re:cinq

The AI Adoption Ladder

From First Licences to the Dark Factory

Nine stages of AI coding adoption, and the bottleneck at each one

The AI Adoption Ladder

From First Licences to the Dark Factory

Engineering teams that adopt AI coding tools usually see the same two things. In the first weeks, developers get faster. A few months later, the company notices that delivery has not changed in the same way: features do not ship noticeably faster, and the roadmap moves at roughly the old pace. The individual gains are there, but somewhere on the way to the company level, most of them get lost.

This paper describes a model we use to explain where they get lost, and how to get them back. We call it the AI Adoption Ladder. It has nine stages, from the first engineers experimenting individually to "dark factories" where agents handle most of the development work end to end. At every stage there is one specific bottleneck that blocks the way up. Teams that find and fix that bottleneck climb to the next stage. Teams that don't stay where they are, no matter how many licences they buy.

We built the ladder from our own engagements across many engineering organisations, including a programme that moved 100 engineers at Odevo, a $3B property management platform, to agentic coding in six weeks. Keep in mind that this paper is a field model, it describes what we have seen, and we treat it as a working hypothesis. Where published research supports it, we cite the research, and where we are less certain, we say so.

The paper covers five things: why the gains stall, the nine stages of the ladder, a short test to find the stage your own team is at, the order in which we have watched teams climb, and the questions the ladder does not answer yet.

Contents

The Plateau Every Dashboard Hides

It starts well. Developers adopt AI tools and get faster at their current work. The engineers feel the difference, and the dashboards confirm it with more pull requests and faster ticket closes. The first weeks feel like a big change.

Then the curve flattens. The developer who got 30% faster in month one is not getting 30% faster again in month four. The gains stop compounding, and it is hard to say with any certainty where they went.

Around the same time, new problems appear. Keeping a shared codebase coherent becomes a new hard problem when ten people and thirty agents push changes to it every day, each with their own tools, prompts, and conventions. Quality becomes harder to judge, because the questions multiply: whether the code is correct today, whether anyone will be able to maintain it in a year, whether the team still understands everything it is shipping. Incidents start appearing in code that wasn't written by hand, where the usual debugging instincts are less useful.

Older weaknesses become more expensive as well. Thin test coverage was survivable when humans wrote all the code at human speed, but at several times the volume, every gap in the safety net is exposed. Manual QA, manual deployments, outdated documentation, and unreliable CI pipelines follow the same pattern. AI did not create any of these problems. They were already there, AI just amplified them.

This leaves leadership questioning: is this the ceiling, or has the team barely started? In most organisations that question is hard to answer, because the metrics that exist were not designed to answer it.

We see this shape in almost every engineering organisation we work with, and the research, which we cover in the next chapter, describes the same pattern at scale. The right question to ask is where the gains went, and what it takes to get them back.

Hidden technical debt exposed

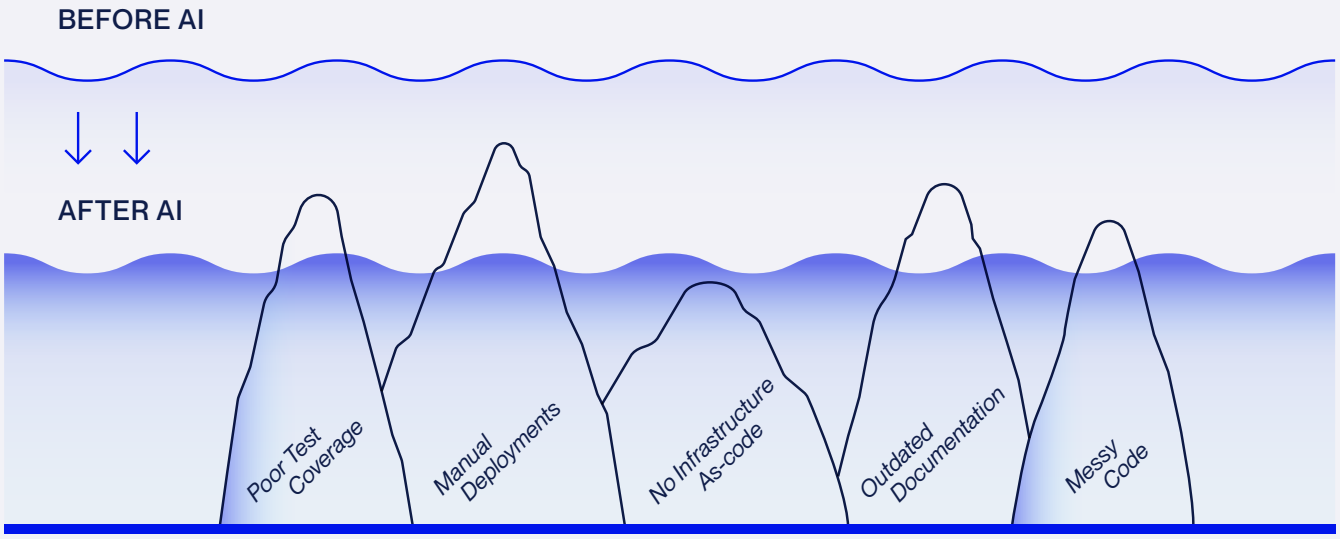


Figure 1. The "Hidden Technical Debt Exposed" iceberg: the waterline drops after AI, exposing poor test coverage, manual deployments, no infrastructure-as-code, outdated documentation, messy code.

