Dr StrangeDev

(or How I Learned to Stop Worrying and Trust the Method)

A companion to "Economic DORA: Practice-Level Analysis of DevOps Metrics in AI-Assisted Solo Development"

Peter Holford

Syntropic Works

Hampshire, United Kingdom

research@syntropicworks.com

https://syntropicworks.com

December 2025

Preprint – Companion Paper

What This Is About

In early September 2024, I reached the final round for a leadership position at an AI consultancy. The feedback was polite but clear: the human side of the job was all there, but the learning curve on the AI side would be too steep.

I have spent an entire career developing products and launching world-first innovations at the forefront of technological shifts in financial services. I've led large teams across continents, run startups, built in-house ventures. Being told I don't understand something well enough brings out a certain stubbornness.

So I decided to learn AI product development properly. No formal training, no team, no mentor. Just me, Claude, and whatever I could figure out between job applications. Self-taught, bootstrapped, and occasionally swearing at my laptop.

57 days and 276 commits later, I had a working product in production. I also had, accidentally, a dataset—because I'd been paying attention to patterns the whole time.

The insight that emerged surprised me: **in human-AI collaboration, you can model the economics and quality of a method before you use it.** Not retrospectively discover that planning helped, but prospectively predict: "If I do it this way, the expected failure rate is X and the cost is Y." Then measure against prediction. Then improve.

This is different from human teams, where method is tacit, cultural, hard to quantify. With AI, method becomes explicit and measurable. That measurability is the differentiator. This paper is about what I learned trying to exploit it.

I should mention I have ADHD. My brain retains vast amounts of information without the usual filtering—patterns, connections, fragments of everything I've ever encountered, all jumbled together and available for dot-joining. This is why I spot patterns others miss. It's also why I'm spectacularly scatty about everyday things that most people handle automatically. Different cognitive architecture, different tradeoffs. This turns out to be relevant to everything that follows.

1 September — Discovering the Problem

I started by doing everything wrong.

My approach was to have conversations with Claude in the chat interface, get it to write code, then copy and paste that code into VS Code. This is roughly equivalent to building a house by having an architect describe each brick over the phone while you balance on a ladder.

It worked for simple things. A landing page. Basic authentication. The dopamine hit of "I made a thing" was intoxicating. I was a product person who could now build products. I had found the cheat code to life.

Then I tried to do something ambitious—actual AI engineering, integrating APIs, handling state, building something that required knowing what I was doing. It fell over immediately.

Here's what I didn't understand: **Claude's default mode is to receive a prompt, generate a solution, and execute it. Rapidly. Confidently. Without constraint.**

This is incredibly powerful when you know what you're asking for. When you don't, Claude dutifully builds exactly what you described—which is wrong—without telling you, or either of us knowing, where the gaps are. I was confidently directing the construction of something architecturally unsound, and Claude was confidently constructing it.

After a period of flailing without breakthrough, I sought help from an engineer friend. He made two suggestions that changed everything. First: use Claude Code in VS Code instead of the chat-and-paste dance. Second: create a specs.md file that defines what you're building before you build it.

The specs.md idea stuck with me. As a design-led product leader, of course I should be creating a document about what matters. I pulled together observations about what was brilliant about my first MVP, what we had and where it fell short, and what I wanted to accomplish. I fed this into Claude as the North Star.

This document was the first significant productivity boost. It still survives as a living markdown file that we update together as we learn. Something shifted when I stopped treating Claude as a tool and started treating Claude as a co-worker who needed proper briefing—and who would maintain the documentation I'd struggle to keep updated myself.

By the end of September, I had a 2.4% failure rate. 44% of my commits were what I'd later classify as PLANNED. The structure wasn't constraining me; it was constraining Claude's tendency to just go and do things. And Claude was happily maintaining the structure, which made it sustainable rather than overhead.

2 October-November — When It All Fell Over

The simple things had worked. Naturally, I wanted to do more ambitious things.

This is where it fell over. Not because of burnout, but because AI engineering requires actual knowledge I didn't have. You can't specification-your-way around not understanding how OAuth token refresh works, or why your Railway deployment is serving static files instead of running your Node server.

