


Towards AI · [Follow publication](#)

🌟 Member-only story

The Hands-On CTO Vol. 2: The Illusion of Measurement




Building a measurement system for the “AGentic Engineering” revolution ahead

12 min read · 6 days ago



Kapil Viren Ahuja

Following ▾

-  Listen
-  Share
-  More



In my [last post](#), I outlined a 90-day journey: a personal mission to transition from boardroom theory to IDE reality. I stepped back from high-level oversight and immersed myself in the code, working closely with my teams to develop a new AI-powered operating system for software development — a system we call PhoenixOS. And now, the honeymoon phase is officially over.

Month one was about figuring out if I could still code. Turns out, yes — but that was the easy part. Now it's month two, and we're deep in the trenches with two critical projects. The demos went smoothly, and everyone nodded along to the vision slides. But here's where it gets real: we actually have to ship this thing with real deadlines. Real consequences.

The response from my team (and others) has been a mix of awe and anxiety. They observe me (their CTO) coding like never before, but they also feel the change. Skepticism remains, yet it has sharpened. The question is no longer, “What is he doing?” but a more pointed one: “Will this new operating system he’s designed actually work under the pressure of our deadlines?”

Your velocity is up 40%, but your team looks exhausted,” my head of engineering pulled me aside after the sprint review. “What exactly are we optimizing for here?”

If she had asked me next week, I would have pointed to the dashboard, but am I glad that she came early. Her comment got me thinking — have I got this right? And this was just the beginning. We were going to pick the pace up by several notches after Diwali. But her question hung in the air like smoke from a fire I couldn’t yet see. All I could muster was “Let me get back to you on that”, knowing that my entire measurement philosophy was about to unravel.

The metrics have ticked up in the last few weeks - sprint velocity, PRs merged, features “done”, yet I have a profound sense of unease. Should’ve felt great, right? I kept thinking about that disaster from six months back — dashboards all green, perfect sprint reports, and we shipped 20% of what we promised: same pattern, different day. Only now I’m not just watching it happen. I’m leading it. And the stakes? Our entire engineering culture.

The initial excitement of building PhoenixOS was giving way to a much more complicated question: “Is this *actually* better?” It turns out, me figuring out how to measure this thing wasn’t going to be easy — I was getting ready for another ride of my life.

Part 1: The “Mirage of Measurement” and “The Metrics That Lie”

I get it; we’re all riding the Gartner Hype Cycle for AI right now. Peak of Inflated Expectations? Yes. Heading for the Trough of Disillusionment? Maybe.

My job is to somehow get us to that Plateau of Productivity without crashing and burning. You need a real compass for that journey; honest data, not wishful thinking.

But what if the compass itself is broken?

Look, my first instinct? Measure every damn thing. Classic CTO move, right? What if, all those green lights on the dashboard were lies? The old metrics are about as useful as a speedometer in a spaceship.

So yeah, picture this nightmare scenario we were about to walk into. Let's humour ourselves for a bit.

- **Adoption Rate:** What if 85% of developers are using PhoenixOS. But is this voluntary adoption?
- **30% reduction in time-to-PR.** I am already seeing pull requests go faster than before. They were smaller, yes, is the team learning? What if we are shipping bugs faster?
- **Developer satisfaction:** High — it seems right now. Team is motivated. But I know they are burning out. But was this a genuine sentiment, or was it a result because their CTO is on the calls with them, and he is invested and hence this is simply a reflection?

Then there's that Stanford study; 19% performance drop when devs use AI wrong. And MIT's research? 95% of AI initiatives failing? I can't find it but I recall reading that even Github's own data shows that while 88% of devs feel more productive, only 26% of their suggestions actually get accepted. And, this is devs using copilot. That's a lot of rejected code.

And then, it got me thinking — are we headed headfirst into the two fundamental paradoxes of measurement? I'm not betting on just two projects here. I'm betting we can beat those odds. The only way through? Stop lying to ourselves about what the numbers mean.

1. The Determinism Illusion

PhoenixOS is supposed to make everything “deterministic”. And, Software development is messy by nature — requirements change halfway through, technical constraints pop up out of nowhere. We need human judgment for the weird edge cases. So what are we really doing? Trying to force predictability onto something fundamentally unpredictable? That perfect consistency we wanted? As I have been building PhoenixOS, there is “One thing” that I am very confident about — AGentic Engineering will not longer be deterministic — there will be some variability. We all have to get ready for that.

