The Architect and The Navigator

Building with AI

Jae S. Jung

Founder, WAM DevTech

EARLY ACCESS EDITION

Chapters 1–2 · February 2026

Table of Contents

Chapters in gray are forthcoming. Early access readers receive all future updates.

Chapter 1: The Architect and The Navigator

The Two Roles That Make AI Collaboration Work

Every time I had an idea, I'd pull out my iPhone and write it down. Used to be Notes. Then I switched to Evernote. The app doesn't matter. What matters is that I had years of these — fragments, outlines, half-baked concepts sitting in folders I rarely opened again. A SaaS product I'd been thinking about for years. An approach to legacy migration I knew would work but never had time to flesh out. Business proposals I could see clearly in my head but couldn't justify spending 40 hours writing.

That's the reality when you're running a business, being a husband and a father, managing clients, putting out fires. The time to sit down and actually execute — to take one of those ideas and turn it into something real — doesn't exist. I used to love sitting in front of my laptop and coding all night. Building things just because I could. That was a lifetime ago. Maybe a slight exaggeration, but it feels that way.

So the ideas pile up. You know they're good. You have the experience to build them. But the gap between what's in your head and a finished product feels like it keeps getting wider. Not enough time. Not enough uninterrupted hours. Not enough energy at the end of the day. Everything feels overwhelming, and the ideas just sit there.

Then AI came along.

ChatGPT was first. Mostly out of curiosity. I'd use it for snippets of code, research and analysis, rewriting emails, drafting responses. Nothing major. But I could feel the difference in how I approached work — small efficiencies that added up. I wasn't fully maximizing it though. I was still treating AI like a smarter search engine. Then I tried Gemini. Similar results. I looked into Claude, explored other AI agents. Each one was useful, none of them changed the game.

Then one day I decided to try something different.

I took a project that had been living in my head — one of those ideas from the phone — and instead of asking AI for a snippet or a quick analysis, I described the whole thing. Or at least I thought I did. Looking back, I was still underestimating what AI could handle. I gave it constraints like "I don't have the luxury of creating separate environments — going directly to production" because I was experimenting, not yet realizing how much complexity AI could actually work with. If I'd known then what I know now, my first message would have been ten times more detailed. But even with those simpler instructions, what came back was astonishing.

I sat there staring at my screen going, "Oh my God." AI did do the work — it executed what I described. But the blueprint was mine. The architecture, the decisions, the constraints, the decades of knowing what works and what doesn't — that all came from me. I communicated it in writing. Not the best writing at first, but I did it. And the output was at a level and speed I didn't think was possible. My jaw hit the floor.

So I kept going. And I kept getting better at it. Each project taught me something new about how to communicate with AI. I started recognizing patterns — more efficient ways to provide context, better ways to steer the conversation, sharper instincts for when to push back and when to let AI run. By the time I got to the fundraising integration project, I wasn't just giving AI constraints — I was asking it to do gap analysis on third-party documentation, identify risks if the integration didn't work as advertised, and surface alternatives before we committed to an approach. I even discovered I could ask AI to help me refine my own instructions — using AI to become a better collaborator with AI. I felt like a conductor waving my hands, and AI was the prodigy playing the instrument. The tune it played was magical.

One project finished, then another, then another. All those ideas that had been sitting in my phone for years — I was actually executing them. A SaaS platform with a 27-table database schema, built in languages I'd never used in production. A comprehensive project blueprint with 46 actionable tasks from a two-page client summary. A full legacy migration plan. A production enterprise application with 150+ endpoints migrated to a modern stack in three weeks. A strategically positioned government proposal. All production-quality.

I shifted my entire business model. And here we are.

But here's the thing that took longer to understand. The results weren't just because AI is powerful. Plenty of people have access to the same tools I do and aren't getting these results. The difference was in how I was collaborating — and when I looked closely at what I was actually doing in those sessions, I found a pattern. A methodology I'd been applying instinctively, built on 30 years of knowing how systems work, how teams execute, and where projects fall apart.

