

Rewriting the Build

How AI Is Changing Software Development End-to-End

A practitioner's field guide for leaders of product and engineering organizations.

The wrong question, and the more interesting one.

The conversation about AI and software development has been dominated by a single question: *can AI write code?* It is the wrong question.

The more interesting shift is happening across the entire software development lifecycle — in how products get defined, how designs get made, how teams plan, how work gets tested, and how systems get shipped and operated. This paper is written from inside that shift. Kollabio builds software for federal agencies and commercial enterprises, and we work with teams navigating these changes in production.

The pattern we see across organizations isn't "AI replacing developers." It is something more fundamental: **the relationship between a practitioner and their work is being rewritten.** The shift is from *doing the work* to *getting the work done with AI*. It sounds small. It isn't.

Inside this paper:

- **A phase-by-phase walk through the lifecycle** — what's actually changing, where the leverage is, where the risk is, and what stays unmistakably human, across Product, Design, Agile, Engineering, QA, and Operations.
- **The team shape that's emerging** — fewer people, higher seniority, blurrier specialties, more emphasis on taste.
- **What this means for leaders** — five practical implications for product and engineering organizations adopting AI deliberately rather than by drift.

The conclusion is not that leaders should adopt AI — that decision is, practically, already being made by practitioners' daily behavior regardless of official policy. The pattern across all six disciplines is the same: AI is **amplification, not replacement**, and the discipline of using it well is harder than the marketing suggests.

CONTENTS

What's inside.

- 01 Introduction: The Wrong Question**
Why “can AI write code” is the wrong way to frame the shift.
 - 02 What's Changing Across the SDLC**
A phase-by-phase walk — Product, Design, Agile, Engineering, QA, Operations.
 - 03 The Team Shape That's Emerging**
Fewer people, higher seniority, blurrier specialties, more taste.
 - 04 What This Means for Leaders**
Five practical implications for product and engineering organizations.
 - 05 Closing: Amplification, Not Revolution**
What this means for leaders in the next 24 months.
-

ABOUT THE GROUND TRUTH SERIES.

*Ground Truth is Kollabio's series of field-level papers on AI's real impact inside software organizations. Paper 1 — this document — examines how AI is changing software development end-to-end, from Product Management through Deployment and Operations. Paper 2 — **Rewiring the Role** — examines how AI is changing the coordination work around that lifecycle. Papers 3 and 4 are in development. Each paper stands alone; together, they reflect Kollabio's view of how modern software organizations change in practice, not theory.*

The wrong question.

The conversation about AI and software development has been dominated by one question: *can AI write code?*

It's the wrong question.

AI coding assistants are real, and they matter. But focusing on them alone misses what's actually changing. The more interesting shift is happening across the entire software development lifecycle — in how products get defined, how designs get made, how teams plan, how work gets tested, and how systems get shipped and operated.

This paper is written from inside that shift. Our team at Kollabio builds software for federal agencies and commercial enterprises. We use AI across our own delivery process, and we watch closely how clients use it in theirs. The pattern we see across organizations isn't "AI replacing developers." It's something more fundamental: the relationship between a practitioner and their work is being rewritten.

What follows is a phase-by-phase walk through the lifecycle. For each, we describe what's actually changing, where the leverage is, where the risk is, and what stays unmistakably human. The pattern that emerges across all six phases is the same one: AI is amplification, not replacement, and the discipline of using it well is harder than the marketing suggests.

The frame for this paper: the shift is from *doing the work* to *getting the work done with AI*. It sounds small. It is not.

Phase by phase.

This is the core of the paper. We take each major discipline in the software development lifecycle in turn, describe what's actually changing, where the leverage is, where the risk is, and what stays human. The same four-part pattern recurs across all six disciplines — because the shift itself is a pattern, not an event.

2.1 Product Management

From requirements to reasoning partners.

WHAT'S ACTUALLY CHANGING

Traditional product management is a bottleneck game. A PM collects signals from users, stakeholders, data, and support — then distills, prioritizes, and writes. Backlogs grow faster than PMs can process them. Discovery gets rushed. Trade-offs go under-examined. AI doesn't change the inputs; it changes how fast a PM can move through them — and that change is large enough to expose habits the bottleneck used to hide.

WHERE THE LEVERAGE IS

Research compression. What used to be weeks of user-interview synthesis can now be done in hours when AI is pointed at raw transcripts, support tickets, and usage logs. The PM still makes the judgment calls; the grunt work of pattern-finding gets offloaded.

Spec quality. A PM writing a PRD with an AI assistant ends up stress-testing their own thinking earlier. The assistant asks the questions a skeptical engineer would ask in review — edge cases, data dependencies, rollout implications — before the work enters the team.