2. Goodhart's Law Death Spiral

You know Goodhart's Law? Once you make a metric a target, people start gaming it. I need to ensure that this doesn't happen — at least not in my pilots. My team knows that I am watching the numbers — much closer that I ever would — hence chances for them making these look good is very high. I want my team to care about the actual improvements and not just hitting their targets. Classic measurement theater — we were optimizing for looking good, not being good — I can't let that happen.

. . .

Part 2: Deconstructing Value — Beyond the Obvious

So I am going to do what any desperate CTO would do — I am torching the idea of the dashboard and just going to kill it. Dead. I am going to tell this to my team after the holidays, and I bet I am going to see some confused looks.

What I am going to do now is, introduce to them, what I call ****The Phoenix Measurement System**** — not just metrics, but metrics about metrics — the famous words!!. We will not count what we do and start measuring what it means.

Here's how I plan to rebuilt our entire approach. From here on, I am planning to break it down into our three core hypotheses, and we will challenge ourselves. Each one increasingly uncomfortable.

Hypothesis 1: PhoenixOS must Reduce Cognitive Load

The promise? AI handles the grunt work so devs can be architects, not bricklayers. Bullshit. I have spent only a few weeks, and you know what I saw? I saw developers spending less time coding and more time writing painstakingly detailed specifications. Less time debugging syntax errors, more time on prompt engineering and trying to understand *why the AI generated what it did*.

Here's what killed me: the cognitive load hadn't disappeared.

It just moved; moved left, moved up. Instead of grinding through implementation details, now my devs were grinding through prompt engineering and spec writing. Is that better? Hell No. How do I know? Because I am also doing the same, and its not fun — trust me. I instead write code then write prompts and read specs all day. Might even be worse; these are builders, not alchemists. They're wired to receive instructions, not write dissertations for an AI.

Better Metric: Net Cognitive Load Assessment.

We will replace our simple satisfaction survey with a weekly, anonymous assessment asking:

- “Rate your mental fatigue. Are you ‘cancelling all weekend plans’ or ‘let’s have a few beers’ tired?”
- How many hours of actual flow state did you achieve? That zone when you forget everything — food, fun, even girlfriend
- Do you need the weekend to recover, or to catch up or are you going to innovate?

We’re not chasing a magic number. We’re watching trends. Are people burning out? **Getting into flow? Remember the flow; we’ll return to it a bit later.**

Hypothesis 2: PhoenixOS must Improve Code Quality

This one gave me actual nightmares. Picture this horror show: PhoenixOS becomes a mediocrity factory. Every function perfectly formatted, excellently documented, perfectly boring. Like if ChatGPT and a compliance officer had a baby.

Hell, it’s worse — at least with human devs you get occasional brilliance mixed with the mess. With AI? You might get perfectly formatted, perfectly predictable, perfectly boring code that technically works but makes you want to cry.

Better Metrics: Quality Floor, Ceiling, and Distribution.

Instead of averaging, we want to look at the whole picture:

1. **Quality Ceiling:** What is the best quality of code we can produce, with or without PhoenixOS? Are we still capable of brilliance?
2. **Quality Floor:** What is the worst quality of code that now makes it into production? Has the safety net improved?
3. **Mutation Testing:** We will stop relying solely on test coverage percentages. Instead, we will start using mutation testing to see if our tests were actually meaningful. A 95% coverage rate is useless if the tests fail to detect when the code is deliberately broken.

The critical question became: Does PhoenixOS raise the floor without lowering the ceiling?

And now comes the metric that makes me lose my cool every single time.

Hypothesis 3: PhoenixOS must Increase Velocity

Velocity. It's the go-to excuse for startups and a comfort for PMs. Don't get me wrong; I'd like to deliver faster too. But then are we delivering product faster or shit? 'Look at our velocity!' - PMs like to boast, even though the builders and alchemists are burning out. It's like dying by a thousand cuts; ship quickly now, but pay the price in tech debt later.

I fear these piles of "quick wins" could quickly become a crushing avalanche of maintenance issues.

A Better Framework: The Sustainable Velocity Index.

Yeah, I know—another formula. But here's the difference — this one actually calls bullshit on the 'ship fast, fix never' mentality. It's like a breathalyzer for technical debt.

$$\text{Sustainable Velocity Index} = \text{Velocity (Sprint N)} / (\text{Velocity (Sprint N+4)} + \text{Cost of Tech Debt Incurred})$$

If our velocity today causes a collapse four sprints from now, that speed wasn't worth it. This (I hope) will force us to measure the second-order effects of our work.

. . .

