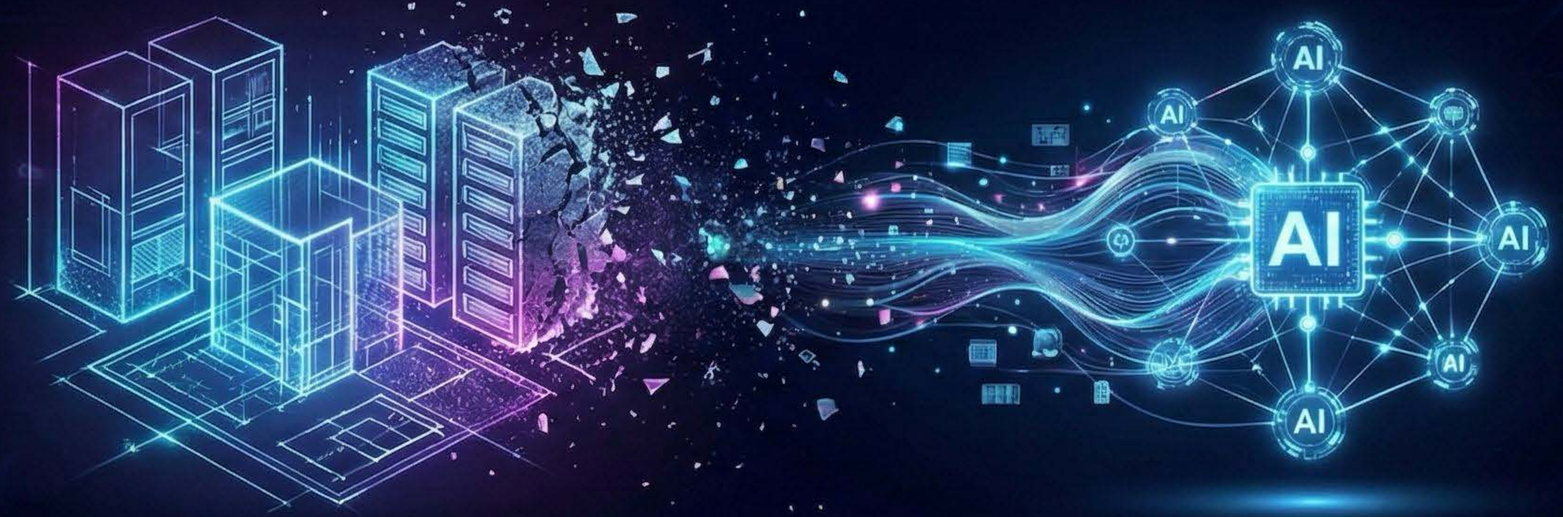
WHY I'M BLOWING UP MY TECH ARCHITECTURE EVERY 6 MONTHS



Miki, CEO & Co-Founder, Ōlelo Intelligence

Why blow up my architecture every 6 months?

1) Because the landscape shifts faster than your roadmap.

Models

Coding Agents

Workarounds

What was cutting-edge six months ago is table stakes today. Models get cheaper & smarter, agents make step-function leaps, workarounds appear weekly.

2) Because the bar keeps rising.

Launch Tech

Competitors

Value

It's easier to launch. Competitors can too. Launch with more value, faster scale, lower even. Yesterday's advantage is tomorrow's baseline. Disrupt yourself, or get disrupted.

