WHITEPAPER

Witted

# AI ASSISTED SOFTWARE DEVELOPMENT: FROM ARTISAN CRAFT TO INDUSTRIAL SCALE DELIVERY

An IT leader's guide to scaling software delivery in the AI era

# CONTENTS

# 1. INTRODUCTION

If you are an IT decisionmaker in a Nordic company, it is highly likely that you have made this move over the past two years: hand developers an AI coding assistant and watch individual throughput climb. The productivity numbers hold up. Individual engineers report writing code two, three, sometimes four times faster than before, and the demos have been convincing enough to get executive level paying attention.

But something breaks down at the organizational level. Individual throughput gains do not translate into faster delivery for the organization. Deployment frequency increases only slightly. Incident rates drift upward. The codebase, written faster than ever, becomes harder to reason about: a proliferation of slightly different patterns, bespoke infrastructure choices, and AI-generated code that no one quite owns.

The trap is not the tool. It is applying an industrial accelerant to a craft production system and expecting it to behave like a factory.

> AI doesn't just accelerate delivery. It accelerates whatever production system it runs on — including the accumulation of technical debt.

The right response to AI-assisted development is not better tooling. It is a better production system: one designed from the start for AI-scale output, where quality, security, and architectural consistency are built into the system rather than depending on each individual engineer to maintain them.

We call it the Software Factory. It is one of the most consequential investments your organization can make in the coming years. The underlying ideas are not new. Your organization is likely already familiar with platform engineering and internal developer platforms. What the factory model changes is that AI acceleration makes the absence of this structure immediately and visibly costly: not a slight continuous drag on delivery, but an active multiplier of whichever flaws already exist. Set the guardrails, build the factory, and watch productivity soar.

## WHY TEAMS GET STUCK

To understand what the factory needs to be, it helps to be precise about why the current model fails under AI acceleration.

Enterprise software operates across three layers with very different rates of change. At the base sit Systems of Records: ERP platforms, identity providers, financial ledgers, data warehouses. These are stable, compliance-heavy, and costly to touch. Above them sit the applications that deliver business value: customer-facing products, internal tools, analytics dashboards, workflow automation. These change constantly, driven by competitive pressure and the business needs of the week.

Between these two layers sits a gap that every team fills differently. Identity wiring, observability setup, data access patterns, security configuration, deployment pipelines. In the artisan model, each team tackles this from scratch, or copies it imperfectly from whatever the previous team did. This work produces no user value, rarely gets tracked, and consumes somewhere between a quarter and half of every team's capacity.

Add AI tools to this picture and the gap does not close. It widens. AI can scaffold a feature in minutes, but it scaffolds into the same disorganised foundation. The infrastructure choices it makes are only as consistent as the context it has been given. Without a shared set of patterns to conform to, AI-generated code inherits the entropy of the environment it operates in.

The inverse is equally true. Where an organization has already invested in architectural direction — clear patterns, defined service boundaries, a shared understanding of what production-ready looks like — AI tools accelerate convergence toward that standard rather than away from it. The factory is designed to create that condition; but in organizations where the groundwork has already been laid, AI's impact is immediate and disproportionately large.

> The real cost and the reason for slow delivery of the artisan model is the invisible tax on every team: the hours spent wiring infrastructure that already exists somewhere else in the organization, the incidents caused by bespoke security configurations, and the cognitive load of working in a codebase where every module was built by a slightly different version of the team.

The factory model addresses this by making the middle layer explicit. Rather than each team improvising the bridge between Systems of Records and Business Applications, the organization invests in a shared platform that standardizes it once, and then lets every team cross at speed.

That structural logic underpins everything that follows. The factory is not a metaphor for just moving fast. It is a specific architectural commitment: invest in the production system so that delivering value becomes repeatable, safe, and scalable. The factory model makes you move fast with high technical quality.

# 2. THE FACTORY

The ideal industrial scale delivery model

The Software Factory has three components, and they work as a system. No single one is sufficient on its own.

## THE PAVED ROADS — THE SOFTWARE AUTOBAHN

The defining question of the factory model is: what should a team never have to think about twice? The answer defines the paved roads: opinionated, pre-validated paths through the complexity of enterprise software delivery.

