aiming for. If I wanted AI to be a true partner in building something substantial, I needed to stop treating it like a magical code vending machine. I needed structure. I needed a plan. I needed a *better way*.

Embracing Structure - The PRFAQ Revelation

Staring at the jumble of incomplete, non-functional code generated by my initial prompts, the path forward seemed murky. The freewheeling, "just ask the AI" approach had led to a dead end. It became starkly clear: I needed discipline. I needed a framework. My mind immediately snapped back to my time at Amazon and a methodology that's practically part of the DNA there: the **PRFAQ**.

If you haven't encountered it, the PRFAQ (Press Release / Frequently Asked Questions) is a deceptively simple yet profoundly effective planning document. Before you embark on building *anything*, you write a press release as if your product has already launched successfully. This forces you to articulate, with crystal clarity, the customer problem,

your solution, and the unique value proposition. What makes it truly powerful are the accompanying FAQs: anticipating every tough question external customers might ask, and every critical query internal teams (engineering, marketing, legal) would raise. It’s a rigorous exercise in working backward from the end-user and ensuring absolute alignment on the *what* and *why* before a single resource is committed to the *how*. For anyone who's worked at Amazon, this structured thinking becomes second nature.

Suddenly, this felt like the missing piece. The PRFAQ demands the very clarity, scope definition, and user-centric focus that my earlier, vague prompts completely lacked. It could serve as the structured input, the detailed brief, that an AI might actually be able to work with effectively down the line. It was time to pivot from asking AI to *build* blindly to asking it to help me *plan* intelligently.

I turned to **Gemini 2.5 Pro**, but this time with a fundamentally different task. I fed it the essence of my app idea (the core problem, the target user, the desired outcome) and asked it to generate a draft PRFAQ.

Gemini delivered a surprisingly strong starting point. The press release captured the general excitement, and the FAQs touched on relevant areas. But this AI-generated draft wasn't the end goal; it was the skeleton. The crucial phase was the **iterative refinement**. I dove into the document, editing heavily. I sharpened the language in the press release to match *my* specific vision, rephrased customer benefits, and significantly expanded the FAQs, adding nuances Gemini couldn't have known and

addressing potential challenges I anticipated. This back-and-forth was incredibly productive. It wasn't just me editing; it was a collaborative process where I used Gemini's structured output as a catalyst for my own deeper thinking, occasionally prompting it for alternative phrasing or ideas for specific FAQ answers.

After several cycles of generation, review, and meticulous editing, I had a finalized PRFAQ document. It felt solid. It was ambitious yet grounded. It articulated precisely what I wanted to build, for whom, and why it mattered. This wasn't just paperwork; it was the bedrock of my project, the North Star providing the clarity and purpose that my initial, scattered efforts had desperately needed. The vague dream was starting to take concrete shape.

Building the Blueprint: “Vibe” Design & Planning

With the PRFAQ providing a solid "North Star," the next logical step was to translate that high-level vision into more concrete plans. I needed to define *how* the app would be built technically and *what* specific features the user would interact with, especially for the Minimum Viable Product (MVP). This meant creating two more essential documents: a System Design document and a set of User Stories. Continuing my new structured approach, I enlisted **Gemini 2.5 Pro** as my planning collaborator once more.

First, I tackled the **System Design Document**. The goal here was to outline the application's architecture: the key components, the technology stack, how data would

flow, and how different services would interact. Using my polished PRFAQ as the core input, I asked Gemini to generate a top-level System Design. Gemini provided a plausible initial draft, suggesting potential architectural patterns and technologies. However, this is where human expertise becomes absolutely critical. Designing robust systems involves nuanced trade-offs, considerations of scalability, maintainability, and cost that AI often simplifies. The draft required **significant refinement**. We went through **several revisions**, with me guiding the process, questioning choices, simplifying complexities, and ultimately shaping a technical blueprint I was confident in. It was a true collaboration: Gemini providing the initial structure and ideas, and me providing the critical analysis and architectural decisions based on experience and project goals.

Next came the **User Stories**. These are vital for defining the actual functionality from the perspective of the end-user, typically following the "As a [user type], I want to [perform an action], so that [benefit]" format. They ensure development stays laser-focused on delivering value. Feeding Gemini the context from both the PRFAQ and our refined System Design document, I tasked it with drafting User Stories specifically for the MVP. It generated a solid list covering the core functionality. But, as before, this was the start, not the end. I meticulously **edited the generated stories**, clarifying acceptance criteria, ensuring they aligned perfectly with the PRFAQ's promise, and adding crucial stories that had been missed. More than just editing, I used this phase to **brainstorm with Gemini**. We explored different facets of the MVP features, debated

priorities, and used the AI as a sounding board to crystallize the exact scope for the initial launch.

