8 west CONSULTING

www.8west.ie | info@8west.ie

Measuring AI Productivity in Software Delivery

Why the Question Is the Problem, and the Questions Leaders Should Be Asking Instead

Every leadership team in software delivery is asking some version of the same question: what productivity gains are we getting from AI, and what should we expect over the next twelve months? These are reasonable questions. They are also, increasingly, the wrong ones. The confident single-number answers they produce are doing more harm than the uncertainty they were meant to resolve. This paper sets out why, drawing on the current research literature and on anonymised findings from our own enterprise engineering work in regulated, large-scale environments.

Authors:

Mark Rainford, VP of Commercial Business Innovation, 8 West Consulting

Colum Sisk, VP of Health InsurTech, 8 West Consulting

Contents

1 Executive Summary

The dominant questions leaders ask about AI in software delivery, namely what gains we are getting and what we should forecast, are framed around rate of output. That framing tends to produce answers that mislead more than they inform. The central points of this paper are as follows.

- The metrics were always proxies. Velocity, story points, code volume and time-to-pull-request were proxies for human cognitive effort. When an AI agent sizes, writes and checks the work, those proxies stop pointing at what they used to measure.
- The evidence is not merely uncertain; it is openly contradictory. Rigorous studies range from a 19% slowdown for experienced developers working in mature codebases to a 56% speed-up on greenfield tasks. That spread is not noise to be averaged away. It is the finding. Achievable gain is overwhelmingly a function of context.
- Acceleration moves the bottleneck rather than removing it. As generation speeds up, the constraints shift elsewhere: intake and requirements gathering, code review and verification, and getting code safely to production. System-level delivery stability can fall even as individual output rises.
- Domain expertise is the validation layer. The depth of domain knowledge available to check the output is what separates real acceleration from plausible-looking work that fails silently. AI makes that expertise more valuable, not less.
- Quality is a time-delayed variable. AI-generated code can look correct for longer before failing; several maintainability signals are trending the wrong way; and AI-assisted developers have been shown to produce less secure code while feeling more confident about it. This makes validation critical, and validation requires domain expertise. Knowing whether what has been produced is actually correct, secure, and maintainable is not something AI can confirm for itself.
- The more important shift is not speed but feasibility. Work that was always valuable but never economically justifiable becomes viable, from surveying an entire legacy estate to building test harnesses at scale and replacing brittle manual processes with proper tooling. This is Jevons' paradox applied to engineering effort.
- The practical conclusion: stop asking for a single productivity number. Ask instead about feasibility, quality and direction, and about where domain expertise can be most effectively applied.

2 Introduction: A Question Worth Reframing

Start with the gap that defines the current moment. The largest annual survey of professional developers, with some 49,000 respondents across 166 countries, found that 84% reported using or planning to use AI tools, up from 76% a year earlier. Yet only around a third say they trust the accuracy of the output, while close to half actively distrust it and a mere 3% express high trust [1]. Adoption is rising while confidence falls, and, tellingly, the most experienced developers are the most sceptical of all. Two-thirds report wrestling with output that is "almost right," and a substantial share now say debugging AI-generated code takes longer than writing it themselves [1].

That gap is the real context for the productivity question. Leaders are being asked to quantify the return on a tool their best engineers use daily but trust least. The instinct is to reach for a number, a percentage gain or a twelve-month forecast, that can be put in front of a board. This paper does

not argue against measurement. It argues for measuring the right things, and offers a more useful frame for that conversation.

A note on the evidence that follows. Alongside the public research, we draw on anonymised findings from our own engineering engagements in regulated, enterprise-scale environments. Client names, systems and projects have been removed; the figures and patterns described are real.

3 What the Numbers Are Actually Measuring

Measuring AI productivity with traditional delivery metrics is like measuring a car in horse-lengths per hour. You will get a number. It will not tell you what you want to know.

Story points were always a proxy for human cognitive effort. Velocity, time-to-pull-request and code volume per story were proxies for the same thing. When an AI agent is sizing the stories, generating the code and surfacing the bugs, the proxy has shifted: it is no longer pointing at the quantity it was built to estimate. Velocity rises. Code volume per story increases. Time-to-pull-request falls. None of that establishes that more value is being delivered.

