huenei
IT services

Building Software with AI Without Losing Control

What separates AI adoption from real AI integration



Opening

Why speed without process always comes at a cost.

Artificial Intelligence has fundamentally changed how software gets built. But the promise of greater speed only materializes when there's a process strong enough to sustain it. Without one, it translates into rework, lost traceability, and decisions made without the business context they need.

At Huenei, we've rebuilt our software development process around this reality, embedding AI as a core component of a delivery model that is predictable, measurable, and auditable at every stage. It's powered by a proprietary platform designed to support the full development lifecycle: prototyping, business specs, task breakdown, execution, testing, and reverse engineering. The platform turns project context into a structured asset that's available to AI throughout, with the whole team collaborating early to build and validate that knowledge.

This approach is grounded in Product-Driven Development (PDD) and Spec-Driven Development (SDD), where specs become the single, traceable source of truth, reducing ambiguity and cutting down on misinterpretation and rework. AI also reshapes the role of the team, through a Human-in-the-Loop model that concentrates human judgment where it matters most: problem framing, business rules, design decisions, and quality and security standards.

IN THIS PAPER

Because AI-generated software doesn't guarantee quality on its own, this paper walks through the measurement and control mechanisms built into the process: mechanisms that catch drift, optimize AI resource usage, and guide generation from the very start. The result is a more controlled and predictable AI-assisted development process: one built to increase build velocity while minimizing the human effort spent correcting misaligned outputs, without sacrificing quality, security, or business alignment.

Table of Contents

01 The problem that speed alone doesn't solve

02 What the data actually shows

03 Spec as the source of truth: PDD and SDD

04 Human-in-the-Loop: where the team steps in

05 The 7 conditions that determine whether AI helps or hurts

06 What kind of team are you today?

07 Where to start based on your archetype

08 How we measure whether it's working

09 What level is your organization at?

10 The five capabilities: how AI transforms each one

11 In practice

12 Sources

The problem that speed alone doesn't solve

AI accelerates. Without process, it accelerates chaos.

A year ago, the question was whether to adopt AI in software development. Today, 90% of tech professionals already use it. That debate is over.

What no one fully anticipated is that the same tool is producing radically different results depending on the team using it. Some teams are delivering faster and with better stability at the same time. Others are generating more code with the same underlying problems, just faster and at greater scale. The difference isn't the tool. It's the engineering system around it.

Research on development teams in 2025 backs this up with numbers: for every 25% increase in AI adoption without supporting governance, delivery stability drops 7.2%. The teams that gain both velocity and quality aren't the ones who picked the best tool. They're the ones who had process, solid specs, and a code review function that could absorb the new volume.

AI doesn't transform teams. It amplifies them. And the primary amplification factor isn't which model you're using. It's how much structured context that model has to work with.



THE PROBLEM THAT SPEED ALONE DOESN'T SOLVE



90%

of tech professionals already use AI in their daily work (14 points higher than the previous year)



–7.2%

drop in delivery stability for every 25% increase in AI adoption without process to support it

What the data actually shows

The gap between adoption and value isn't a hypothesis. It's been measured.

Analysis of more than 600 million lines of code confirms that AI-assisted code, without reinforced review and standards, tends to accumulate rather than get refactored. AI-generated code now accounts for roughly 50% of everything committed to active repositories. What hasn't improved over four years of measurement is the share of that code that passes security checks: 56%. The rest introduces known vulnerabilities. Not because the models are careless, but because no one asked them to be secure.

While pull requests grew 20% year-over-year, commit comments dropped 27% in the same period. Code review isn't keeping pace with code generation. The bottleneck isn't writing speed. It's the process's ability to absorb what AI is producing.



of everything committed to active repositories is AI-generated code



the share of that code that passes security checks



+20%

pull requests grew year-over-year



−27%

commit comments dropped in the same period

Code quality metric trends — before and after mass AI adoption

Code churn rate (code revised/discarded within 2 weeks)	≈3.3%	**5.7% → 7.1%**
Lines of code refactored	≈24–25%	**under 10%**
Copy/paste lines	8.3%	**12.3%**
Duplicate code blocks (per million lines)	40.3 (2023)	**73.0 (2026, +81%)**

"

AI writes code faster than most teams can review, integrate, and refactor it. That imbalance is exactly what structured context and process are here to fix.

Spec as the source of truth: PDD and SDD

AI works as well as the context it receives. The spec is that context.