Prioritization transparency. Scoring frameworks (RICE, WSJF, whatever the team uses) become cheap to run on large backlogs. This doesn't eliminate political fights over what ships next, but it makes the data layer of those fights much richer.

WHERE THE RISK IS

Synthesis isn't judgment. An AI summarizing two hundred interviews can produce themes that sound plausible and aren't actually there. The PMs getting real value run synthesis more than once, from different angles, and go back to raw transcripts when a finding matters enough to ship against. The failure mode isn't laziness — it's trusting a confident summary that was never challenged.

Spec inflation. Because generation is cheap, PMs sometimes ship PRDs that read like every edge case has been thought through and end up burying engineers in detail that doesn't help them build. The discipline of *deciding what not to specify* gets harder when the marginal cost of more specification is near zero.

Data theater in prioritization. Running a RICE score on two hundred backlog items can substitute for the harder act of picking the right five. The numbers create a sense of objective decision-making while the underlying judgments — what counts as “reach,” what we mean by “impact” — are still entirely human and entirely contestable.

WHAT STAYS HUMAN

Knowing what the product is for, who it serves, and why. AI sharpens these judgments. It doesn't make them. Customer conversations the PM has personally are still the most reliable signal in the building. The PM who substitutes AI synthesis for hours-with-users gains speed and loses the only thing that actually matters.

2.2 UX Design

From artifacts to interfaces.

WHAT'S ACTUALLY CHANGING

Design is where the visible transformation is happening fastest. Tools like Figma Make, v0, and Claude artifacts now produce working interfaces from text prompts in seconds. It's tempting to declare designers obsolete. It's also wrong. The artifact-production layer of design — static mockups, prototypes, handoff specs — is collapsing in cost. What that exposes is what the artifact was for: transmitting design intent to engineers, and forcing the designer to think before drawing. Both purposes survive, but neither lives in the artifact anymore.

WHERE THE LEVERAGE IS

Sketch-to-prototype velocity. A designer can now go from a rough sketch to a working, clickable prototype without a week of pixel-pushing. The cost of testing an idea against a real user has dropped from days to minutes. Teams that used to ship two design rounds before usability testing now ship five.

Variant exploration. Instead of producing one polished comp, designers can produce ten variants — different layouts, different IA structures, different microcopy treatments — and bring them to a critique session as input rather than as a defended position. The conversation moves from “is this right?” to “which of these is right, and why?”

Handoff acceleration. Specs, redlines, and developer-ready states that used to take a designer half a day to assemble can be generated from the working prototype itself. The designer's time gets reclaimed for the parts of handoff that actually matter — the edge cases, the empty states, the failure modes.

WHERE THE RISK IS

Plausible but wrong. Generated interfaces look polished. They are also often wrong in ways that don't fail loudly — broken information architecture, inaccessible color contrast, microcopy that violates brand voice. A designer who treats generation output as draft material catches these. A designer who treats it as finished work ships them.

The convergence problem. Different teams using the same generative tools produce strikingly similar designs. The aesthetic of “AI-suggested” interfaces is real, and it tends toward generic SaaS visual language regardless of the brand it's supposedly serving. Differentiation requires designers willing to override the model's defaults.

Atrophy in the juniors. Junior designers who never spent time hand-building layouts, hand-tuning typography, or hand-pushing pixels may never develop the eye that lets a senior designer recognize what's wrong with a generation. The craft was always learned by doing the slow work. The slow work is exactly what AI now skips.

WHAT STAYS HUMAN

Information architecture, interaction patterns, accessibility, brand coherence — these don't emerge from a prompt. They emerge from designers who understand the user, the system, and the constraints, and who can evaluate generated output against that understanding. The designer of today looks more like a design director: curating,

editing, setting constraints, rejecting generations that look plausible but feel wrong. The title on the org chart doesn't need to change. The daily work already has.

2.3 Agile Practices

Less ceremony, more signal.

WHAT'S ACTUALLY CHANGING

Agile practices were built for teams of humans coordinating human work, in an era when status was scarce and had to be forcibly synchronized. When a meaningful share of the team's output is AI-assisted and a meaningful share of activity is already visible in source systems, the ceremonies start to creak. Estimation gets noisier. Standups feel ritualistic. Retros depend on memory rather than data. The ceremonies don't disappear, but their justifications change — and the teams using AI most effectively are quietly redesigning them.

