Agile Returns To Speed

The odd thing about Agile is that the people who wrote the Manifesto spent the next twenty years apologizing for it. Dave Thomas, one of the original signatories, gave a talk in 2015 called “Agile is Dead” (Thomas, 2015). He wasn’t attacking the speed. He wasn’t attacking the intention. He was attacking the religion — the certifications, the coaches, the frameworks of frameworks, the two-week sprint treated as a law of physics. His point was that “agile” started as an adjective and ended as a noun, and somewhere in that grammatical shift the whole thing went sideways.

I’ve been thinking about this lately because LLMs have quietly finished the job Dave Thomas started. They haven’t killed Agile. They’ve made Agile’s central artifact — the sprint — look embarrassingly slow.

Consider what a sprint actually is. It’s a timebox. Two weeks, usually. Why two weeks? Because that was the shortest interval in which a cross-functional team of humans could plan, build, test, and release something worth looking at. The number wasn’t handed down from heaven. It was a compromise with the biology of meetings and the latency of human typing. Scrum took the Toyota Production System, swapped “cars” for “features,” and called it a methodology. The rugby metaphor was supposed to be charming; in practice it described the level of finesse pretty accurately.

Now imagine you told a 2005 Scrum Master that in 2025 a developer could conceive a feature at 9 a.m., describe it in English, and have a working implementation before lunch. They would have said: “Then why do we need the ceremony?”

Exactly.

The sprint was a workaround

Most of what we call “process” is a workaround for something we couldn’t do. We invented sprints because we couldn’t iterate faster. We invented backlogs because we couldn’t remember everything. We invented standups because we couldn’t see what the person next to us was working on. These were all solutions to problems of human throughput.

LLMs don’t eliminate those problems. But they change the time constant by one or two orders of magnitude. Rosenthal and Zuckerman make this point nicely for logistics: experience used to follow production, and now it precedes it (Rosenthal & Zuckerman, 2025). You can simulate a million outcomes before you ship a box. The same thing is happening in software, except software was already mostly simulation to begin with. The learning curve hasn’t steepened. It’s collapsed.

When the learning curve collapses, the sprint collapses with it. What used to take a two-week sprint — a real slice of working software, tested, reviewed, demoed — now takes somewhere between a morning and an afternoon. Not always. Not for everything. But often enough that the ceremony built around the old cadence starts to look like a Civil War reenactment.

The religion, not the speed

Here's where I want to be careful, because this is the part where people get upset. I am not against iteration. I am not against releasing early. I am not against talking to users. These are good ideas. They were good ideas before Agile and they'll be good ideas after it. The Manifesto, read charitably, is mostly a list of reasonable preferences.

What I'm against is the part where reasonable preferences calcified into a clerisy. The part where you couldn't ship a feature without a Jira ticket, and you couldn't have a Jira ticket without an epic, and you couldn't have an epic without a quarterly planning offsite. The part where the ritual of the retrospective replaced the act of reflection. The part where "we're Agile" became a sentence people said with the same conviction as "we're Lutheran."

Michael Burnett put it well: the Agile Manifesto, read in 2023, reads like a set of excuses for not doing the work (Burnett, 2023). Welcome changing requirements — sure, also, we'll change our minds whenever. Working software is the primary measure of progress — sure, as long as "working" means "compiled." The principles weren't wrong so much as they were convenient. And conveniences, given enough time, become theologies.

What LLMs actually change

The interesting claim isn't that LLMs make Agile faster. It's that LLMs make Agile unnecessary for the reason it was invented and return us to the disciplines it was invented to avoid.

What were those disciplines? They were the boring ones. Research. Architecture. Design. Strategy. Thinking about a whole system before you build a piece of it. The things Waterfall did badly and Agile decided not to do at all. For twenty years we pretended you could skip the thinking phase because the market was moving too fast, and we called the result "discovery." Mostly it was just guessing in public.

LLMs change this because they make the thinking phase fast. Not just the coding phase — the thinking phase. You can sketch a whole system in an afternoon. You can argue with yourself about the data model. You can ask the model to find the holes in your plan, and it will, because models are quite good at finding holes in plans. The expensive part used to be

exploring the option space. Now the expensive part is knowing which options are worth exploring. That's a judgment problem, not a throughput problem.

And judgment — taste, discipline, knowing what good looks like — is exactly what the software craft tradition cared about before we all went to Agile church.

