How We Built a Way to Measure AI's Impact on Engineering—Without Fooling Ourselves

The story of creating TDPI with a client who refused to guess

The phone call that started it

"We've spent six figures on AI coding tools. Everyone says they're faster. But are we actually shipping more value?"

The VP of Engineering sounded tired. Their company had rolled out GitHub Copilot, then added Cursor, then experimented with AI code review bots. Developers loved them. Anecdotes were glowing. But when the board asked "what's the ROI?"—silence.

They weren't alone. Every engineering leader we spoke to in late 2024 faced the same problem: **AI tools promise productivity gains, but traditional metrics can't prove it.**

Story points? Teams estimate faster now, so points-per-sprint looks flat even though they're shipping more. Lines of code? AI writes verbose code, so LOC goes *up* while quality goes sideways. Velocity charts? Meaningless when the work itself is changing.

We needed a measurement system that could:

1. **Capture real outcomes**, not proxies
2. **Account for quality**, not just speed
3. **Adapt to strategy shifts** (launch mode vs. hardening mode)
4. **Work with imperfect data** from real-world tools
5. **Resist gaming**, because smart people game smart metrics

This is the story of how we built it together.

What we learned in the first month (mostly by being wrong)

Attempt 1: "Just measure cycle time"

We started simple. Pull request cycle time seemed obvious—if AI helps developers code faster, PRs should close faster.

What actually happened: Cycle time *did* drop, but defect rates spiked. Developers were moving fast and breaking things. One team cut their PR time by 40% while their bug backlog doubled.

Lesson: Speed without context is just speed. We needed a **composite view**.

Attempt 2: "Track AI usage vs. output"

We tried correlating Copilot acceptance rates with story completion. High AI usage should mean more stories shipped, right?

What actually happened: The correlation was weak and noisy. Some developers used AI heavily for boilerplate but still got stuck on architecture decisions. Others barely used AI but were extremely productive because they were working in familiar codebases.

Lesson: AI is a tool, not a magic wand. We needed to measure **outcomes**, not tool adoption.

Attempt 3: "Let's survey the developers"

Surely the people using AI daily could tell us if it was working?

What actually happened: Developers *felt* more productive (and they probably were), but feelings don't satisfy a CFO asking about budget allocation. We needed numbers that could stand up to "prove it."

Lesson: Qualitative insights are vital, but we needed **quantitative guardrails** alongside them.

The breakthrough: Four truths, one score

We were in a workshop room—whiteboards covered in failed metric sketches—when the engineering director said something that stuck:

> "I don't need one perfect number. I need to know: Are we faster? Are we shipping real value? Are we maintaining quality? And can I trust the plan?"

That became our framework. Four dimensions that, together, paint an honest picture:

1. Speed – Are we getting code to customers faster?

We settled on **PR cycle time**: from "ready for review" to "deployed." It's imperfect—doesn't capture thinking time—but it's measurable and meaningful. AI should help here by reducing boilerplate and speeding up reviews (when used for code explanation).

2. Value – Are we shipping outcomes that matter?

We tracked **EPICs per week** (big, meaningful features) and **stories per quarter**. This was the antidote to "busy work." If AI made teams faster but they just churned through trivial tasks, this would catch it.

3. Quality – Are we creating problems for later?

Defects per unit of work became our North Star. We normalized it (issues per story, or per point) so teams of different sizes could compare. If AI was injecting subtle bugs through unreviewed suggestions, this would show it.

4. Predictability – Do we deliver what we commit to?

Delivery accuracy: what we delivered divided by what we planned. AI should help here too—better code suggestions mean fewer "oh, that's harder than we thought" moments mid-sprint.

Why normalisation saved us from madness

Early on, we tried comparing raw numbers across teams. Chaos.

- Team A's "fast" PR cycle (1.2 days) looked worse than Team B's (2.1 days), but Team A worked on a legacy monolith while Team B had a greenfield microservice.
- Team C shipped 10 stories in a sprint; Team D shipped 4. But Team D's stories were massive platform changes.

The fix: We stopped comparing teams to *each other* and started comparing each team to *their own past*.

We normalised every metric against that team's **worst historical period** (their "floor"). The worst period = 1.00. Anything above 1.00 = improvement.

Suddenly, we could say: "Team A is 1.8× better than their worst quarter" without arguing about whether their work is "really" harder than Team D's.

