The Fastest-Learning Team Wins

A CTO's guide to leading the AI transformation in Software Development

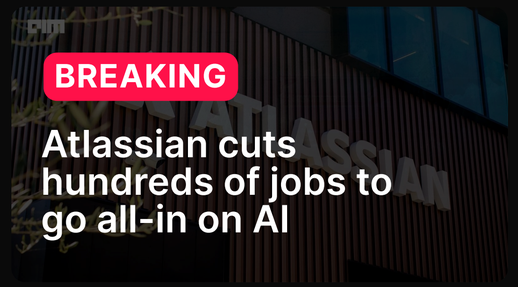
NaNLABS
YOUR TECH SIDEKICK

Table of Contents

"We now have a shared understanding of the sorts of questions we should be asking. That might be the most valuable outcome of all."

Martin Fowler, Thoughtworks Future of Software Development Retreat, February 2026

The Headlines Aren't Talking To You



You have seen the headlines.

Jack Dorsey cut nearly half of Block's workforce, close to 4,000 people, and explicitly tied the decision to AI reshaping labor productivity. The board saw a 24% stock pop the day the announcement dropped. Atlassian reorganized entire engineering divisions. Other companies announced they were replacing software roles with AI agents outright. The narrative is loud, urgent, and building momentum.

Here is what it leaves out: those companies have thousands of engineers, dedicated platform teams, and legal and HR infrastructure built specifically to absorb that kind of disruption. They can run that experiment because they have the margin for error that comes with operating at that scale. When a company with 8,000 engineers restructures 400 roles, it barely registers on the org chart.

You have a completely different set of constraints.

You are managing a team of 5, 20, maybe 60 engineers. Every hire has a name. Every decision lands directly on real

people, real clients, and real revenue. You probably don't have a dedicated people operations function to absorb the fallout of a structural reorganization. The playbook being written for enterprise engineering organizations is not your playbook; and treating it as if it were is one of the more reliable ways to damage something that took years to build.

This ebook is for you.

At **NaNLABS**, we have spent 13 years embedded inside engineering teams, operating across 10 to 15 client companies simultaneously, mostly at exactly the team sizes we are describing. We have watched this transformation unfold from the inside, not from a conference stage or an analyst briefing. What we have seen does not match the headlines. It is more nuanced, more human, and frankly more interesting than the narrative of replacement and disruption would suggest.

What follows is our honest read on what is actually happening in software development, why it matters, and what it genuinely requires from you as a leader.

We will draw on research, data, and patterns built across 13 years of client work. We will not pretend there is a universal playbook, because there is not one. What there is, however, is a clear direction. **And to understand it fully, you need to start much further back than most people realize.**

Chapter 1

This Moment Has Roots

The Argument That's Been Building Since 1999

1999

Extreme Programming Explained and The Pragmatic Programmer

Frame software as a social discipline and a craft. Ownership, collaboration and responsibility take center stage.

2001

Software Craftsmanship: The New Imperative

Pete McBreen argues that software cannot be industrialized without losing quality.

2008

Clean Code

Robert C. Martin popularizes disciplined coding practices and raises the bar for technical excellence.

2009

Software Craftsmanship Manifesto

Formalizes the philosophy: well-crafted software, continuous value, professional communities, and strong partnerships.

2014

Software Craftsmanship by Sandro Mancuso

Synthesizes the idea: the real job is not writing code, but owning outcomes and understanding the business.

2026

In the AI era

The argument doesn’t change.
The economic pressure does. What remains is judgment, ownership, context and trust.

The most important thing to understand about what AI is doing to software development is this: it is not a new argument.

It is the same argument the industry has been having since the late 1990s, finally arriving with enough economic force that it cannot be ignored.

Go back to 1999. Two books were published that would quietly reshape how serious practitioners thought about what the discipline actually required.

Kent Beck's Extreme Programming Explained argued that software development was fundamentally a social and collaborative activity, that the customer belonged inside the team, that ownership of outcomes should be collective, and that technical excellence was a professional obligation rather than a personal preference.

That same year, Andy Hunt and Dave Thomas published **The Pragmatic Programmer,** coining a phrase that has outlasted almost everything else from that era: "sign your work." Own what you build. Take pride in it. Treat your craft as something worth putting your name on.

Two years later, in 2001, Pete McBreen pushed further in **Software Craftsmanship: The New Imperative.**

His argument was explicit and, at the time, provocative: quality software cannot be manufactured. It has to be built by people who care about the outcome and take responsibility for it. One of his chapter titles stated it plainly; "Software Craftsmanship Is a Rejection of Narrow Specialization", and he made the case that the industry's obsession with decomposing work into narrow, interchangeable tasks was not efficiency. It was a category error about what the work actually was.

By 2008, the idea had enough institutional weight that Robert C. Martin proposed adding a fifth value to the Agile Manifesto **“Clean Code”.**

In 2009, a group of practitioners met in Libertyville, Illinois, and drafted what would become the **Manifesto for Software Craftsmanship,** formalizing the argument: well-crafted software, steadily adding value, a community of professionals, productive partnerships.