By the end of this phase, the vague idea had evolved significantly. I now possessed a powerful triad of planning documents:

The **PRFAQ**: Defining the *why* and *what* from the customer's perspective.

The **System Design**: Outlining the *how* from a technical perspective.

The **User Stories**: Detailing the *specific features* and user value for the MVP.

Each document was initiated with AI's help but heavily curated, refined, and validated by human oversight. This comprehensive blueprint felt robust. The path to implementation was no longer a guess; it was mapped out. I was finally ready to translate these plans into actual code.

The Master Plan - Orchestrating Implementation

With the comprehensive blueprint now established through the PRFAQ, System Design document, and User Stories, the abstract vision felt tangible. The 'what', 'why', and architectural 'how' were defined. But for an LLM to build software it requires more than just understanding the destination and the vehicle; it requires a detailed itinerary – a step-by-step guide for the journey. The final step before diving into code was to synthesize all this planning into an actionable **Implementation Plan**.

This felt like the ultimate test for my new AI-assisted workflow. Could an AI take these distinct, detailed planning documents and weave them into a coherent execution strategy? It was time to truly "**feed the beast.**" I gathered my finalized PRFAQ, the refined System Design document, and the prioritized User Stories for the MVP. I presented this entire package to **Gemini 2.5 Pro**.

My request was specific: analyze these documents and **generate a detailed, step-by-step implementation plan** to build the MVP. I needed a logical sequence of tasks, breaking down the larger features defined in the User Stories into manageable development steps, all while respecting the technical decisions laid out in the System Design. Gemini processed the inputs and delivered precisely that: a structured sequence of tasks, logically grouped, transforming the high-level plans into a concrete list of actions.

This initial plan from Gemini was impressive, providing a clear path forward. However, in complex projects, a second perspective can be invaluable for catching omissions or suggesting optimizations. To ensure robustness and add another layer of validation, I decided to perform a **cross-check using Claude 3.7**. I provided Claude with the implementation plan generated by Gemini (likely along with the source documents for full context) and asked it to review the plan critically: Were there missing steps? Could the sequence be improved? Were dependencies correctly identified? Essentially, I asked Claude to act as a peer reviewer for Gemini's plan.

Claude's review proved highly valuable. It offered suggestions for refining the task order, pointed out a couple of dependencies that weren't explicit in Gemini's plan, and helped break down a few larger tasks into more granular steps. This wasn't about pitting one AI against the other, but about leveraging the strengths of different models to achieve a more polished result. I carefully evaluated Claude's feedback, compared it with Gemini's original structure, and incorporated the suggestions that made the most sense, **refining and perfecting the plan** with my own judgment overlaying the AI recommendations.

The outcome of this multi-AI, human-guided process was the **master implementation plan**, a detailed, logically sequenced, and cross-validated roadmap (living in a file I'd soon call **DevPlan.md**). It broke down the entire MVP development into concrete, actionable tasks. This wasn't just a list; it felt like the operational orders derived directly from the strategic documents created earlier. The bridge from planning to execution was finally built. With this detailed guide in hand, the daunting task of coding felt organized and achievable. I was ready to start building.

Execution & Adaptation - Coding with Cursor

Armed with the detailed **DevPlan.md**, the phase shifted from planning to doing. It was time to translate the meticulously crafted steps into functional code. My primary tool for this execution phase was **Cursor**, leveraging its ability to understand project context and generate code. My core strategy was methodical: **one chat window per**