Part 3: The Hidden Costs and Unseen Benefits

Now that I have clearer view of our immediate hypotheses, I hope to see the deeper, more systemic challenges; the ones that won't show up in any sprint report.

The Innovation Killer Problem

Here's what keeps me up at night: spec-driven development works great when you know what you're building. But what about when you don't? The best code I've ever written came from those "wait, what if we tried ...". These were the codes which made way into products (at least architecture/pattern did). Those accidents when the original plan goes to hell and you stumble onto something brilliant. I hope that we aren't killing that? Trading away all our serendipity for predictability? And how the hell do you measure innovations that never happened?

In the 8 weeks of coding that I have done while using PhoenixOS, I have had a few of these, and here I wish I had tracked those. So, here is our first metric.

Attempted Metric: Solution Novelty Score

I want to introduce a new practice in our peer reviews. Alongside “looks good to me”, I want to ask: “Did this solution teach you something new or solve the problem in a way you wouldn’t have thought of?”. So now everytime we do a PR review we will answer “LGTM” and “IWHT (I Wouldn’t Have Thought (of that))”.

We want to start tracking the frequency of IWHT (“I wouldn’t have thought (of that)”) moments. We were trying to measure whether PhoenixOS was creating competent but ultimately unimaginative code.

The next one I love, because I know that AGentic engineering will either fix atrophy or will cause it.

The Skill Atrophy Problem

This is the Pilot Paradox, and it scares the shit out of me. Yeah, autopilot made flying safer - but one look at the [Air France 447’s disaster](#), it tells you that the pilots forgot how actually to fly. That’s my nightmare scenario for PhoenixOS. We build this AI system, and then what? Our people turn into button-pushers who can’t code their way out of a paper bag?

Think about it: Junior devs won’t learn to code; they’ll just learn to babysit an AI. Can’t write specs, can’t validate output, can’t debug when the magic stops working.

Senior devs? They’ll atrophy into code curators, like museum guards who know where everything is but couldn’t paint to save their lives.

Everyone becomes a goddamn prompt engineer. But here’s the kicker; when you’ve forgotten how the sausage gets made, how the hell do you write prompts that make sense?

Critical Test: The Manual Override

So here’s my non-negotiable test. Everyone takes it, including me.

Every six months, everyone codes raw. No PhoenixOS, just you and the IDE. Junior to principal and to CTO, no exceptions. Some people think I’m paranoid. I call it insurance. Track the scores over time - are we getting dumber or staying sharp? It’s not punishment; it’s a fire drill. Because if AI eventually craps out - I need to know my team can still fly manual. Otherwise we’re just building a house of cards, waiting for the wind.

The Organizational Change Complexity

This isn't really about the tech. It's about people. So I'm going full anthropologist. Truth is, PhoenixOS itself doesn't matter that much. What matters is whether we can change how we think about building software. That's the real battle.

. . .

Part 4: The Reality Compass Framework

Facing this mountain of complexity, I need a new plan. Not a list of 20 KPIs, but a structured way to learn over time. I call it **The Reality Compass Framework**—a three-phase approach to navigating from measurement theater to genuine truth. I will present this framework to my engineering leads.

The Calibration Phase: Are We Doing It Right? (Weeks 1–8)

Look, the first eight weeks? It's all about not tripping over our own feet. I have been doing this for a while now, but we will continue to do this for another 4 weeks or so. Track the basics: Are people even using this thing? Where are they rage-quitting? What commands are they typing wrong fifty times a day? Forget strategic value - we're just trying to make sure the damn thing works. It's like tuning a guitar before you play. Can't make music if you're out of tune, right?

The Validation Phase: Is It Actually Better? (Months 2–6)

Months 2 through 6, shit gets real. Time for A/B testing, and I mean actual science, not the “feels faster” we usually do. Twenty user stories, split down the middle - half with PhoenixOS, half without. Track everything: merge times, bug counts, code quality scores. Anonymous reviews so nobody's kissing ass. Yeah, the politics suck - no manager wants to run the control group, thinks it'll make them look bad. Too bad. This is where we find out if we're brilliant or full of shit - no wiggle room, no excuses. The whole thing - it's non-negotiable.

The Evolution Phase: Does It Scale and Sustain? (Months 6–18)

The long game - months 6 to 18. This is where we separate the winners from the wannabes. Forget velocity charts. Are our devs getting smarter or turning into button-pushers? Is our tech debt exploding while we're high-fiving over story points? Are we solving real problems or just shipping faster garbage? Here's the test: If we killed PhoenixOS tomorrow, would we be better engineers than we were 18 months ago? That's evolution. Not just a system that works — a system that makes us dangerous. Otherwise we just built an expensive crutch.

