The Second wake-up call: Speed is one thing – Control is another

Paul Bevan ©Bloor Research May 2026

ARTICLE REPRINT

Last month, when I wrote my original AI wake-up call, it was prompted by a very real moment of surprise. Listening to Robin Bloor describe what he was doing with Claude 4.6 from Anthropic, I realised – rather abruptly – that I had underestimated both the pace and the direction of travel. What I thought was incremental progress was, in fact, something closer to a step change.

That blog was deliberately a jolt. It was aimed at software developers and IT organisations – people who build and run the systems the rest of the business depends on. The intention was simple: recognise how fast this is moving, because it's faster than you think.

But having had time to reflect and having spoken to a number of people already putting these tools to work, I think there's a second message that needs to sit alongside the first.

Yes, the pace is extraordinary. But that is precisely why a degree of caution is now essential. What concerns me slightly is that, in some quarters, the wake-up call is being interpreted as a green light to accelerate as quickly as possible into AI-driven development – automating code generation, wiring up systems, pushing prototypes into production – without fully understanding where the risks lie. I'm already hearing examples of this happening, and while some of the results are impressive, some are, frankly, a little worrying.

Because for all their capability, these systems are not infallible. And more importantly, they are not accountable. One of the first issues that becomes apparent when you look closely is the tendency to equate well-formed output with correctness. AI-generated code often looks clean, structured, and entirely plausible. In many cases, it runs. But "runs" is not the same as "works as intended," particularly when you move beyond straightforward scenarios into edge cases, integrations, or performance-sensitive environments.

We are, I think, in danger of mistaking fluency for accuracy.

A second challenge is context. Tools like Claude 4.6 are undeniably better at maintaining context than earlier generations, but they are still heavily dependent on the inputs they receive. If the architectural intent isn't clearly defined, if constraints are vague, or if the surrounding system isn't properly understood, then the outputs, however polished, can quietly diverge from what is actually required. And because they look convincing, those divergences are not always picked up early. That creates a subtle but important risk: errors that are harder to detect because they don't look like errors.

There is also a longer-term concern that I don't think we should ignore. If developers begin to rely too heavily on generated code without fully understanding how it works, there is a danger that some of the underlying engineering knowledge starts to erode. Over time, that could leave teams very good at producing solutions quickly, but less well equipped to diagnose and fix them when things go wrong.

And things will go wrong. That's the nature of complex systems.

Integration is another area where reality bites. Generating a piece of code is one thing. Integrating that code into an existing environment, ensuring it behaves correctly alongside other components, meets performance expectations, adheres to security requirements, and remains maintainable, is quite another. AI can help with parts of that process, but it doesn't remove the need for discipline. If anything, the volume of generated artefacts increases the burden on testing, validation, and governance.

This is where I think we need to be very clear. The opportunity here is not simply to automate coding. It is to rethink how we build and manage software in a world where code can be generated at scale. That requires stronger architectural thinking, not less. It requires better testing practices, not shortcuts. And it requires developers to remain firmly in the loop—not as passive recipients of output, but as active owners of the systems being created.

In practical terms, that means being selective. There are areas where AI can safely accelerate ▶

development; boilerplate code, documentation, and early-stage prototyping for example. And there are areas where a more cautious approach is justified; core business logic, security-critical components, or regulated environments. Treating all use cases as equal is, in my view, a mistake.

It also means putting proper guardrails in place. Clear architectural patterns, well-defined interfaces, robust testing frameworks, and explicit review processes become even more important when the rate of code generation increases. Speed without structure is not an advantage; it's a risk multiplier.

None of this is to suggest that organisations should hold back. Quite the opposite. Those that learn how to use these tools effectively will move faster and deliver more. But effectiveness here is not just about how quickly you can produce something. It's about how confidently you can rely on it once it's in production.

And that comes back to a simple point that perhaps didn't come through strongly enough in my first blog... the technology may be accelerating, but the responsibility hasn't gone anywhere.

We are still accountable for the systems we build. We are still responsible for their behaviour, their security, their maintainability and their impact. AI can assist, accelerate, and in some cases transform how we develop software—but it does not absolve us of the need to understand, validate, and stand behind what we deploy.

So yes, take this as a continued wake-up call. The pace of change is real, and it is not slowing down. But don't confuse urgency with recklessness. If the first message was "things are moving faster than you think," then the second is this: make sure you're still in control when you try to keep up.