Every time I corrected AI's output, I was articulating a judgment call I'd been making unconsciously for years. Every time I gave it context that changed its approach entirely, I was naming expertise I didn't know I had. The collaboration became a mirror. What it reflected back was a methodology I'd been using all along but never put into words.

That methodology is what this book is about.

What Most People Get Wrong

Here's what I see when I watch experienced professionals use AI: they ask it to do small things.

A developer needs a utility function. A consultant needs a paragraph rewritten. A project manager needs a comparison table. Each request is self-contained, each response is useful, and each interaction ends there. The AI does the small thing, the professional moves on, and nobody thinks much about it.

This isn't wrong. It's just leaving 90% of the value on the table.

The leap that most people never make is handing AI the entire problem — the full scope, the messy context, the half-formed vision — and working through it together from beginning to end. Not "write me a function" but "here's the system I'm building, here are my constraints, here's what I know and what I don't know — let's figure this out."

That leap is hard for a specific reason. It requires you to articulate things you've never had to articulate before. When you work alone, your expertise is implicit. You make decisions based on pattern recognition you can't fully explain. You know a database schema is wrong before you can say why. You sense that a proposal angle won't land before you can articulate the alternative.

AI can't read your mind. So if you want it to operate at the level of a genuine collaborator, you have to externalize all of that. You have to become both the person who knows the answer and the person who can explain how they got there.

That's the real skill. And it's organized into two roles.

But before I explain those roles, I need to address something about AI itself — because how you think about what AI is shapes how you collaborate with it.

More Than a Tool

Most people describe AI as a tool. That undersells it.

What we have access to is something closer to a mind that has absorbed more knowledge than any single human could in a hundred lifetimes. Not a deity — not something to worship or fear. But a collaborator operating at a scale and depth that we're still learning to comprehend. And you get to work with it as much as you want, whenever you want, on whatever you want.

I don't think most people have fully grasped what that means. I'm not sure I have either. We're harnessing something we don't yet have the vocabulary to describe accurately. What I do know is this: it makes me better. It makes small businesses competitive in ways they never could be before. It takes 30 years of experience and amplifies it to a degree that still surprises me every time I sit down to work.

When a junior professional uses AI, they get help with tasks they don't yet know how to do. That's valuable. But the output is limited by what they can evaluate — if they can't tell whether a database schema will scale, AI's schema goes unchallenged. When a senior professional uses AI as a full collaborator, something different happens. Every year of experience becomes leverage. Your ability to spot a flawed architecture in seconds means AI's output gets better because you're better. AI doesn't replace your judgment. It gives your judgment a vehicle to operate at a scale and speed it never had before.

Thirty years of building systems didn't become less valuable when AI entered the picture. It became the thing that makes AI collaboration actually work.

But here's the catch. AI is the ultimate amplifier — and an amplifier without someone at the controls is just noise. An orchestra without a conductor. A road without a civil engineer. A vision for a house without an architect to turn it into blueprints. The power is real. But power without direction produces nothing. It needs someone who knows what to build, why to build it, and what "done right" actually looks like.

That someone is you. And the way you show up in that collaboration comes down to two roles.

The Two Roles

This is the pattern I didn't know I was following until I looked back at what I'd been doing across those five projects.

The Architect communicates what they're trying to build — except instead of drawing a blueprint, they're writing one in words. You're setting direction: providing context, defining constraints, making the strategic decisions that shape everything downstream. You're not telling AI what to build line by line. You're giving it the full picture — here's the problem, here's what I know about it, here are the boundaries, here's what success looks like. The Architect works at the level of intent and judgment.

The Navigator steers through that blueprint and finds the gaps in it. Every blueprint has them — assumptions that don't hold, edge cases nobody thought of, places where the plan meets reality and something doesn't fit. AI doesn't know those gaps exist. It can't anticipate what it hasn't been told. Maybe that's where AI is heading one day, but right now it needs a Navigator — someone in the output, evaluating it in real-time, pushing back when something feels off, redirecting when the approach needs to change, iterating until the result matches what your experience tells you is right. The Navigator works at the level of quality and reality.