The synthesis came in 2014, when **Sandro Mancuso published The Software Craftsman.** No AI. No LLMs. No predictions about automation. Just a clear-eyed argument about what the role had always actually required: communicating directly with clients, taking ownership of outcomes, understanding the business you were building for, forming productive partnerships rather than executing tasks in isolation. Understanding the why, not just the how.

Why the industry kept ignoring it.

Here is the part that matters: the industry heard this argument every time. And every time, it drifted back toward narrow specialization and task execution.

Why? Because narrow specialization was easier to hire for. Easier to measure. Easier to slot into a project plan. And, crucially, there was enough demand for software development in the 2000s and 2010s that you could sustain a business on task execution alone. The market did not punish the drift, so the drift continued.

The **"10x engineer"** mythology of the 2010s is a useful case study. It celebrated raw individual technical output as the highest form of contribution. It was the exact opposite of what Beck, McBreen, and Mancuso were arguing and it dominated the hiring culture of a generation of tech

companies. Thousands of engineering organizations structured themselves around finding and retaining individual technical superstars, while systematically underinvesting in the collaborative, communicative, ownership-oriented capabilities that would have made their teams resilient. That trade-off is now reversing with speed.

AI isn't the disruption. It's the deadline.

AI is not a new argument about what software development requires. It is the moment the same argument finally has enough economic pressure behind it that the industry cannot look the other way.

Here is why: when AI handles the straightforward execution layer, boilerplate code, repetitive patterns, well-defined transformations, what remains is exactly what the craftsmanship movement said was always the real work:

- **Communicating with clients**
- **Understanding business context**
- **Making judgment calls in ambiguous situations**
- **Owning outcomes**
- **Forming productive partnerships.**

The narrow specialist who was valuable precisely because they could execute well on a well-defined task is now competing with a tool that executes faster, doesn't need onboarding, and is available at 3am. That competitive position is deteriorating. Not instantly, and not uniformly, but the direction is clear.

The engineer who was always the most valuable, the one who understood the problem before writing a line of code, who could sit across from a client and build trust, who took genuine ownership of whether the product worked in the real world , that person is becoming more valuable, faster.

This means something significant for how you think about your team and your role: you are not reinventing the software development function from scratch. You are completing an evolution that was already underway. The destination hasn't changed, but the urgency has.

The best minds in the field declined to write a new manifesto.

In February 2026, Thoughtworks hosted the Future of Software Development retreat in Utah, deliberately timed to mark the 25th anniversary of the Agile Manifesto, held just miles from where the original was written. The attendee list read like the discipline's Hall of Fame: Martin Fowler, Kent Beck, Steve Yegge, Gene Kim, Laura Tacho, Gergely Orosz, roughly 50 people in total.

The question everyone expected them to answer:

Would they produce a new manifesto for the AI era?

Their answer was explicitly no.

Fowler and Thoughtworks CTO Rachel Laycock said so directly. What they produced instead was a 17-page summary of eight themes, including "**Where does the rigor go?**", "**The middle loop: a new category of work**", and "**The human side: roles, skills, and experience**", and a shared acknowledgement that they left with more questions than answers.

Fowler's framing: **"We now have a shared understanding of the sorts of questions we should be asking. That might be the most valuable outcome of all."**

Sit with that for a moment. In 2001, the people in that room felt confident enough to write a manifesto that shaped an industry.

In 2026, the same caliber of people, including the person who wrote Extreme Programming, explicitly declined to do so. They considered the question and concluded that intellectual honesty required admitting they didn't know enough to prescribe.

That posture is more useful to you as a leader than any confident prescription would be. Anyone selling you certainty about where this goes is either misinformed or selling something. The practitioners closest to the work are sitting with the complexity and being honest about it.

That is the tone we want to bring to this guide.

Chapter 2

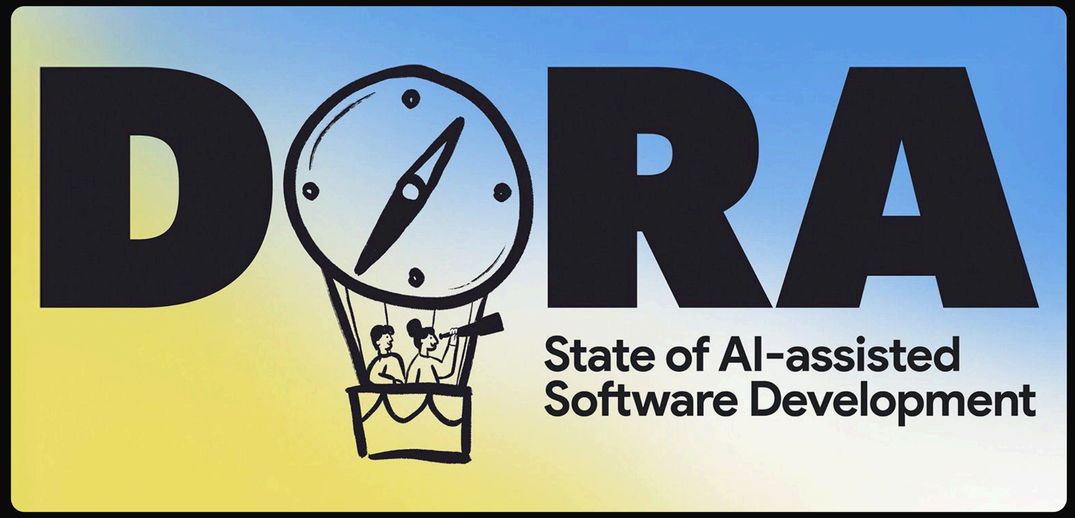
What's Actually Happening On The Ground

Strip away the noise.