The whole, not the piece

Burnett has a diagram in his essay showing Agile as a circle and Waterfall as a staircase (Burnett, 2023). His point is that Agile assumes the feature will be revised and Waterfall assumes the product won't be. Both are wrong, but in opposite directions. The honest answer is that you want to think about the whole product, and then ship it in pieces, and then actually revise it. None of those three steps is optional, and Agile quietly dropped the first one.

LLMs let you put it back. You can now hold a whole system in working memory — yours and the model's together — in a way that a human alone genuinely couldn't. You can describe the shape of the thing at the level of the whole product, not the level of the next ticket. This is a real cognitive shift and it's worth sitting with. For most of my career, "architecture" meant "the stuff we wished we'd thought about six months ago." Now you can think about it six months ago in about an hour.

This is the part that kills the sprint. The sprint existed because we couldn't see very far ahead. When you can see further ahead, the timebox stops being a tool and starts being a cage.

Common sense, in partnership

I want to be clear about one thing. The answer isn't to replace Agile with another framework. If some consultant is already drafting a Manifesto for AI-Native Software Development, I'd like to politely ask them to stop. The whole lesson of the last thirty years is that the frameworks weren't the point. The disciplines were the point.

So what should you actually do? Roughly what good engineers did before anyone invented a capital-letter methodology for it.

Think about who the user is and what they need. Do the research. Write it down. Design the system end to end, at least as a sketch. Show the sketch to the model and argue with it. Show it to a human and argue with them too. Decide what the thing is before you build it. Then build it — often with the model doing the boring parts — and ship it in slices thin enough that you learn from each one. Revise based on what you learn. Have opinions about quality. Refuse to ship things you wouldn't be proud of. Repeat.

Notice that none of this requires a two-week sprint. None of it requires a standup. None of it requires a story-pointing exercise where the team argues about whether something is a 3 or a 5. Those weren't the goal. They were scaffolding, and LLMs have made the scaffolding heavier than the building.

The tongue-in-cheek part

I'll admit I find the current moment darkly funny. For twenty years we were told that long-cycle thinking was a luxury only the dinosaurs at big enterprise companies could afford. You had to ship, ship, ship. You couldn't stop to design a system because the market would eat you alive. And now the tool that actually eats you alive — the LLM — rewards the exact opposite behavior. It rewards the person who thought carefully about the whole problem before opening the editor. It punishes the person who types "let's just get something working" and then discovers at hour six that their data model can't actually support what the product needs to do.

Agile told us to ship first and think later. The LLM is a truth serum for that advice. Ship first and think later, with an LLM, just means you generate your mistakes faster.

Think first, with the LLM, and ship second — and you discover the old disciplines were right all along. They just didn't have a catchy name or a certification track.

What "agile" means again

If we want to rescue the word, we can. Drop the capital A. Let "agile" go back to being an adjective, the way Dave Thomas wanted (Thomas, 2015). An agile team is one that moves quickly, changes direction when it should, and doesn't confuse motion with progress. That's it. That's the whole definition. It doesn't need a Scrum Master, a Product Owner, a Sprint Planning ritual, or a Jira license. It needs people who know what they're doing and a tool — now, mercifully, a very good one — that makes the doing faster.

You could argue this returns us to something older than Agile, older than Lean, older than the Toyota Production System. It returns us to craft. A carpenter doesn't have sprints. A carpenter has a plan, a set of tools, a standard of quality, and the good sense to notice when the wood is warping. Software, for a brief and strange period, pretended it was manufacturing. It isn't. It never was. The LLM is the thing that finally lets us admit it.

The kicker

The irony of all this is that the Agile Manifesto's four values, if you strip away the theology that grew up around them, point at the same place LLMs do. Individuals and interactions. Working software. Customer collaboration. Responding to change. These were never bad ideas. They were just bad as a religion.

Maybe the right way to think about the next decade is this: we finally have a tool fast enough to let us be slow where it matters. Slow about the research. Slow about the design. Slow about deciding what the thing should actually be. And fast — genuinely, shockingly fast — about everything else.

Agile, the adjective, might finally get its name back.

References

Burnett, M. (2023, February 12). *The age of Agile must end: 30 years ago the technology industry attempted to import Lean practices — it failed*. UX Collective. Retrieved from https://uxdesign.cc

Rosenthal, D., & Zuckerman, M. (2025). *AI destroys the old learning curve: Wright's Law is being rewritten, and leaders who don't adapt to this new world will be replaced*. ESP Logistics Technology / Boston Consulting Group.

Thomas, D. (2015). *Agile is Dead (Long Live Agility)* [Conference talk]. GOTO Conferences. Based on the essay at https://pragdave.me/thoughts/active/2014–03–04-time-to-kill-agile.html