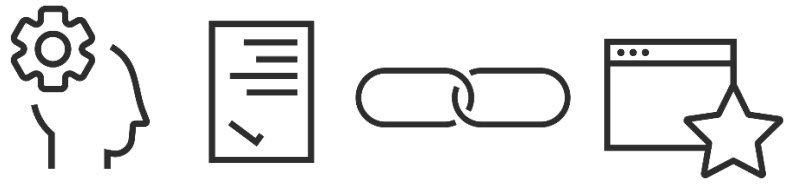
code4thought ®

AI-Assisted Software Engineering: The New Delivery Paradox

A snapshot of the Software Engineering profession repositioning itself through structural changes in real time



About code4thought: code4thought is a technology company focused on rendering AI and large-scale software systems trustworthy and thoughtful. Through our proprietary platforms, iQ4AI and Orisign and expert advisory services, we help organizations assess, govern and improve AI systems and AI-assisted software engineering practices across their lifecycle. Our work combines software quality expertise, AI testing and assurance, international standards awareness and practical enterprise advisory, empowering organizations to build, adopt and scale technology with confidence, compliance and control.

Contents

Editorial

Yiannis Kanellopoulos, Founder and CEO, code4thought

When we published the open invitation that seeded this paper[1], we were careful to frame it as a set of questions rather than a thesis. Six long conversations later, I am more convinced than ever that this framing was the right one. What follows is not a verdict on AI-assisted software development. It is a snapshot of a profession reasoning its way through a structural change in real time.

In this paper, you will find no consensus. The people who contributed have their own perspectives and opinions. They differ about what the new primary artifact of our craft actually is. They disagree about how quickly organizations can responsibly absorb AI-generated output. They even disagree about whether code itself still matters in the way it used to. What they share is something more valuable than consensus: a refusal to reduce this moment to either hype or fear.

It would be a mistake not to thank all the contributors to this paper: Ejona Preci, George Marinos, John Fox, Markus Borg, and Meri Roboci.

A few threads are unmistakable. The first is that velocity without governance is not progress — it is, as one contributor put it, entropy with a ticket number attached. The second is that AI is an amplifier: it makes disciplined organizations more effective and undisciplined ones more dangerous, at the same speed. The third is that the locus of engineering judgment is moving upstream. Whether we call the new artifact a prompt, a spec, a way of working, or simply the solution to the user's problem, the work that matters is increasingly the work of framing intent clearly, testably, and with an eye on consequences.

There is also a quieter, more humane thread running through these interviews. Several contributors spoke, unprompted, about the junior engineers entering the profession now, about the cohort of mid-career developers whose craft is being reshaped mid-flight, and about the risk of de-skilling that accompanies any powerful abstraction. This is not a technical footnote. It is the single most important governance question our industry faces, and it will not be solved by tooling alone.

At code4thought, our bias is towards measurement, assurance, and the boring disciplines of software quality. That bias is on display in some of the framings below. But the reason we commissioned this paper was precisely to test our own assumptions against practitioners who see the problem from different angles — security, product engineering, academic research, enterprise adoption, and regulated industry. The result, I hope, reads less like a company point of view and more like a field report from the frontier.

We are not at the end of this conversation. We are barely at the beginning. If this paper helps you ask sharper questions inside your own organization — and argue about the answers with better evidence — it has done its job.

Yiannis Kanellopoulos
Patras, May 2026

[1] https://code4thought.eu/2026/02/25/the-future-of-ai-assisted-software-development-why-its-time-to-ask-better-questions-an-open-invitation-for-answers/

Executive Summary

The future of AI-assisted software development is not arriving as a single event. It is arriving as a structural change in what software engineering produces, how organisations absorb it, and who in the profession holds the advantage. This paper distils that change through six long conversations with practitioners who see the problem from different angles: security, product engineering, academic research, regulated industry, retail banking, and our own perspective from inside code4thought.

The headline finding.

AI does not break software engineering — it amplifies whatever discipline, or lack of it, is already there. The locus of engineering value is moving upstream, from code to intent. Most organisations are not yet ready for that move.