The Ultimate KPI: The Courage to Fail

After all this measuring and second-guessing, I landed on something that made me uncomfortable as hell. The most important metric? It's not on any dashboard. Can't track it in Jira.

I call it the Intellectual Honesty Index.

It's a simple question: Do we have the courage to abandon PhoenixOS if the evidence suggests we should?

How do you measure guts? Count the "this sucks" conversations—track who's brave enough to publish failure reports. Watch who speaks up in retros versus who stays quiet. If people won't tell me PhoenixOS is broken when it's broken, then we've already lost. The scariest metric? How many people are too scared to say to the CTO his pet project is shit. That number better be zero, or we're just playing corporate theater with better props.

Look, if you can't admit when your pet project is failing, then all these fancy metrics are just BS. You're not measuring to learn anymore - you're measuring to justify something you've already decided. That's not data science. That's just politics with spreadsheets.

We're still in the middle of this mess. Haven't shipped yet. But here's the thing - I don't care about proving PhoenixOS works anymore. What I care about is this: Can we build a team that knows the difference between real value and bullshit metrics? Can we measure what actually matters? And when the data tells us we're wrong, will we have the guts to listen?

The Rise of DevX

Agentic AI

Claude

Thought Leadership

Cto

Leadership



Follow

Published in Towards AI

89K followers · Last published just now

Making AI accessible to 100K+ learners. Find the most practical, hands-on and comprehensive AI Engineering and AI for Work certifications at academy.towardsai.net - we have pathways for any experience level. Monthly cohorts still open—use COHORT10 for 10% off!



Following ▾

Written by Kapil Viren Ahuja

92 followers · 37 following

Creating Architectural POVs | Nurturing architects of the future.


No responses yet



Bgerby

What are your thoughts?

More from Kapil Viren Ahuja and Towards AI

 In Generative AI by Kapil Viren Ahuja

Top-Down AI Adoption Beats Bottom-Up Every Time (Except When It Doesn't)

I have been watching leaders struggle with AI adoption for years now, and I need to tell you something that might sound contradictory...

Sep 15  57



 In Towards AI by Ashish Abraham

No Libraries, No Shortcuts: LLM from Scratch with PyTorch

The no BS guide to build, train, and fine-tune a Transformer architecture from scratch



Oct 2



1.1K



14



In Towards AI by Devi

Running Small Language Models (SLMs) on CPUs: A Practical Guide

The what, why, and how with a practical example, to solidify your learning!

Sep 29



271



4



In CodeToDeploy by Kapil Viren Ahuja

Claude-code vs Codex-CLI

The Showdown between Claude-code and Codex-CLI—what works for me

✦ Oct 13 🖱 68



See all from Kapil Viren Ahuja

See all from Towards AI

Recommended from Medium


 Tim Wang

Spec-driven AI coding: Spec-kit, BMAD, Agent OS and Kiro

For AI-driven development, Spec-kit, BMAD, and OpenSpec are toolkits that enforce a structured, spec-driven approach to minimize the...

Oct 17 🖱 5




 In ITNEXT by Mario Bittencourt

Up your AI Development Game with Spec-Driven Development

Spec Driven Development is new promising way to adopt AI and keep the developer in the loop. I will cover all aspects of SpecKit here.

Oct 20  365  2



 In Coding Nexus by Code Coup

The Claude Skills Cookbook: Anthropic's New Context Engine Outperforms MCP??

Anthropic quietly released the Claude Skills Cookbook on GitHub. Initially, I assumed it was just another dull API document. But after...

★ 3d ago 🖱️ 45



 In Predict by iswarya writes

Claude Haiku 4.5: The AI Coding Revolution That Just Changed Developer Economics Forever

The Collapse No One Saw Coming

★ 5d ago 🖱️ 66 💬 5





Lee Munroe

Empowering Design Agency: How AI is Revolutionizing Early-Stage Prototyping

My team has been diving deep into the AI playground lately. We're seeing a real shift in our initial design approach—jumping straight...

Apr 30 🖱️ 91



Dibya Darshan Khanal

Building a Streamlit Weather Chat App with AWS Kiro: My Journey, Benefits, and a Comparison

“From vibe coding to viable code”—that’s one of Kiro’s taglines, and over the last few weeks, I’ve put it through its paces building a...

Oct 5 🖱️ 1



See more recommendations