A paved road is a set of architectural contracts: the API schemas new services must implement, the event structures the data platform expects, the security patterns the compliance team requires, the observability defaults that make a new service production-ready from day one. These contracts allow AI coding tools to produce architecturally coherent output, because the context they are given is coherent. A well-specified platform environment is the single most effective lever for improving AI-generated code quality at scale.

Teams on the paved road move fast because the system removes the decisions that are not relevant for differentiating from competition. Authentication configuration, logging format, deployment pipeline design — these have been made, validated, and encoded. What remains is the work that actually matters: understanding the domain, building the right thing, serving the customer.

In practice, context quality matters as much as tool quality. A repository with consistent directory structure, predictable naming conventions, and working examples of the expected patterns already in place gives an AI tool the information it needs to generate coherent new code. If every module reflects the preferences of a different team at a different moment, an AI tool will get the information it needs to generate consistent inconsistency. The platform does not just set standards for what ships to production. It shapes the context that determines what gets generated in the first place.

## EMBEDDED GUARDRAILS

Speed without safety is not our goal. The factory model deals with this by moving quality and compliance controls from the end of the process, where they slow delivery, to the start, where they are invisible.

Embedded guardrails in Software Factory mean security scanning runs on every commit, not at the end of a sprint. A new service cannot reach production without meeting observability standards, because the pipeline checks automatically. Data classification is enforced at the infrastructure layer, so a developer cannot accidentally store sensitive data in the wrong place — regardless of what their AI tool suggests.

The shift this enables is from a governance culture of "no, unless approved" to one of "yes, because the system keeps you on track". Developers gain genuine autonomy because the platform has already removed the ways they could go seriously wrong. The guardrails are what makes sustained speed possible — not restrictive.

## AUTOMATED QUALITY CONTROL

AI-generated code is fast, but it is also probabilistic. A factory that runs at AI speed but ships unreliable output is not a factory. It is a debt engine.

The volume AI introduces is qualitatively different from what human-paced development produces. An AI-assisted team generates far faster than it can carefully examine. The output tends to be syntactically valid, passing surface checks, conforming to obvious conventions, but it can introduce subtle inconsistencies that accumulate across a codebase over time. The factory's quality controls are designed specifically for this dynamic, to catch at the system level what the pace of generation would make impossible to catch at the individual review level.

The third component is automated verification: contract tests that confirm AI-generated services conform to platform specifications, test-generation tooling that produces test suites alongside application code, and readiness gates that check production standards before a service is ever promoted. Quality is not a downstream activity. It is built into the production process from the first commit.

## ARTISAN MODEL VS. SOFTWARE FACTORY

| | Artisan Model | Software Factory |
|---|---|---|
| **Infrastructure setup** | Each team wires identity, observability, and deployment from scratch. Repeated across every project. | Platform encodes these as defaults. Teams start production-ready on day one. |
| **AI coding tools** | LLM assistants generate code into an inconsistent environment. Quality depends on the individual team's conventions. | LLM tooling generates code against a well-specified platform contract. Consistency is a property of the system. |
| **Security & compliance** | Reviewed at the end of delivery cycles. Creates bottlenecks and late-stage rework. | Embedded in the pipeline. Automated checks run continuously; human review reserved for high-stakes changes. |
| **Scaling delivery** | More output requires more engineers. Technical debt grows proportionally. | Platform investment scales across teams. Output grows; marginal cost per feature falls. |

# 3. RAMPING UP

How to scale development to industrial level

The factory is not built in a quarter. It grows through a sequence of deliberate investments, each extending the platform's reach and proving its value to the next set of teams. The cardinal rule: avoid big-bang transformation. Evidence comes first, then scale.

The transition to the industrial model takes way more than changing the tooling. It is a change management project of transforming how value is delivered. We recommend a three-phased approach: 1) The Lighthouse Project, 2) Standardizing the Assembly Line, and 3) Scale and Specialization.

This should not be a big bang reorganization, but a progressive expansion of the factory footprint based on measurable friction reduction.