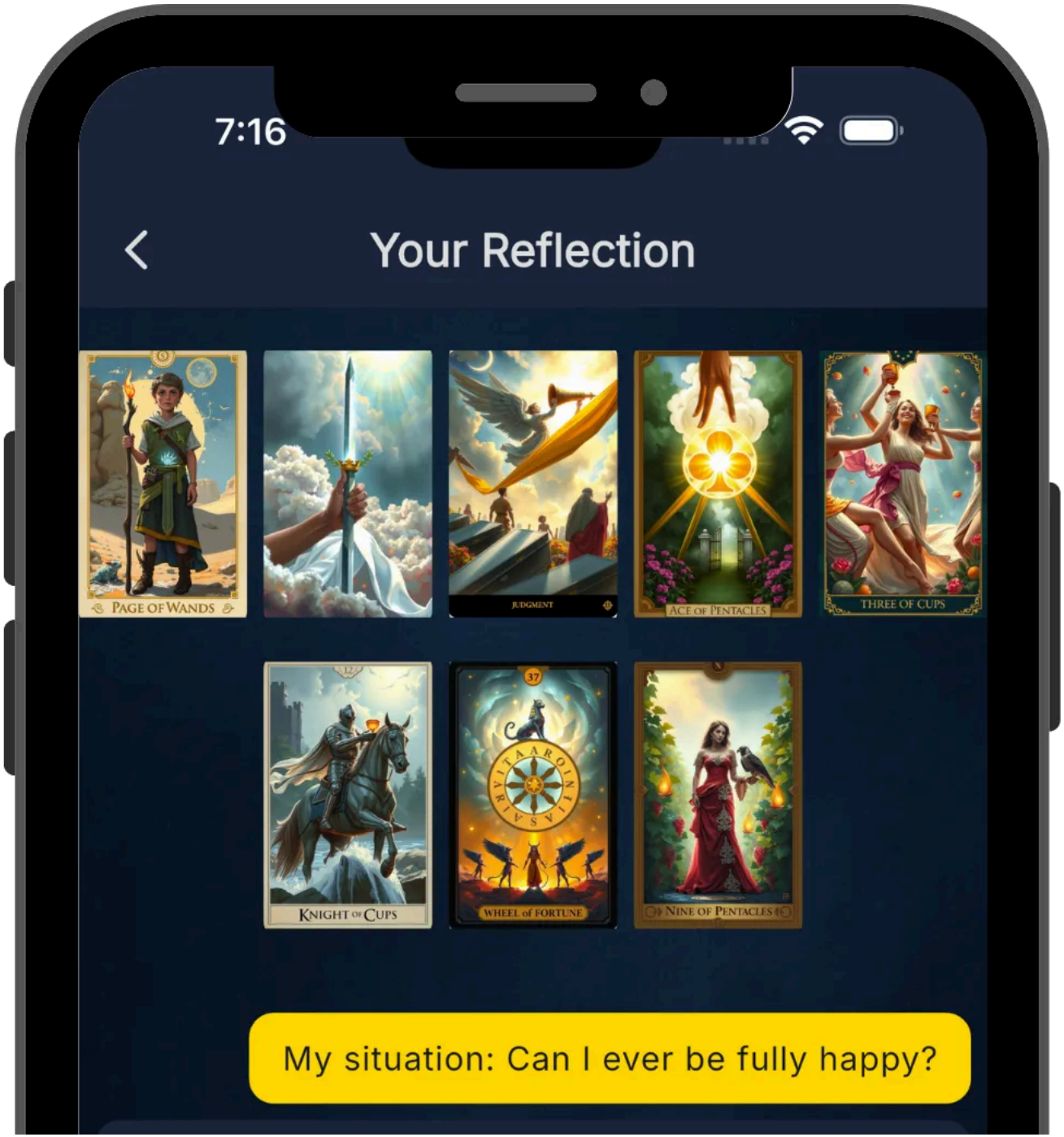
task. For each item listed in **DevPlan.md**, I'd open a dedicated chat in Cursor, feed it the task description (always including System Design and User Stories for context), and prompt it to generate the necessary code.

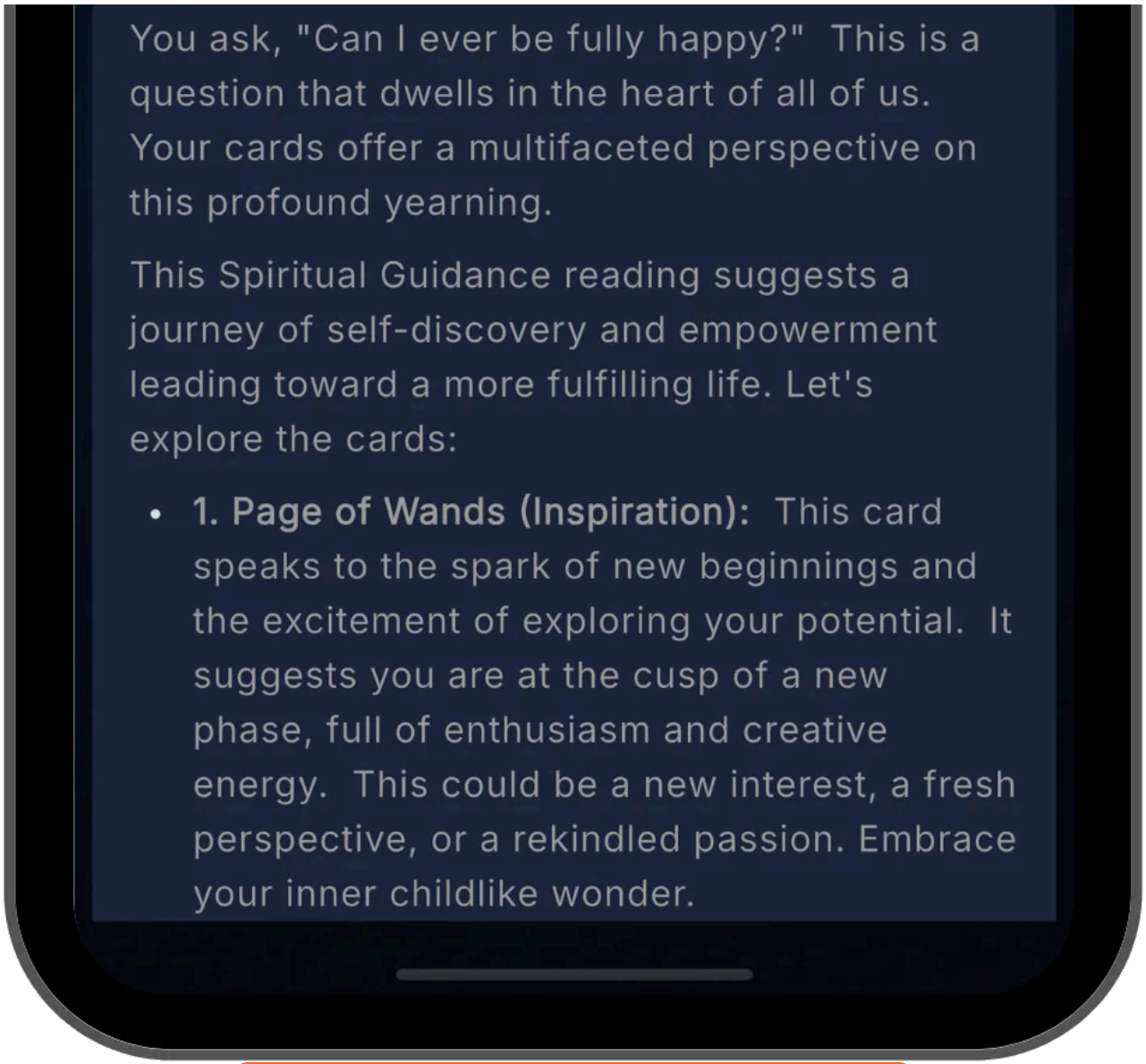
Initially, this felt like the smooth sailing I had originally hoped for. But reality soon introduced new complexities. While Cursor handled smaller, well-defined tasks reasonably well, I quickly ran into trouble with more substantial steps. Asking it to implement a complex feature in one go often resulted in buggy, incomplete, or logically flawed code, leading straight back into the **painful debugging sessions** I'd hoped to leave behind.