Think of it this way. You're a conductor, and you've just been given access to a prodigy — not just any prodigy, but one who has mastered every instrument ever made and studied every composition ever written. The talent is almost incomprehensible. But without a conductor who knows what the audience needs to hear, who sets the tempo, who shapes the performance in real-time — that prodigy plays everything and nothing at the same time. Technically perfect. Emotionally empty. The Architect chooses the piece. The Navigator conducts the performance. The prodigy plays.

Most people only play one role. Architects who set great direction but accept whatever AI produces. Navigators who nitpick output but never gave AI the context to produce something good in the first place. The professionals getting exceptional results are the ones playing both — and switching fluidly between them.

The Architect: Setting the Direction

This is the role that experienced professionals are naturally built for — even if they don't realize it yet.

The Architect's job is to ensure AI is solving the right problem within the right constraints. This is where your experience and professional judgment come into play — before AI generates a single line of output.

Giving AI the Full Picture, Not Just a Task

When I started building FileCourier — a B2B SaaS file transfer platform you'll work through in Chapter 3 — my first message to AI wasn't "build me an API." It laid out the infrastructure direction: Aurora PostgreSQL for the database, API Gateway and Lambda for the REST API. I asked for language recommendations and whether tools like Replit could expedite development.

AI came back with Aurora Serverless v2 for cost efficiency, Node.js with TypeScript, and SST as the infrastructure framework — and steered me away from Replit for this particular use case, since the architecture needed to be AWS-native from the start. As the collaboration progressed, I layered in operational constraints: no staging environment, database private and accessible only via Lambda. Each piece of context narrowed AI's output toward solutions that fit my reality.

Here's what that looks like in practice:

> Architect prompt: "I'm planning to use Aurora PostgreSQL, API Gateway, and Lambda. What language and tools do you recommend for a B2B SaaS file transfer platform?" Not this: "Build me an API."

The difference isn't just detail — it's direction. The first prompt gives AI enough context to make recommendations that fit your actual constraints. The second gives AI nothing to work with except its defaults.

Starting with Strategy, Not Execution

On the government RFP (Chapter 7), I uploaded the full document and asked: "Should I even submit a proposal for this?" Anyone who's written a proposal knows the writing is the easy part. Understanding what the evaluators care about, positioning against invisible competitors, deciding which weaknesses to address head-on — that's the real work. The Architect does that work before the first word is drafted.

Making Decisions AI Can't Make

On a legacy migration for a ColdFusion-based CMS platform (Chapter 5), AI would have happily recommended a modern decoupled architecture — REST API backend, separate frontend. But this was a monolith where everything rendered server-side. Converting the code to Python and simultaneously converting the architecture would stack two massive transformations on top of each other.

So I chose Flask specifically because its server-side rendering mirrors how ColdFusion works — keeping the migration focused on the code, not redesigning the entire architectural pattern. Same logic for choosing raw psycopg2 over an ORM: ColdFusion uses inline SQL through cfquery tags, so a 1:1 translation to raw database calls keeps the conversion predictable.

These decisions came from understanding the system being migrated from, not just the system being migrated to. AI has no way to weigh those tradeoffs without you providing that judgment.

What the Architect Is Really Doing

The Architect's job isn't to have every answer. It's to frame the problem so clearly that AI can contribute at the level you need. That framing comes from experience — years of seeing what works, what breaks, and what looks fine until it hits production. You're not micromanaging AI. You're giving it the benefit of everything you've learned, so its output starts from a foundation of real-world judgment instead of generic best practices.

The Navigator: Steering Through the Mess

This is the role I had to learn the hard way — by catching mistakes that good Architect work should have prevented.

Here's where most people fall short. They set up the conversation well enough, but then accept whatever AI produces. They don't navigate. And navigating is where abstract becomes tangible.