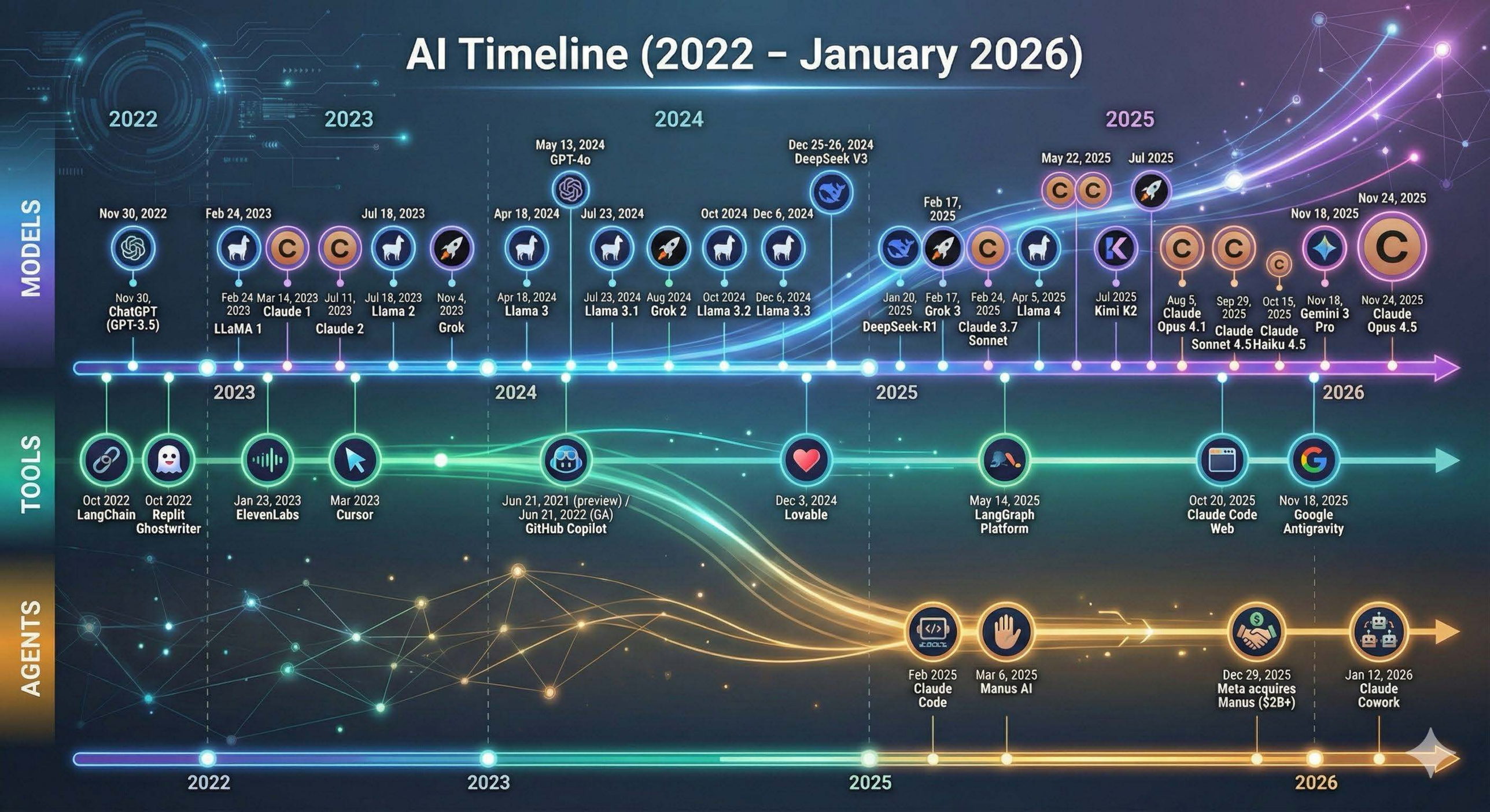
3) Because legacy is the enemy of value.

Customer

Today's Delivery

Tech Debt

Customers care about delivery today. Tech debt, "fix it later," and attachment to old decisions are friction. Legacy is a barrier to the value your customers need.