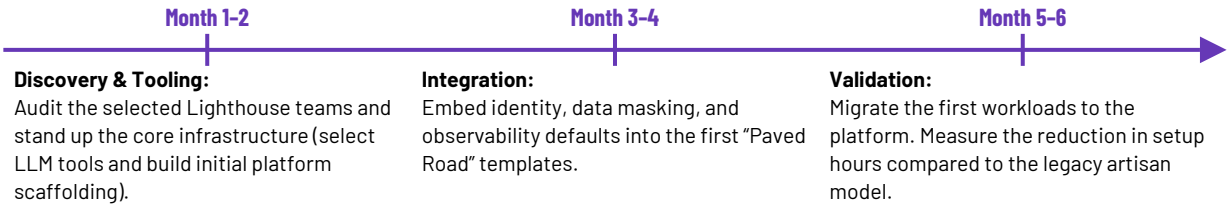
## PHASE 1: THE LIGHTHOUSE PROJECT (THE FIRST 6–9 MONTHS)

Identify 1–2 value streams where manual coordination and technical friction currently delay delivery. The goal of Phase 1 is not a complete platform. It is undeniable evidence that the factory model works and evidence is strong enough to justify the Phase 2 investment.

This means picking one value stream where the pain is visible and the team is motivated. Deploy a minimal platform slice: one golden-path template, one CI/CD pipeline with embedded security scanning, observability defaults and measure what changes. Good metrics are lead time before and after, developer time spent on infrastructure setup or incident rate attributable to bespoke configuration choices.

Identity and access management, data governance, and AI developer experience are all worth building. Each is also a substantial project on its own terms. Identity integration alone, centralised SSO, secrets management, RBAC alignment with an existing enterprise IdP, typically takes three to six months once compliance review and migration of existing service accounts are included. Attempting multiple major capabilities simultaneously in a six-month window will lead to delivering none of them credibly.

> What does Phase 1 success look like: the Lighthouse team's lead time has fallen measurably. At least 80% of their new work starts from a platform template rather than bespoke scaffolding. Developers on the team can articulate, unprompted, why working inside the platform is better than working outside it.



### KEY CAPABILITIES TO BUILD

| Capability | Strategic "Why" |
|---|---|
| **Chat-based DevEx** | Lowers the barrier for developers to scaffold, troubleshoot, and follow compliant practices via natural language. |
| **Design System** | Ensures consistent UX and accessibility across all products by providing reusable building blocks. |
| **Identity & Access** | Centralizes SSO, secrets patterns, and role-based access to remove bespoke security wiring. |
| **Data Governance** | Implements controlled access, masking, and lineage to ensure data safety by default. |
| **Observability Defaults** | Automates logs, metrics, and traces so operational readiness is not an "add-on" task. |

## PHASE 2: STANDARDIZING THE ASSEMBLY LINE (6–9 MONTHS)

Phase 2 is about codifying what the Lighthouse taught and extending the platform's reach. Paved roads that emerged from Phase 1 are formalized into reusable templates. The risk-tiered governance model is wired into the pipeline. Three to five additional teams are brought on.

Two essential capabilities deferred from Phase 1 should arrive now: identity and access management that reaches full enterprise integration, and data governance — meaning controlled access, data classification enforcement, lineage tracking — is embedded into the platform's data access patterns.

Phase 2 is also where organizational resistance tends to peak. Some senior engineers will find the platform constraining. Some business units will argue their domain is too distinctive for standardized tooling. Some middle managers will experience the boundary between Platform Builders and Solution Engineers as a threat to their team structures. These are not engineering problems: they are change management problems, and they require as much deliberate attention as the technical roadmap.

The most effective response is not to overrule the resistance but to dissolve it by making the platform genuinely good to work with, creating a structured process for teams to propose extensions to the golden paths, and making the productivity gap between platform-native and non-platform delivery visible and impossible to argue with.

## PHASE 3: SCALE AND SPECIALIZATION (12 MONTHS)

In the final phase, the platform becomes the default environment. Not only because it's mandated, but also because it is clearly the best place to build. The organizational bottleneck has shifted from technical implementation to domain expertise. This is the outcome the factory model was always designed to produce.