This led to **Lesson Learned #1: Embrace Recursive Planning for Complexity.** I realized that if a task felt too large or intricate for a single prompt, it probably was. The solution? When faced with such a task, I learned to pause the coding and first ask **Cursor itself** to *create a detailed, step-by-step implementation plan for that specific task.* Essentially, I was applying my planning methodology recursively. By breaking down the complex task into a mini-project with its own checklist within the main chat window, I could guide Cursor through smaller, more manageable steps, leading to much more reliable results. Recognizing *when* to double-click and create these sub-plans became a crucial skill, honed by experience and the occasional Cursor mishap.

As development progressed, another challenge emerged: keeping track of exactly what had been implemented versus what was initially planned, especially when minor

deviations or improvements occurred during coding. This brought me to **Lesson Learned #2: Document As You Go.** Simply marking a task "done" wasn't enough. After successfully completing a task (or a sub-task from a mini-plan), I developed the habit of asking Cursor to **document the specifics of the implementation directly within the DevPlan.md file** under the relevant task. This real-time documentation proved invaluable later, providing precise context for dependent tasks and ensuring the plan always reflected the current state of the codebase.


7:16
Your Reflection
PAGE OF WANDS
JUDGMENT
ACE OF PENTACLES
THREE OF CUPS
KNIGHT OF CUPS
WHEEL of FORTUNE
NINE OF PENTACLES
My situation: Can I ever be fully happy?



Finally, the structure provided by the initial planning phase extended to the development rhythm itself. **Lesson Learned #3: Phased Development and Rigorous Version Control Are Non-Negotiable.** As suggested by Gemini during the planning phase, the DevPlan.md was already divided into logical phases. At the completion of each phase, I enforced a strict routine: **Thorough Testing -> Git Commit -> Push to GitHub.** This disciplined cycle was absolutely essential. Why? Because, let's be honest, AI code generation isn't perfect. Cursor *would* occasionally mess things up, sometimes subtly, sometimes significantly. Having clean, tested commit points at the end of each phase provided a safety net, a known good state to revert to if needed, preventing catastrophic setbacks and ensuring steady, verifiable progress.

This execution phase was far from passive prompting. It was an active process of guiding the AI, adapting the plan, learning from mistakes, and establishing robust workflows for documentation and version control. The initial framework was proving its worth, not as a rigid set of rules, but as an adaptable system for navigating the complexities of AI-assisted development.

Ensuring Quality & Security

Following the disciplined rhythm of phased development, testing, and version control, the core features of the Minimum Viable Product (MVP) were functionally complete. The DevPlan.md, diligently updated by Cursor after each task, served as a living record of progress. According to the plan Gemini had helped structure, we had now

arrived at a critical juncture: the **Production Readiness** phase. This meant shifting focus from building *more* features to ensuring the existing application was stable, performant, secure, and truly ready for end-users.