AI Timeline (2022 – January 2026)
2022
2023
2024
2025
MODELS
Nov 30, 2022
Nov 30, ChatGPT (GPT-3.5)
Feb 24, 2023
Feb 24 2023 LLaMA 1
Mar 14, 2023 Claude 1
Jul 11, 2023 Claude 2
Jul 18, 2023
Jul 18, 2023 Llama 2
Nov 4, 2023 Grok
Apr 18, 2024
Apr 18, 2024 Llama 3
May 13, 2024 GPT-4o
Jul 23, 2024
Jul 23, 2024 Llama 3.1
Aug 2024 Grok 2
Oct 2024
Oct 2024 Llama 3.2
Dec 6, 2024
Dec 6, 2024 Llama 3.3
Dec 25-26, 2024 DeepSeek V3
Jan 20, 2025 DeepSeek-R1
Feb 17, 2025
Feb 17, Grok 3
Feb 24, 2025 Claude 3.7 Sonnet
Apr 5, 2025 Llama 4
May 22, 2025
Jul 2025
Jul 2025 Kimi K2
Aug 5, Claude Opus 4.1
Sep 29, 2025 Claude Sonnet 4.5
Oct 15, 2025 Claude Haiku 4.5
Nov 18, 2025
Nov 18, Gemini 3 Pro
Nov 24, 2025
Nov 24, 2025 Claude Opus 4.5
2023
2024
2025
2026
TOOLS
Oct 2022 LangChain
Oct 2022 Replit Ghostwriter
Jan 23, 2023 ElevenLabs
Mar 2023 Cursor
Jun 21, 2021 (preview) / Jun 21, 2022 (GA) GitHub Copilot
Dec 3, 2024 Lovable
May 14, 2025 LangGraph Platform
Oct 20, 2025 Claude Code Web
Nov 18, 2025 Google Antigravity
AGENTS
Feb 2025 Claude Code
Mar 6, 2025 Manus AI
Dec 29, 2025 Meta acquires Manus ($2B+)
Jan 12, 2026 Claude Cowork
2022
2023
2025
2026

Preparing to Blow Stuff Up

1) Assemble your toolkit.

Coding Agent

Claude Code · Cursor · Antigravity · Codex

Pick one. Try others weekly.

Capable Model

Opus 4.5 · GPT-5 · Gemini 3 Pro

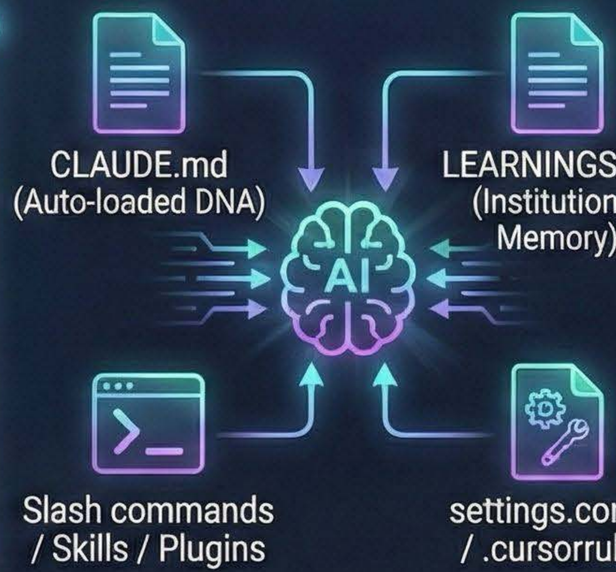
Agent is only as good as the brain.

Your Repo & IDE

GitHub · GitLab · Cursor · VSCode

Agent needs access. Cursor doubles as both.

2) Prime your environment.



Onboarding for a brilliant teammate with perfect recall. Better priming, better output.

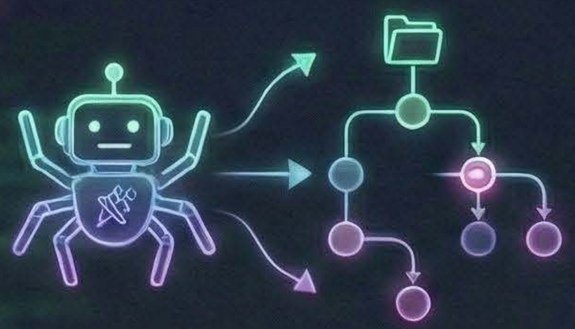
3) Abandon Agile. Embrace Waterfall.