This is not a new problem; AI has merely made it acute. The settled academic answer to “how do we measure developer productivity” is that you cannot reduce it to one dimension. The SPACE framework, which covers Satisfaction, Performance, Activity, Communication and Efficiency, was published precisely because single-metric approaches consistently mislead: Activity is the most visible and easiest dimension to count, but volume of activity says nothing about whether the work mattered [2]. Lines of code reward verbosity; commit counts reward triviality. Under AI, the Activity dimension becomes actively misleading, because volume can now be produced faster than it can be understood, and the metric has no way to distinguish authored-and-reasoned code from generated-and-unreviewed code.

There is a more general law at work, long predating AI: when a measure becomes a target, it ceases to be a good measure [3]. Set a velocity target and AI will help a team hit it without telling anyone whether the underlying delivery improved. The measure moves; the thing it once stood for does not necessarily move with it.

A concrete illustration from our own work makes the point. In one engagement, an internal productivity claim circulated of a roughly fifty-fold improvement in database research and analysis: tasks estimated at two to three weeks were completed in hours, and a body of performance testing estimated at several hundred person-days was completed in days. The figure was real, in the sense that the work genuinely was completed in that time. But it rested on an invented baseline. The “two to three weeks” was a manual estimate of how long the task would have taken had anyone attempted it that way. For much of the work, no one ever had, because it had never been economically justifiable to attempt. The honest internal note attached to the claim said exactly that: this was novel work, with no pre-AI comparator to measure against. A fifty-times number with no genuine denominator is not a productivity metric. It is a signal that something more interesting is happening, a point we return to below.

4 The Evidence Is Contradictory by Design

If the metrics are unreliable, perhaps the controlled studies can supply the number leaders want. They cannot, and the reason is instructive.

In 2025, METR ran a randomised controlled trial, the design used in clinical drug trials, with sixteen experienced open-source developers working on mature repositories they knew well, averaging five years of prior experience on the codebase. Each of 246 real tasks was randomly assigned to allow or disallow AI tooling. The developers expected AI to make them about 24% faster. After the study, they estimated it had made them roughly 20% faster. In fact, allowing AI made them 19% slower [4]. The result held in precisely the setting many enterprises care about most: large, complex, high-standard codebases where the cost of a subtle error is high.

Set against that, an equally cited controlled experiment points the other way. When developers were asked to implement an HTTP server in JavaScript from scratch, those with an AI pair-programmer completed the task 55.8% faster than those without [5]. Larger field studies at major firms found more modest but real gains, with developers completing in the order of 13% to 22% more pull requests per week with AI assistance [6]. A six-week controlled study at a large bank reported tasks completed around 42% faster, with the least experienced benefiting most [7].

These findings are routinely presented as a contradiction to be resolved, as though one set of researchers must be wrong. They are not in contradiction. They are measurements taken at different points on a single spectrum. The greenfield HTTP server is unconstrained: no legacy, no integration surface, no downstream dependencies, a clean problem with a well-trodden solution. The mature repository is the opposite: deep context, a high quality bar, surgical rather than additive change. AI helps most where constraint is lowest, and helps least, or hurts, where constraint is highest. The same tool produces a 56% speed-up and a 19% slowdown depending only on where it is pointed.

This is why the single-number question fails. There is no organisation-wide AI productivity figure, because there is no single kind of work. The honest answer to “what gain are we getting” is another question: getting it on what?

Study	Setting	Task type	Effect on speed
METR RCT (2025) [4]	Mature, complex repositories	Real issues; surgical change	**~19% slower**
GitHub experiment (2023) [5]	Greenfield, no dependencies	Build an HTTP server from scratch	**~56% faster**
Field studies, major firms (2024) [6]	Mixed enterprise codebases	Day-to-day delivery tasks	**~13–22% more PRs/week**
Large-bank study (2024) [7]	Algorithmic tasks, training given	Self-contained problems	**~42% faster**

Table 1. The same class of tooling, measured across different environments, spans a roughly 75-point range. The spread is the result, not an artefact to be averaged away.