By this stage, the platform has evolved from a project into a product with its own roadmap, its own users, and its own feedback loops. Teams contribute through an inner-source model, submitting improvements and extensions via a governed pull-request process rather than depending on a central team for all changes. Architectural governance becomes federated: clear decision rights about what belongs in the platform versus the application, with a review process for edge cases.

The strategic outcome is what the investment was always designed to produce: organizational output that scales without a proportional increase in headcount. The factory absorbs more teams, more complexity, and more AI-generated volume without requiring proportionally more engineers to hold the foundation together.

# 4. THE NEW ORGANIZATION

Talent, roles, and the investment case


The factory model requires a different organization, but not necessarily a larger one. The change is in how talent is focused.

In the artisan model, the most experienced engineers spend substantial time on infrastructure: setting up environments, configuring security, building deployment pipelines. That is expensive expertise applied to work that does not differentiate the organization. The factory model redirects it.

## THE PLATFORM BUILDERS

Platform Builders are the engineers who build and maintain the factory floor. Their output is not features: it is the environment in which features can be built safely, quickly, and consistently. They think in systems. How does a new team get from zero to production-ready in a day? What should a developer never need to think about? Where are the friction points that push teams to build their own tooling?

Their measure of success is the inverse of what most engineering teams track. They succeed when their work becomes invisible. When developers using the platform never have to think about what lies underneath. A Platform Builder who constantly fields questions about infrastructure configuration has not failed personally; the platform has failed to absorb the complexity it should.

This profile is scarce. There are DevOps teams, platform chapters, SRE functions in most organizations, but few whose explicit mandate is developer experience as a product, with the same discipline around user research, feedback loops, and iterative improvement that product teams apply to customer-facing software.

## THE SOLUTION ENGINEERS: DOMAIN ASSEMBLERS

Solution Engineers drive on the paved road. They are domain experts who use AI and platform capabilities to deliver business outcomes at a pace the artisan model could not match.

The shift in their role is not a reduction in skill, it is a reorientation of where skill gets applied. Less time on infrastructure wiring, more time on understanding the business domain well enough to ask the right questions from an AI tool, validate its output against real requirements, and compose platform capabilities into something that genuinely serves the customer and the organization.

The hardest transition for experienced engineers is cultural. There is a deep professional identity bound up in the craft of writing code: the mastery of syntax, the satisfaction of a well-constructed algorithm. The factory model asks engineers to measure themselves differently: by the outcomes they enable, not the lines they write. For many, this is genuinely hard. Change management that ignores this reality is change management that will fail.

## THE NEEDED PROFILES

| Role | Phase | Type | Description |
|---|---|---|---|
| **AI Coach** | Phase 1–2 | Platform Builder | Performs discovery audits on 'Lighthouse' teams and drives adoption within delivery teams. Owns the change management workstream, coaching engineers through the shift to AI-assisted development and platform-native ways of working. Reports to the Program Director. |
| **AI Platform Engineer** | Phase 1 | Platform Builder | Generalists capable of setting up initial platform scaffolding, including Identity Management, Data Governance and pipelines, and Observability defaults. |
| **AI Platform Engineer** | Phase 2–3 | Platform Builder | Codifies learnings from Phase 1 into reusable 'Paved Roads' and standardized templates for web services, APIs, and data pipelines. Automates operational readiness checks to verify logging and tracing standards. |
| **AI Development Process Engineer** | Phase 2–3 | Platform Builder | Builds chat-based Developer Experience (DevEx) interfaces to lower the barrier for developers to scaffold and troubleshoot via natural language. Implements private-by-default AI using controlled tenants and self-hosted LLMs. |
| **AI Security Architect** | Phase 2–3 | Platform Builder | Embeds governance and risk-tiered approval workflows directly into the delivery pipeline. Implements automated quality and security checks for low-risk changes while maintaining manual oversight for high-risk architectural shifts. |
| **AIOps Engineer** | Phase 2–3 | Platform Builder | RevOps but for AI and LLM models. Optimizes costs and outputs by building workflows to decide on most cost-efficient model usage, and builds caches and infrastructure to minimize token usage. |
| **Domain Specialist** | Phase 2–3 | Solutions / Business | Focuses exclusively on business logic and customer outcomes by using platform primitives to assemble applications. Uses Generative AI to handle complex implementations by translating intent into assembly. |
| **Solutions Architect** | Phase 2–3 | Solutions / Business | Applies domain-centric expertise to understand business risk and compose platform capabilities into outcomes. Extends platform capabilities through a hub-and-spoke model to ensure autonomous delivery. |