AI agents need clarity upfront. Vague requirements produce vague output. Classical engineering principles return.

Assess Before You Destroy

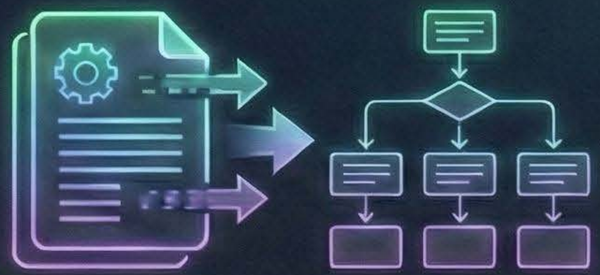
Kick off agents—in parallel.

Repo analysis agent



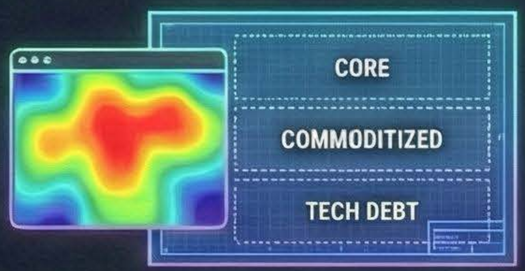
Use Claude Code plugin like Superpowers. Crawls codebase, surfaces actual architecture (not what you think).

Requirements extraction agent



Context7 documents functional requirements from existing system.

Business architecture agent



Custom skill generates capability heat map. Core? Commoditized? Technical debt masquerading as “critical infrastructure”?

Parallel Execution

But wait. Did the agent say “Done!” but… it isn’t?

The Ralph Wiggum Loop: Solves Premature Completion in Agentic Coding

Named after the Simpsons character who keeps trying despite failures.

The Problem: Agents Lie (Prematurely)

Not malicious, just declare victory too early. 'Done!' No it isn't. 'All tests passing!' Three skipped. 'Implementation complete!' Edge cases ignored. Manual verification defeats the point.

The Solution: Loop Until Actually Done

Each iteration sees the previous attempt. Failures become data. The agent learns from mistakes within session.

Why It Works & Benefits

No human bottleneck: You're not the feedback loop.

Persistence beats perfection: Automated retries.

Real Results (Investor CRM Example)

A full Investor CRM was built and deployed in just 2 days. The Ralph Loop autonomously iterated on complex data models and UI states until all 150+ acceptance criteria were met without human intervention.

In Practice & The Catch

/ralph-loop "Implement user authentication with OAuth" --completion-promise "All auth tests pass AND login flow works end-to-end" --max-iterations 50
Walk away. Come back to a working implementation or a clear log.

The Catch: Need verifiable completion condition. "Make it good" fails. "All 47 unit tests pass" works. PRD-first matters.

One-liner version: 'Don't ask the agent if it's done. Define what done means, and let it loop until it gets there.'

Building the PRD: Where You Earn Your Keep

You've kicked off agents, analyzed repo, extracted requirements, mapped architecture. Now, synthesize everything.

1. Synthesize Everything Into a PRD

Repo analysis
(technical constraints, patterns)

Functional requirements
(what system does/should do)

Business architecture
(strategic, commodity, dead)

PRD
Draft

Synthesize into structured **PRD DRAFT**

2. This Is Where You Show Up (Review Intensely)

Everything before was setup. This is execution.

Your experience, judgment, domain expertise matter.

Review line by line.

Ask:
Does this capture need?
Are requirements complete?
What's ambiguous?
Where are gaps?

⚠ Agent is brilliant at synthesis, terrible at knowing what you forgot.

3. What Waterfall Means Now & The Quality Bar

Bureaucracy

Define Upfront

Not bureaucracy. Define everything upfront before agents build. Agents are executors, not guessers.

Ambiguity in, garbage out.
Clarity in, magic out.

The Quality Bar:
Hand it to a stranger without clarifying questions.

That's the bar for the agent.