Most of the problems that surface six sprints after adopting AI aren't problems with the tool. They're problems with what the tool was given to work with: ambiguous requirements, business rules that lived in informal conversations, and technical decisions no one documented. AI doesn't read minds. It operates on what's written down.

Product-Driven Development (PDD) and Spec-Driven Development (SDD) are the two methodological practices that address this at the root. They're not theoretical frameworks. Together, they provide a framework for turning project context into a structured, available, and traceable asset: for the team and for AI.

Product-Driven Development (PDD)

The product (its goals, its users, its business constraints) defines the technical work, not the other way around. AI operates on product-oriented specs: it knows what to build and why before proposing how.

Spec-Driven Development (SDD)

The spec is the single source of truth. Every requirement, design decision, and acceptance criterion exists as a structured asset before a line of code is written. AI works from that foundation.

How context flows through the development lifecycle

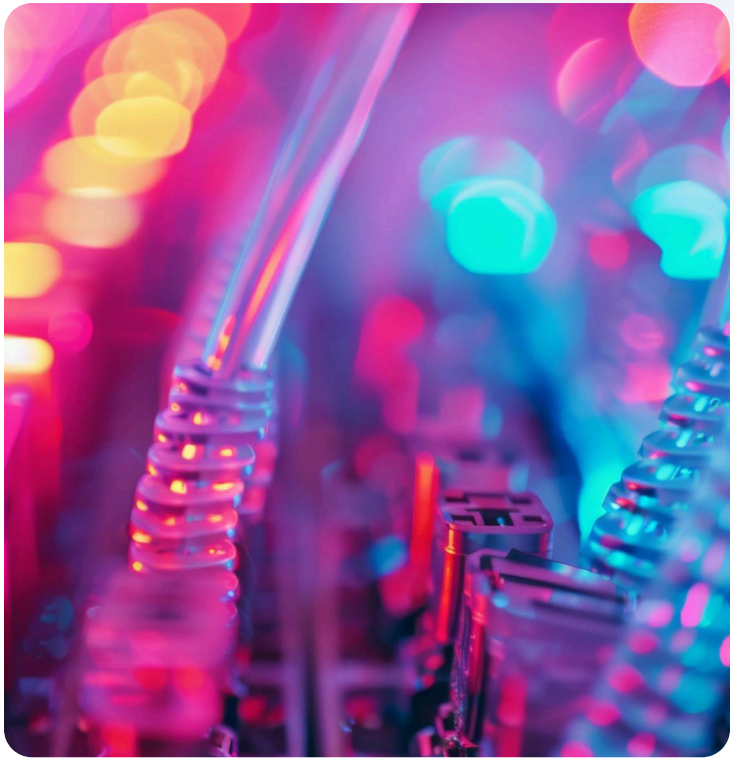


Structured context as a living asset — available to AI at every stage

Prototypes · Specs · Task breakdown · Testing · Reverse engineering

The result of applying PDD and SDD isn't just less ambiguity. It's that AI can participate at every stage of the cycle with the context it needs to make relevant suggestions, generate coherent specs, produce test cases that cover real business criteria, and document decisions with genuine traceability. The spec stops being a document someone writes once and forgets. It becomes the central asset everything else derives from.

> **Rework doesn't disappear because AI is faster. It disappears when the spec resolves ambiguity before the team starts building.**



Human-in-the-Loop: where the team steps in

AI doesn't replace human judgment. It concentrates it where it matters.

One of the most common risks when integrating AI into the SDLC is confusing "AI does it" with "the team doesn't need to." The right model is the opposite: AI absorbs the repetitive execution work so the team can focus its energy on the five points where human judgment is irreplaceable.

These aren't optional checkpoints. They're the gates that determine whether AI-assisted development produces predictable, measurable, and auditable outcomes. Without them, it's just volume without control.

01

Problem contextualization

The team defines the problem to be solved, its business constraints, and the success criteria. AI operates within that frame. It doesn't infer it or assume it.

02

Business rules

Business rules aren't in any training dataset. The team documents them, structures them, and makes them part of the context available to AI before any generation starts.

03

Design decisions

AI can generate architecture alternatives and model the impact of each decision. The choice, along with its consequences, belongs to the team. It's not delegated.

04

Quality standards

Acceptance criteria, coverage thresholds, and review gates are defined by the team. AI applies them; it doesn't set them.

05

Security guidelines

Security instructions in the prompt aren't optional. 44% of AI-generated code without them introduces a known vulnerability. The team defines what each output must meet before it reaches review.