# THE INVESTMENT CASE

Building a factory costs money. A minimum viable platform team of an AI Platform Engineer, an AI Development Process Engineer, a Security Engineer, and transformation support is a real commitment at Nordic market rates. The question is not whether this investment is large. It is whether the alternative is larger.

The alternative is paying the artisan tax indefinitely: infrastructure setup repeated across every project, incidents caused by bespoke configurations, onboarding time lost as new developers navigate a different stack in each team, compliance reviews piling up at the end of delivery cycles. That tax is already being paid. The factory model makes it visible and finally eliminates it.

## WHERE THE ARTISAN TAX ACCUMULATES

Three cost categories dominate when organizations measure the artisan baseline honestly:

The first is **infrastructure setup time**. In a team without a platform, getting a new service to production-ready state — identity wiring, observability, CI/CD, security configuration — takes one to three weeks of senior engineer time per service. At €120K fully-loaded annual cost for a senior engineer in the Nordics, three weeks equals roughly €7,000 per service. An organization shipping twenty new services a year spends approximately €140,000 annually on work the platform could eliminate entirely.

The second is **developer onboarding friction**. A new developer joining a team without a platform typically needs four to eight weeks before reaching confident, independent delivery. Most of that time is not spent learning the domain. It is spent learning the team's unique infrastructure conventions, which differ from the conventions of the team they just left. At €90K average annual cost for a mid-level developer, an eight-week ramp represents €14,000 in reduced-productivity salary per hire, before counting the senior time spent mentoring which could take, for example, 5 hours a week for a four-week period. An organization hiring fifteen developers a year carries roughly €210,000 in onboarding overhead that a standardized platform cuts to two or three weeks.

The third, and hardest to forecast, is **incident cost from bespoke configuration**. For a fifty-developer organization, a conservative estimate is two to three incidents per year attributable to configuration inconsistency, each requiring two to five days of senior engineering time to diagnose and remediate. At Nordic rates, that is €30,000–€80,000 in remediation cost annually, before any commercial, regulatory, or reputational consequences.

| ILLUSTRATIVE BASELINE: 50-DEVELOPER ORGANIZATION | |
|---|---|
| Infrastructure setup (20 services/year @ €7K each) | €140,000 |
| Onboarding friction (15 hires/year @ €14K each) | €210,000 |
| Onboarding: mentoring from senior (15 hires/year @ €1,200 each) | €18,000 |
| Incident remediation (2–3 incidents/year) | €50,000 |
| **Total annual artisan tax** | **~€418,000** |

## WHAT THE FACTORY COSTS

A minimum viable platform team for a fifty-developer organization consists of two AI Platform Engineers and a part-time security and AI engineering contribution, either employed or brought in through staff augmentation. At Nordic market rates, this totals approximately €280,000–€340,000 annually in personnel costs. Tooling and infrastructure licenses — CI/CD platform, observability stack, enterprise LLM tenant — could add €30,000–€60,000 per year depending on vendor choices.