4. PRD Content & Acceptance Criteria (Ralph Loop)

Technical
- (Architecture
- Data, API
- Performance
- Security)

Non-Technical
- (Business Rules
- Compliance
- Data retention
- Audit)

Usability
- (User flows
- Edge cases
- Accessibility
- Response time)

One More Thing: Write acceptance criteria as Completion Promises for Ralph Loop. Not 'user can log in". Instead: "OAuth flow completes < 2s, persists 7 days, handles refresh, fails gracefully".

The Irony: We spent a decade escaping waterfall. Now advanced AI demands it. Agents aren't humans; they need the spec. Your job is to write the PRD so well that code writes itself.

Kick Off the Build: Orchestrating Your Agent Army

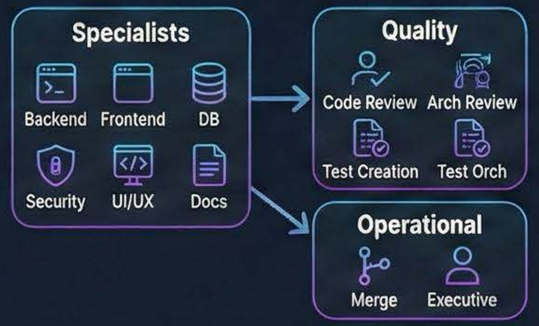
Deploying Your Locked PRD with Parallel, Specialized AI Agents

1. Executive Agent: Your Dashboard

Tracks progress, surfaces blockers, reports status, coordinates handoffs. Your AI project manager.

2. The Agent Orchestra & Atomic Tasks

Agent Specialists & Quality



The Atomic Task Principle

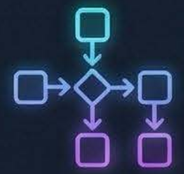
Task 1: Create Schema
Task 2: Implement JWT
Task 3: Build Login Endpoint

Breakdown into small, independent units. Fresh context, clean commits, parallel execution, clear completion.

3. Scaling Patterns & Merge Flow

Specialist vs. Volume Parallelism

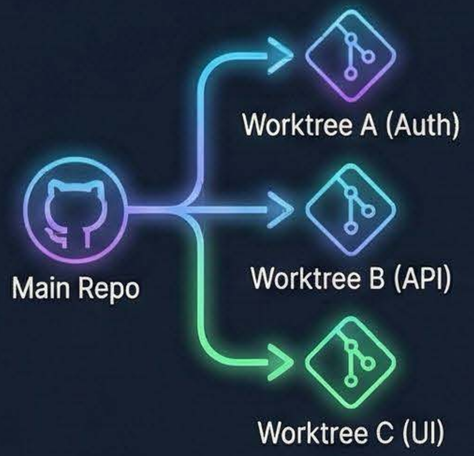
Different domains or multiple attempts at same task.



The Merge Flow

Review -> Arch -> Security -> Test -> Merge PR -> Human Approve

4. The Isolation Problem (Solved)



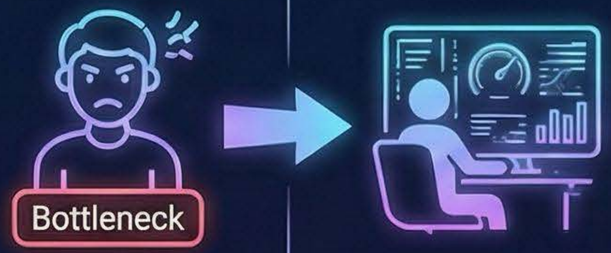
Git Worktrees: Separate branches & working dirs. No collisions. Full parallelism for simultaneous Claude Code sessions.

Orchestrated Delivery → Parallel Execution

Human OVER the Loop: Monitor or Walk Away

The choice that separates "Babysitting" from "Orchestrating".