The maths (one worked example, so you can sanity-check us)

Let's say Team Alpha's **worst quarter** (before AI tools) looked like this:

Pillar	Worst Period
Speed (PR cycle time)	2.3 days
Value (EPICs/week)	2 EPICs/week
Quality (defects/unit)	3.0 defects/unit
Predictability (accuracy)	0.60

Three months after adopting AI tools (Copilot, Cursor, AI-assisted reviews), their **current quarter** looks like this:

Pillar	Current Period
Speed	1.3 days
Value	6 EPICs/week
Quality	1.8 defects/unit
Predictability	0.90

Now we normalise:

- **Speed** (lower is better): 2.3 ÷ 1.3 = **1.77** → × 20% weight = **0.354**
- **Value** (higher is better): 6 ÷ 2 = **3.00** → × 30% weight = **0.900**
- **Quality** (lower is better): 3.0 ÷ 1.8 = **1.67** → × 20% weight = **0.333**
- **Predictability** (higher is better): 0.90 ÷ 0.60 = **1.50** → × 30% weight = **0.450**

TDPI = 0.354 + 0.900 + 0.333 + 0.450 = 2.04

Translation: Team Alpha is now operating at **~2× their previous worst performance**, with most gains coming from **Value** (shipping more meaningful work) and **Predictability** (hitting their commitments).

That's a story you can tell a CFO.

What actually changed on the ground

The metric didn't improve teams. The **conversations** the metric enabled did.

Month 2: The "fast lane" experiment

TDPI showed Team Beta's **Speed** was lagging—PRs sat in review for days. We introduced a "fast lane": any PR under 50 lines with passing tests got 30-minute SLA reviews.

Result: Speed score jumped from 1.2 to 1.9 in one sprint. Developers started writing smaller, AI-assisted PRs because they knew they'd get fast feedback. A virtuous cycle.

Month 4: The quality wake-up call

Team Gamma's **Quality** score dropped (1.3 → 0.9, meaning *worse*). Digging in, we found developers were accepting Copilot suggestions without reading them carefully—copying subtle bugs in error handling.

Fix: We added a "AI-generated code" checkbox to PR templates and made reviewers specifically check those sections. Quality rebounded to 1.5 within three sprints.

Month 6: The predictability breakthrough

Team Delta's **Predictability** was stuck at ~0.7 (delivering only 70% of planned work). The root cause? Mid-sprint scope creep. AI made coding feel so fast that product managers kept adding "just one more thing."

Fix: We instituted a **sprint freeze**: no new work after day 2 unless something is dropped. Predictability climbed to 0.92. Ironically, they shipped *more* over the quarter because they finished things instead of starting ten.

The weights: Strategy in four percentages

We set default weights—**Speed 20%, Value 30%, Quality 20%, Predictability 30%**—but the real power came from *adjusting* them.

Example: Launch mode

Client was releasing a new product. For that initiative, we shifted to:

- **Speed 25%** (get it out fast)
- **Value 35%** (ship the MVP features)
- **Quality 15%** (we'll harden it later)
- **Predictability 25%** (stay on schedule)

This made trade-offs explicit. Everyone knew quality was temporarily de-prioritised, and we had a plan to re-weight it post-launch.

Example: Platform hardening

Six months later, technical debt was biting. We shifted to:

- **Speed 15%**
- **Value 20%**
- **Quality 35%** (fix the bugs)
- **Predictability 30%** (deliver the debt paydown we scoped)

TDPI still worked—it just measured what *currently mattered*.

The AI effect, quantified (what we learned after 12 months)

After a year of measuring, we could finally answer the original question: **"What's the ROI of AI tools?"**

Across five teams (anonymised composite data):

Pillar	Pre-AI (baseline)	Post-AI (12 months)	Improvement
Speed	1.00	1.68	**+68%**
Value	1.00	2.40	**+140%**
Quality	1.00	1.52	**+52%**
Predictability	1.00	1.44	**+44%**

Aggregate TDPI: 1.00 → 1.76 (**76% improvement**)

But the real insights were in the **decomposition**:

- **Speed gains were real but modest.** AI helped, but process changes (faster reviews, smaller PRs) mattered more.
- **Value gains were huge.** Teams shipped 2.4× more EPICs. Why? AI eliminated grunt work, freeing time for high-impact features.
- **Quality improved despite speed increases.** This surprised everyone. Theory: AI caught silly bugs (typos, missing null checks) that used to slip through.
- **Predictability improved.** Fewer "this is harder than we thought" surprises. AI helped with unfamiliar libraries and frameworks.

What we got wrong (and how we fixed it)

Mistake 1: Over-explaining the math

Early scorecards were dense. Executives glazed over.

Fix: We created a **one-page dashboard**—four pillar scores, one TDPI number, one trend arrow, two bullet points on what changed. That's it.

Mistake 2: Comparing teams publicly

Some teams had higher TDPI scores. Others felt bad. Morale dipped.

Fix: TDPI became **team-private**. Only aggregate, anonymised trends went to leadership. Teams competed with their own past, not each other.