5 Where the Work Moves: Bottlenecks and the Perception Gap

Even where AI clearly accelerates code generation, it does not follow that the system delivers faster. Acceleration tends to relocate the constraint rather than remove it.

When generation speeds up, review becomes the bottleneck, and reviewing AI-generated code is its own discipline, one not every team has developed. Junior developers ship more features; senior developers slow down to catch subtle errors before they reach production. Individual and team productivity begin to diverge in ways existing reporting does not capture.

The largest industry dataset on this bears it out. Across two successive years, Google's DORA research found that AI adoption raised individual productivity, flow and job satisfaction while, at the same time, being associated with reduced delivery stability and throughput at the system level, to the point that a 25% increase in AI adoption was associated with an estimated fall of around 7% in delivery stability [8]. The mechanism is unglamorous: AI encourages larger change sets, and large batches are harder to review, test and recover from. The 2024 research added a rework-rate measure specifically to capture the unplanned fixes this produces; the 2025 research found AI adoption was not resolving instability but tracking with more of it [9].

There is a second, subtler issue: people are poor judges of their own AI-assisted productivity. In the METR trial, developers believed AI had sped them up even after it had measurably slowed them down [4]. The same pattern appears in quality: in a Stanford user study, developers with access to an AI assistant wrote less secure code than those without, and were simultaneously more likely to believe their code was secure [10]. The tool does not only change the output; it changes the operator's confidence in the output, and the two move in opposite directions. Self-reported productivity, which is the basis of most internal AI assessments, is exactly the measure the evidence says to distrust.

Our own delivery data illustrates where the real value tends to sit when the work is done well, and it is not raw speed. In one anonymised case, a feature was taken from requirement to implementation-ready in roughly a day, against a conventional estimate of several sprints and a team of five to seven. But the figure that mattered was not the calendar compression. It was that six or more handoffs, each of which normally introduces a wait and a loss of context, were reduced to zero, a partner team was unblocked on day one rather than after implementation, and a production-grade defect was caught before any production code was written. The speed was the headline; the eliminated handoffs and the early-surfaced risk were the value. A velocity report captures the first and is blind to the second.

6 Quality Is a Time-Delayed Variable

Defect rates under AI can move in either direction, and the problematic direction is often delayed. AI-generated code can appear correct for longer than human-written code before an issue surfaces, which means the metric that looked good last quarter may be concealing something that will matter next quarter.

The maintainability signals are not reassuring. Analysing 211 million changed lines of code from 2020 to 2024, one large study found that the share of changed lines associated with refactoring fell from around a quarter in 2021 to under a tenth in 2024, while copy-pasted code rose over the same period and the frequency of duplicated blocks increased roughly eight-fold; 2024 was the first year in which copy-pasted lines exceeded "moved" lines, the signature of consolidation and reuse giving way to duplication [11]. Code churn, lines reverted or rewritten within two weeks of being authored, rose in parallel. The interpretation requires care, because this analysis infers AI involvement from the timing and composition of changes rather than observing it directly, and at least one repository-level study of self-declared AI usage found more nuanced patterns [12]. But the direction of travel is consistent with what one would expect from a tool that makes inserting new code trivial and proposing reuse comparatively hard.

The security dimension matters most in exactly the environments we work in. Beyond the Stanford finding that AI assistance correlated with less secure code and greater misplaced confidence [10], industry surveys report developers spending more time debugging AI-generated code and resolving vulnerabilities it introduces [13]. The usual caveat applies in both directions: the Stanford study used an earlier-generation model, and tooling has advanced since; equally, capability advances do not remove the verification burden, they relocate it.

A single anonymised example from our own work shows why traditional verification is not sufficient on its own. During performance profiling of a new query path, a data-type mismatch on a query parameter was found to be forcing an implicit conversion at the database layer. Corrected, the query dropped from 464 milliseconds to 11, a roughly forty-fold improvement. The point is not the speed-up; it is what would have caught it. Unit tests, running on mocked data, would not. Integration tests, running on small datasets, would not. Code review would not, because no reviewer routinely checks a parameter's declared type against a column's. Functional QA would not, because the code was functionally correct. The only thing that surfaced it was validation at production scale, a step that, in this instance, was a natural part of the agent-assisted workflow rather than an afterthought. AI did not introduce the defect and did not, on its own, catch it. What caught it was a workflow that pushed verification to where the defect actually lived, operated by someone who knew what to look for.