1. The Shift in Mindset



Human IN the Loop (Old Way)	Human OVER the Loop (New Way)
Approve every action/command	Configure allowed actions & safety rails
Constant monitoring	Periodic check-ins or "Walk Away"
Works only when you are awake/active	Agents work 24/7 while you sleep
Reactive intervention (Manual control)	Proactive guardrails (Strategic control)

2. Configuring for Autonomy

```
{
  "permissions": {
    "allow": [
      "Read",
      "Write",
      "Edit",
      "Bash(npm install*)",
      "Bash(git commit*)",
      "Bash(pytest*)"]
    },
    "deny": [
      "Bash(rm -rf*)",
      "Bash(sudo*)",
      "Bash(*prod*)",
      "Bash(*deploy*)"]
    }
}
```

Give agent power to build, remove power to destroy.

3. The 4-Week Trust Gradient

Week 4:
Full Orchestration
(Walk Away unlocked)

Week 2:
Safe Autonomy
(Auto-approve read/write/test)

Week 3:
Commit Autonomy
(Auto-approve read/write/test)

Week 1:
Full Supervision
(Approve manually)

4. The Four Safety Rails

- **Scoped Permissions** (Specific white-lists)
- **Completion Promises** (Loop until verified)
- **Iteration Limits** (Max iterations/cost)
- **Review Output, Not Process** (Focus on final PR)

Key Takeaway:
Delegate execution, don't abdicate responsibility. The skill is building the system so watching becomes optional.

The Part Nobody Talks About: This Is Exhausting (Managing AI Agents)

Let me be real: You're not writing code anymore. It sounds like less work. It isn't. A Different Kind of Tired.

Like watching five chess games simultaneously & responsible for all.

The New Cognitive Load: What's Happening in Your Brain

Context Switching

Context Switching Amplified
Switching between agents' tasks, each with its own state.

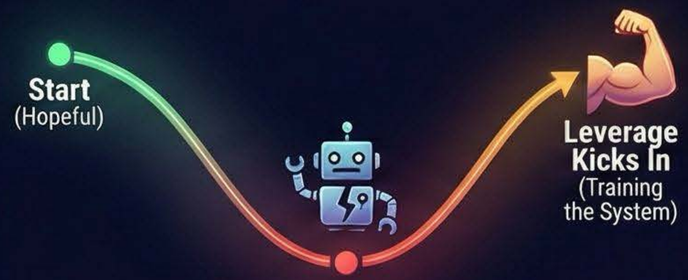
Pattern Recognition Under Uncertainty

Intuition for when to trust, when to intervene. Is output wrong or intermediate?

Decision Fatigue

The Anxiety of Letting Go
Tension of being built by something you can't fully predict. Learning to sit with it.

The First Few Weeks Are Brutal

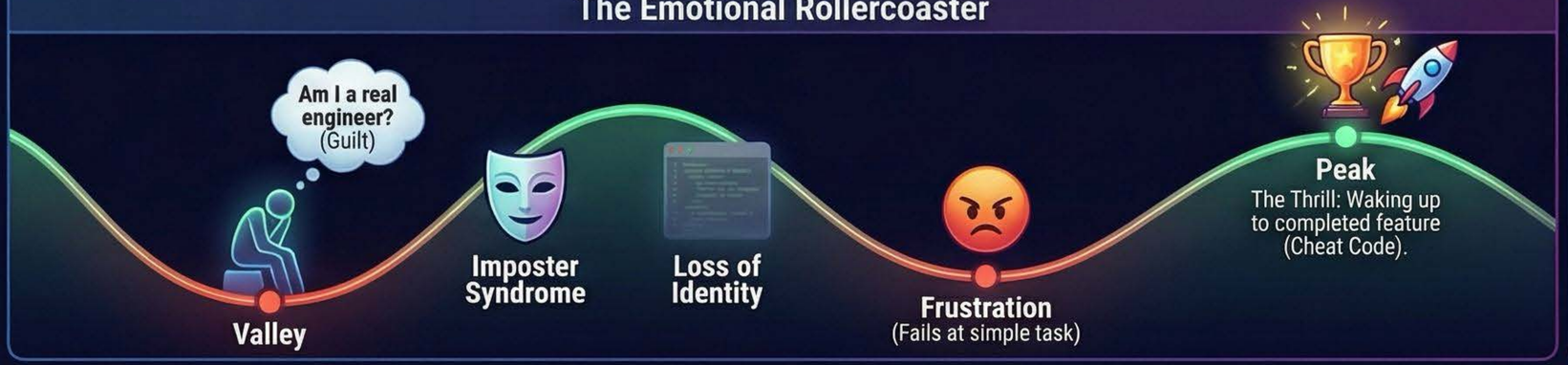


Not optimizing for one-time fix, but building a system.