The five questions, in one line each:

- **Primary artifact**. The deliverable is no longer code. It is the upstream artifact — prompt, specification, solution, way of working — and most of it is currently unversioned, unreviewed, and unaudited.
- **Abundance**. The bottleneck moves to comprehension. "Knowledge debt" — code that nobody understands — is now a sharper risk than technical debt, and rebuilding what you already buy is rarely the saving it appears to be.
- **Trust**. Inspect-and-approve does not survive at machine speed. Trust must be architected into the rails — through deterministic tooling, policy-as-code, and a human who can still stand behind the output.
- **Prerequisites**. AI amplifies what you already have. Most organisations have done neither the human housekeeping (literacy, safety, adoption support) nor the technical housekeeping (code health, identity, model risk) required to deploy agents safely.
- **The engineer**. The role is moving from code producer to architect, orchestrator, and problem framer. The transition will be uneven, and the disappearance of the entry-level rung is the open question the industry has not yet answered.

What to do on Monday morning.

Each section closes with a short list of questions to ask inside your own organisation. Start there.

1. If AI generates the code, what becomes the primary artifact?

For three decades, source code has been the unambiguous artifact of software engineering — the thing we reviewed, measured, secured, and handed over at release. When AI writes a meaningful share of that code, the question is no longer academic. If the implementation is generated on demand, what are we actually producing?

The contributors to this paper reach similar conclusions by different routes. Two of them place the prompt squarely at the center.

> *"The primary artifact becomes the prompt. When AI generates the code, the code is a derivative. The intent, the constraint, the context — all of that, lives upstream in the prompt. And prompts today are unversioned, unreviewed, and almost entirely unaudited."*
>
> — Ejona Preci, Group CISO, LINDAL

Meri Roboci frames the same shift in linguistic terms. If programming was once the language we used to speak to the machine, prompting has become that language, and the differences between AI tools are increasingly differences in how well they understand our instructions.

John Fox pushes the analogy further. He compares today's generated code to compiled machine code, a downstream artifact that no one thinks to version-control because it is reproducible from a higher-level source.

> *"In the same way that you have source code and you have documentation, really now what you're saying is context and well-crafted prompts... and the generation of source code is the same thing as the generation of machine code from a higher-level programming language."*
>
> *— John Fox, Senior UI Engineer, Netflix*

Yiannis Kanellopoulos takes the argument one level further still. The artifact, he insists, was never really the code in the first place. Software has always existed to solve someone's problem; the reason source code became the de facto deliverable was that getting a processor to do anything was the hardest part of the job. With that barrier collapsing, the deliverable is becoming what it should have been all along, the solution itself.

George Marinos takes the argument into a more uncomfortable territory. Having watched code decay into disposability long before AI arrived — frameworks churning, programming languages upgrading, half of any given codebase rewritten within a few years — he argues that even the design documents and specifications we might be tempted to elevate in code's place are themselves now AI-generated. The real asset, he says, is the way of working itself: the institutional method by which a team converts intent into outcome.

Markus Borg offers a useful counterweight. He accepts that the point of interaction has moved upstream, but refuses to write off the code itself, particularly in safety-critical contexts, where what executes on the target platform is still what affects users and must still be tested, statically analyzed, and held to account.

Taken together, the answer is not that code no longer matters, but that its role has changed. Code is becoming the compiled output of something more abstract and more human: intent, context, constraint, and process. The profession's assurance machinery will need to reach further up the

chain, and Ejona Preci's warning about unversioned, unaudited prompts as a supply-chain vulnerability deserves to be taken seriously before, not after, the first major incident.

Practical Takeaway: Questions to ask your organisation:

- What are the artifacts we are actually preserving when AI generates our source code — the prompts, the context, the specifications, or just the output?
- Where do our prompts live today? Are they versioned, reviewed, and auditable in the same way our source code is?
- If a regulator or customer asked us tomorrow to reproduce *why* a piece of generated code does what it does, could we?
- Have we updated our definition of "intellectual property" and "engineering deliverable" in contracts, job descriptions, and assurance processes — or are we still assuming it is just the source code?
- Who in the organisation owns the upstream artifact (intent, prompt, specification), and is that person empowered or skilled the way a lead engineer used to be?

2. Can organizations absorb the abundance of AI-generated features?

If AI reduces the marginal cost of producing a feature to near zero, the bottleneck moves. Every contributor to this paper identified a downstream constraint that organizations are nowhere near ready for.