7 The Real Shift Is Feasibility, Not Speed

The productivity conversation in software delivery is not new. It is the latest in a long sequence of baseline shifts. C++ raised the floor above Assembly; Java and .NET raised it again; libraries, frameworks and IDEs moved the level of abstraction upward with each generation, and what counted as a reasonable day's output moved with them. AI looks, at first, like another such shift.

But the more consequential change is not speed. It is feasibility, and here the relevant idea is more than a century old. In 1865, the economist William Stanley Jevons observed that as steam engines became more efficient in their use of coal, total coal consumption did not fall but rose: making a resource cheaper to use expanded the range of things it was worth using it for [14]. The principle has been rediscovered for AI, most visibly when Microsoft's chief executive, Satya Nadella, invoked it in early 2025 to argue that as AI becomes cheaper and more capable, demand for it will not shrink but expand [15]. Applied to engineering effort, it reframes the entire productivity question. When the cost of doing a piece of work collapses, the binding question is no longer "how much faster is the work we already do," but "what work is now worth doing that never was before."

This is where our fifty-times figure from earlier resolves. The reason there was no genuine baseline is that the work had never been economically justifiable to attempt by hand. Consider a large, complex relational database estate of the kind common in regulated insurance and health technology. Historically, performance analysis meant targeting the obvious offenders, such as the worst few stored procedures and the queries causing the most visible pain. Reviewing thousands of procedures for marginal inefficiencies was not avoided because it lacked value; it was avoided because the effort could never be justified. The work sat in the "good idea, never quite worth it" pile. AI changes that calculation. Marginal gains across an entire estate become findable. Performance test harnesses can be built at scale. The output is not merely a faster version of existing work; it is access to work that was previously off the table, and it converts into lower

infrastructure cost, better operational performance, and a codebase that has finally been surveyed properly rather than patched at its edges.

The same logic applies to processes, not just code. In one environment, the overwhelming majority of a complex configuration workload was being delivered by hand-written scripts run directly into production, a slow, review-heavy, error-prone path that persisted because the user-facing tooling that would have replaced it had never been worth building. The screens were unfit for purpose; bulk operations barely existed; staff had to be database experts as well as domain specialists simply to function. None of this was a mystery to the people involved. It endured because building proper tooling had always cost more than living with the workaround. Lower the cost of building that tooling, and the calculus inverts: a body of internal enablement work that was permanently below the line becomes not just viable but obviously worth doing.

The most striking version is sheer scale. One legacy adjudication component still in production was first built around two decades ago and now processes in the order of two hundred thousand transactions a day, peaking well above that, against a database measured in terabytes and tens of billions of records. A core operation took just over a second per item, ran with no performance monitoring, and carried years of accumulated unmanaged data alongside orphaned work items consuming cycles on a fixed interval. On analysis, it was reducible to under fifty milliseconds. What had changed was not the difficulty of the fix but the feasibility of finding it: the estate could finally be surveyed, at production scale, without the survey itself being uneconomic. That is a different category of value from anything a velocity report was designed to capture.

8 Why the Twelve-Month Forecast Falls Apart

The forecast version of the productivity question, what gains should we expect over the next year, fails for the same underlying reason, compounded by a moving target.

The achievable gain in any given environment depends on a specific set of variables: existing architecture; regulatory context; how fault-tolerant the application is; codebase size and how well it is understood; integration footprint; data quality; and whether the typical change is additive or surgical. These interact, and they differ sharply from one system to the next. A single forecast across a portfolio of such systems is an average over things that have almost nothing in common.

A more useful frame is to view the estate as a portfolio with distinct tiers, each with its own realistic range.