Building the Mental Muscle (Deliberate Practice)

1. **Shorter Autonomy Windows**
   
   30m → 1h → Overnight
   Build tolerance gradually.
2. **Develop Intervention Instincts**
   
   Recognize early signs. Distinguish "wrong" from "different".
3. **Accept Imperfect Outputs**
   
   Not exactly your code, but faster. Refactor later.
4. **Protect Recovery Time & Rituals**
   
   Schedule actual breaks. Batched reviews, not constant checking.

The Emotional Rollercoaster



The Honest Truth & Preparation

Different Tired, More Shipping
Exhaustion comes with results not possible before. Trading effort type.

Prepare Yourself
Accept the difficulty. Block learning time. Find community. Patience. Push through until patterns become automatic. It's worth it.

CONFESSION: I WAS THE STUBBORN ENGINEER

The Three Months I Wasted Being Comfortable

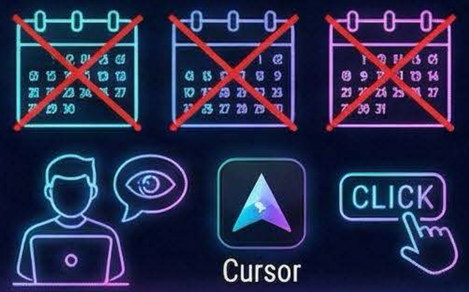
1. THE COMFORT ZONE & THE LIE

IntelliSense

2024: Productive in PyCharm. Comfort, muscle memory, plugins. Thought Cursor/Windsurf weren't worth setup time.

The Lie: "The ROI isn't there."

2. THE THREE-MONTH DELAY & THE SWITCH

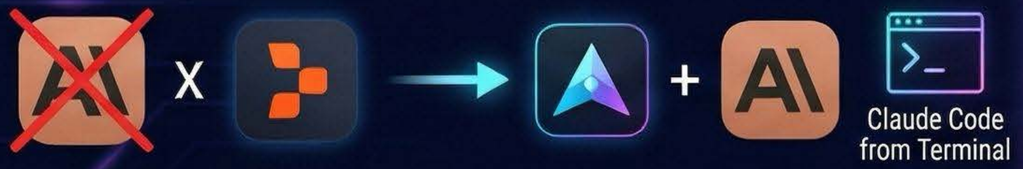


Cursor

Saw multi-file context, Composer. Clicked. Wasted 3 months delaying.

Switched: Setup took an afternoon. Muscle memory adapted in days. Leap in productivity.

3. THE CLAUDE CODE SKEPTICISM & INTEGRATION



Claude Code from Terminal

Early try: "I could write this faster." Shelved it.

Embedded in Cursor. Terminal + IDE = Full autonomous agent power. Off to the races.

4. THE MULTIPLICITY & THE REAL COST

Cursor (Daily Driver)
Claude Code (Autonomous)
Antigravity (UI)
Copilot (Completions)

Use them all.
It's a toolkit, not a marriage.

3 Months Delay = 18 Days Lost Productivity

The Real Cost: 18 days lost for an afternoon of setup. The ROI was enormous.

THE LESSON: DON'T DISMISS TOOLS. TRY AGAIN. YOU DON'T HAVE TO CHOOSE ONE. WHAT'S YOUR CURSOR? TRY IT THIS WEEKEND.

30-year tech veteran. Still got stuck. If it happened to me, it can happen to you.