Why the Gains Stall

Our explanation is a working hypothesis. We built it from our engagements, from the practitioners we talk to on our podcast Waves of Innovation, and from our own engineering work, and the published research below is consistent with it. Parts of it may still turn out to be wrong, and we expect to revise it as more teams reach the later stages. The mechanism, as far as we can tell, is amplification: AI changes the speed at which a team's existing strengths and weaknesses produce results, much more than it changes the strengths and weaknesses themselves. A team with solid foundations gets the full benefit of that speed, while a team with weak foundations discovers its problems faster than it can repair them.

The underlying idea isn't new, it's been around for more than forty years. Eliyahu M. Goldratt's Theory of Constraints argues that the throughput of any system is determined by its biggest bottleneck. Improving other parts of the system doesn't increase overall throughput; it simply causes work to pile up where the real constraint still exists. AI can accelerate many parts of the software development lifecycle, from writing code to drafting requirements, reviewing changes, and testing. But most teams apply it to code generation first, which is why that's where the initial productivity gains appear. If writing code was your primary bottleneck, overall throughput improves. If it wasn't, you've only made one part of the process faster while the real constraint remains unchanged. Goldratt's recommendation was straightforward: identify the bottleneck, remove it, then repeat the process. Sometimes that means applying AI to the next constraint. Other times it requires better automation or changes to the way the team works. The principle stays the same – the bottleneck never disappears, it simply moves, and progress comes from continually addressing whatever is slowing the system down next.

Same pipeline each row. The bottleneck moves downstream as each prior constraint is fixed.