> *"If we ever get past technical debt, every study now shows we have a new debt, the knowledge debt. We are producing code that nobody understands. And the moment you need to evolve it, you discover you cannot evolve what you never understood in the first place."*
>
> *— Yiannis Kanellopoulos, Founder and CEO, code4thought*

Yiannis Kanellopoulos's warning lands sharply because it reframes a familiar industry term. Technical debt might be painful, but nowadays it has become measurable (so we can control it) and is solvable; in the sense that we have the proper tooling in place to mitigate it.

Knowledge debt is different: it begins the moment AI-generated output enters production without a human ever truly comprehending what it does. Kanellopoulos cites a related trap he increasingly sees in organizations that announce they will replace their SaaS estate by rebuilding it with AI agents. "They forget that the initial cost of software is roughly 20% of its total lifetime cost; the remaining 80% is evolution." Cheap generation does not pay that bill.

Ejona Preci describes three cohorts of organizations:

- those that have bolted AI onto existing human-paced processes and will drown in their own output;
- those that have paused adoption entirely and will lose the competitive race;
- and a third group rebuilding their operating model around AI-native assumptions, automated policy enforcement at the pipeline level, continuous validation in place of periodic review, and risk scoring that keeps pace with commits.

Only the third group, she argues, is actually adopting AI. "Velocity without governance," in her phrase, "is entropy with a Jira ticket."

> *"The cost of creating code is approaching zero. There's no way a human can keep up with that."*
>
> *— Markus Borg, Adjunct Associate Professor, Lund University*

Markus Borg agrees that few organizations are ready and points to the adjacent bottlenecks nobody budgeted for: testing teams working in old rhythms, marketing unable to keep pace with releases, and monetization models that assumed a slower cadence of capability. His caution is sharper than it first appears: "companies that treat AI purely as a cost-cutting lever", he warns, "will fail to benefit from the disruption, while those that use it to change what they can build will steamroll the competition."

Meri Roboci brings a regulated-industry perspective. "In regulated environments or in critical infrastructure, any AI-generated capability is evaluated against business value, data flows, model ownership, and proximity to a workflow. 'Less is more,' especially as governance complexity grows with every additional feature."

John Fox reframes the question altogether. The interesting metric is not lines of code produced, but lines removed, bugs resolved, and the quality of what executes, “whether the rocket comes back.” George Marinos pushes back from the opposite direction: “the average quality of AI-generated code”, he argues, “is already higher than the average quality of the code being written by the global developer population, because most software in the world is written in bulk, under pressure, by people with fewer than five years of experience.” Seen from that angle, AI is less an accelerant of chaos than a standardiser of mediocrity if organizations build the traceability and measurement to prove it.

Practical Takeaway: Questions to ask your organisation:

- Which of Ejona Preci's three cohorts are we honestly in — bolted-on, paused, or AI-native? What evidence supports that answer?
- What are our downstream bottlenecks — code reviews, testing, product-market fit, targeted marketing, monetisation — and have we resourced them for the new pace?
- Are we just measuring AI productivity by volume (lines, commits, tickets closed) or by outcome (bugs resolved, code removed, customer problems solved)?
- How much of the code we shipped this quarter does *no one in the organisation actually understand*? What is our exposure to "knowledge debt" as distinct from technical debt?
- Before we replace any SaaS tool with an in-house AI build, have we costed the 80% of lifecycle cost that comes after the first release?

3. How do we ensure trust at machine speed?

Once the cost of producing code collapses, the cost of inspection becomes the binding constraint. Every contributor in this paper arrived at the same conclusion, from different starting points: the traditional model of review-and-approve does not survive contact with agentic development.

> *"Trust isn't something you grant after a review. At machine speed, trust is something you have to architect into the rails from the start."*
>
> *— Ejona Preci, Group CISO, LINDAL*

Ejona Preci's prescription is uncompromising: stop trying to inspect everything and start enforcing invariants. Policy expressed as code, cryptographically enforced. Immutable audit trails written at generation time, not reconstructed afterwards. Kill-switches that are real, not policy documents that assume a human will approve at 3 a.m. The architecture she describes is closer to hardware security module design than to traditional application security engineered for an actor that is fast, tireless, and incapable of moral judgment.

Yiannis Kanellopoulos's answer is deliberately systemic. Trust, in his framing, must be built around a triad:

- deterministic non-AI tooling, such as static analysis and established quality measurements;
- the AI agents themselves; and
- the human in the loop.

None of the three is sufficient on its own. Forward-looking teams are actively experimenting with combinations: deterministic analysis to anchor what is verifiably true, paired with non-deterministic agentic review to find what humans miss. The wider point: a developer who runs an agent without being able to read and stand behind what the agent produces has not delegated work, only liability.