Before looking at patterns, it is worth acknowledging a specific problem with almost all AI-in-engineering research: **the signal-to-noise ratio is poor.**

Studies showing 30–55% productivity gains from AI coding tools circulate widely. So do studies showing negligible gains on complex tasks. Both are probably accurate, they are measuring different things in different contexts, and the aggregate numbers hide more than they reveal.

A study measuring how fast engineers can complete a well-defined coding exercise in a controlled environment tells you very little about how those same engineers will perform on the ambiguous, context-dependent, collaboration-heavy work that makes up most of what high-performing teams actually do.



The 2025 DORA Report, one of the most rigorous and methodologically consistent annual studies of software delivery performance, is more useful precisely because it measures the full system. Its central finding on AI is not about productivity in isolation. It frames AI's role as that of an amplifier: a force that reflects back the quality of your existing engineering pipeline, culture, and practices. That framing deserves more attention than it typically gets.

Two patterns, and what they share:

When we look across our active client base, two distinct patterns emerge. They look different on the surface and involve completely different kinds of organizations. They point in exactly the same direction.

1. **Established companies using AI to grow, not cut.**

Companies with revenue to protect, clients who depend on them, and reputations built over years. They are not AI-native organizations, and they aren't moving recklessly, nor should they. What they are doing is using AI as a force multiplier: more efficiency per engineer, higher output per sprint, more market share within the same headcount.

Nobody in this group is talking about replacing their engineering team. They are asking how to make it stronger.

Sometimes they are driving the transformation themselves; sometimes we are helping them push it forward. In either case, what we consistently find is that where it is working, it is working because the fundamentals were already solid. Good communication between engineers and clients. Clear ownership of outcomes, not just tasks. Engineering practices that hold up when things get complicated.

AI, in these environments, is not creating the conditions for success. Its accelerating something that was already there.

2. **Greenfield projects built on the assumption that AI will solve the hard parts.**

These are newer engagements , younger companies, clients starting from scratch, sometimes seed-stage startups who have read the same headlines about AI dramatically reducing development costs and timelines.

The expectation coming in is often that AI will compress the path from idea to product. Sometimes that expectation is realistic. More often, it is not, and the gap between expectation and reality produces a specific failure mode.

What we find underneath that expectation is almost always the same thing: the absence of a clear product vision. They do not yet fully know what they want to build or why their intended users would choose it over alternatives. AI can accelerate execution dramatically. What it cannot do is replace clarity of direction. When that clarity is missing, adding more tools makes things worse faster, because you can generate code at speed in the wrong direction.

Part of our job, before a single line of code gets written, becomes helping them develop that clarity. Defining what success looks like. Who the user is. What problem the product is genuinely solving. That is not a technical problem. It is a product thinking problem. And it is one that AI makes more urgent, not less.

The amplification effect in practice.

The **DORA** finding about AI as an amplifier becomes more concrete when you see it play out in real teams.

Laura Tacho, former CTO of DX and one of the more rigorous researchers working on engineering productivity, presented corroborating data at The Pragmatic Summit in San Francisco in February 2026.

Incidents reduced by 50%

Laura Tacho, The Pragmatic Summit (Feb 2026)

Her research found that organizations with healthy engineering systems, strong deployment practices, clear ownership, good feedback loops, **are seeing up to 50% fewer incidents as AI tools are introduced.** Organizations that were already experiencing dysfunction are becoming more dysfunctional, faster.

The mechanism is straightforward once you see it: AI tools accelerate whatever loop your team is already running.

If your team has strong code review practices, fast feedback, and engineers who take genuine ownership of what they ship, AI tools make that loop faster and the outputs better. If your team has weak code review, unclear ownership, and engineers who treat their work as done when the pull request is merged, AI tools produce more code of that quality at higher velocity. More surface area. More debt. More incidents.

This is not a pessimistic finding. It is a clarifying one. **It tells you exactly where to focus.**

The fragmented adoption trap.

There is a third pattern we observe regularly, one that cuts across both of the above and represents one of the more common ways well-intentioned AI adoption goes wrong.

Call it **fragmented adoption.** Every engineer experimenting individually with their own AI tools, their own workflows, their own prompting strategies. Individual setups, individual discoveries, individual preferences, but no shared direction, no systematic knowledge transfer, no organizational intelligence being built.

The engineer who spends three weeks developing a genuinely useful approach to AI-assisted code review

keeps that knowledge in their head or their notes. When they leave, or move to a new project, it goes with them.

This is not a competitive advantage.
It is the engineering equivalent of everyone on a sports team developing their own fitness routine independently, with no shared playbook.