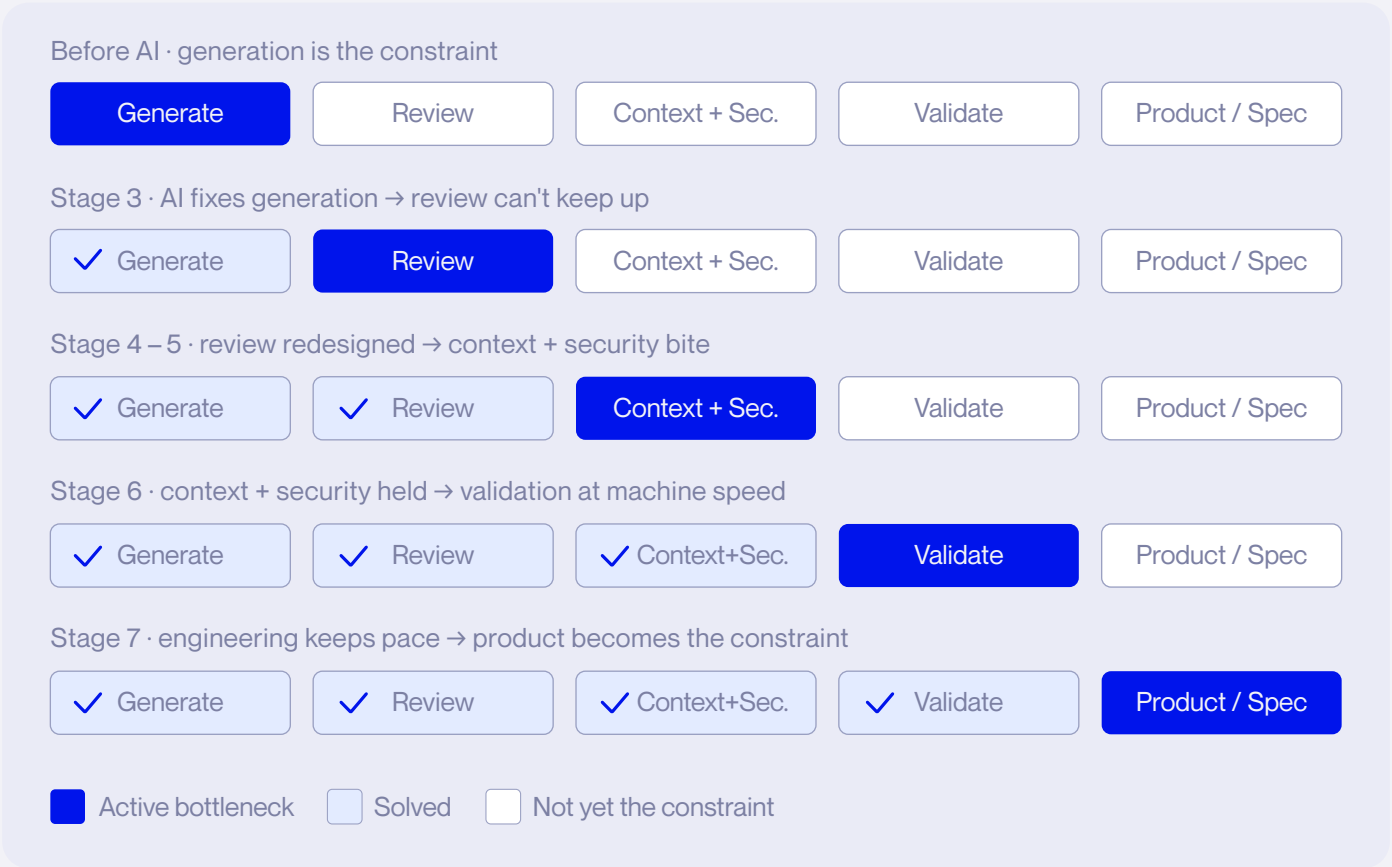


Figure 2. The bottleneck moves downstream. Once AI speeds up code generation, the constraint shifts to review, then context & security, and finally validation.

Multiple studies point to the same conclusion

DORA 2025 asked nearly 5,000 technology professionals about AI-assisted development. Its conclusion: AI works as an amplifier, magnifying the strengths of strong organisations and the problems of struggling ones.

Faros AI measured 10,000 developers across 1,255 teams. On teams with heavy AI use, developers finished 21% more tasks and merged 98% more pull requests, while company-level delivery did not move: lead time, failure rate, and recovery time all stayed flat. The same data shows where the extra volume went - review time went up 91%, and pull request size went up 154%. Faros named the pattern the AI Productivity Paradox.

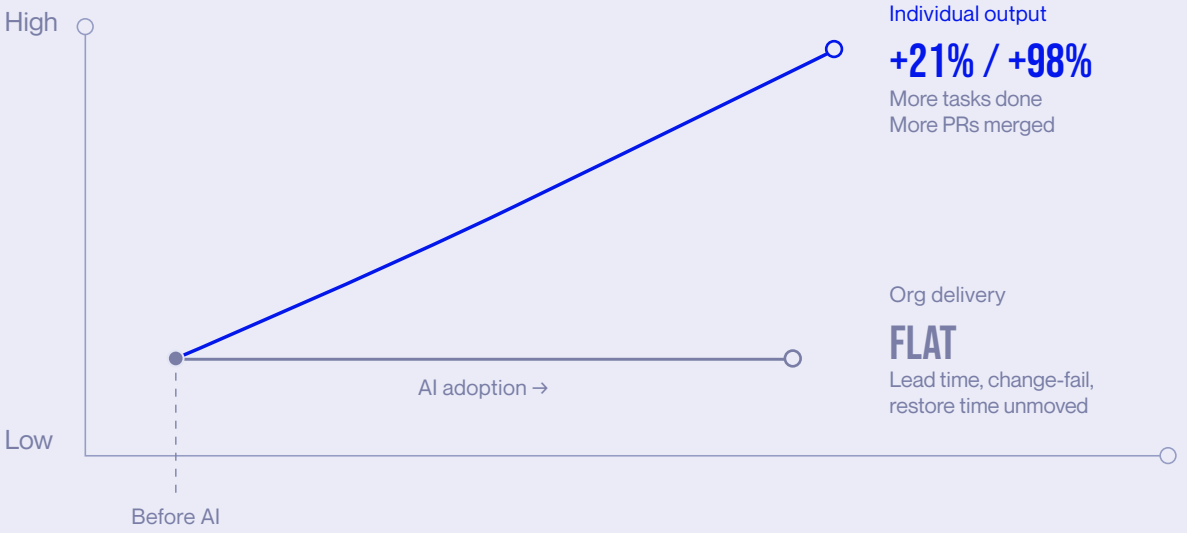
METR ran a randomised controlled study with experienced developers on code they knew well. With AI assistance they were around 19% slower, while estimating that AI had made them about 20% faster. The gap between how productive a team feels and how productive it is can be large.

MIT CISR studied 721 enterprises. Companies in the lower two stages of AI maturity performed below their industry's financial average, and companies in the upper two stages performed above it.

McKinsey found 88% of organisations using AI in at least one business function, while 1% call their AI strategy mature.

The scissors: output climbs, delivery doesn't move

What AI changes, and what it doesn't - Faros AI telemetry, ~10,000 developers, mid-2025.



See the find-and-change table in Part 2.