AI generates. The team decides. That division isn't a limitation of the model. It's what makes the outcome defensible to the client, to the regulator, and to the team itself six months down the road.

The 7 conditions that determine whether AI helps or hurts

These aren't nice-to-haves. They're preconditions. Without them, AI amplifies what's already broken.

Research across nearly 5,000 tech professionals identified seven conditions that separate teams gaining both velocity and quality with AI from those gaining only velocity. Viewed through the lens of this paper, most of them are different expressions of the same thing: structuring and making available the context AI needs to work well.

CONDITION 01

A clear, communicated AI policy

Without an explicit policy on which AI tools can be used, for what tasks, and with what oversight, teams improvise. That improvisation shows up later as code without context, without standards, and without traceability.

CONDITION 02

Healthy data ecosystems

AI is only as good as the data it can access. Fragmented or outdated data produces equally fragmented suggestions, just faster.

CONDITION 03

Internal data accessible to AI

Context that lives in three people's heads isn't available to any AI assistant. Documenting and structuring that knowledge isn't just a prerequisite for integration. It's part of the integration itself.

CONDITION 04

Solid version control

As AI-generated change volume grows, version control stops being hygiene and becomes the only way to maintain traceability over what changed, why, and which agent proposed it.

CONDITION 05

Small batch work

A 2,000-line pull request doesn't get reviewed. It gets approved blindly. Small batches are what keep human review real, not ceremonial.

CONDITION 06

Genuine user focus

AI accelerates execution, not direction. If the team isn't clear on what the user needs, more velocity just means more unnecessary work delivered sooner.

CONDITION 07 · THE MOST CRITICAL

Internal quality platforms

These are the "paved roads" that let AI's benefits scale without getting lost in downstream bottlenecks. Without a solid platform, an individual productivity gain in code gets absorbed by a review, testing, or deploy process that wasn't designed to handle the volume.

What kind of team are you today?

Your starting point determines your path. The same tool produces radically different results depending on the engineering system around it.

Research identified seven team archetypes. The most relevant column through the lens of this paper isn't the general description.
It's how structured context (or its absence) defines what AI can and can't do in each case.



20%

Harmonious High-Achievers

High performance, stable systems. More structured context = higher velocity with lower risk.



20%

Pragmatic Performers

Consistent velocity. Risk: code review gets overwhelmed if context isn't filtering the volume of changes.



15%

Stable and Methodical

High quality at a deliberate pace. Upfront specs reduce the ambiguity that slows the team down.



17%

Constrained by Process

Inefficient processes create friction. AI on a broken process just accelerates it.



11%

Legacy Bottleneck

AI speeds up code writing, but the underlying system absorbs every gain. Context first, generation second.



10%

Foundational Challenges

The most fragile starting point. Without the foundational conditions, AI amplifies problems before it amplifies solutions.



7%

High Impact, Low Cadence

Strong results, high instability. Context governance is what stabilizes cadence.

"

Legacy Bottleneck teams have the most severe context problem: it's not just that specs are missing. The knowledge of the system that should be documented isn't there either. AI operating in that vacuum amplifies the wrong assumptions instead of correcting them.

Where to start based on your archetype

Your archetype isn't just a diagnosis. It determines the right order of operations.

Harmonious High-Achievers / Pragmatic Performers

WHERE TO START

Development & Deploy. The context already exists. The next step is structuring it into shared prompts and activating quality gates.

⚠ CRITICAL WATCH-OUT

Establish prompt standards, CI/CD quality gates, AI churn monitoring.

Stable and Methodical

WHERE TO START

Testing. Upfront specs make it possible to generate edge cases before the sprint, not at the end.

⚠ CRITICAL WATCH-OUT

Run a QA pilot with edge case generation. Expand to Discovery only once Testing is stable.

Constrained by Process

WHERE TO START

Discovery. AI-assisted requirements synthesis reduces friction between business and engineering.

⚠ CRITICAL WATCH-OUT

Fix the most inefficient processes first. AI on a broken process just accelerates the dysfunction.

Legacy Bottleneck

WHERE TO START

Design/Modernize. AI-assisted legacy code analysis delivers the highest ROI before touching delivery.

⚠ CRITICAL WATCH-OUT

DON'T start with code generation. First structure the context of the existing system.

Foundational Challenges

WHERE TO START

The 7 conditions come first. No SDLC stage delivers positive AI ROI without the foundational conditions.