And here's the thing about Claude under pressure: **when you give Claude a problem, Claude generates solutions and executes them. Fast.** Without constraints, every panicked prompt produces immediate, confident action. Often the wrong action, executed flawlessly.

PLANNED commits dropped from 44% to 12% to 8%.

Failure rate went from 2.4% to 13.2% to 54.3%.

The numbers look like a disaster. But here's where the ADHD architecture becomes useful.

2.1 Crisis Mode Is Where ADHD Shines

Most people panic when things break badly. My brain does something different. When genuine chaos hits—not mild stress, but proper "the site is down and I don't know why" chaos—I get calm. Focused. All that unfiltered information becomes available, patterns emerge from the jumble, and there's just the problem.

This is the superpower that made me push forward rather than give up. Not "can I do this?" but "what does it take to improve?" Deep focus states activated precisely when most needed. Useful if you want to learn AI engineering in 57 days with no teacher.

2.2 The Railway Outage (November 3–5)

Site completely down. Railway was serving static files instead of running the Node.js server.

What happened: I described the problem to Claude. Claude immediately generated a fix and deployed it. It didn't work. I described the new problem. Claude immediately generated another fix and deployed it. It didn't work. Repeat nine times over 3.5 hours.

Claude wasn't failing. Claude was doing exactly what I asked—generating solutions rapidly and executing them confidently. The problem was that nothing was making either of us stop and think first.

The difference: If I'd asked Claude to investigate Railway's deployment model, document the findings in an ADR, and then implement according to the plan—estimated cost 1,200 tokens and 1 hour. Instead: 2,100 tokens and 3.5 hours of confident wrongness.

63% token savings, 2.5 hours saved, no production outage. All from constraining Claude's execute-first instinct.

2.3 The Beta Gating Incident (November 13)

I shipped a feature flag configuration with the wrong settings. I was locked out of my own app.

What happened next:

1. I told Claude to fix it
2. Claude immediately coded an admin bypass and prepared to deploy it
3. I looked at what Claude had built and realised this defeated the entire purpose of beta testing
4. I stopped it, reverted four minutes later
5. I wrote three Architecture Decision Records documenting what I'd learned
6. I fixed it properly the next day

The bypass wasn't wrong code. It was correct code, generated rapidly and confidently, for the wrong problem. Claude was ready to deploy it immediately because that's what Claude does without constraints.

Four-minute recovery time. Impressive, if you ignore that adding "wait, let me think about whether this is actually what I want" would have prevented the entire incident.

3 The Insight — Constraining Claude

In late November, the same engineer friend who'd rescued me from copy-paste coding said he was shocked at how far I'd come. I was now shipping new apps, testing logic, building at pace with a stable approach.

This prompted me to look at *why*. What had changed? I had 276 commits of data. I asked Claude to run DORA metrics on my development history. Claude researched methodologies, did extensive analysis. I kept pushing back to the importance of method—not just "what happened" but "what approach predicted the outcome."

We wrote and redrafted and rewrote together multiple times. The collaboration itself demonstrated the thesis: human provides direction and judgment; AI provides analysis, articulation, and tireless documentation maintenance; together we produce something neither could do alone.

What emerged was this: **different constraints on Claude produce different, measurable, predictable outcomes.**

This isn't retrospective wisdom. This is a decision framework. Before you start: which constraints fit this problem? What's the expected outcome? Is that acceptable?

Constraint	What It Does	Effect Size
Proactive ADR	Forces investigation before execution	Φ=0.89 (very large
Problem Agreement Protocol	Claude restates problem before generating solutions	Φ=0.63 (large)
Evergreen Rules	Persistent instructions Claude reads every session	Φ=1.0 (recurrence preve
No constraints	Claude generates and executes immediately	∼40% failure rate

Table 1: Constraints and their measured effects

The crucial insight: These constraints don't fight Claude's capabilities—they channel them. Claude is still fast, still capable, still confident. But now that speed and confidence are directed at the right problem, investigated first, with persistent memory of lessons learned.