Figure 3. Individual output climbs while organisation-level delivery stays flat. Faros AI telemetry, ~10,000 developers, mid-2025.

We heard the same story from the engineers themselves. Before training 100 engineers at Odevo, we asked them what concerned them most about AI. These were their answers, in their own words:

- Losing the fun of the work.
- Losing the ability to solve problems.
- Coding worse things faster.
- Using AI without learning about it first.
- No longer understanding the code.
- Loss of pride in work.
- Getting replaced.
- Losing the human touch.
- Competitors using AI to outcompete us.
- Security and new attack vectors.

These concerns come from people experiencing the shift firsthand, and they point to the same conclusion as the data: the way we work has changed faster than the systems around it.

The Symptoms

Inside a team, the pattern shows up as a series of symptoms, and they tend to appear in a recognisable order.

The first thing that usually goes sideways is code review. PR throughput can triple within months while the reviewers' capacity to read carefully stays the same, so reviewers compensate by skimming, by trusting the tests, or by trusting the author. The tests themselves prove less than they used to, because AI-generated tests tend to check the code that was just written rather than the requirement the code was supposed to meet. The result is test coverage that rises while the team's actual confidence in the code falls.

The second symptom is code that the team finds hard to explain. Developers start approving pull requests they could not have written themselves, eventually the difficulty becomes visible, when the code has to change and nobody really knows how it works.

The third is context files turning into dumping grounds. AGENTS.md grows past the point where the model can make good use of it, and the team responds by adding more to it, which makes the agent's behaviour worse rather than better.

The fourth is security. An agent running with skip permissions holds the developer's credentials, the team's data, and network access at the same time - the combination Simon Willison calls the lethal trifecta. In our experience, teams reach this exposure earlier than they realise.

The fifth arrives once the team has solved some of the earlier problems: the bottleneck leaves engineering. Development becomes fast enough that the product turns into the slowest stage, because specifications that were detailed enough for human-speed work do not carry enough information for agents. Legacy systems show a related limit, since they cannot absorb fast change safely, which makes them the place where AI helps least.

The Ladder

If AI amplifies whatever is already in place, then the practical question for a leader is what it amplifies first, what it amplifies next, and in what order the problems will arrive. Across our engagements, the same bottlenecks have surfaced in roughly the same sequence, and we have organised that sequence into a ladder of nine stages. Each stage has a name, a description you can recognise your team in, and the bottleneck that blocks the way up. Timelines differ between companies, and some teams skip a stage, but the order has held well enough across enough teams to plan against.

Several existing frameworks describe parts of this landscape. Ours is designed to complement them, not replace them. Dan Shapiro's "Five Levels: from Spicy Autocomplete to the Dark Factory" describes how an individual developer progresses. DORA's AI Capabilities Model names seven capabilities that help AI amplify in the right direction. MIT CISR shows that an organisation's stage matters for financial outcomes. What our ladder describes is the journey of the organisation as a whole, and the bottleneck it meets at each step.

The ladder works at two levels. On a single codebase, it shows where one team stands and which bottleneck it meets next. Across an organisation, it shows how far the whole engineering function has moved, because an organisation only climbs as fast as its codebases align. So you assess codebase by codebase first, then read the organisation as the sum of those positions. Bringing every codebase onto the same rung is what a transformation is really working towards; once they align, the organisation can move up the ladder as one.

Each stage reveals the next constraint. Progress comes from solving them one by one. Each pill highlights the key constraint that emerges at that stage.