Tier	Typical environment	Realistic effect	What to measure
The Core	Legacy, regulated, deeply integrated with core business data; surgical change; high cost of error	Modest, of the order of 5–20%, and only with investment and tolerance for instability	Defect direction; stability; whether previously infeasible analysis is now done
The Edge	Greenfield or greenfield-adjacent: new features, clean data, few downstream dependencies	Large, 50–100%+; if you are not seeing it, ask why not	Throughput; cycle time; speed to validated
Below the Line	Internal tooling, harnesses, automation and analysis never viable to build by hand	Effectively unbounded, the comparator is “not done at all”	Risk avoided; cost removed; work newly surfaced

Table 2. A portfolio view of achievable gain. The right question is not “how much,” but “which tier, and what should we therefore measure.”

Two worked cases mark the range. A legacy stack in a regulated industry, deeply integrated with core business data, is genuinely difficult to accelerate; gains in the 10–20% region are achievable but demand investment and a tolerance for periods of instability, and anyone quoting 50–100% in that context is not being straight. At the other end, a stable internal dashboard, a bolt-on feature on a web portal, or a greenfield tool with clean data and no downstream dependencies may find 50–100% an underestimate, and there the better question is why the gain is not higher still.

Layered on top is the fact that the tools themselves shift quarterly. What is true of AI tooling today has a real chance of looking outdated within months, in either direction. The researchers behind the most rigorous slowdown finding were explicit that their result was a snapshot of one moment in capability, not a constant, and when they attempted to repeat the measurement months later with newer tooling, they found the signal had become unreliable enough that they changed the experimental design [16]. A twelve-month number anchors expectations to a target that is moving underneath them, and leaves teams either disappointed or falsely reassured.

9 Domain Expertise Is the Validation Layer

Running through all of the above is a single dependency that determines whether AI acceleration is real or illusory: the depth of domain knowledge available to validate what the AI produces. The agent amplifies expertise; it does not substitute for it.

The contrast is sharpest when set out directly. In the anonymised feature work described earlier, the gains were realisable only because the operator carried context the agent did not: that a particular identifier column was a non-Unicode type, so a parameter had to match it or silently force a costly conversion; that a particular scoping value mattered for correctness; that a safe, production-representative schema existed against which to validate without touching sensitive data; that the platform imposed specific payload limits; and that a partner team needed an interface contract before it could begin. None of that is in the model. It is in the people who have worked the system.

With domain expertise	Without it, the “AI-capable contractor” risk
Matches parameter types to the schema; the costly conversion never ships	Accepts a plausible default type; a latent performance defect reaches production
Validates at production scale against a safe schema	Does not know such a schema exists; output is untested at scale
Optimises the operations that actually affect users	Optimises the wrong procedures; no user-visible improvement
Knows a partner needs a contract first; unblocks them on day one	Produces code in isolation; integration stalls later
Reads the output critically and catches what is merely plausible	Produces plausible-looking code that fails silently in production

Table 3. The same agent, two operators. The difference is not tooling fluency; it is whether the operator can tell correct from nearly-correct.

This is also what the research predicts. AI struggled most in METR’s mature, high-context repositories, the environments where deep comprehension is the scarce input. It helped most on

the greenfield task where little context was required [4][5]. The Stanford result is the same warning from the other side: without the expertise to tell secure from insecure, the operator not only ships the flaw but grows more confident while doing so [10]. An AI-capable contractor can operate the agent. What they cannot do, without the domain knowledge, is validate the output, and in a regulated, enterprise-scale system, the validation is the job. For organisations whose engineers genuinely know a complex domain, this is not a defensive observation; it is the strategic point. Domain comprehension is the part of the work that AI makes more valuable, not less.

10 Better Questions for Leadership

If the standard questions generate noise rather than signal, the more useful move is to ask what you are actually trying to learn.

For engineering teams. Are defects trending in the right direction? Is the team tackling work that was off the table six months ago? Is review capability keeping pace with generation capability?

For delivery. Where can you realistically target step-change improvement, and what specifically is blocking it today? Which parts of the estate are greenfield-adjacent, and which carry the full weight of legacy constraint?

For architecture and infrastructure. What work has become economically viable that was not before? Where are the marginal gains that were previously impossible to justify, and what would it mean operationally to capture them?

For leadership. Recognise that the single-number target is the one most likely to produce a frustrated team and a board that concludes either that AI is overhyped or that the organisation is moving too slowly. Neither conclusion is useful, and both are artefacts of the wrong question. The questions that stay useful regardless of what the models do next quarter are the ones focused on feasibility, quality and direction, not rate of output.

