AI GOVERNANCE SERIES | PART 2

The New Control Layer

How Spec-Driven Development Makes AI Governance Operational

By Greg Aldrich | Global CIO & Strategic Advisor | April 2026

In my previous article on AI governance, I argued that organizations should not wait for regulation to define responsible AI. They must develop internal governance capabilities now. Several clients and colleagues raised a valid question: What does this look like at the development level?

This is my answer.

Organizations I work with are increasingly deploying AI coding agents that generate, iterate, and refine software faster than any human team. Most encounter the same issue: governance models designed for human-paced development do not work at machine speed. Human-centered validation models do not scale at machine speed.

Spec-driven development is the most practical response I have seen to this problem. This is not a development methodology—it is a governance architecture that enables scalable control of AI development without sacrificing the speed that makes it valuable.

> ***"Spec-driven development is how governance becomes operational at engineering speed — not a development practice, but a control system."***

Why the Old Control Model Is Breaking

Traditional software governance was built around a set of assumptions that made sense when humans wrote every line of code: a finite rate of change, direct human visibility into implementation, and clear accountability at the point of creation. Code reviews worked because a senior engineer could read a junior engineer's code and catch problems before they reached production.

Coding agents invalidate all three assumptions simultaneously.

A capable agent can generate thousands of lines of code in the time it takes a reviewer to read hundreds. These implementations may be functionally correct but subtly misaligned with intent, not due to error but interpretation. Accountability also becomes less clear when no human authored the code.

I have observed this in client environments. A team deploys an agent to accelerate a development backlog, and velocity improves significantly. Weeks later, a behavior appears in production that no one specifically designed. This is not a traditional bug, but an emergent result of the agent's interpretation of its instructions. The code review process did not catch it because no reviewer could fully grasp the scope of the generated system.

> ***“This is not a failure of AI, but a failure to adapt human-era controls to machine-scale systems.”***

When code is generated rather than written, reviewing code is no longer sufficient as a control mechanism. Control must move upstream.

The Shift: From Code to Specification

In traditional development, control resides in the code that is written, reviewed, and merged. In an agent-driven model, control must shift upstream to what is defined before the agent begins.

This is the core insight of spec-driven development: the specification becomes the primary governance artifact. It is not documentation of what was built, but a structured definition of system requirements, constraints, acceptable outcomes, and compliance measures.

Code becomes an implementation detail, while the specification serves as the system of control. To be precise: the spec governs intent; the code remains the executable record. Both matter — but in an agent-driven environment, the spec is where governance lives.

For executives, this maps directly onto governance. Discussions about risk boundaries, compliance conditions, and accountability are precisely what specifications formalize. Spec-driven development is not a technical abstraction; it is an engineering discipline aligned with governance objectives.

This mirrors what happened with infrastructure a decade ago. Control moved from manual provisioning to infrastructure-as-code. Spec-driven development applies the same principle to software: control moves from implementation to definition, and from review to enforcement.

What This Looks Like in Practice

The practical shift involves four interconnected changes to how development is structured:

1. Specifications as Structured Control Documents

Effective specifications are not prose requirements documents. They define functional behavior, non-functional constraints, risk boundaries, and compliance conditions in formats that can be evaluated by both humans and automated systems. The specification is the basis for governance, not a document to be archived after project completion. In practice, these specifications increasingly take structured, testable forms—ranging from schema definitions and policy constraints to executable test suites and validation rules. The critical distinction is that they are not read—they are enforced.

2. Validation Before Generation

The development flow inverts. Instead of build, test, and fix, the sequence becomes define, validate, generate, and verify. Validation criteria are set before the agent writes any code. This shift changes the accountability model: if the specification is incorrect, it is a governance failure; if the implementation deviates from the specification, it is a system failure. Both are visible and attributable.

3. Automated Verification Pipelines

Human reviewers cannot validate agent-generated systems at the speed agents operate. Automated verification infrastructure, which continuously evaluates whether implementations meet specifications, respect constraints, and avoid unintended behaviors, is essential in an agent-driven environment. It serves as the governance mechanism.

4. Agents as Bounded Actors

Business leaders should not view agents as open-ended tools awaiting instructions. Instead, agents should be seen as actors operating within a bounded system. The quality of output depends less on task description and more on how well boundaries, expectations, and validation criteria are defined. Constraints serve as the control mechanism.

> ***"Agents don't need better instructions. They need well-defined constraints — and systems that verify those constraints are being respected."***

The Governance Connection

At this point, spec-driven development shifts from a development topic to a strategic approach.