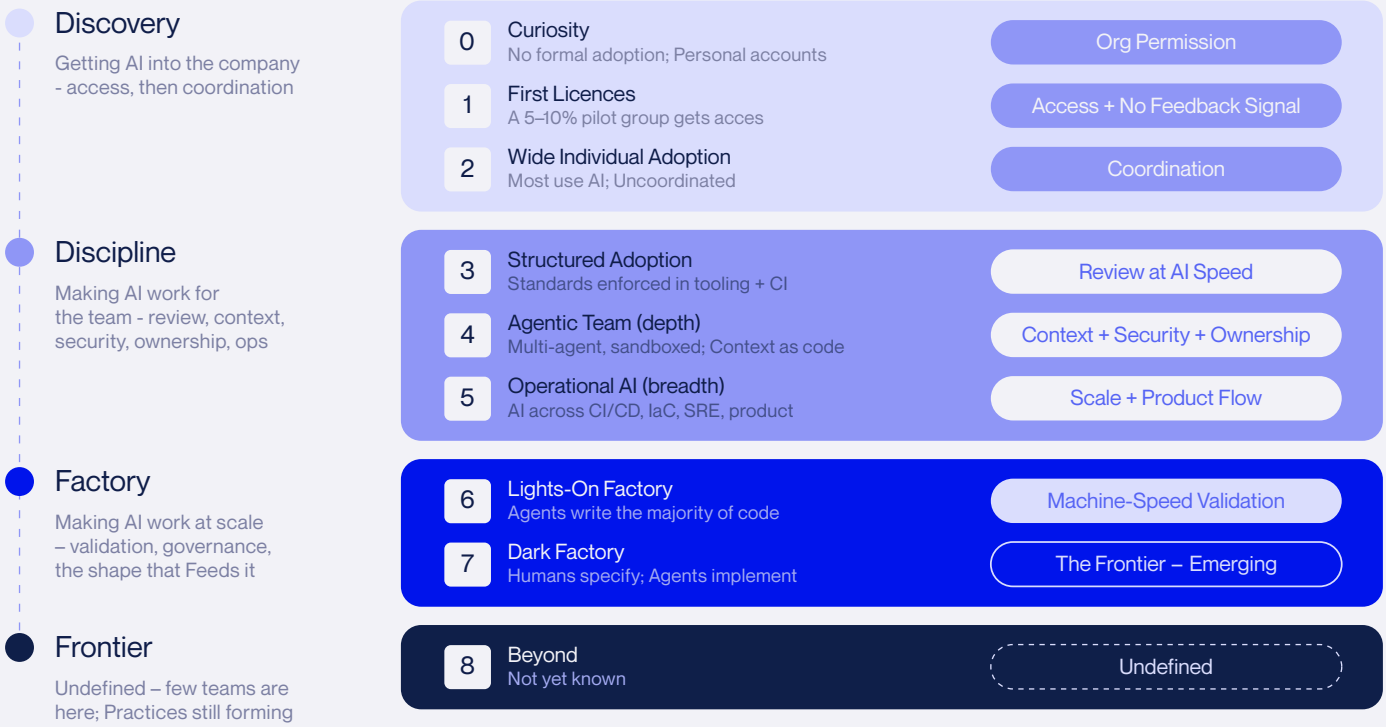


Figure 4. Most teams sit around stage 2. Illustrative field observation across re:cinq engagements

Download the complete AI Adoption Ladder Framework as a PDF

A detailed guide to every stage of the AI Adoption Ladder Framework

The four phases:

- Discovery *(stages 0–2)* – getting AI into the company at all. The first bottleneck is permission and access; once most people have access, the bottleneck becomes coordination.
- Discipline *(stages 3–5)* – making AI work reliably. The bottleneck moves through review, context, security, ownership, and operations.
- Factory *(stages 6–7)* – making AI work at scale. The bottleneck moves to validation and governance.
- Frontier *(8+)* – unknown so far. We haven't seen any teams go past the dark factory yet.

The nine stages:

Stage 0 – Curiosity

No formal adoption yet. Engineers experiment individually, and leadership is aware of it without acting on it. Most companies left this stage in 2023 or 2024. Bottleneck: organisational permission. The stage ends when leadership sees enough to fund a pilot.

Stage 1 – First licences

A pilot group, usually 5–10% of engineers, gets formal access. The most motivated engineers learn quickly, while the rest of the organisation has no reliable signal about whether any of it is working. Bottleneck: that missing signal. The stage ends when the pilot produces credible evidence on adoption, satisfaction, and output.

Stage 2 – Wide individual adoption

Most engineers have access and use AI at least sometimes, but everyone uses it their own way, with different tools, prompts, and conventions on the same codebase. Output rises, quality varies, and the dashboards look good. Bottleneck: coordination. The stage ends with shared standards, structured education, and a pilot team that redesigns its workflow first.

Stage 3 – Structured adoption

Standards exist and are enforced through tooling, linting, and CI. Education has run as a structured programme over several weeks, long enough for behaviour to actually change, and most developers have moved from autocomplete to agentic work. Bottleneck: review, because output now arrives faster than senior engineers can carefully read it. The stage ends when review has been redesigned and the test suite is trusted.

Stage 4 – Agentic team *(depth)*

Several agents per developer is normal, running in sandboxed environments, and context files, architecture decision records, and schemas are treated like production code. Bottleneck: context quality, security, and ownership. This is the stage where the lethal trifecta needs a real answer, and where hard-to-explain code accumulates if review and ownership rules lag behind. The stage ends with scoped agent permissions and clear ownership rules.

Stage 5 – Operational AI *(breadth)*

AI spreads beyond the IDE into CI/CD, code review automation, infrastructure as code, on-call, and security scanning, and beyond engineering into product, design, and sometimes finance. Bottleneck: scale, and product flow, because engineering stops being the slowest stage. The stage ends with factory infrastructure.

Stage 6 – Lights-on factory

Agents produce most of the code, and humans focus on review and approval. Stripe's Minions merge more than 1,300 AI-generated PRs a week, and Ramp's Inspect handles around half of the company's PRs. Bottleneck: validation at machine speed, because agents cannot reliably check their own work, so the validation pipeline sets the pace for everything.