Catching What AI Misses

During the FileCourier build, I needed multiple user roles: Admin, Developer, Manager. Since I hadn't specified roles upfront, AI created separate tables for admin users and regular users. It worked. But would it scale? Would it be maintainable when we added new roles? Not a chance.

When I provided the context I actually wanted — a normalized model with a roles lookup table — AI implemented it correctly. The code wasn't broken. The architecture was. And AI had no way of knowing that without me catching it.

Pivoting When the Plan Hits Reality

On the ColdFusion-to-Python migration, AI generated 102 tasks. That's not a plan — that's noise. AI can generate and track hundreds of granular items without breaking a sweat. But developers have to actually execute those tasks, and 102 items is overwhelming — a team sees a list like that and doesn't know where to start.

Here's what navigating looked like:

> Navigator prompt: "This is too granular for a development team to execute against. Group these into logical work units — each unit should represent a meaningful deliverable a developer can pick up and complete. Aim for 15-25 tasks, not 102." Four iterations later, we had 19 meaningful tasks a team could actually execute against.

That's the Navigator at work. Not fixing bugs — fixing usability. Not correcting code — correcting the gap between what AI produces and what humans can work with.

Demanding Honesty, Not Validation

On the proposal, I asked AI to play red team: "What are the weaknesses? Where would a competitor attack us? Be brutal." It identified three vulnerabilities I hadn't fully confronted. Requesting criticism instead of comfort made the final product significantly stronger.

Injecting What Only You Can Bring

AI generates professional-sounding documents all day. The parts that actually differentiate — the ones that make a client say "this team gets it" — come from you. On the fundraising platform project (Chapter 4), the third-party platform's terminology didn't match how the client thought about their own system — "campaigns" and "profiles" meant different things to them than what the platform intended. AI had no way of catching that disconnect. I did, because I understood the client's world, not just the platform's documentation.

What the Navigator Is Really Doing

The Navigator's job is to close the gap between AI's output and human reality. AI doesn't know that 102 tasks will paralyze a development team. It doesn't know that a client uses "campaign" to mean something different than the platform does. It doesn't know that a database schema that works technically will become a maintenance nightmare in six months. You know these things. The Navigator is the role where that knowledge becomes action.

Why You Need Both

Playing only the Architect means you set great direction and accept whatever comes back. The code compiles but the assumptions are wrong. The documents look polished but fall apart when a real team tries to execute against them.

Playing only the Navigator means you're constantly fixing problems that better direction would have prevented. You're spending your energy on corrections instead of creation.

The two roles aren't sequential — they're interwoven. In a single session, I might play the Architect when I set infrastructure constraints, switch to Navigator when I catch a flawed database design, back to Architect when I redirect the strategic angle, then Navigator again when I push AI to stress-test its own recommendations. That constant switching is what real collaboration looks like.

And here's what I've learned about resistance. The people who struggle most with this model aren't beginners. They're the veterans — senior developers, experienced architects, proposal writers with decades of wins. There's something deeply personal about the way experienced professionals work. A senior developer doesn't just write code — they think in code. The act of building line by line isn't mechanical labor to them. It's where their problem-solving happens. Asking them to hand that process to AI feels like asking a surgeon to describe an operation instead of performing it.

But that framing misses the point. AI doesn't take the craft away from you. It changes where in the process your craft matters most. The developer who lets AI handle the mechanical implementation can focus on architecture — the system-level thinking that actually determines whether a project succeeds. The proposal writer who lets AI handle first drafts can focus on strategy and positioning — the judgment that wins contracts.

The shift isn't from craftsperson to passenger. It's from practitioner to architect.

The 16 Principles

Through those five projects, I identified 16 principles that consistently separate productive AI collaboration from wasted conversations. They aren't theoretical — every one emerged from real projects, real mistakes, and real results.

They fall into two layers that map directly to the two roles:

Layer 1: Communication Principles — The Architect's Toolkit. These govern how you interact with AI — the mechanics of setting direction effectively. Leading with constraints instead of vague goals. Making decisions while letting AI provide options. Sharing raw output instead of summaries. Keeping context flowing across a conversation. Eight principles in total, each one determining whether AI understands your situation clearly enough to produce useful output.