Launching software involves more than just code that works; it requires crossing a threshold of quality and reliability. To navigate this crucial pre-launch phase systematically and avoid overlooking critical details, I decided a comprehensive **Production Readiness Checklist** was essential. And who better to help draft it than the AI that had been involved throughout the planning and documentation?

I gathered all the project artifacts: the foundational PRFAQ, the technical System Design, the defining User Stories, and, most importantly, the **DevPlan.md** which now reflected the *actual* state of the implemented code thanks to Claude's documentation efforts. I fed this rich context bundle to **Gemini** with a specific prompt: "Generate a comprehensive Production Readiness Checklist tailored to this application, based on all the provided documentation."

The result Gemini returned was **fantastic**. Because it had access to the specific architecture (Flutter + Firebase), features, and even implementation notes, the checklist wasn't a generic template. It was remarkably relevant, covering aspects like final integration testing, error handling verification, code cleaning, performance checks on key user flows, confirmation of logging mechanisms, and deployment procedure dry-runs; all derived directly from the context I provided.

A major pillar of production readiness is **security**. This wasn't something to be taken lightly, especially with user data involved. Given my specific tech stack of **Flutter and Firebase**, I needed to ensure I was adhering to current best practices. I tasked **Gemini again, this time explicitly leveraging its search capabilities**, to research and outline key security considerations for Flutter/Firebase applications. I asked it to look into areas like secure Firestore/Storage rules, proper authentication token handling, input validation (client-side and potentially Cloud Functions), protection against common vulnerabilities (like cross-site scripting if web components were involved), dependency security scanning, and best practices for managing API keys and sensitive configuration.

Gemini integrated these security-specific findings directly into the Production Readiness Checklist it had already generated. Now I had a single, unified document covering both general quality checks and targeted security measures for my specific stack. The final step involved methodically working through this checklist, performing verifications, running security scans, tightening Firebase rules, reviewing code for potential issues, and addressing any identified gaps.

This structured, AI-assisted approach to quality assurance and security hardening was incredibly effective. It transformed a potentially overwhelming pre-launch phase into a manageable process, leveraging the project's own history to ensure thoroughness. Completing that checklist didn't just mean the app was "done"; it meant it was *ready*, giving me much greater confidence to proceed towards launch.

AI Beyond the Code - Crafting the Brand & Marketing

While the core development was progressing, guided by the **DevPlan.md** and the iterative coding process, I realized that building a successful app involves more than just functional code. It needs a presence, a personality, and a plan to reach its audience. Astonishingly, the same structured, AI-assisted approach I was using for development proved equally powerful for these seemingly more creative tasks. I decided to run a **parallel track**, leveraging AI, specifically **Gemini 2.5 Pro**, for branding and marketing while the coding continued.

First, I tackled the **Marketing Plan**. Feeding Gemini the now-rich context of the PRFAQ (which already defined the target audience and value proposition), I asked it to draft a foundational marketing plan. This included identifying potential channels, suggesting initial messaging angles, and outlining key launch activities. As with the technical documents, this wasn't a final product but an excellent starting point that I refined with my own insights and goals.

Next, I focused on the app's identity. A consistent voice is crucial for user experience. I prompted Gemini to help create a **Tone of Voice guide**. Based on the app's purpose and target audience described in the PRFAQ, we defined key attributes: should it be formal or informal? Playful or serious? Encouraging or purely instructional? Gemini helped articulate these guidelines clearly.

Similarly, I needed a **Visual Style Guide**. While I wasn't expecting Gemini to design the UI itself, I asked it to suggest color palettes, typography styles, and general aesthetic principles that would align with the app's intended feel and target user. It provided several options and rationales, which helped me solidify the visual direction.

With the tone and visual style defined, I turned to **AI image generation** tools like **Flux or ChatGPT's DALL-E** for the actual creative assets. Crucially, I used the principles defined in the Tone of Voice and Visual Style guides (with Gemini's help refining the prompts) to generate the **app's logo, splash screen visuals, and other key art**. The AI wasn't just creating random images; it was generating assets aligned with the pre-defined brand identity.

The real magic, however, was integrating these branding elements back into the development process. I didn't just file these guides away; I actively used them. Within **Cursor**, I instructed it to **use the defined Tone of Voice for *every* single user-facing**

string: button labels, error messages, onboarding text, everything. Furthermore, I configured Cursor to treat the **Visual Style Guide as project rules**, ensuring that colors, fonts, and spacing adhered to the design specifications during code generation.

This parallel process of AI-assisted branding and marketing, tightly integrated with the AI-assisted development, ensured remarkable consistency. The app wasn't just functional; it was starting to look and feel cohesive and professional, reflecting the defined identity across both its code and its communication. The AI wasn't just a coder; it was becoming a collaborator across the entire product lifecycle.