Stage 7 – Dark factory

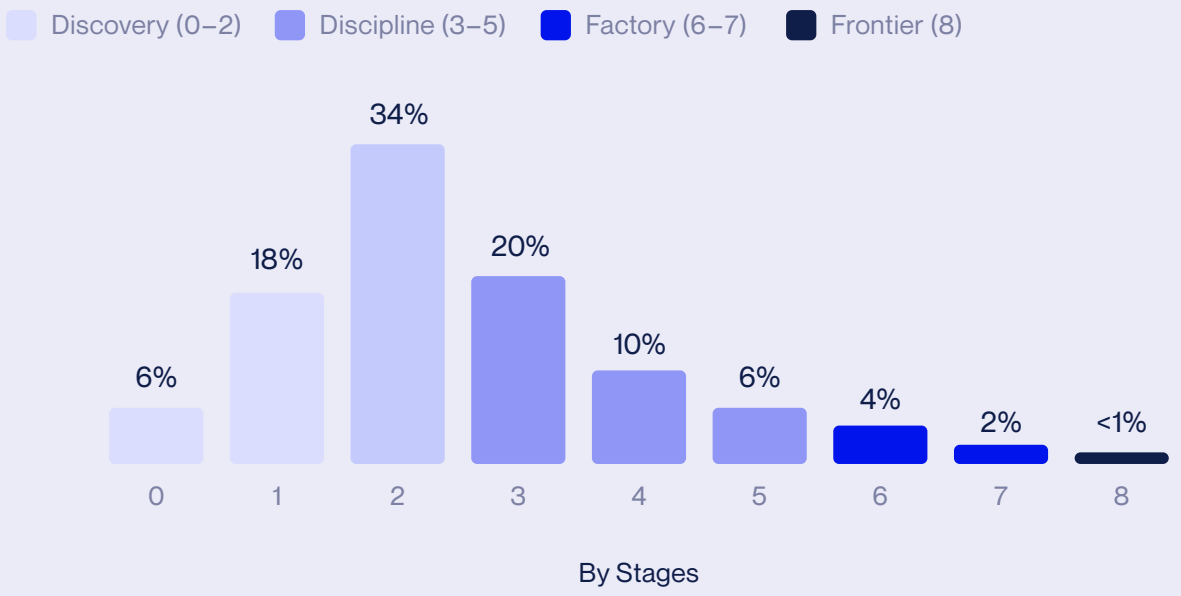
Humans specify, agents implement, and validation carries most of the weight that code review used to carry. StrongDM's manifesto states it as two rules: code must not be written by humans, and must not be reviewed by humans. They budget at least $1,000 of model tokens per engineer per day. OpenAI's Harness Engineering team works in a similar direction. We include these as evidence that the rung exists rather than as a recommendation; for most organisations today they are not a sensible target. Bottleneck: too few teams have reached this stage yet to say what they hit next.

Stage 8+ – Beyond

Unknown. Stage 7 itself may turn out to be temporary, so we keep 8+ on the map as a placeholder rather than a prediction.

Finding Your Stage

Illustrative field observation across re:cinq engagements.



The dashboards look healthy at Stage 2, which is exactly why the plateau stays invisible from the top.

Figure 5. Most teams sit around Stage 2.

Most companies we meet sit around stage 2. Output is up, the dashboards look good, and the bottleneck has moved downstream to places leadership rarely looks, which is why the plateau is easy to miss from the top.

To be clear, no published research measures these stages directly. Our distribution comes from field observations. However, it aligns well with the available evidence. McKinsey reports that only 1% of organisations have a mature AI strategy, while Faros shows that delivery performance remains largely flat. Both support the view that most organisations are still operating at Stage 2.

Where is your team?

Answer *Yes* or *Not Yet* for each of the eight questions below. The count places you in one of five groups; treat it as an approximation rather than an exact position.

1. Most of your developers use AI daily, not just the eager 5–10% who got there first.

2. You can describe, in writing, how AI-generated code is reviewed differently from human-written code, and the rules live in tooling, not only on paper.

3. Your agentic tools have scoped credentials and run in sandboxes. They do not run as the developer, with access to production secrets.

4. You have a structured context file (AGENTS.md or equivalent) per repository, containing only what the model does not already know: invariants, exceptions, conventions that go against popular patterns.

5. Your CI is fast and trusted enough to work as a real-time feedback loop instead of a late-stage gate.

6. Your test suite catches behaviour regressions reliably enough that AI-generated code adds no hidden review burden.

7. You measure delivery throughput: lead time, change failure rate, time to restore, next to any AI productivity claim. PR counts and token usage are not your success metrics.

8. Agents produce a real share of merged code, in isolated environments, behind validation pipelines, and you have written down what ships automatically and what needs human review.