Mistake 3: Not documenting context

One quarter, Team Epsilon's Speed crashed. Panic. Turns out they'd onboarded three juniors. Context matters.

Fix: Every score now has a **context note field**. Hiring spikes, audits, outages—if it's not normal, write it down.

Mistake 4: Changing definitions mid-flight

We tweaked the "defect" definition in month 5. Historical comparisons broke.

Fix: Definitions are now **versioned and locked per quarter**. Changes only apply to *future* periods.

The rituals that made it stick

Metrics don't change behavior. **Rituals** do.

Weekly 15-minute stand-up

- What's the current TDPI?
- Which pillar moved most (up or down)?
- What's one experiment we could run this week?

No blame. No deep dives. Just: observe, hypothesise, try something small.

Monthly steering committee

- Show trend lines (3-month rolling view).
- Highlight one team doing something clever.
- Adjust weights if strategy shifted.

Quarterly retrospective

- Review the year-over-year TDPI.
- Celebrate improvements.
- Pick *two* focus areas for next quarter (not ten).

The hard questions (and our honest answers)

"Aren't you just measuring what's easy to measure?"

Guilty as charged. We don't capture "thinking quality" or "architectural elegance" directly. But we believe these show up indirectly—good architecture leads to fewer defects and faster changes over time. If TDPI trends down despite faster PRs, that's a signal something's off.

"Can't teams game this?"

Yes, but it's hard. Gaming Speed (tiny PRs) hurts Quality. Gaming Value (lots of small EPICs) hurts Predictability. Gaming Quality (over-testing) hurts Speed. The four pillars create tension *by design*. We also watch for anomalies—sudden jumps get investigated.

"What if AI just makes bad code faster?"

That's exactly what Quality and Predictability catch. If AI-generated code ships fast but breaks in production (Quality drops) or requires more rework than estimated (Predictability drops), TDPI will flag it.

"Do developers resent being measured?"

Some did at first. Two things helped:

1. **No individual scores.** TDPI measures teams and initiatives, not people.
2. **Transparency.** We shared the full methodology. When developers understood *why*, resistance dropped.

How to try this yourself (90-day pilot)

Weeks 1–2: Set the foundation

- Pick one team or product area.
- Define the four pillars in plain English (one sentence each).
- Agree on initial weights based on your current strategy.
- Publish a draft scorecard and get feedback.

Weeks 3–6: Baseline & instrument

- Pull data from existing tools (GitHub, Jira, Linear, whatever you use).
- Choose a "worst period" (could be last quarter, or the quarter before AI tools arrived).
- Calculate normalised scores for each pillar.
- Set your baseline TDPI = 1.00.

Weeks 7–12: Improve & learn

- Identify your lowest-scoring pillar.

- Pick *one* experiment to improve it (e.g., faster reviews, clearer acceptance criteria).
- Recompute TDPI weekly.
- Hold a 15-minute "what moved?" meeting.

Success criteria: Not "TDPI went up" (though that's nice). Success is: **"We made a decision based on this data that we wouldn't have made otherwise."**

The question we still can't answer (and that's OK)

"How much of the improvement is AI, and how much is just better process?"

Honestly? We don't know. And we've made peace with that.

AI tools were the catalyst—they made speed gains *possible*, which created slack, which let teams tackle process debt. But TDPI measures the **combined effect** of tools + process + people.

For our client, that was enough. They weren't trying to write an academic paper. They needed to know: "Should we renew these AI tool licenses, and should we invest more in training?"

TDPI said: **Yes, and here's where to focus next.**

Where we go from here

We're now running TDPI across twelve product teams at three companies. The patterns are consistent:

- **Value** improves most dramatically (AI eliminates grunt work).
- **Speed** improves modestly (AI helps, but process matters more).
- **Quality** improves slightly *if* teams adapt their review practices.

- **Predictability** improves when planning gets more honest.

We're also experimenting with:

- **Leading indicators** (can we predict TDPI changes before they happen?).
- **Team health overlays** (correlating TDPI with engagement scores).
- **Cost normalisation** (TDPI per dollar spent on tooling).

But the core framework—four pillars, normalised to past performance, weighted to strategy—has proven robust.

The real bottom line

If you'd asked us two years ago "how do you measure AI's impact on engineering?", we'd have shrugged.

Today, we have an answer—not a perfect one, but an **honest** one.

TDPI doesn't tell you whether to adopt AI tools. It tells you whether they're *working*. And when they're not, it shows you where to look.

For the VP who called us, that was worth every hour we spent arguing about defect definitions in that whiteboard room.

Want to pilot TDPI with your team? We'll help you set it up—and if it doesn't make decisions clearer within 90 days, we'll help you turn it off.

ThoughtFox: Navigating complex organisations with AI so value flows faster.

Learn more at www.thoughtfox.ai