Meri Roboci's answer is complementary: zero trust as a default, borrowed wholesale from cybersecurity practice.

> *"We still apply a zero-trust approach. Just as in cybersecurity, the same discipline extends to AI. In environments where critical information is at stake, we move deliberately. Trust is not a constraint on innovation; it's the condition for sustainable innovation."*
>
> *— Meri Roboci, Cyber Security and AI Enablement, DWS Group*

Markus Borg brings in the human layer. Trusting AI-generated code is not unlike trusting any other component that occasionally fails, since the discipline of building trustworthy systems out of individually fallible parts is not new. What is new is the de-skilling risk that accompanies heavy reliance: when you, as a software developer, get disconnected from your own code, so does some of the knowledge that lets you spot when something is wrong.

George Marinos makes an industrialization argument. Software has, until now, been a pre-industrial craft — home-workshop production at a civilizational scale. Agentic development is the first thing that could plausibly industrialize it: consistent patterns, massive-scale pattern detection across thousands of repositories, and process traceability that was never feasible when every developer's contribution was a one-off. John Fox returns to an older formulation — trust but verify — and the

Cold War phrase translates cleanly: “one can trust that something useful is being produced because the end-user result is visible, but verification is non-negotiable.” That, too, is a design requirement, not a review step.

Practical Takeaway: Questions to ask your organisation:

- Where is trust currently *granted by review* in our pipeline, and where could it instead be *architected into the rails* (policy-as-code, immutable audit trails, scoped credentials, real kill-switches)?
- Do our developers understand and stand behind every piece of AI-generated code that reaches production — or have we quietly delegated comprehension along with implementation?
- Have we paired our agentic tooling with deterministic tooling (static analysis, established quality measurement) so that one verifies what the other cannot?
- In which parts of our business are we willing to *trade speed for verifiability*, and have we said so explicitly?
- What is our plan for the slow-burn de-skilling risk — the loss of judgment that comes from never having to read your own code?

4. What prerequisites separate success from chaos?

The question the industry most wants answered — "Do we need to clean-up first, or can the agents clean-up for us?" — turns out to be less interesting than the question underneath it: what foundations make agentic development safe to attempt at all?

Yiannis Kanellopoulos opens with a prerequisite that almost no technical roadmap captures: most developers, in most organizations, are not yet using these tools. He cites a striking working figure from the engagements he sees regularly: "only 20% of developers in a given organization have meaningfully adopted AI assistance. The rest are watching, waiting, and quietly worried about their jobs."

No technical prerequisites matter until that human prerequisite is addressed: making people feel safe with the change, properly training them, and measuring adoption in ways that help teams rather than punish individuals. Once that is in place, he is unequivocal about what comes next.

> *"We are entering the industrial era of software engineering. That requires a new kind of discipline, not obedience, but standardization. All those best practices we have been preaching for decades: now is the moment they actually have to be followed. Civil aviation operates by the book. So must we."*
>
> *— Yiannis Kanellopoulos, Founder and CEO, code4thought*

The amplification metaphor recurs across every contribution.

Ejona Preci lists the foundations she sees most often missing: "data hygiene before AI touches anything, identity and scoped credentials for non-human principals, a genuine model risk framework, and, crucially, executive understanding that AI security risk is enterprise risk, not an IT problem to be delegated downward."

Her formulation is blunt: AI amplifies whatever you already have, good and bad, and feeding unpatched vulnerabilities into an AI-accelerated pipeline does not bury the problems; it multiplies them. Yiannis Kanellopoulos sharpens the same point with a memorable image: agents are like Ferraris, and most old codebases are dirt roads. Buying the car does not give you a track.

Markus Borg, whose research group has studied this directly, reports a strong correlation between code health and agent effectiveness. "Healthy, modular code allows agents to do precise work; messy code trips them up for the same reasons it trips up humans." His practical conclusion: "Do not expect agents to fix your worst legacy systems magically, but do use deterministic tooling, such as static analysis and quality metrics, to give them the context they need."

John Fox argues that hygiene is really about proving comprehension. Before writing code, the best engineers he has worked with always wrote pseudocode, not for the compiler but for the humans who would inherit the work. "AI changes nothing about this discipline except its importance," he notes.