Your score:

- 0 – 2 Yes – Discovery *(Stages 0–2)*
  Individual productivity without coordination. First investment: an assessment, then a pilot team, then foundations.
- 3 – 4 Yes– Discipline Entry *(Stages 3–4)*
  Structured adoption is starting, and review and context are the active bottlenecks. First investment: structured education, review redesign, AGENTS.md discipline.
- 5 – 6 Yes – Discipline Mature *(Stages 4 – 5)*
  Operating at an agentic level, with security, context, ownership, and operational breadth as the bottlenecks. First investment: the security model, scoped permissions, ownership transfer.
- 7 Yes – Approaching the Factory *(Stages 5 – 6)*
  The bottleneck is shifting to product flow and validation at scale. First investment: product-engineering integration, factory infrastructure.
- 8 Yes – Factory *(Stages 6 – 7)*
  Operating at system level, where the bottleneck is validation at machine speed. First investment: software factory patterns, an internal developer platform, spec discipline.

Printable scorecard

		Not Yet	Yes
01	Most developers use AI daily — not just the eager few	☐	☐
02	Written, tooling-enforced rules for AI-code review	☐	☐
03	Agentic tools have scoped credentials and run sandboxed	☐	☐
04	A structured AGENTS.md per repo – only non-default rules	☐	☐
05	CI is fast, trusted, used as a real-time feedback loop	☐	☐
06	Test suite catches regressions; AI code adds no review debt	☐	☐
07	You measure delivery throughput, not PR or token counts	☐	☐
08	Agents merge real code behind validation pipelines	☐	☐

0 – 2 Yes	3 – 4 Yes	5 – 6 Yes	7 Yes	8 Yes
Discovery (0–2)	Discipline Entry (3–4)	Discipline Mature (4–5)	Approaching Factory (5–6)	Factory (6–7)

Figure 6. The eight-question diagnostic as a printable scorecard.

Where the companies we Know sit *(anonymised engagements and public references)*.

- A European platform-engineering-mature company at stage 2: "It's going in all directions, and I'm not sure how we fit."
- A British engineering company recently taken private, at stages 1–4 depending on where you look: modern code at 3–4, the legacy monolith at 1.
- A European SaaS with a strong agentic product team at stages 3–5 by team. Odevo at stages 3–5.
- A European product company at stage 6, with its own cloud-isolated environment system.
- Public references: Stripe and Ramp at stage 6; StrongDM, OpenAI's harness team, and Glowforge at stage 7.
- A European product company at stage 6, with its own cloud-isolated environment system.
- Public references: Stripe and Ramp at stage 6; StrongDM, OpenAI's harness team, and Glowforge at stage 7.

What the Ladder Doesn't Cover

The ladder describes the typical journey of an engineering organisation, but like any model, it simplifies reality. It's worth calling out six important limitations.

Different teams within the same company are often at different stages. The British engineering company from the previous page is a good example: one modern product operates at Stages 3–4, while its legacy monolith remains at Stage 1. The ladder applies to a team working on a specific codebase, not an entire organisation. That's why a more useful question is, "What stage is this team at, on this codebase?" rather than, "What stage is your company at?"

Progress is not always forward. A company can move backwards after a leadership change, an acquisition, or a technology shift, and some teams skip stages entirely; a greenfield team built around AI from day one can land at stage 4 without ever doing 2–3. The ladder describes the typical sequence, and it does not prescribe a mandatory path.

The ladder is about engineering. Product, design, marketing, finance, legal, and support each follow their own trajectory. The engineering ladder reaches the point where product becomes the bottleneck, while the company-wide journey goes further, and we treat that as a separate track.

Model improvements move teams forward on their own. A new model release can push a team half a stage overnight, without any organisational change. The ladder describes the choices an organisation makes, while the model sets the limit of what is possible at each rung.

The talent pipeline. As teams reach the factory rungs, the entry-level work that used to produce senior engineers starts to disappear. If a stage-6 team consists of four senior specification engineers working with agent fleets, it is not clear where the next generation of seniors will come from. We plan to return to this in a later paper in this series.

The work itself changes shape. Working with three agents at once is closer to engineering management than to engineering, and reviewing AI-generated code is a different task from writing code. The natural pauses the work used to provide, such as slow compiles and thinking time between keystrokes, are mostly gone. The ladder measures what a team does at each stage, and it does not yet capture what working at each stage feels like or how managers should adjust to keep it sustainable. This is another topic we plan to cover in the series.

No company sits on a single rung. Each team, on each codebase, has its own.

How Teams Climb

Buying more licences does not move a team up the ladder; we have watched several companies try. The sequence below is the one that worked for the customers we took from stages 1–3 to 4–5, drawn from Odevo and engagements like it. Teams skip steps, run two in parallel, and return to earlier ones when new bottlenecks surface, but this is the order that has succeeded most often.