⚠ CRITICAL WATCH-OUT

The priority is building the conditions, not integrating AI.

High Impact, Low Cadence

WHERE TO START

Governance & Operate. Instability is the primary risk.

⚠ CRITICAL WATCH-OUT

Don't expand AI adoption until change failure rate is under control.

Sequencing principle

Before scaling AI adoption to additional stages, validate that metrics from the previous stage are stable: change failure rate under control, AI code churn ratio within benchmark (1.5×), and PR review time not falling alongside comment volume. When all three signals are green, the next stage has a foundation to scale from.

How we measure whether it's working

Without full-system metrics, AI code generation speed is noise, not signal.

Measuring only lines generated or writing speed produces a false picture of progress. What matters is whether the full process from requirement to production got faster, more stable, and more predictable. Those are the metrics a CTO can bring to a business conversation.

THE FOUR KEY DELIVERY CYCLE METRICS

Lead time for changes

How long a change takes from commit to production. AI should compress it. If it doesn't, there's a bottleneck AI is feeding, not fixing.

Deployment frequency

A jump in frequency without a jump in stability is a warning sign, not a win.

Change failure rate

If it's rising while generation speed increases, context isn't working as a filter. It's failing as an anchor.

Time to restore (MTTR)

Predictive AI monitoring should bring this down. If it doesn't, the integration is still shallow.

AI-SPECIFIC SIGNALS TO WATCH

AI code churn ratio

AI-generated vs. human-written code. Keep it below 1.5×. Above that, context isn't filtering correctly.

Refactoring rate

Lines refactored vs. lines added. A sustained drop means AI is stacking technical debt, not solving problems.

PR review time

If it's falling alongside comment volume as AI output grows, human review has become a rubber stamp.

> **A team can use AI to generate 30% more code and watch that gain disappear entirely in a code review process that was already saturated before AI entered the picture. Full-system metrics catch that problem before it becomes a crisis.**

What level is your organization at?

The maturity model reframed: it's not about tools. It's about how available and structured your context is.

LEVEL 1

Loose tools

Everyone uses AI on their own terms. Context isn't structured or shared.

· No spec as single source of truth
· Individual, non-standardized prompts
· Highly variable output quality

LEVEL 2

Documented context

There's a stated AI policy and specs are starting to take shape, but they're not yet the team's source of truth.

· AI posture documented
· Partial specs in place
· No quality gates or churn metrics

LEVEL 3

Context as an asset

Specs are the source of truth. Context is structured and available to AI at every stage. Metrics and quality gates are active.

· PDD/SDD as team practice
· Structured, available context
· DORA + churn metrics monitored

LEVEL 4

Systemic AI with Human-in-the-Loop

AI embedded across the full SDLC, with human oversight at the moments that matter and complete process traceability.

· Human-in-the-Loop at every gate
· Full AI output traceability
· VSM active across the org

Most organizations that think they're at Level 3 are at Level 2. They have the policy written down, but they don't have the spec as a source of truth or the platform to prove the policy is actually being followed.



The five capabilities: how AI transforms each one

Each capability is a self-contained delivery unit. Without the right role configuration, it's not a capability — it's a client dependency.

The composition of an engineering team determines what it can deliver independently. What PDD, SDD, and Human-in-the-Loop describe as methodology, the five capabilities translate into concrete roles, deliverables, and intervention points.

THE FIVE CAPABILITIES IN THE DELIVERY CYCLE — WITH ENABLE AI & DATA AS THE FOUNDATIONAL LAYER

Discover & Build	Modernize	Operate	Quality & Test	Integrations

Enable AI & Data — foundational layer

Shared context · Prompt standards · Real-time metrics · End-to-end traceability

THE FIVE CAPABILITIES: HOW AI TRANSFORMS EACH ONE

Discover & Build

Not a phase: a discipline. A POC without a Business Analyst translating business constraints into technical requirements isn't a discovery process: it's a prototype that will need a full redesign before it reaches production. The output isn't a demo. It's a validated MVP with a real path to production.

Modernize

The most structurally demanding capability. It requires an Architect who can map what exists, a Tech Lead to manage the gap between legacy behavior and the target architecture, and DevOps that can run both systems in parallel. Without all three, modernization becomes migration. And migration without architecture becomes the next legacy system.

Operate

The most underestimated capability at the start and the most expensive to rebuild later. Operating requires continuous monitoring, regression discipline, and CI/CD ownership inside the team, not outside it.