11 Conclusion: What Has Actually Changed

The productivity framing implies that AI is an accelerant applied to the work we already do. That is part of the story. The more important part is that the barrier to doing the work properly has come down.

Deeper analysis that was previously too expensive to commission. Technical debt that never reached the top of the prioritisation list. Consistency checks across a surface area too large to inspect by hand. Quality gates teams always wanted but could never afford to staff. That work was always valuable. It simply was not always feasible. When AI makes it feasible, the value shows up in places that story points and velocity reports were never designed to capture: reduced infrastructure spend, risk avoided, client relationships preserved, codebases finally in the condition they should have been in years ago.

That is the shift worth tracking, and it cannot be tracked with a single number. It requires different questions, and the discipline to keep asking them as the tools keep changing.

12 Where to Start

None of this argues against measuring AI's impact. It argues for measuring the right things, and that implies a different first move. Before committing to a forecast, map the estate: which systems

are Core, which are Edge, and which hold work that was never feasible before. For each, identify the binding constraint and the specific points where AI changes the economics enough to matter. That map, not a single productivity figure, is what tells a leadership team where to invest and what to expect from the investment.

This is the work 8 West does with engineering teams in regulated, enterprise-scale environments: a scoped, evidence-led reading of feasibility, quality and direction, and a sequenced path that captures value where it genuinely exists rather than where a velocity report suggests it should. The organisations that get the most from AI will not be the ones reporting the highest velocity. They will be the ones asking the better questions, and acting on the answers.

13 References

[1] Stack Overflow. “2025 Developer Survey.” 2025. survey.stackoverflow.co/2025

[2] Forsgren, N., Storey, M-A., Maddila, C., Zimmermann, T., Houck, B., and Butler, J. “The SPACE of Developer Productivity.” ACM Queue, 2021.

[3] Goodhart, C. (1975); popularised formulation by Strathern, M. (1997): “when a measure becomes a target, it ceases to be a good measure.”

[4] Becker, J., Rush, N., Barnes, E., and Rein, D. “Measuring the Impact of Early-2025 AI on Experienced Open-Source Developer Productivity.” METR, 2025. arXiv:2507.09089. metr.org

[5] Peng, S., Kalliamvakou, E., Cihon, P., and Demirer, M. “The Impact of AI on Developer Productivity: Evidence from GitHub Copilot.” 2023. arXiv:2302.06590. (Microsoft Research.)

[6] MIT Exploration of Generative AI. “The Productivity Effects of Generative AI: Evidence from a Field Experiment with GitHub Copilot.” 2024. (Field experiments across c. 1,974 developers at Microsoft and Accenture.)

[7] Chatterjee, et al. Six-week controlled experiment on AI-assisted development at ANZ Bank, 2024, as reported in the software-engineering literature.

[8] Google Cloud / DORA. “Accelerate State of DevOps Report 2024.” DORA.dev

[9] Google Cloud / DORA. “State of AI-assisted Software Development 2025.” dora.dev

[10] Perry, N., Srivastava, M., Kumar, D., and Boneh, D. “Do Users Write More Insecure Code with AI Assistants?” ACM CCS, 2023. arXiv:2211.03622. (Stanford University.)

[11] GitClear. “AI Copilot Code Quality: 2025 Research” (analysis of 211 million changed lines of code, 2020–2024). gitclear.com

[12] “Self-Admitted GenAI Usage in Open-Source Software.” 2025. arXiv:2507.10422. (Finds more nuanced code-quality patterns where AI usage is self-declared.)

[13] Harness. “State of Software Delivery 2025.” (Reports increased time debugging AI-generated code and resolving AI-introduced vulnerabilities.)

[14] Jevons, W. S. The Coal Question: An Inquiry Concerning the Progress of the Nation, and the Probable Exhaustion of Our Coal-Mines. 1865.

[15] Nadella, S. Public remarks on the Jevons paradox and rising AI demand, January 2025. (Chief Executive, Microsoft.)

[16] Becker, J., et al. “We Are Changing Our Developer Productivity Experiment Design.” METR, February 2026. metr.org