A 2026 LeadDev analysis of AI adoption in engineering organizations found that fragmented tool usage, where individuals experiment without organizational structure around learning and sharing, consistently underperforms relative to teams that systematize adoption, even when the systematized teams are using less cutting-edge tools.

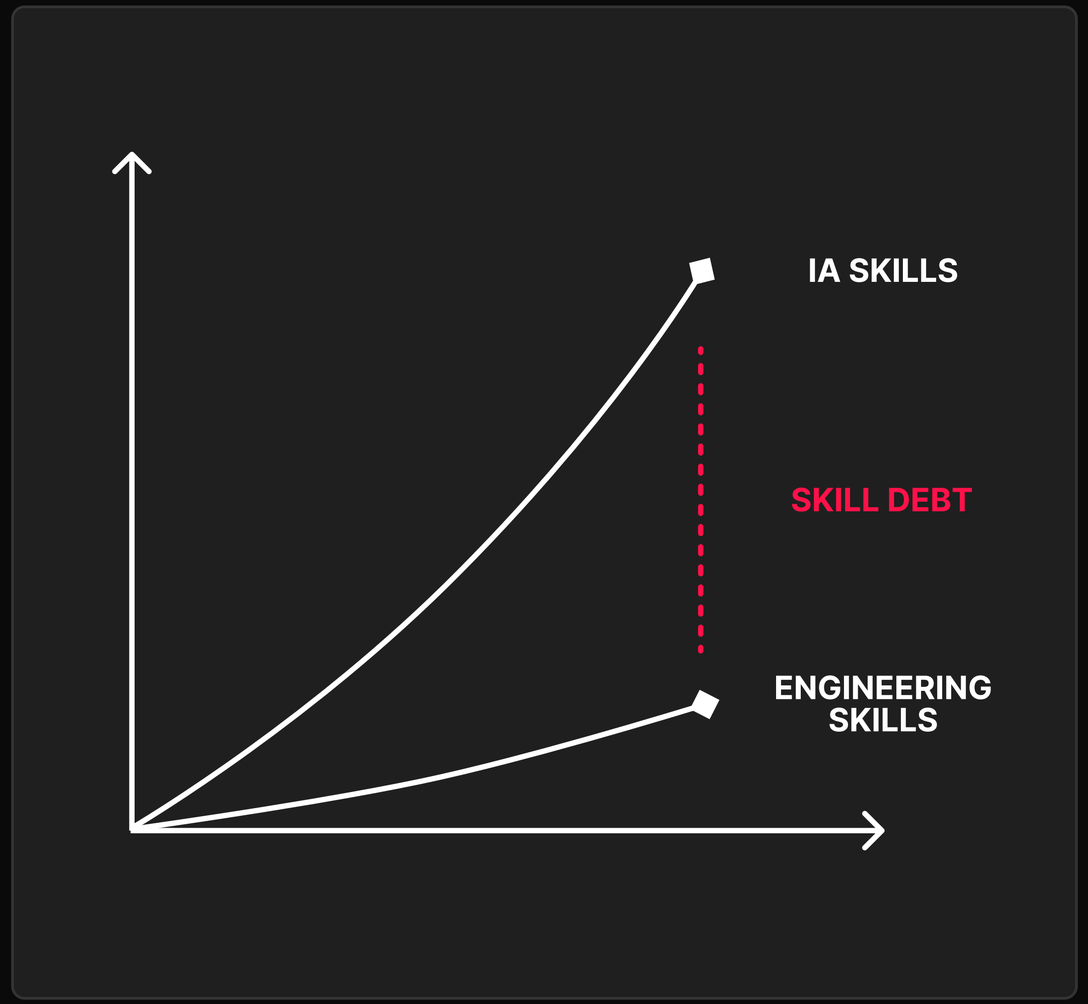
The discipline of making learning collective turns out to matter more than the sophistication of the tools being adopted.

The teams pulling ahead are not the ones with the best individual tools. They are the ones where learning happens at the organizational level. Where one person's discovery becomes three people's practice. Where the system gets smarter, not just the individuals.

That distinction is almost entirely a leadership variable... Which means it is yours to own.

Chapter 3

You Can't Hire Your Way Out Of This



The reflex that no longer works.

When engineering organizations hit a capability gap, the instinct is to hire for it. For most problems in engineering leadership, that instinct is sound. Need more backend capacity? Hire. Need mobile expertise you don't have in-house? Hire. Need someone who knows your particular cloud platform deeply? Hire. The external talent market is the natural and usually correct solution to a skills deficit.

That logic is breaking down in a specific and important way when it comes to AI capabilities, and the breakdown is structural, not cyclical.

The skills most in demand right now, building production-grade AI agents, evaluating LLM output reliability, orchestrating multi-step workflows, integrating real-time inference into existing systems, became a real, distinct

discipline roughly twelve months ago. The talent pool that has done these things well, in production, at scale, is thin. The engineers who do it well are expensive, are receiving multiple competing offers, and are extremely difficult to retain. Supply is simply not keeping pace with demand, and the gap is not closing quickly.