In a well-implemented spec-driven model, the governance artifacts your organization needs are produced as a natural byproduct of the development process:

- Specifications become auditable records of intent.
- Validation logs become evidence of compliance.
- Verification pipelines become mechanisms of enforcement.
- Development histories become traceable audit trails.

Instead of applying governance after development through reviews, it becomes embedded within the system itself. Governance is not a gate; it is the architecture.

For organizations that established AI governance structures early and designed for auditability from the outset, spec-driven development is the engineering model that brings these commitments to life at the delivery layer. It bridges the gap between governance policy and practice.

To be direct, most organizations I work with have not reached this level of maturity. Many are still in early AI-assisted development, where agents support human developers but humans remain primary authors. Transitioning to a spec-driven model requires investment in specification discipline, validation infrastructure, and a fundamental rethinking of development team roles. While this is not trivial, organizations that invest now will gain a significant structural advantage as agent capabilities advance.

What Most Organizations Will Get Wrong

The failure patterns are already visible in early adopters:

- Treating specifications as documentation rather than control systems — writing them after the fact, filing them away, never validating against them
- Continuing to rely on code review as the primary quality gate — scaling human reviewers rather than rethinking the review model
- Deploying agents without defined constraints — optimizing for velocity while creating unmanaged risk

- Underinvesting in verification infrastructure — building the generation capability without building the validation capability
- Assuming governance redesign can follow the velocity gains — by the time the problem is visible, the technical debt is already significant

This pattern is common in AI governance: organizations apply new capabilities to old operating models and are surprised when controls fail. The issue is not the technology, but the governance architecture.

A Maturity Framework

Organizations tend to move through four recognizable stages:

- **Stage 1 —** Code-Centric: Human-written, manually reviewed. Traditional controls apply. This is where most organizations still operate.
- **Stage 2 —** AI-Assisted: Agents augment development, but humans remain the primary authors and reviewers. Velocity improves; governance gaps begin to emerge.
- **Stage 3 —** Spec-Driven: Specifications define system operation. Validation becomes the central control mechanism. Development and governance begin to converge. This is where the real transformation starts.
- **Stage 4 —** Self-Governing: Continuous validation and adjustment are embedded. Governance is automated and auditable. Oversight by human staff is repositioned from reviewing outputs to defining systems.

Most organizations are between stages 1 and 2. Moving to stage 3 is primarily a governance and organizational design decision, not a technology investment. The technology for spec-driven development exists today; the greater challenge is the willingness to restructure.

What Good Looks Like

The organizations I've seen execute this well share a common set of practices:

- Specifications are tied to business outcomes — not simply technical requirements — and are treated as living governance documents throughout the development lifecycle.
- Automated validation pipelines enforce constraints continuously, not just at release gates.
- Agents are deployed within clearly defined boundaries, with explicit validation criteria established before generation begins.
- Development workflows are designed for auditability from the start — not retrofitted after the fact.
- Human review is intentionally shifted from monitoring completed work to defining system requirements and verifying compliance.

This approach does not eliminate human decision-making. Instead, it places decision-making where it is most effective: upstream in definition and constraint, rather than downstream in review and remediation.

Final Thought

A decade ago, automation enabled us to scale infrastructure. Cloud, containers, and infrastructure-as-code shifted control from provisioning to definition. This pattern is now repeating at the development layer, with equally significant governance implications.

The question is no longer how to control the code, but how to control the systems that produce it. The answer, as with infrastructure, is to move control upstream, make it executable, and build verification systems for continuous enforcement.

Organizations that adopt this approach early will not only build faster but also achieve control, consistency, and auditability at a scale manual governance cannot match. More importantly, they will demonstrate that governance and AI capability, when properly architected, are aligned.

> ***"Control is no longer in what is written.***
> ***It is in what is defined — and how it is enforced."***

This is Part 2 of an ongoing series on AI governance and organizational readiness.
Part 1: The AI Governance Tightrope

About the Author

Greg Aldrich is a Global CIO and Strategic Advisor with 30+ years of experience helping boards, executive teams, and C-suite stakeholders navigate IT strategy, AI governance, and digital transformation. He serves as Senior Strategy Advisor at Blue Tree Technology Group and Senior Transition Architect at SDS Consulting, and currently serves as CIO at Andrew Wommack Ministries & Charis Bible College, where he chairs both the AI Committee and IT Steering Committee. He has advised organizations across financial services, gaming, healthcare, higher education, logistics, and nonprofit sectors.

Connect: linkedin.com/in/galdrich