> *"When we led with governance, people experienced it as obstruction because they couldn't see the reasoning behind the rules. When we shifted to building AI literacy first, the governance that followed felt logical, not imposed. Sequence changes everything."*
>
> *— Meri Roboci, Cyber Security and AI Enablement, DWS Group*

Meri Roboci's sequence — literacy first, governance second — dovetails with Yiannis's human-first prerequisite. George Marinos's version of the same argument lands at standards: a shared, team-

wide discipline for where the agent keeps design decisions, specifications, and open items, so that overlapping or stale information stops being the default failure mode.

Practical Takeaway: Questions to ask your organisation:

- What percentage of our developers are actively using AI tools today, and how do the non-adopters describe their reasons? (If we have not asked, that itself is the finding.)
- Are we leading our AI rollout with literacy and psychological safety — or with governance documents and checklists people will work around?
- How healthy is the codebase we are about to point our agents at? Have we done the housekeeping, or are we taking a Ferrari off-road?
- Is AI risk owned at the executive level as enterprise risk, or has it been delegated to IT and Security as a tooling problem?
- Do we have shared standards for *where* the agent keeps design decisions, specifications, and open items — or is every team improvising?

5. What happens to the software engineer?

On the question of the profession itself, our contributors are the most emphatic — and the most unified.

> *"There's very little middle ground for software engineers, and the determining factor is whether they evolve from code producers to architects. The engineer who can't explain why the system should behave a certain way is already being automated."*
>
> *— Ejona Preci, Group CISO, LINDAL*

Ejona Preci's concern is less with the eventual shape of the role than with the transition period. An entire cohort of engineers has been selected and rewarded for implementation speed; the new bottleneck is judgment, and judgment takes longer to build than keystrokes. Organizations that invest in retraining — critical evaluation, security reasoning, system thinking — will keep their talent. Organizations that cut headcount because AI writes the first draft will discover that accountability without expertise is just liability.

Markus Borg is pragmatic. "Software engineering was never only about writing code," he points out, "and the code-monkey archetype was already being disrupted before agents arrived." What is genuinely new is the need to master the agents themselves; the engineer to be what he variously calls "the architect, the integrator, the orchestrator, or, more plainly, the person who knows when to read the generated code and when to trust it."

Yiannis Kanellopoulos is more optimistic. Having spent years away from hands-on coding, he describes the experience of returning to it with agentic tools as a rediscovery: an assistant that returns ideas, suggests improvements, and offers a perspective on his own work that he would otherwise lack. "That," he argues, "is the real promise of the moment, not faster typing, but the resurrection of creative bandwidth in a profession that had quietly been losing it. The future engineer," in his phrase, "has no excuse for not being creative."

> *"The best results with agents are being achieved by people with managerial skills. The agent is effectively a junior collaborator, and what matters is how clearly you can describe the request, the evaluation criteria, the constraints. The era in which the developer was a scarce, expensive resource to be protected is over."*
>
> *— George Marinos, Assistant General Manager, Innovation & Digital Partnership, National Bank of Greece*

George Marinos reports, based on direct observation of his own teams, that the engineers achieving the best results with agents are those with managerial skills. "The bet going forward," in his reading, "is on productivity of thought, not productivity of typing."

Meri Roboci sees prompt engineering as the natural successor to the long arc that took programming from binary to assembly to high-level languages to something close to plain English. "The really good engineers were always the ones who could program their own thinking before they wrote a line; those same people will be the ones who prompt well." Her worry, shared by several contributors, is the entry-level job — the meeting notes, first task list work that taught juniors the shape of the organization — and the absence of an obvious replacement rung on the ladder.

John Fox ends where he began: the least difficult part of what he does as a software engineer has always been writing the code. "The advantage has shifted, decisively, to those who can understand the problem and talk to humans about it."

Practical Takeaway: Questions to ask your organisation:

- Are we investing in the upskilling our existing engineers — judgment, system thinking, security reasoning, problem framing — or just measuring their typing speed in a new tool?
- Where, today, is our entry-level rung on the engineering ladder, and what work is still genuinely formative for a junior?
- Is our hiring criteria still optimised for the engineer we needed five years ago?
- How are we recognising and rewarding the engineers whose value is now mostly upstream — describing problems clearly, briefing agents well, evaluating outcomes — rather than producing volume?
- If AI removes completely the friction of writing code tomorrow, what would our best engineers actually create with the bandwidth they get back?

