When Implementation Becomes Trivial: What Agentic Coding Really Changes

By Laurent ASSOULY, January 2026

Software development is undergoing a fundamental shift. Writing code no longer defines professional expertise; instead, value is moving toward the ability to oversee its production. The debate about whether AI can code is essentially over. Implementation is no longer the bottleneck.

For a long time, coding was the primary proof of skill and the main source of professional identity. Agents are now making that model obsolete.

1 Implementation as a commodity

Writing code that works is now a fast, cheap, and routine process. Languages or frameworks are no longer meaningful barriers. An agent can jump between different syntaxes instantly, applying industry standards that developers used to spend years learning.

"Closed" problems are a good example of this change. Coding a biquad filter is a math challenge, but the technical implementation is a solved problem. Since there are thousands of known, high-performance versions available, an agent can produce a perfect one in seconds. Knowing how to write that code yourself is no longer a competitive advantage. The work is now purely utilitarian.

2 The role of design

Design is much harder to automate. It isn't about following best practices; it is about making hard choices between conflicting requirements. It means setting invariants and deciding which trade-offs to accept regarding cost or speed.

These decisions don't come from statistical models. While an agent can suggest common structures, it doesn't know which one is vital for a specific project without being told. Without that human intent, AI simply defaults to an average. A design is never an average; it is a deliberate choice made under risk.

3 The case of audio plugins

Audio software makes this distinction very clear. In a plugin, signal processing and the user interface follow completely different rules. They cannot block each other. Keeping them separate is an architectural choice driven by real-time needs.

An agent might "know" to avoid locks in an audio thread, but it can't guess the right latency target or whether to prioritize code simplicity over raw optimization. You can easily get code that passes tests and runs fast but is still wrong because it violates a requirement that was never made explicit.

4 From craftsman to pilot

The developer's role is shifting. Value no longer comes from clever syntax or knowing a language inside out, but from judgment.

The job is now about setting boundaries, assessing technical risks, and rejecting solutions that don't fit the product vision. Long-term maintainability is still a major point of contention. An agent can give you an immediate answer, but ensuring that the code can evolve over time still requires constant human supervision. Without clear instructions on topics where there is no consensus, AI eventually hits the genuine Murphy's Law: it has a 50

Conclusion

Agentic coding doesn't make development obsolete. It just makes implementation secondary. Code is a tool, not the goal.

What matters now are the plans: the invariants and the difficult compromises. Agents execute fast, but they cannot decide what should be true for the software. The responsibility for those choices remains human.