The Result: A Professional App in Record Time

The culmination of this structured, AI-assisted journey wasn't just theoretical progress; it was a tangible, functional application. After navigating the phases of planning, execution, refinement, and quality assurance, the outcome was genuinely remarkable: a **fully working, professional-looking app, ready for launch.**

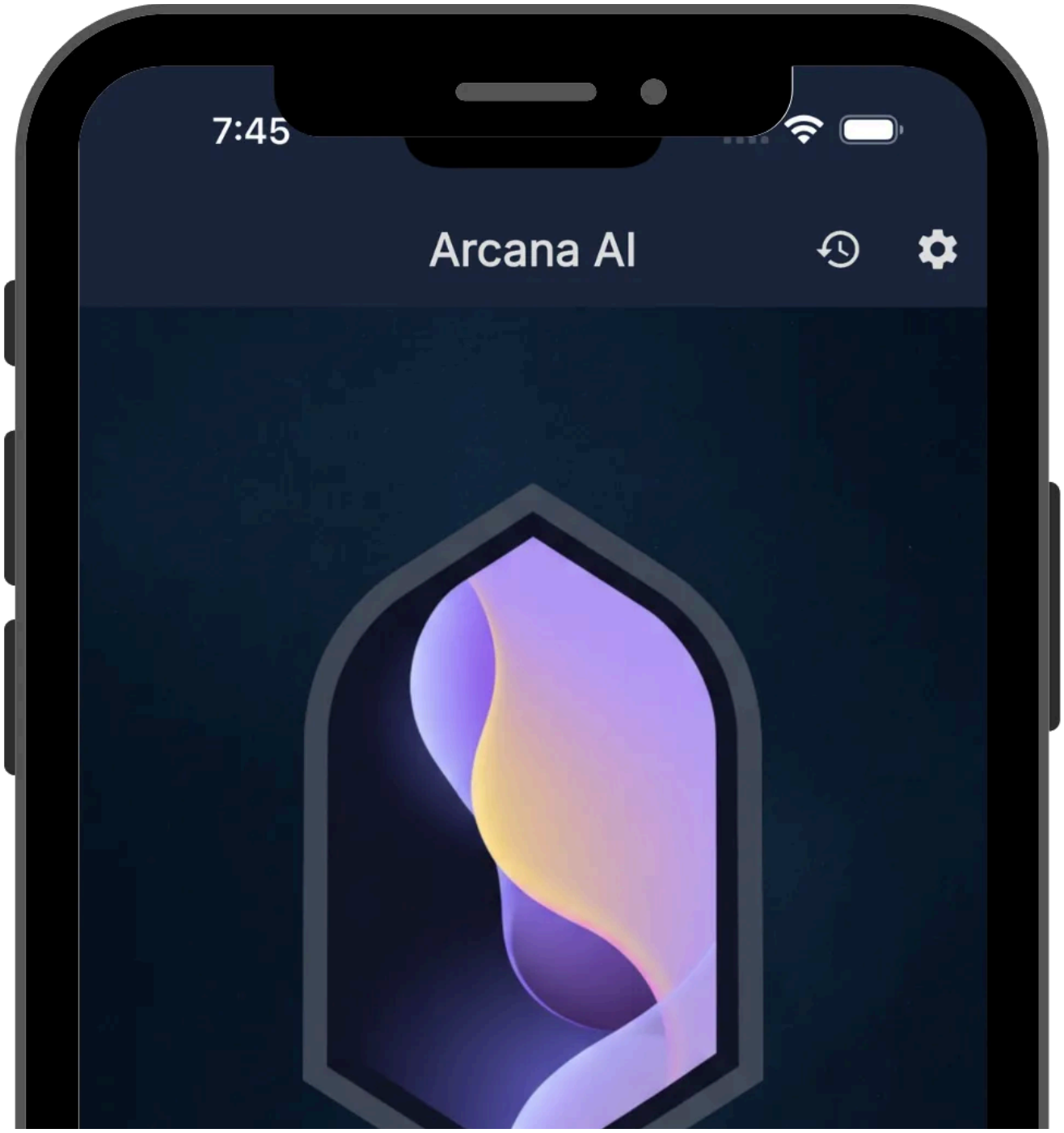
What truly astonished me was the timeframe. From the initial PRFAQ brainstorming session to having a production-ready MVP, the entire process took roughly **two weeks of focused effort in my spare time**. This pace would have been utterly unthinkable using traditional methods alone, especially considering the biggest hurdle I overcame: the tech stack.