But here is what matters more than supply: the half-life of AI tooling knowledge is now measured in months, not years. Gergely Orosz and Elin Nilsson documented this in a rigorous survey of AI tooling published in The Pragmatic Engineer in March 2026: engineers who were considered competent practitioners of specific AI-assisted workflows in mid-2025 found significant portions of those workflows obsolete or superseded by Q1 2026. New models, new frameworks, new capabilities , the landscape shifted enough in six months that expertise had a meaningful expiration date.

This creates a problem that hiring cannot solve. Even if you find the right person today, in three months you will need something different. A framework that didn't exist last quarter. A capability that just became possible because of a model released last week. You will be back in the same position, looking externally for skills your team doesn't have. **Hiring defers the problem, it does not resolve it.**

The concept of skill debt.

There is a useful analogy here to technical debt, a concept every engineer understands intuitively.

Technical debt accumulates when teams take shortcuts in implementation that make future work harder. It is not inherently wrong to incur it; sometimes the right trade-off at a given moment is to move fast and clean up later. The problem comes when debt accumulates without

acknowledgment, compounds over time, and becomes a structural drag on everything the team tries to do.

Skill debt works the same way. Every time an organization responds to a capability gap by hiring rather than building internal learning capacity, it is incurring skill debt. Every time upskilling is deferred because the current sprint is too demanding, it accumulates. Every time knowledge gained by one engineer fails to transfer to the team, the debt compounds.

At moderate levels, skill debt is manageable. At the pace of change the current AI environment is producing, unmanaged skill debt becomes existential faster than most CTOs have experienced before.

The organizations building durable competitive advantage right now are not doing so by making better individual hiring decisions than their competitors. They are doing so by developing the organizational capacity to learn faster , to turn external change into internal capability continuously, not episodically.

The question that cuts through.

There is a diagnostic question worth putting to yourself, and eventually to your engineering leads, with honesty:

Does your engineering organization learn fast enough to stay ahead of a market that changes every 90 days?

Not your best individual engineer. Your organization. The system, not just its most capable nodes.

If the answer is yes, you have an asset that compounds. If the answer is no, or even "I'm not sure", everything else you are doing is downstream of that gap. Tool decisions, architecture choices, hiring plans: all of it rides on the organization's ability to absorb and apply new information. If that capacity is limited, the compounding works against you regardless of what else you do well.

This reframes upskilling in a way most engineering organizations have not fully internalized. Upskilling is typically treated as a professional development benefit, something that appears in an annual review conversation, gets allocated a modest budget line, and is measured by courses completed or certifications earned. It is seen as something you do for engineers.

That framing needs to change. Upskilling your team is a strategic investment in your competitive position, in exactly the same category as infrastructure decisions or architectural choices. The way you prioritize it, fund it, and hold yourselves accountable to it needs to reflect that.

Teams that treat learning as a productivity lever, not a benefit, are the ones moving fastest right now. That is not an accident.

Chapter 4

The Profiles That Are Pulling Ahead

The prediction most CTOs would get wrong.

Ask a room of CTOs to describe the engineer who is thriving in the current environment, and most would describe something like: an experienced senior engineer who adopted AI tools early, is technically adventurous, and has deep expertise in modern infrastructure.

That description correlates weakly with what we actually observe. The engineers navigating this transition well look somewhat different , and understanding why matters for how you think about your team's development. The consistent profile is not defined by seniority, tenure, or early tool adoption. It is defined by three characteristics that often appear together: **strong technical fundamentals, active intellectual curiosity that extends beyond their immediate domain, and a pull toward broader ownership of outcomes rather than just outputs.**

What doesn't predict success	What does
Seniority / years of tenure	Strong technical fundamentals
Early AI tool adoption	Active intellectual curiosity beyond their domain
Deep narrow specialization	A natural pull toward broader ownership
Using AI to produce faster	Using AI to learn faster

These engineers use AI to learn faster, not just to produce faster. That distinction is significant. An engineer who uses AI to produce more code more quickly is gaining velocity. An engineer who uses AI to understand a domain they don't know, to reason through an architectural trade-off they haven't faced before, to develop mental models that will be useful across many future problems.

In Martin Fowler's framing, these are the Expert Generalists, T-shaped engineers whose horizontal bar keeps expanding, not because they were told to become generalists, but because the nature of the work kept pulling them outward and they followed that pull. The depth is still there. The breadth keeps growing.

Why strong fundamentals matter more, not less.

There is a counterintuitive pattern in how AI tools interact with technical skill levels, and it has significant implications for how you think about your team.

The conventional narrative suggests AI tools are a great equalizer, that they bring junior engineers up closer to senior level by handling the parts of the work where experience normally matters. There is some evidence for this on well-defined, narrow tasks.

On complex, ambiguous, or high-stakes work, the dynamic reverses. Engineers with strong fundamentals use AI to move faster through familiar territory and explore new territory more confidently, because they have the judgment to evaluate what the tool produces. Engineers with weaker fundamentals struggle to distinguish good AI output from plausible-looking but subtly wrong output, and in complex systems, plausible-but-wrong is more dangerous than obviously wrong.