Closing note

This paper was commissioned as an open question, not as a position. The six practitioners who contributed were generous enough to express their own diverse opinions openly in public, with us and with each other, and the paper is stronger because of it. What they have in common, beneath the different vocabularies, is a conviction that this moment rewards clarity of intent and punishes ad-hoc improvisation. The tools are changing faster than the operating models that wrap around them. Closing that gap is, in the end, the work.

Contributors

(in alphabetical order)

Ejona Preci, Group CISO, LINDAL

> Ejona Preçi is Group CISO at Lindal Group and Responsible AI Governor for Germany at GCRAI. She operates at the intersection of Cybersecurity and AI, driving innovation that is secure, transparent, and human-centric. Recognized as Cybersecurity Woman of the Year 2024 (Barrier Breaker), Global 40 under 40 in Cybersecurity, and Top Cybersecurity Voice 2025, Ejona inspires global audiences as a thought leader, author, and keynote speaker committed to advancing a responsible digital future.

George Marinos, Assistant General Manager, Innovation & Digital Partnership, National Bank of Greece

> George Marinos is Assistant General Manager of Innovation and Digital Collaborations at the National Bank. He is responsible for developing and adopting AI technologies at NBG. From May 2017 to December 2023, he was the Director of Digital Transformation, leading the technological development of the Bank's digital services. Over the past 15 years, he has contributed to nearly all of the National Bank's technological and digital modernization projects. He has previously worked in the technology sector, including both the private and broader banking industry. He has studied at the School of Electrical and Computer Engineering at the National Technical University of Athens.

John Fox, Senior UI Engineer, Netflix

> John Fox is a Senior iOS UI Engineer at Netflix, where he has spent the last decade building interface experiences at scale. A true pioneer in the Apple development space, John's career ranges from the early days of NeXT WebObjects to modern iOS frameworks. He previously co-founded WebWare Corporation, serving as CTO and architecting an award-winning Digital Asset Management platform used by brands like Sony Pictures and Martha Stewart. He is also the founder of GroupSmarts, the creator of the Macworld "Best of Show" winning app MemoryMiner.

Markus Borg, Adjunct Associate Professor, Lund University

> Dr. Markus Borg is a principal researcher with CodeScene and an adjunct associate professor at Lund University. His research interests reside in the intersection of software engineering and applied artificial intelligence. He serves on the editorial board of Empirical Software Engineering and is a department editor for IEEE Software.

Meri Roboci, Cyber Security and AI Enablement, DWS Group

> Meri Roboci is a Cyber Security and AI Enablement strategist with deep expertise in AI governance, behavioral security, and human-centred defense, specialized is financial services ecosystems. With a background spanning business informatics, information security, and digital transformation, she brings an interdisciplinary perspective to some of the most complex challenges at the intersection of AI and cybersecurity. An international keynote speaker and panelist, Meri speaks regularly on AI enablement, organizational resilience, and why the human layer remains the most critical variable in any security architecture. She is based in Frankfurt, Germany.

Yiannis Kanellopoulos, Founder and CEO, code4thought

> Yiannis Kanellopoulos is the Founder & CEO of code4thought, a technology company specializing in AI testing, software quality, and trustworthy AI governance. With a background spanning deep-tech entrepreneurship, software engineering, and AI assurance, he has spent more than 17 years helping organizations build reliable and accountable digital systems. A former Practice Leader Greece at Software Improvement Group (SIG), he holds a Ph.D. in Computer Science from the University of Manchester. Yiannis is also a Board Member of Orange Grove, supporting innovation and entrepreneurship in Greece, and is an active advocate for responsible AI and technology-driven societal impact.

Disclaimer: The views expressed in this report were based on interviews with the contributors. All the contributors, except for Yiannis Kanellopoulos, are not affiliated with code4thought in any way.

Take the Next Step Toward Governed AI-Assisted Software Engineering

AI-assisted software engineering is changing how organizations create and govern software. As code generation accelerates, technology leaders need stronger evidence, clearer ownership and better control over quality, knowledge, intent and risk.

code4thought helps organizations move from AI coding experiments to governed, evidence-based software engineering practices through its AI Software Delivery Governance service and Orisign, our proprietary governance platform.

By combining team-level evidence, expert interpretation and recurring governance reviews, we help organizations scale AI-assisted software delivery responsibly and prepare for more autonomous and agentic engineering models.

Visit code4thought.eu to learn more!