Quality & Test

Quality isn't a final phase. It's a practice that starts at requirements. A QA function that only tests finished code finds defects when they're already expensive to fix.

Integrations

Integration failures are one of the most documented causes of cost overruns in software projects. This requires an Architect governing the API layer, a Tech Lead making the technical calls, and developers with specific integration experience. Without all three, every connected system becomes a maintenance liability.

FOUNDATIONAL LAYER · Enable AI & Data

Not a sixth capability: it's what makes the other five work with AI instead of working around it. It includes shared context across the squad, prompt standards, operationalized data pipelines, and real-time metrics visible to the client. Roles: Tech Lead · Dev · DevOps · PM.

How AI transforms each capability when it's integrated as process

Discover & Build	Modernize	Operate	Quality & Test	Integrations
BA · UX/UI · Architect · Dev · QA	Architect · Tech Lead · Dev · DevOps	Tech Lead · Dev · DevOps · QA	QA · BA · DevOps	Architect · Dev · Tech Lead
Stakeholder feedback synthesis; detection of conflicting requirements before design begins	Legacy code analysis, dependency mapping, and documentation of the existing system	Predictive monitoring that catches degradation before incidents; auto-generated operational docs	Automatic generation of edge cases; real-time coverage analysis	Automatic dependency mapping; API contract conflict detection before production
Less rework: requirements arrive validated before the team starts building	**The most expensive bottleneck in any modernization (the initial analysis) shrinks from weeks to days**	**AI spots the problem before it becomes an incident: the team acts, not reacts**	**Coverage stops being a snapshot at sprint end and becomes a continuous signal**	**Integration errors (the cause of over a third of migration failures) are caught before they reach production**

"

As AI absorbs more of the implementation work, the human role shifts from writing code to validating and verifying what AI proposed. Prompt engineering and context engineering are emerging as core skills for the modern software engineer.

In practice

A framework is only worth something if it can be executed with real roles and measurable results.

The scenario below illustrates how PDD, SDD, and Human-in-the-Loop work together across a delivery cycle. This isn't a story about adopting tools. It's a story about integrating process.

FOR ILLUSTRATION PURPOSES

What this model looks like in practice

A product team needed to digitize a business workflow with complex rules and frequent requirement changes, without sacrificing control over quality or security. Before structuring context, each developer prompted on their own terms: code volume was growing, but review comments weren't keeping pace — the same pattern described in section 02. The team adopted SDD as a practice: every story was consolidated as a single, traceable spec before any code was generated, and PDD kept those specs anchored to product goals rather than just technical tickets. All five Human-in-the-Loop checkpoints became mandatory gates, and the CI/CD pipeline started monitoring churn and coverage on every release.

1.3× AI-to-human code churn ratio — within the 1.5× benchmark: generated code was reviewed and refactored at the same pace as the rest, instead of accumulating

–30% change failure rate over two quarters, while deploy frequency increased — stability wasn't traded for speed

5/5 Human-in-the-Loop checkpoints enforced as mandatory gates on every release, not optional reviews

This scenario reflects patterns typical of how this model is applied — it does not correspond to a specific client or project.

30 years of track record in regulated industries

Three decades building software for banking, healthcare, insurance, and financial services. ISO 9001, 27001, and 45001 certified.

Operations in 7 countries, U.S. legal entity since 2019

We operate in Argentina, Brazil, Chile, Colombia, Mexico, Peru and USA. 300+ experts ready to integrate into any engagement model.

— Sources

Verified references used in this whitepaper.

DORA / Google Cloud. State of AI-Assisted Software Development, 2025.

DORA / Google Cloud. The ROI of AI-assisted Software Development (v.2026.1), 2026.

DORA / Google Cloud. AI Capabilities Model, 2025.

GitClear. AI Copilot Code Quality Research, 2025 and 2026 (The Maintainability Gap).

GitHub. Octoverse 2025.

Veracode. GenAI Code Security Report, 2026.

Amazon Web Services (AWS). AI-Driven Development Lifecycle (AI-DLC), 2025.

Hymel, C. The AI-Native Software Development Lifecycle. arXiv:2408.03416, 2024.

National Institute of Standards and Technology (NIST). AI Risk Management Framework (AI RMF).

OWASP GenAI Security Project. LLM Top 10 / Agentic Top 10, 2026.

ISO/IEC 42001. AI Management System.

Stack Overflow. Developer Survey, 2025.

AUTHORS



Alejandra Ochoa
Software Engineering Director



Jorge Attaguile
Chief Operating Officer