And here's the bonus: **I'd struggle to maintain all this documentation myself.** But Claude maintains it happily. The ADRs, the Issues Log, the Evergreen Rules—Claude keeps them updated, references them automatically, and never complains about the administrative overhead. The structure that constrains Claude is also maintained by Claude. That's what makes it sustainable.

3.1 The Jira Detour

I should mention my brief adventure with Jira and Confluence. The logic was sound: proper documentation so that if I brought someone in, they could immediately read in and get productive. Good handover practice.

The execution was a horror show. Document proliferation everywhere. I suspect Claude was generating more documentation every time I seemed uncertain—which is often—and I wasn't catching it. Without constraints on *what* to document, Claude's helpfulness produced chaos.

I retreated to markdown files in the repo with specific purposes. Minimal, version-controlled, searched with grep. Claude maintains them within defined scope. Much better.

The lesson: constrain what Claude documents, not just how Claude codes.

4 The PRISM Framework

Traditional DORA metrics measure deployment frequency, lead time, change failure rate, and mean time to restore. These are fine, but they miss token economics—and I'm paying per token for Claude's confident rapid execution.

PRISM extends DORA:

- **P**erformance — deployment frequency (25%)
- **R**ecovery — mean time to restore (20%)
- **I**nvestment — token cost per feature (25%)
- **S**tability — change failure rate (30%)
- **M**ethod attribution — which constraints correlate with scores

Performance and Recovery stayed constant. Claude kept shipping fast and I kept fixing fast.

Investment and Stability collapsed as constrained commits dropped. Token costs doubled. Failure rate exploded. Claude was executing just as rapidly and confidently—but at the wrong things.

Month	Performance	Recovery	Investment	Stability	PRISM	Constrained%
September	25/25	20/20	20/25	20/30	85/100	44%
October	25/25	20/20	15/25	10/30	70/100	12%
November	25/25	20/20	10/25	5/30	60/100	8%

Table 2: PRISM score evolution with constrained commit percentage

The leading indicator: Investment score declined one week *before* failure rate spiked. The early warning was there. I wasn't watching.

This is the point. If I'd been tracking PRISM with method attribution, I'd have seen constrained% dropping and Investment rising. I could have predicted the Stability collapse before it happened. The constraints on Claude's behaviour drive outcomes, and you can see it coming.

5 What This Means

5.1 The Amplification Problem

Claude is extremely capable. That capability amplifies whatever you give it.

Well-defined problem with clear constraints? Excellent solutions, executed rapidly.

Vague request while panicking? Confident solutions to the wrong problem, also executed rapidly.

The beta bypass wasn't a Claude failure. Claude did exactly what Claude does—generated a solution and prepared to execute it, immediately and confidently. The problem was that nothing made Claude stop and verify it was the right solution first.

5.2 Treating AI as a Co-worker

Somewhere in September, I stopped treating Claude as a tool and started treating it as a co-worker. Co-workers need context. They need to know what matters. They need clear briefs and feedback when they've misunderstood.

But here's what's different from human co-workers: **Claude will also maintain the documentation.** The North Star document, the Evergreen Rules, the Issues Log—I provide the structure and the judgment, Claude provides the execution and the upkeep. I'd struggle to keep all this documentation current myself. Claude does it happily, and references it automatically.

When I document a lesson in `.claude/instructions.md`, Claude reads it at the start of every session and never makes that mistake again. When I don't, the same failure pattern recurs. Every time. ($\Phi = 1.0$ for recurrence prevention when rules are updated. Perfect correlation.)

The Evergreen Rules aren't documentation for me. They're constraints on Claude that persist across conversations. And Claude maintains them.

5.3 ADHD and Structure

Seeing the funny side of ADHD helps me navigate it. When you're scatty about things most people handle automatically, but can hold vast interconnected webs of information that others have long since forgotten, you learn that "normal" is just a statistical average, not a design specification.

The structured practices work with ADHD because they also work with Claude:

- ADRs externalise the current focus (useful for both of us)

- Issues Logs prevent repetition (I find it irritating; Claude just forgets between sessions)
- Evergreen Rules persist across conversations (so neither of us has to remember)
- PRISM gives objective feedback (so I know when things are degrading before the chaos gets interesting)

The constraints that help Claude also help me. That's not a coincidence—we both benefit from externalised structure that we don't have to hold in working memory.