Layer 2: Thinking Principles — The Navigator's Toolkit. These govern how you reason as a collaborator — the judgment that produces expert-level results. Iterating instead of accepting first drafts. Challenging assumptions when something doesn't match your domain knowledge. Consolidating for flexibility. Keeping scope tight. Eight more principles, each one sharpening the gap between what AI generates and what actually works in production.

Communication gets you good input. Thinking gets you good output. You need both layers working together — just like you need both roles.

The next chapter breaks down all 16 principles with concrete examples from each case study, so you'll have the full vocabulary before diving into the projects themselves. But the principles only matter if you see them in action. That's what the rest of this book is for.

What Comes Next

This book is organized in two parts.

Part 1: The Framework is what you're reading now. The next chapter lays out all 16 principles in detail — the full vocabulary of effective AI collaboration, with concrete examples from each case study. By the end of Part 1, you'll have the mental model and the language to start applying the methodology immediately.

Part 2: The Principles in Action is where you see it all work for real. Each chapter walks through a complete production project — not a tutorial exercise, not a toy example, but actual work that shipped or was delivered to real clients. For each one, you'll see:

- The project context — what we were building and why - Architect decisions — what direction I set and the reasoning behind it - Navigator moments — where I pushed back, redirected, or caught errors - Actual prompts and AI responses — the real conversation, not a sanitized version - What the collaboration produced — the tangible deliverables - Principles in focus — the 3–4 principles most critical to that project, shown in action

Here's what's ahead:

Chapter 3: FileCourier — Building a B2B SaaS file transfer platform using Aurora PostgreSQL, API Gateway, Lambda, and Node.js/TypeScript. I built this in languages I'd never used in production. The Architect set the infrastructure direction; the Navigator caught a flawed database design, corrected authentication patterns, and steered AI through dozens of implementation decisions.

Chapter 4: The Fundraising Platform — A car wash company needed a fundraising platform. We transformed a vague two-page client summary into a comprehensive project blueprint, recommending integration with a third-party fundraising platform. Business proposal, technical integration guide, executive presentation, risk assessment, and 46 actionable tasks across 6 epics — from nothing to a complete blueprint in 24 hours. The Navigator caught platform terminology mismatches that AI couldn't see.

Chapter 5: The Legacy Migration — The Roadmap — Planning a ColdFusion/SQL Server to Python Flask/PostgreSQL migration for a CMS platform built exclusively for law firms. Both the Architect and Navigator were essential — the Architect made framework choices AI would have gotten wrong, and the Navigator consolidated 102 AI-generated tasks into 19 meaningful work units through four iterations, pushing back on guidelines and structure until the plan was airtight. This was a team effort — me, a Director of Technology, and senior developers — where my job was to provide the roadmap, the structure, and the guardrails so the team could track progress and execute with confidence.

Chapter 6: The Legacy Migration — The Execution — Migrating a production ColdFusion REST API with 150+ endpoints to Go on AWS Fargate. This is the other side of the coin from Chapter 5. Here I was a one-man team — hands on keyboard, working through the code end to end with AI as my collaborator. Three weeks, 88% cost reduction. The lessons I learned executing this migration are what allowed me to tighten up the roadmap in Chapter 5. A developer on the Chapter 5 team will go through exactly what I went through here — and the roadmap I built for them was designed to make that experience structured and trackable.

Chapter 7: Mooresville RFP — A government website redesign proposal that started with "Should I even submit?" Strategic analysis, red team assessment, competitive positioning, incumbent analysis — the Architect work that happens before a single word of the proposal is written.

Chapter 8: When AI Helps vs. When It Slows You Down — A decision framework for knowing when to collaborate with AI and when to do the work yourself. Not everything benefits from AI collaboration, and knowing the difference is part of the methodology.

Let's start with the principles.