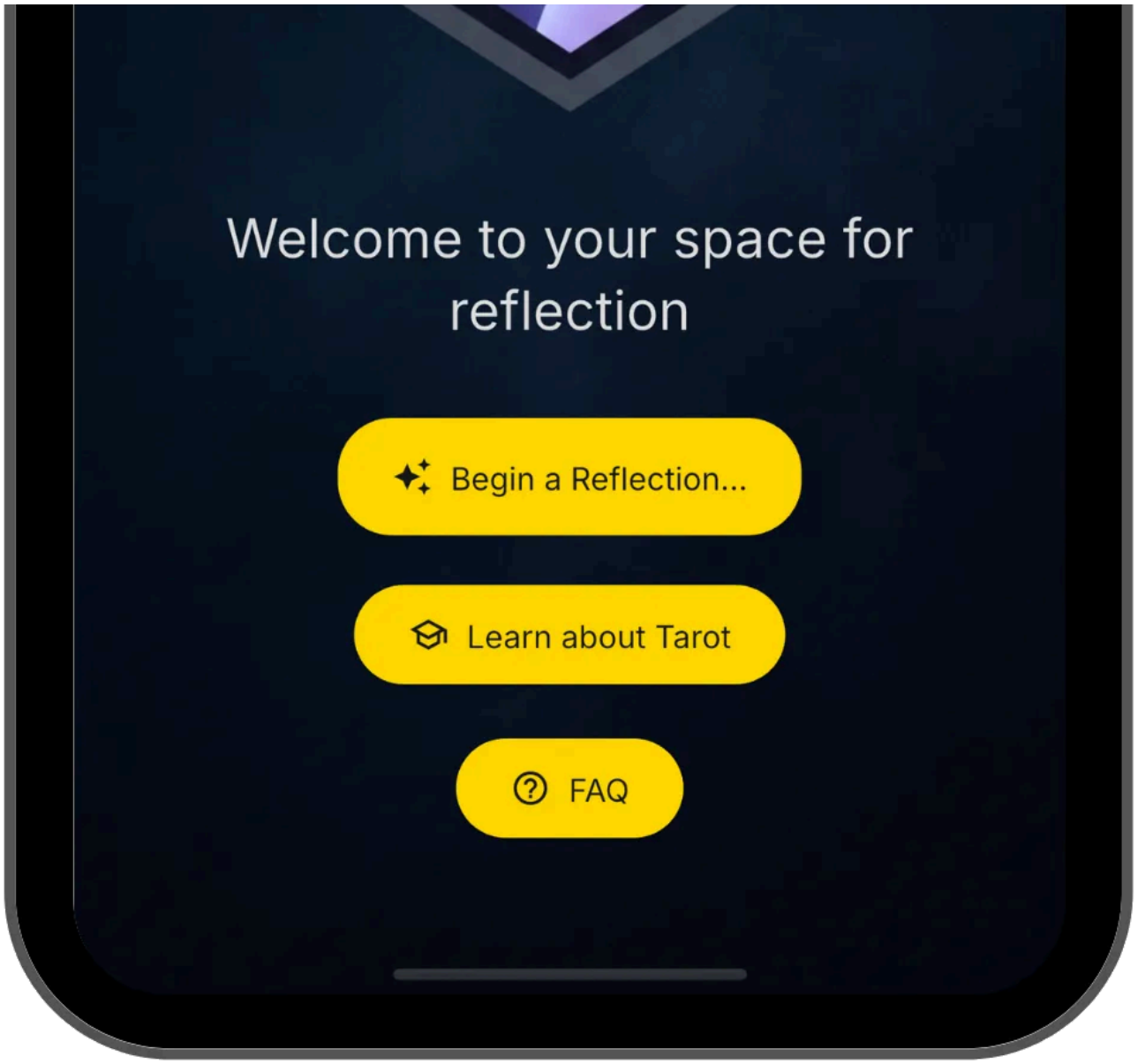
When I started, I had no experience with **Flutter and Firebase**. Yet, guided by the structured plans and leveraging AI tools like Cursor for code generation (within the framework I'd established), I was able to build a robust application using these very technologies. The AI wasn't just writing code; it was effectively bridging my knowledge gap, allowing me to implement features in an unfamiliar ecosystem by following the detailed steps and leveraging generated code snippets (always reviewed and integrated carefully, of course).

The final product wasn't a clunky prototype cobbled together with AI glue. Thanks to the integrated approach (defining the system design, enforcing the visual style guide and tone of voice during generation, and adhering to the production readiness checklist) the app felt polished and cohesive. The user interface was consistent, the user-facing text matched the defined personality, and the underlying architecture, planned and reviewed with AI assistance, was sound.

This success wasn't accidental, nor was it the result of finding some magical "perfect prompt." It was the direct outcome of the **structured "Vibe Software Engineering" framework** I had iteratively developed out of sheer necessity. The initial failures with single prompts weren't just setbacks; they were the catalyst for building a methodology that treated AI not as a code vending machine, but as a powerful collaborator *within* a rigorous planning and execution process. The contrast between the chaotic, failed attempts I described in the begining and the smooth, productive

workflow that ultimately delivered the app couldn't have been starker. The framework worked.