WHERE THE LEVERAGE IS

Estimation as range. With AI assistance, velocity is bimodal — some work is 5x faster, some unchanged, and the difference isn't always predictable at grooming time. Teams have started treating estimates as ranges rather than points, or skipping estimation altogether on small stories. The result isn't sloppier planning; it's more honest planning.

Standups shift focus. “What did you do yesterday” matters less when AI can produce a commit log. “What are you *deciding* today” matters more. The fifteen-minute standup that used to be a status round-robin becomes a coordination conversation focused on blockers, judgment calls, and help requests.

Retros with real data. AI can summarize sprint activity, flag bottlenecks in cycle time, and surface patterns across multiple retrospectives that a human facilitator would miss. The retro conversation becomes about what to *do* about the patterns, not whether the patterns exist.

Backlog hygiene becomes cheap. Stale tickets, duplicate stories, orphaned epics — the kind of cleanup product teams always deferred — becomes a Saturday-afternoon task instead of a quarterly project. The backlog actually reflects what the team is working on.

WHERE THE RISK IS

Ceremony automation kills the signal. It is technically possible to have AI generate every standup update, every status report, every retro summary. Teams that go this far discover that the ceremonies' real purpose was the human contact — the moment someone says “I'm stuck” and someone else offers help. Automating the artifact eliminates the artifact's only useful side effect.

Velocity inflation hides debt. Teams running AI-assisted feel faster — and they are, on the work AI handles well. But the work AI handles poorly (legacy untangling, ambiguous requirements, architectural decisions) doesn't get faster, and it gets squeezed out of the sprint by the work that does. The team's velocity number rises while its hardest work falls behind.

Process theater migrates upward. When the mechanical parts of agile get cheap, teams that were marginal at agile sometimes respond by adding more ceremonies — quarterly OKR reviews, monthly portfolio syncs, weekly cross-team alignment meetings — to fill the calendar that AI freed up. The leverage gets eaten by overhead.

WHAT STAYS HUMAN

The team's relationship with its own work. Reading the room when someone is fatigued or overwhelmed. Judging when the team is lying to itself about what's blocking. Calling the shot on what to cut when the sprint slips. Holding people accountable in ways that build rather than corrode trust. None of this is AI-replaceable. Most of it gets harder, not easier, as ceremonies become more efficient — because the efficiency removes the slack that used to surface the human signal.

2.4 Software Engineering

From author to architect.

WHAT'S ACTUALLY CHANGING

The most-discussed shift, and the most misunderstood. AI coding tools work well for a specific profile of work: well-scoped, well-tested, bounded in blast radius. They work poorly on ambiguous requirements, tangled legacy systems, and architectural decisions that ripple across a codebase. The practical effect is that the engineering role bifurcates by where the work lives on that spectrum — and the bifurcation is happening faster than most engineering organizations are tracking.

WHERE THE LEVERAGE IS

Senior-with-AI is the new high-leverage profile. A senior engineer pairing with an AI assistant ships dramatically more — sometimes 2–3x more on bounded work, with comparable quality. The senior brings the judgment about what to build; the AI brings the speed of building it. Together they outperform either alone by a meaningful margin.

Boilerplate evaporates. CRUD endpoints, data-mapping code, configuration scaffolding, format conversions — the verbose, pattern-heavy work that used to consume real engineering hours — gets generated and reviewed in minutes. Those hours show up elsewhere.

Refactoring becomes tractable. Architectural improvements that used to be too expensive to undertake — renaming a deeply-used type, splitting a tangled module, migrating a deprecated API across hundreds of call sites — can be drafted by AI and reviewed by the engineer. Refactors that would have lived on the wishlist for years now get done in afternoons.

WHERE THE RISK IS

The senior-junior gap is widening. A junior engineer pairing with AI ships code that looks plausible and is subtly wrong, in ways the junior may not yet have the judgment to catch. The apprenticeship models that produced today's senior engineers — juniors doing rote work, slowly building intuition — are being disrupted exactly because the rote work is what AI now does. The pipeline that produces seniors in five years is breaking now, and few organizations have a replacement.

Code review becomes the bottleneck. When generation is cheap and review is not, throughput is gated by reviewers. The PR backlog grows. Reviewers burn out. The team's effective velocity sits far below its generation velocity, and the gap looks like productivity loss rather than what it actually is — a pipeline mismatch.

Architectural debt compounds at the new pace. When teams ship features twice as fast, debt accumulates twice as fast. The teams getting real leverage from AI are the ones that were already disciplined about boundaries, contracts, and observability. The teams without that discipline are accelerating into a wall — and the wall arrives sooner than they expect.

Vibe coding has a shelf life. Prototypes built with loose prompting and no structure work beautifully until the second developer tries to touch them. The serious

application of AI in engineering looks less like magic and more like a very fast, very literal colleague who needs clear instructions and well-tested boundaries.