Total platform investment at this scale: roughly €310,000–€400,000 per year.

| Year | Platform Coverage | Artisan tax eliminated | Platform cost | Net this year | Cumulative position | Productivity increase |
|---|---|---|---|---|---|---|
| Year 1 (build phase) | **25%** | €104,500 | €350,000 | −€245,500 | −€245,500 | **+5%** |
| Year 2 (scale phase) | **80%** | €334,400 | €300,000 | €34,400 → net positive | −€211,100 | **+10%** |
| Year 3 (full adoption) | **95%** | €397,100 | €300,000 | €97,100 | −€114,000 | **+20%** |
| Year 4+ | **95%** | €397,100 | €300,000 | €97,100 | −€16,900 → break-even ~after Year 4 | **+30%** |

The table above models a steady-state scenario where productivity gains are reinvested as increased delivery capacity. The base idea is that the factory outputs more without growing headcount. Organizations can, however, make a different choice: capture the same gains as cost reduction by not backfilling attrition or by consolidating redundant roles. Under that model, the platform investment can break even during Year 2 rather than Year 4, as the cost base shrinks faster than the investment accumulates. Neither choice is universally correct. The right answer depends on whether the constraint is

budget or delivery velocity.

These figures are illustrative, built on conservative assumptions and Nordic market rates. The numbers should be treated as indicative order-of-magnitude estimates rather than exact benchmarks. The actual break-even depends on how aggressively Phase 2 onboards teams and whether the onboarding and incident figures above are higher or lower in the specific organization. The right starting point is a discovery audit: a structured two-to-four-week assessment of where time is actually going, what the incident record shows, and what developers say when asked how much of their week goes on work that creates no direct customer value. Those numbers, not industry benchmarks, are what make the case undeniable in any executive team meeting.

## INVESTMENT PROFILE BY ORGANIZATION SIZE

| Organization size | Investment profile |
|---|---|
| **Under 20 developers** | A dedicated platform team is premature. Shared golden-path templates and a standard CI/CD toolchain deliver most of the benefit without the overhead. Revisit at 25–30 developers. |
| **20–50 developers** | The break-even zone. A two-person platform team focused on CI/CD, security defaults, and one golden-path template typically breaks even within 12–18 months. The artisan tax is real but the platform cost must be kept lean. |
| **50–200 developers** | Clear positive return from Year x, strong return from Year x+1. The fixed platform cost serves an expanding estate; each additional team onboarded increases the return on the same base investment. |
| **Over 200 developers** | The calculus shifts from ROI to competitive risk. At this scale, the cost of not having a platform is measured not just in wasted engineering time but in the speed and reliability disadvantage relative to the competitors who have built one. |

A credible investment case starts with a discovery audit: a structured assessment of how much time teams spend on infrastructure work versus value-creating work, where the incident record points to configuration failures, and what it costs to bring a new developer to their first confident deployment. These numbers make the cost of inaction concrete, and are almost always more persuasive than a projection of future savings.

# 5. GOVERNANCE

Security and risk that enables rather than obstructs

The most common objection to moving fast is risk. In regulated industries such as financial services, healthcare, public sector, compliance teams exist for good reasons, and “move fast” has caused real damage in the past. The factory model does not dismiss this concern. It addresses it directly.

The key insight is that governance does not need to be proportional to activity: it should be proportional to risk. A color change on an internal dashboard and a modification to a payment processing flow are both software changes. Treating them identically is not rigor; it is friction. The factory model separates them deliberately.

## RISK-TIERED GOVERNANCE

| Risk tier | Characteristics | Governance approach |
|---|---|---|
| **Tier 1: Low** | Internal tools, UI changes, low-sensitivity data, documentation. | Fully automated: pipeline checks run, guardrails pass, code ships. No manual sign-off required. |
| **Tier 2: Medium** | Customer-facing features, third-party integrations, moderate data exposure. | Hybrid: automated checks supplemented by targeted peer review of the AI-generated logic. Lightweight sign-off. |
| **Tier 3: High** | Financial transactions, PII handling, core infrastructure changes, regulated workflows. | Strict: deep manual review, security penetration testing, multi-stakeholder approval with full audit trail. |

This tiering changes compliance from a gate at the end of the process to a property of the work itself. Tier 1 work never enters a compliance queue. Tier 3 work gets the full weight of human expertise applied exactly where it belongs. The governance function becomes more effective, not less, because it stops burning attention on low-risk changes.