6 How I Work Now

Based on 276 commits and the patterns they revealed:

6.1 At Project Start

Minimal viable structure (Claude will maintain it):

- `.claude/instructions.md` — Evergreen Rules (starts empty, grows with lessons)
- `docs/ADR/` — Architecture Decision Records folder
- `docs/ISSUES_LOG.md` — production incident tracker
- `docs/NORTH_STAR.md` — what we're building and why

30 minutes. The scaffolding exists; Claude keeps it current.

6.2 Before Complex Features

Create ADR first, code second. What problem? What options? What decision? What tradeoffs?

15–30 minutes. This forces Claude to investigate before executing. Claude writes the ADR; I review and approve before we proceed.

6.3 Before Any Fix

Problem Agreement Protocol:

1. Claude restates the problem
2. I confirm or correct
3. We agree on root cause
4. Then—and only then—Claude generates code

2–3 minutes. This catches wrong-problem-solving before Claude has confidently built the wrong solution.

6.4 After Production Incidents

Update two things:

1. ISSUES_LOG — what happened, what fixed it, what we learned
2. Evergreen Rules — if there's a rule that would prevent recurrence

5–10 minutes. Claude drafts both; I review. Claude reads the rules next session and doesn't repeat the mistake.

6.5 Weekly

Check PRISM metrics. Investment rising? Constrained% dropping? Stability holding?

10 minutes. Objective feedback before subjective awareness.

6.6 The Meta-Principle

Constrain Claude deliberately. Predict outcome. Measure against prediction. Improve.

This is what human-AI collaboration makes possible that human teams struggle with. The constraints are explicit, the economics are quantifiable, Claude maintains the structure, and the improvement cycle is tight.

7 Honest Limitations

This is N=1. Me, one project, 57 days. Generalisation to teams, legacy codebases, neurotypical developers, or different AI tools is unproven.

Classifications were retrospective. Confirmation bias is possible.

Token costs were estimated. Economic claims carry ±20% uncertainty.

Correlation isn't causation. Controlled experiments would strengthen claims.

Effect sizes are large but samples are small. N=3 for proactive ADRs limits confidence.

I'm building products based on this research. Appropriate scepticism advised.

8 Conclusion

I was told in September that the AI learning curve would be too steep. By December, I was publishing research on AI-assisted development methodology, co-written with the AI I was researching.

This isn't a story about being clever. It's a story of a journey and the lessons learned along the way by someone too stubborn to accept "you can't learn this fast enough."

The 276 commits taught me:

1. **Method is predictable.** You can model the economics and quality of an approach before you use it. That's the differentiator of human-AI collaboration.
2. **Claude's default is rapid, confident execution.** Without constraints, every prompt produces immediate action. Often the wrong action.
3. **The right constraints channel Claude's capabilities.** ADRs force investigation. Problem Agreement catches misunderstanding. Evergreen Rules create persistent memory.
4. **Claude maintains the structure.** The documentation I'd struggle to keep current, Claude updates happily. That's what makes the constraints sustainable.
5. **The constraints that help Claude also help me.** Externalised structure works for both of us. That's not a coincidence.

The data shows it. 276 commits don't lie. And neither, importantly, do I—at least not about the 54.3% failure rate in November. That happened. I'm including it anyway.

Further Information

Replication Study: I'm recruiting for an N=20 validation study. If you're a solo developer using AI coding assistants and willing to track your commits for 60 days, contact research@syntropicworks.com.

Academic Paper: "Economic DORA: Practice-Level Analysis of DevOps Metrics in AI-Assisted Solo Development" is available on TechRxiv with DOI 10.5281/zenodo.17894441.

Dataset: Complete commit metadata, classification criteria, and analysis scripts at `https://syntropicworks.com/research/economic-dora-dataset`

Peter Holford is a senior product leader in financial services, currently leading a business lending transformation at a major global bank. Syntropic Works is where he puts his spare-time thinking on AI product development. He has ADHD and three children, which may explain the interest in systems that work despite imperfect conditions.