WHAT STAYS HUMAN

Architectural reasoning across the whole system. Judgment on ambiguous requirements. Debugging in tangled legacy where the AI doesn't have full context. The ability to push back on a generated solution and say "this works but it's the wrong abstraction." Knowing when the test passes for the wrong reason. These are the muscles that took years of doing the slow work to build, and they remain the differentiating skill on every team we work with.

2.5 Testing and QA

From scarcity to abundance.

WHAT'S ACTUALLY CHANGING

Testing has always been resource-constrained. Coverage gaps were tolerated because comprehensive testing was prohibitively expensive. AI changes the economics. Tests that used to cost an engineer a day to write get generated and reviewed in an hour. The constraint on coverage stops being capacity. The new constraint is judgment about which tests matter — and that constraint is harder to manage than the old one.

WHERE THE LEVERAGE IS

Test generation at scale. AI can generate unit tests, integration tests, and property-based tests from existing code in minutes. The quality varies, but generated tests provide a floor of coverage most projects never had — particularly on legacy code that was never adequately tested in the first place.

Edge-case enumeration. Asking an AI to enumerate weird inputs, failure modes, and security boundaries of a function produces a list most developers wouldn't generate on their own. Not because developers aren't smart — because they're tired and time-pressured. The AI doesn't get tired.

Exploratory testing automation. AI agents can drive a UI, follow user journeys, and report anomalies. This doesn't replace experienced QA — the judgment of "this feels broken even if it technically works" is still human — but it scales the mechanical parts of exploratory testing in ways small QA teams couldn't before.

WHERE THE RISK IS

Tests assert what the code does, not what it should do. Generated tests are written by reading the implementation, which means they often validate the wrong thing — the bug, codified. A test that passes against buggy code provides false confidence, and the confidence is harder to undo than no test at all. Teams that ship generated tests without review create a coverage number that lies.

Coverage theater. It's possible to ship high coverage percentages with generated tests that wouldn't catch the bugs that actually matter. Coverage as a metric was always weak; with AI-generated tests, it's worse. The metric gets gamed without anyone deciding to game it, simply because generation makes coverage cheap and judgment doesn't keep pace.

QA judgment atrophies. When mechanical testing gets handled by AI, the strategic side of QA — defining what good coverage means, prioritizing risk, owning quality posture — can wither in teams that were already weak at it. The QA engineers who lean into the strategic role become more valuable. The ones who don't are at real risk of being mistaken for redundant.

WHAT STAYS HUMAN

Defining what “good coverage” actually means for a given product, in a given context, against a given risk profile. Curating which generated tests deserve to live in the suite and which need to be rewritten. The “this feels broken even if it technically works” judgment that catches the bugs no test would have predicted. Owning the overall quality posture in a way no automation can. The QA function isn't shrinking — it's moving up the abstraction ladder, fast.

2.6 Deployment and Operations

The quiet revolution.

WHAT'S ACTUALLY CHANGING

DevOps and operations receive less attention in AI conversations than engineering. Over a five-year horizon, the impact here may be larger. The work is verbose, pattern-heavy, and frequently done at 3 a.m. by tired humans — which is exactly the profile AI handles well. The teams that integrate AI into their operational stack see compounding gains over time, in ways that don't show up in a single quarter but become structural advantages.

WHERE THE LEVERAGE IS

Infrastructure-as-code becomes conversational. Writing Terraform, Kubernetes manifests, and CI/CD pipelines is the kind of verbose, pattern-heavy work AI does well. Teams ship infrastructure changes faster — and catch more mistakes in review, because reviewers can focus on intent rather than syntax.

Incident response compresses. AI assistants triage logs, correlate events, and propose hypotheses during incidents in ways a tired on-call engineer at 3 a.m. cannot. Time-to-detection and time-to-mitigation improve measurably in teams that have integrated AI into their observability stack.

Documentation stops being aspirational. Runbooks, architecture diagrams, post-mortems — the perpetually-neglected artifacts of operations — get produced automatically and maintained continuously. The quality ceiling is lower than a great human-written runbook. The quality *floor* is vastly higher than what most teams actually have.

Security expands its surface area. AI-assisted static analysis catches more vulnerabilities at commit time. AI-assisted runtime monitoring catches more anomalies in production. Neither replaces the security team, but both stretch its effective coverage in a discipline where coverage gaps are existential.

WHERE THE RISK IS

Cargo-cult infrastructure. Generated Terraform and Kubernetes manifests look right. They sometimes are. They sometimes embed assumptions — about IAM scope, network topology, secret management — that don't fit the environment. Teams that ship generated infrastructure without engineer review create blast radius they don't understand until the incident.