### AI DATA SECURITY

There is a second dimension to governance in the AI era: data security for AI systems themselves. The answer is not a binary choice between cloud APIs and fully self-hosted models. It is a tiered deployment architecture that matches data sensitivity to the appropriate infrastructure: public API endpoints with data-loss-prevention scanning for non-sensitive workloads; enterprise cloud tenants with data residency configuration for standard business logic; self-hosted models for regulated or highly sensitive data. In Nordic markets, this maps directly onto GDPR data residency requirements and the practical constraints.

The factory does not leave these decisions to individual teams. It encodes them into the platform, so routing happens automatically based on the workload's data classification.

# 6. MEASURING THE FACTORY

Metrics for the industrial age

The factory model also changes what should be measured, not only how fast things move. An organization that tracks only deployment frequency and feature throughput will miss the health signals that predict whether the factory is sustainable and will be surprised when adoption stalls, developer satisfaction drops, and technical debt reasserts itself through AI-generated output that nobody owns.

A complete measurement framework covers four levels.

## PLATFORM HEALTH

Is the factory floor in good condition? These are the leading indicators for everything downstream: golden-path availability and uptime, the coverage of application types with supported templates, time to onboard a new team, and the volume of support requests the platform team receives. A proxy for where the platform is still generating friction rather than removing it.

## DEVELOPER EXPERIENCE

Do developers actually want to work inside the platform? This is the leading indicator of adoption. A platform that technically works but is painful to use will be quietly circumvented. Teams build local tooling, find workarounds, and recreate the artisan model inside the factory's perimeter. Quarterly developer experience surveys, time-to-first-deployment for new team members, and escalation rate (i.e. how often developers need to contact the platform team just to make progress) are all meaningful signals.

## DELIVERY PERFORMANCE

Is the factory producing faster and more reliably? Lead time from idea to production, cycle time broken out by risk tier, template adoption rate, and change failure rate are the standard engineering effectiveness metrics and they remain important. In the factory model, though, they are lagging indicators: outcomes of a healthy platform and good developer experience, not the primary management focus.

## BUSINESS OUTCOMES

Is the organization decoupling its growth from its headcount? That is the question the factory model is designed to answer at board level. Revenue per developer, time-to-market for new products, and customer-facing incident rate translate the engineering investment into business terms. Tracking cost per feature over time, and specifically how that cost moves as AI generation scales up, tells the story of whether the factory's efficiency is compounding or eroding.

> The factory's ultimate measure: are we doing more with the same people — or the same with fewer?

# 7. THE CHOICE

Summary and call to action

The AI tools are not going away. The question every engineering leader in the Nordics faces is not whether to adopt them. It is whether to let them run on the existing production system or to build the production system those tools actually deserve.

The artisan model was designed for a world where the limiting factor was how fast a skilled human could write code. That world does not exist anymore. The new limiting factor is the quality of the production system: how consistently it enforces architectural standards, how safely it handles AI-generated volume, how quickly it brings new teams to full speed without re-building infrastructure that already exists somewhere else.

The organizations that build this system now will compound their advantage over the next several years. Not because their developers are faster than competitors' — everyone will have similar tools — but because their factory floor is better: more consistent, more trustworthy, more productive per engineer than the improvised infrastructure of teams still building by hand.

Building the Software Factory is not a quick win. It is a long-term investment with a three-phase roadmap, real change management challenges, and genuine organizational resistance to work through.

The companies that will define the next era of software delivery are the ones who recognized, early enough, that the game had changed, and that running faster on the same track was not the answer.

## AI DISCLOSURE

The text you are reading has been written by humans, because we hate AI slop too. Artificial intelligence has been used for summarizing background material, visual formatting, as support in ideation, and proofreading. As in AI assisted software development, we welcome AI's assistance as long as humans stay in charge.

Witted

## ABOUT WITTED MEGACORP

All companies will either become software companies or perish. The sheer force of software is reshaping every sector, dismantling legacy paradigms and forging new realities. Witted is the partner for the companies riding this wave of digital revolution. We are the catalyst of change in the IT services industry, tearing down legacy and building new ways of working — here to reinvent the work life. Our clients are the architects of this future: companies across diverse sectors, united by a shared understanding that they are, at their core, software companies.

Personnel 430 · 53 M€ turnover 2025 · Network of 3,500+ senior IT professionals

**www.witted.com**