1. Start with an assessment

Establish where each team actually is, codebase by codebase, because different parts of the organisation will be at different rungs and the plan should be built on the real picture.

2. Form a small, mixed pilot team

Three to five people, including at least one skeptic. The motivated adopters will succeed anyway; the signal you need is whether the skeptic moves.

3. Fix foundations before features

If test coverage is poor, use AI to fix the coverage before pointing it at feature work, and if CI is unreliable, repair that first. "Often we recommend: if you don't have good test coverage, then use AI to create this coverage. Don't go and create features." - Pini Reznik

4. Run education as a structured programme

Boot-camp format, hands-on, spread over several weeks so new habits can settle. At Odevo this meant seven modules of four hours each, hands-on exercises, a peer-support Slack channel, and weekly check-ins over several months, across all five cohorts.

5. Make tool and guardrail decisions before expanding licences.

Choose the agentic tools, the security model, the context standards, and the review process first; expanding access before deciding produces eighty developers doing eighty different things on the same codebase.

6. Use train-the-trainer to build internal capability

Pick the engineers who absorbed the training best, teach them to deliver it, and hand the programme over, so that the capability stays when the external partner steps away

7. Move into the agentic stages deliberately

Scoped credentials, sandboxed environments, and structured context kept short enough to stay useful. This is also the right moment to deal with the lethal trifecta properly.

8. Invest in platform and factory patterns

An internal developer platform, specialised agents, and spec-driven development. The durable advantage of AI adoption is built at this stage, and it depends on the previous seven steps having happened first.

Where to Go From Here

No one knows yet where AI Native development will ultimately lead. The tools, the infrastructure, and the working patterns around them are all still changing, and the factory patterns are still being established. What we can say, from our customers and from our own engineering work, is that the teams making durable progress are the ones fixing one bottleneck at a time, in the order their stage demands.

re:cinq runs an AI Coding Adoption Reality Check – a two-to-four-week assessment that locates your teams on the ladder, identifies the active bottleneck, and tells you what to fix first. It covers developer workflow, review and testing maturity, context infrastructure, AI usage patterns, security boundaries, infrastructure as code, product flow, and measurement. The output is a written assessment and a roadmap built around the stage you are at.

> *re:cinq made me rethink how to use AI in scaled organisations*
>
> BJÖRN BRAUEL
> CTO, Odevo

This paper is the first in a series we are writing through 2026. Software Factories at Enterprise Scale covers the platform architecture behind stages 6–7, and papers on education and behaviour change *(stages 1–4)* and on code review, testing, and validation *(stages 3–5)* follow.

To find out where your organisation sits on the ladder, write to us at info@re-cinq.com with the subject line **AI Coding Adoption Reality Check.**

re:cinq, JJuly 2026.

References

Research and Data

DORA, State of AI-assisted Software Development 2025
https://dora.dev/dora-report-2025/

Faros AI, The AI Productivity Paradox
https://www.faros.ai/blog/ai-software-engineering

METR, Measuring the impact of early-2025 AI on experienced open-source developer productivity
https://metr.org/blog/2025-07-10-early-2025-ai-experienced-os-dev-study/

MIT CISR, Building Enterprise AI Maturity (Weill, Woerner & Sebastian, Dec 2024)
https://cisr.mit.edu/publication/2024_1201_EnterpriseAIMaturityModel_WeillWoernerSebastian

McKinsey, The State of AI in 2025
https://www.mckinsey.com/capabilities/quantumblack/our-insights/the-state-of-ai

Security Guidance

Simon Willison, The Lethal Trifecta
https://simonwillison.net/2025/Jun/16/the-lethal-trifecta/

Software Factories and the Frontier

Stripe Engineering, Minions
https://stripe.dev/blog/minions-stripes-one-shot-end-to-end-coding-agents

OpenAI, Harness Engineering
https://openai.com/index/harness-engineering/

Dan Shapiro, The Five Levels: from Spicy Autocomplete to the Dark Factory
https://www.danshapiro.com/blog/2026/01/the-five-levels-from-spicy-autocomplete-to-the-software-factory/

StrongDM's software factory *(manifesto documented)* Simon Willison, How StrongDM's AI team build serious software without even looking at the code
https://simonwillison.net/2026/Feb/7/software-factory/

re:cinq Case Reference

Daniel Jones & Tomasz Maj, "More software, faster – Odevo's AI Native transformation,"
AI Native DevCon *(London, June 2026)*
https://www.youtube.com/watch?v=mB74LGAmmV0

Building one, or competing with someone who is?

We help enterprise engineering organisations design and build software factories – the architecture, governance, and validation behind agentic development at scale. Wherever you are on that path, we should talk.

REACH OUT > **hello@re-cinq.com**

re:cinq

re:cinq is an engineering consultancy helping medium and large organisations adopt AI across software development: building software factories, training teams, and modernising legacy systems.