Confident wrong hypotheses during incidents. AI-assisted triage works most of the time, which is the worst possible reliability profile in incident response. The cases it gets wrong are the cases where its hypothesis sounds plausible enough to send the on-call engineer down a thirty-minute rabbit hole that the right human instinct would have skipped. Mean time-to-resolution improves on average; tail-latency on resolution sometimes gets worse.

Documentation that's wrong but looks authoritative. Auto-generated runbooks have a particular failure mode: they describe what the code or infrastructure was, not what it is now, in the same confident tone. Engineers who trust the generated doc as ground

truth waste time. Teams that don't actively review and prune their generated documentation accumulate quiet inaccuracy that surfaces during incidents.

Alert fatigue from AI-driven scanners. Static analysis and runtime monitoring at AI scale produces more alerts than humans can triage. Teams that don't invest in the calibration layer — what severity, what's noise, what's worth waking someone for — drown in low-signal findings and miss the high-signal ones.

WHAT STAYS HUMAN

The architectural intent behind the infrastructure. Calling the shot in an incident, especially when the AI hypothesis and the gut feeling disagree. Reading the actual operational pattern, not just what the dashboard says. Judgment about which security findings to prioritize and which to accept. The shape of the on-call rotation, the contracts between teams, the post-mortem culture — all of this is human and remains so. AI extends operational reach. It does not replace operational judgment.

The team shape that's emerging.

Put these shifts together and the outlines of a new team shape emerge:

- **Fewer people doing more work.** Not a slogan — a practical observation. Teams of 5 are doing what teams of 8 did three years ago.
- **A higher seniority average.** The work AI handles well is much of the work junior roles traditionally did. Teams are rebalancing toward more senior, more judgment-heavy profiles.
- **Blurrier specialties.** A PM who can prototype, a designer who can ship production code, an engineer who can write specs — these profiles are becoming common because AI reduces the cost of stepping outside one's discipline.
- **More emphasis on taste.** When everyone can produce more, what separates a great team from a mediocre one is the ability to tell the difference between “works” and “right.”

This is not a prediction. It's a description of teams we are already working with.

Five practical implications.

If you lead a product or engineering organization, a few practical implications follow from the patterns this paper has described.

- 1 Invest in senior judgment.** The ROI on hiring senior practitioners has gone up, not down. Budget accordingly.
- 2 Rebuild the apprenticeship path.** Juniors won't learn the way they used to. If you don't design a new path, you won't have seniors in five years.
- 3 Tighten your architecture.** AI accelerates whatever direction you're already headed. If that's good, great. If it's bad, that's bad twice as fast.
- 4 Treat review as strategic.** Where review capacity is the bottleneck, throughput of the whole organization is gated. Resource it as such.
- 5 Measure outcomes, not activity.** Lines of code, commits, and ticket counts were always weak metrics. With AI they're worse. Move to outcome metrics — cycle time, defect rates, customer-observed quality.

Amplification, not revolution.

The honest frame for AI in software development is not “revolution” and not “hype.” It's **amplification**.

AI makes good teams better faster. It makes bad teams worse faster. And it makes the difference between those two kinds of teams much more visible.

The question for every software organization is no longer whether to adopt AI. It's whether your practices, your people, and your product strategy are ready to be amplified.

This paper reflects what we see across our engagements with federal agencies, GovCon primes, and commercial enterprises. If you're navigating these shifts inside your own organization, we'd be glad to compare notes.

About Kollabio

Kollabio is a design and engineering firm that creates human-driven, AI-empowered solutions for federal agencies, GovCon primes, and commercial enterprises. We combine **Experience Thinking** — practical design thinking with the how-to included — with Agile delivery and continuous deployment to ship measurable value in multiple increments and with continuous feedback loops.

Our core competencies span strategy, experience design, product development, cloud architecture, and DevSecOps. We hold 8(a), HUBZone, CMMI DEV ML3, CMMI SVC ML3, ISO 9001:2015, and ISO 20000-1:2018 certifications, and are a GSA Contract Holder (MAS) as well as a holder on CMS DASH BPA and SBA EADS BPA contract vehicles.

This is Paper 1 in Kollabio's *Ground Truth* series — field-level papers on AI's real impact inside software organizations. If you are navigating the shifts described in this paper inside your own organization, we welcome the conversation.

CONTACT

sales@kollabio.com
571-295-7209

OFFICE

23710 Schooler Plz Ste 275
Room #2058
Brambleton, VA 20148