The **2026 CACM paper** by Russinovich and Hanselman on redefining the software engineering profession makes this point precisely: as AI handles more of the execution layer, the premium on engineering judgment increases, not decreases. Knowing what to build, knowing whether what was built is right, and knowing when to trust and when to verify AI output, these are judgment capabilities, not execution capabilities. They come from strong fundamentals, accumulated experience, and the kind of breadth that lets engineers reason across contexts. This is an argument for investing in your engineers' depth.

The Forward Deployed Engineer, and what it means for your team.

In February 2026, First Round Review published a detailed examination of a role profile that has been gaining traction: the Forward Deployed Engineer.

The FDE is an engineer who operates at the intersection of technical capability and direct client engagement, someone who can sit across from a stakeholder, understand the business problem without requiring it to be translated into a technical specification first, and come back with a direction that is both technically sound and commercially meaningful. Not a project manager. Not a solutions architect. An engineer who can do the technical work and hold the full context of why it matters.

At larger organizations, Palantir made the role famous, this is a distinct function. You can hire for it, structure teams around it, develop a career path for it.

At your team size, that structure doesn't exist. But the capability is arguably more important. When every engineer on your team needs to be moving quickly and making good decisions, the ability to understand the client's world, and

bring technical judgment to bear on real business problems, cannot be siloed in one function. It needs to be distributed.

The practical question this raises: who on your team has the raw materials to develop this capability?

Not the finished product, it takes time and the right conditions. But the instincts: the curiosity about why the product exists, the willingness to engage with client-facing conversations, the ability to hold ambiguity without defaulting immediately to "let me get the requirements written up."

Identifying those people and deliberately creating the conditions for them to grow into this space is one of the highest-leverage investments available to a CTO at your team size.

The bottleneck with your name on it.

Here is something worth being direct about, because it tends to be the unspoken dependency in many of the engineering organizations we work with.

There is probably one person on your team who can walk into a client conversation, understand the business problem, reason across the full technical stack, and come back with a direction, without needing to escalate. That person is almost certainly you. And that is not a compliment, it is a structural vulnerability.

As long as you are the only person operating in that space, combining technical judgment, business context, and the credibility to represent both to a client or to a board, you have not built a team. You have built a bottleneck with your name on it. Will Larson, in a March 2026 post for Irrational

Exuberance, makes the argument directly: **in the AI era, the scarce resource is not technical skill. It is judgment, the capacity to apply technical understanding to ambiguous real-world situations and make good calls.If that capacity lives entirely in you, your team's ceiling is your availability.**

This is a useful provocation because it reframes what developing your people actually means. It is not about creating managers. It is about creating engineers who can own problems, not just tasks, who can sit with a client's confused request, ask the right questions, and return with a direction rather than a list of clarifying tickets.

The teams where this is happening have something in common: the CTO made it structurally possible. They created the conditions, client access, domain exposure, the psychological safety to make judgment calls and sometimes be wrong, that let capable engineers grow into this space. **That is a leadership decision.**

Chapter 5

This Is A Change Management Problem, Not A Technology Problem

What the practitioners doing this at scale say.

We work with a client whose entire business is helping large enterprises bring AI into practice. Not talking about it. Actually doing it , with companies everyone reading this would recognize. What the owner tells us consistently is the same observation, regardless of which organization they are working with:

> *"This is a change management problem, not a technology problem.*
>
> **The AI part is maybe 10% of the project. The other 90% is getting people aligned, helping stakeholders understand what they're actually trying to accomplish, and creating the conditions for the change to stick."**

That observation comes from someone who does this at scale. If change management represents 90% of the problem at enterprise scale, it does not get smaller at your team size. In some ways it is harder, because you do not have a dedicated organizational effectiveness function, a change management lead, or an internal communications team. The function that handles all of that is you.

The research is consistent on this point. **Harvard Business School professor Amy Edmondson's** decades of research on team performance identifies psychological safety, the shared belief that the team is safe for interpersonal risk-taking, as the single most consistent predictor of team learning and adaptability. Technical capability or tooling. The degree to which people feel safe to raise concerns, admit uncertainty, and try things that might not work.
AI transformation requires all three in high volume. If the environment punishes them, the transformation stalls.

A **2026 LeadDev** analysis corroborates this from the engineering-specific angle: the primary barriers to successful AI integration are not technical.

They are cultural and organizational:

- **Resistance to change**
- **Misalignment between tool adoption and workflow design**
- **The absence of shared direction consistently outrank technical challenges as the reasons AI initiatives underdeliver.**

This is useful framing for CTOs who are spending the majority of their time on technical decisions about AI. The technical decisions matter. But the organizational conditions that determine whether those decisions translate into outcomes matter more.

The three levers that actually move teams.

Based on 13 years of watching engineering organizations navigate significant transitions, and the more recent experience of AI transformation specifically, we have found that three practices consistently distinguish teams making genuine progress from teams generating noise.

1. **Normalize experimentation publicly, including failure.**

This one is harder than it sounds, because it requires a specific kind of vulnerability from you as a leader. If you only share wins, the AI tool that worked, the experiment that paid off, the approach that landed exactly as intended, your team will mirror that behavior. They will show you wins. They will absorb the cultural signal that failure is not safe to surface, and the real friction in your organization will become invisible to you.