7:45
Arcana AI



10. Conclusion: More Than Just Prompting, A New Development Paradigm

Looking back on the journey from those initial, frustrating failed prompts to the launch-ready application sitting before me now (as of late April 2025), the transformation in my approach feels profound. My experience underscores a critical takeaway for anyone venturing into AI-assisted development: building substantial, quality software with today's tools requires significantly **more than just clever prompting**. The dream of a single-sentence request yielding a complete app remains, for now, largely an illusion.

Success didn't come from finding a magic wand, but from building a skeleton. It required embracing structure, discipline, and a willingness to adapt. The "Vibe Software Engineering" framework that emerged wasn't pre-planned; it was forged in the crucible of trial and error. Its core tenets: meticulous upfront planning inspired by methodologies like PRFAQ, leveraging AI as a partner in generating *plans* (System Design, User Stories, Implementation steps), rigorous cross-validation, iterative execution with crucial adaptations like sub-task planning and real-time documentation, and disciplined quality gates with version control; were all necessary components. Removing any one piece likely would have led back to the chaos of the early attempts.

Let's be clear: this approach is still **work**. It requires significant engagement, critical thinking, domain expertise, and careful oversight. You're not passively watching code generate; you're actively architecting, directing, reviewing, refining, and managing the entire process. However, it's fundamentally *different* work. It shifts the bottleneck from

typing syntax or wrestling with unfamiliar code to higher-level strategic thinking, planning, and quality control. The AI handles much of the heavy lifting in drafting and initial implementation, allowing you, the human developer, to focus on the bigger picture and navigate the complexity far more rapidly. It felt less like traditional coding and more like guiding a highly capable, if sometimes unpredictable, apprentice.

This entire experience feels like a glimpse into a **new paradigm for software development**. It's a truly collaborative model where human intelligence sets the vision, defines the structure, makes critical decisions, and ensures quality, while Artificial Intelligence accelerates the execution, handles repetitive tasks, and provides drafts and suggestions at incredible speed. It's not about replacing developers; it's about augmenting them, potentially changing *how* we develop software and dramatically increasing our velocity and ability to tackle new challenges, even in unfamiliar territory. The journey was demanding, but the result: "a professional app built in record time", speaks volumes about the potential of this structured, collaborative approach.

What's Next? Brainstorming the Future with AI

Here I am, in late April 2025, with my app polished, tested, and on the cusp of launching. There's a unique thrill in seeing something you envisioned, planned, and guided into existence finally ready to meet its users; especially when it was built using a process that felt experimental just weeks ago. The speed and efficiency were

remarkable, but the real satisfaction comes from having navigated the challenges and forged a repeatable framework that actually delivered.

So, what's next? First and foremost, launching this app and seeing how it fares in the real world! But beyond that, this experience hasn't just given me an app; it's given me a powerful new way to build. The "Vibe Software Engineering" framework feels robust, and I'm incredibly excited to apply it to future projects. The question inevitably becomes: what project *should* be next?

And naturally, my mind turns back to the AI toolkit. Why not leverage AI for the very first step: ideation and market discovery? I'm already contemplating using LLMs to brainstorm potential app ideas, analyze market trends, and identify underserved niches. Perhaps I'll even explore specialized AI tools, maybe digging into something like those **Manus reports** I've heard about, to generate in-depth analyses searching for opportunities ripe for innovation. It feels like the logical extension of this journey: using AI not just to build the solution, but to help discover the problem worth solving.

The possibilities feel wide open. This blend of human strategy and AI acceleration has fundamentally changed my perspective on software creation. The learning curve was steep, and the process required constant adaptation, but the potential for rapid, high-quality development is undeniable. I can't wait to fire up the planning documents, collaborate with my AI partners, and see what we can build next. The future of

development feels like an ongoing conversation between human creativity and artificial capability, and it's an incredibly exciting conversation to be a part of.

4 Likes · 1 Restack

Discussion about this post

Comments Restacks

Write a comment...

Garrett Vargas Apr 26

Liked by Armando Maynez

Great article. The way you describe the process, it feels like you took on a PM / engineering manager role with the various AI tools filling in the role of your development team. The human interactions you

describe feel a lot like doing code/design reviews, and layering in your expertise to guide the project forward

LIKE (1) REPLY SHARE

1 reply by Armando Maynez

Marcelo Calbucci Apr 25

Liked by Armando Maynez

I love the use of the PRFAQ to help LLMs understand the purpose behind it and make decisions that align with vision and strategy. That's exactly what you'd do for your team if you were working with software engineers, designers, and product managers. Kudos!

btw, I wrote a book about PRFAQ - www.theprfaq.com

LIKE (1) REPLY SHARE

1 reply by Armando Maynez

2 more comments...