The teams moving fastest have leaders who talk openly about what didn't work. We tried X and it underdelivered for this reason. We ran this experiment and it taught us something we didn't expect, here's what it was. That behavior models the intellectual honesty the moment requires, creates permission for your engineers to do the same, and, critically, keeps you visible to the real texture of your organization's experience with this transition.

2. Make learning collective, not individual.

One engineer going deep on a new capability and keeping it to themselves is not organizational learning. It is individual learning, which has real value but compounds slowly and is fragile in the face of attrition.

One engineer going deep, then bringing three colleagues into a working session where they build on it, that creates organizational learning. The knowledge becomes embedded in multiple people. The team's collective capability improves. The competitive advantage accrues to the organization, not just the individual.

This requires deliberate structure. It doesn’t happen spontaneously. The teams getting this right have built regular knowledge-sharing rituals into their operating rhythm: short internal demos, AI tool retrospectives, rotation of research responsibilities. The specific format matters less than the consistency and the expectation that individual discovery is the beginning of a learning cycle.

3. Create the direction explicitly , and then repeat it.

Your team is almost certainly already feeling the pressure of AI transformation. They are reading the same headlines you are, seeing the same discussions on LinkedIn and in

developer communities, fielding the same questions from friends outside the industry. That pressure is present whether you address it or not.

What most CTOs have not done is give that pressure a direction. The question is not just "Are we using AI?" It is: “What kind of engineering organization are we building, and what does excellence look like for us in 18 months?”. If you have not answered that question clearly, and communicated the answer in terms your team can actually navigate by, they are feeling pressure without a compass. That combination produces anxiety and defensive behavior, not momentum.

This does not require a polished strategy document or a roadmap with false precision. It requires honest, direct communication about where you are taking the team and why. What you are moving toward. What good looks like. What role each person plays in getting there. And then it requires repeating it, because people absorb direction over time, not in a single announcement.

The anxiety question.

There is a distinction that matters more than most CTOs realize: the anxiety visible in engineering teams right now is not primarily about the change itself. People are capable of adapting to significant change. What they adapt to poorly is sustained pressure without understanding the direction.

The engineers on your team who seem most resistant to AI transformation are, in most cases, not resistant to AI. They are experiencing a specific form of uncertainty that shows up as resistance: they do not understand what the change means for their role, their career, and their value. They are making a rational calculation.

The intervention is not more tools, more training mandates, or more urgency. **It is a direct, honest conversation about where the organization is going, what their role in that future looks like, and what they personally gain from engaging with the transition rather than waiting it out.**

Gergely Orosz noted in his February 2026 analysis of AI's impact on the engineering profession that the engineers most successfully navigating this shift are not the ones who feel no anxiety, they are the ones whose leaders gave them enough clarity to channel that energy productively. The difference between anxiety as paralysis and anxiety as fuel is largely a function of whether the person has enough context to act. That context is yours to provide.

What the 90-day question reveals

One of the most useful diagnostics we have found for an engineering organization's readiness for this transformation is a simple question asked of the whole team: **What did we do this week to learn faster than our competitors?**

The first time you ask this question in most organizations, the answers are thin. People describe individual tool experiments or link to articles they read. The organizational learning infrastructure, the mechanisms by which individual discovery becomes collective capability, is either absent or invisible.

Asking the question consistently, over time, and treating the answers as seriously as you treat delivery metrics begins to shift the culture.

It signals to your team that learning velocity is a performance dimension you are measuring and investing in. That signal, repeated, changes what people prioritize.

Chapter 6

What It Looks Like To Have A Sidekick

The wrong model, and why it persists.

There is a version of technical partnership that most engineering leaders have experienced at some point: a consulting firm that arrives with a framework, runs a series of workshops, delivers a document, and leaves. The knowledge transfers to a deck. The implementation is yours to figure out. The relationship ends when the SOW expires. That model persists because it is easy to procure, easy to budget, and easy to measure. A defined project with a defined deliverable produces clear contracts and clean expense reports.

What it does not produce, in most cases, is capability in your organization. The framework exists in the document. The judgment that informs it walked out the door.

The kind of transformation this guide describes, building organizational learning velocity, developing the Forward Deployed Engineer profile, creating the cultural conditions for AI to amplify your team's strengths rather than its weaknesses, does not fit into a project with a defined end date. It is ongoing, iterative, and deeply dependent on understanding your specific people, your specific clients, and the organizational patterns that make your team what it is. **That requires a different kind of partnership.**

What the partnership actually looks like.

We are not a consulting firm, we are engineers who embed inside your team. In practice: we work inside your existing workflows, alongside your existing people, on your actual problems. We do not arrive with a predetermined solution, we arrive with deep experience across the contexts most relevant to where software development is going, and we apply that experience to your specific situation.

After 13 years of operating this way, across more than 100 engagements, mostly at team sizes of 5 to 60 engineers, we have developed a clear view of where the leverage points are, what the common failure modes look like, and what the conditions for durable success require. We have seen the patterns described in this guide play out many times, in many variations. That pattern recognition is not available in any framework or research report. It comes from being present when things get hard.

We are an AI-first agency. That means AI is not a service line we added, it is the lens through which we approach every engagement. We build AI-native products for organizations that want to move fast without accruing the kind of technical and organizational debt that slows teams down later. And we help engineering organizations transform how they work, so that AI becomes a genuine capability embedded in their culture, not a tool a few individuals are experimenting with in isolation. Our technical depth spans Cloud Data Engineering and Applied AI/ML, and we work with AWS and Databricks as our core platform partners.

Engagements that create lasting value tend to share a structure that unfolds in roughly three phases.

In the early phase, the most important work is diagnostic rather than delivery-oriented. Understanding where your team's learning and collaboration patterns actually are , not where they are assumed to be, and identifying where the gaps between current state and where you need to be are largest. This is also where we establish the working relationship: learning your codebase, your clients, your organizational rhythms.

In the core phase, we are operating as integrated members of your team. Building things alongside your engineers, not for them. The distinction matters enormously for knowledge transfer. When we solve a problem alongside your engineer, they understand it. When we solve it for them, we have the understanding and they have the output.

In the later phase, the goal shifts explicitly toward making ourselves less necessary. The patterns we introduced become your team's patterns. The capabilities we brought in become capabilities your engineers own. The organizational learning velocity is higher than it was when we arrived , not because we are still there, but because the conditions for it are now embedded.

That is what we mean by a sidekick. Not a dependency. A force multiplier. Not someone who does the work while you watch, but someone who makes your team significantly better at doing the work.

The sidekick framing, and why it matters.

The word we use, sidekick, is deliberate, and it is worth being direct about what it means.

In the traditional client-vendor relationship, the vendor is implicitly positioned as the expert and the client as the recipient of expertise. That structure, however politely managed, creates a dynamic where the client is subtly cast as the one with the problem and the vendor as the one with the solution. It is not a partnership of equals.

We reject that framing entirely.

Our clients are the heroes of their own story. They have the domain knowledge, the client relationships, the industry context, and the vision that makes the work meaningful. We bring technical depth, cross-industry patterns, and the specific expertise that the current moment demands. The value we create is in the combination, in embedding our capability inside their context, and building something neither of us could build alone.

That framing is not marketing. It is a structural commitment that shapes how we engage, what we measure, and how we know when we have succeeded.

Closing

The Question For Monday

We said at the outset that we would not leave you with a ten-step framework. We meant it.

The teams figuring this out are building their own playbooks, iteratively, with their specific people, clients, and constraints. There is no shortcut to that process. The specificity is the point, what works for a 12-person SaaS engineering team in a vertical market is not what works for a 55-person platform team at a scale-up. Anyone packaging this into a universal prescription is oversimplifying in ways that will cost you.

What we have tried to offer instead is a honest read on the terrain: where this moment comes from, what is actually happening in real teams, where the leverage points are, and what it requires from you as the person responsible for leading the transformation. The closing provocation is a simple one.

Am I leading this transformation, or am I waiting for it?

Waiting has costs that are not always visible in the short term. The team that builds organizational learning velocity in 2026 will not feel its competitive advantage until 2027 or 2028, when the market has moved again and their capacity to absorb the change is faster than their competitors. The compounding takes time to show up. The investment has to precede the return.

And one concrete question to bring to your team this week, not as a rhetorical device, but as a genuine weekly operating question: **What did we do this week to learn faster than our competitors?**

Because in a market that changes every 90 days, the durable advantage is not the best current stack. It is not the best current hire. It is the fastest-learning organization, the one that turns market change into internal capability continuously, compounds that capability over time, and arrives at each new inflection point better prepared than the competition.

That organization is buildable. It does not require firing anyone, reorganizing your structure, or adopting a specific set of tools. It requires you deciding that building it is your job. And then doing it.

Who's in Your Corner?

Every hero needs a sidekick. Not because they can't do the work alone, but because the best outcomes happen when someone who genuinely understands your world is beside you, not observing from the outside. That is the role NaNLABS was built to play.

We are an AI-first engineering partner for organizations serious about building something that lasts. We build AI-native products,helping companies move from bold ideas to working systems without accumulating the technical and organizational debt that compounds into drag. And we help engineering teams transform from the inside, developing the learning velocity, ownership culture, and AI fluency that make the advantage durable rather than momentary.

Our technical depth spans Cloud Data Engineering and Applied AI/ML, with AWS and Databricks as our core platform partnerships. **We work primarily with teams and industries where the cost of getting this wrong is high and the opportunity for those who get it right is significant.**

But the what matters less than the how. We don't arrive with a framework and leave before the hard part starts. We embed inside your team, learning your codebase, your clients, your organizational rhythms, and we build alongside your engineers, not for them. The knowledge stays. The capability compounds. When the engagement ends, your team is stronger than it was.

That is what a sidekick actually does. Not take over the mission. Strengthen the hero running it.

If the questions in this guide are ones you are sitting with, about learning velocity, about the profiles on your team, about what kind of engineering organization you are building, we would be glad to think through them with you.

Every hero needs a **sidekick.**
Let’s power up together.

 ask@nan-labs.com

 nan-labs.com

 NaNLABS

 github.com/nanlabs

NaNLABS
YOUR TECH SIDEKICK