

Architecture at agent speed

Why the high-level design can no longer sit still, and what governing it properly now requires

Jason Cruz Falzon · creator of the Cognitive Delivery Framework

Enterprise architecture governance was built for a world where change was rare. Agent-assisted development has made change the constant. The design document now sits still while the system underneath it moves, and the process built to protect the design is the very thing that lets it fall out of date. This paper argues that the answer is not to abandon governance, but to move it. From approval before the work to evidence during it. From a document that describes the past to one that keeps pace with the present.

A change lands in the design. It is a good change. An agent, working through a new ingestion path, has found that the original module boundary does not hold. The data has a shape nobody modelled at the start. The boundary needs to move. The agent records why, adjusts the code, and carries on.

Three weeks later the architecture review board meets. The high-level design that describes the old boundary is on the agenda for sign-off. By the time the board approves it, the codebase has moved on again. Two more boundaries have shifted. The document the board has just approved describes a system that no longer exists.

This is not a story about one organisation. It is a pattern, and it is becoming the normal condition of enterprise delivery. The design document sits still while the system underneath it does not. The gap

between the two widens. And the process built to protect the design is the very thing that lets it fall out of date.

The question this paper sets out to answer is simple to state and uncomfortable to sit with. When your delivery moves at agent speed, what happens to the documents that are meant to govern it? And what does it take to keep them honest without slowing everyone down?

The document that used to sit still

For most of the history of enterprise architecture, the high-level design was fixed early and treated as settled. You did the modelling upfront. You socialised it. You got sign-off across the domains that cared. Then you built against it. Deviation from the design was treated as a problem to be corrected, not a signal to be captured.

There was a reason for this, and it was a good one. Someone has to be accountable for whether the system that ships matches the system that was designed. If the design drifts without being formally updated, you lose traceability. You cannot answer what changed, or when, or why anyone thought it was a good idea. In a regulated enterprise, that is not a documentation gap. It is a liability.

Data architecture inherited this instinct more strongly than most. The frameworks that shape the discipline are built around deliberate, sequenced, upfront design. You define the target state, you map the transition, you govern the move from one to the other through formal change control. It is a careful way of working, and it was well matched to a world where change was rare and expensive.

The problem is that the world it was matched to no longer exists.

What agents changed

Software is not built overnight. It is not built in a week. It is built over months, sometimes years, and across that span there is always deviation. There are always updates. There is always something new that the original design did not anticipate. This was true before agents. Agents have made it the dominant fact of delivery rather than an occasional exception.

When an agent implements against a design, it makes hundreds of small structural decisions along the way. Where a boundary sits. How a contract is shaped. Which assumption to hold and which to relax. Each decision is minor on its own. Together they accumulate into a system that has quietly moved away from the document that describes it.

Model capability rises across the same span. What an agent could not attempt at the start of a project it can attempt by the middle. The design that was right for the capability you had is not always right for the capability you have gained. So the pressure to pivot is not a failure of planning. It is a consequence of building over time with tools that keep getting better.

This leaves the enterprise with two poor options. Hold the design static and accept that it will describe what you meant to build rather than what you actually built. Or let the design evolve without control and lose the traceability that made it worth having. Most organisations pick the first without admitting it, and then act surprised when the document turns out to be fiction.

There is a third option. It is harder to describe and harder to build, and it is the one worth having.

The high-level design is not just a diagram

Before we get to the third option, it is worth being precise about what a high-level design actually is, because the answer changes what governing it requires.

A high-level design is not a solution design. It is not a schema. It is the one document that gives you a full overview of where the system is. It reflects every domain at once, at least at a high level. Security constraints sit alongside operational assumptions. Deployment topology sits alongside strategic intent. The whole point of the document is that you can look at it and understand the system as a single coherent picture rather than as a stack of disconnected specialist views.

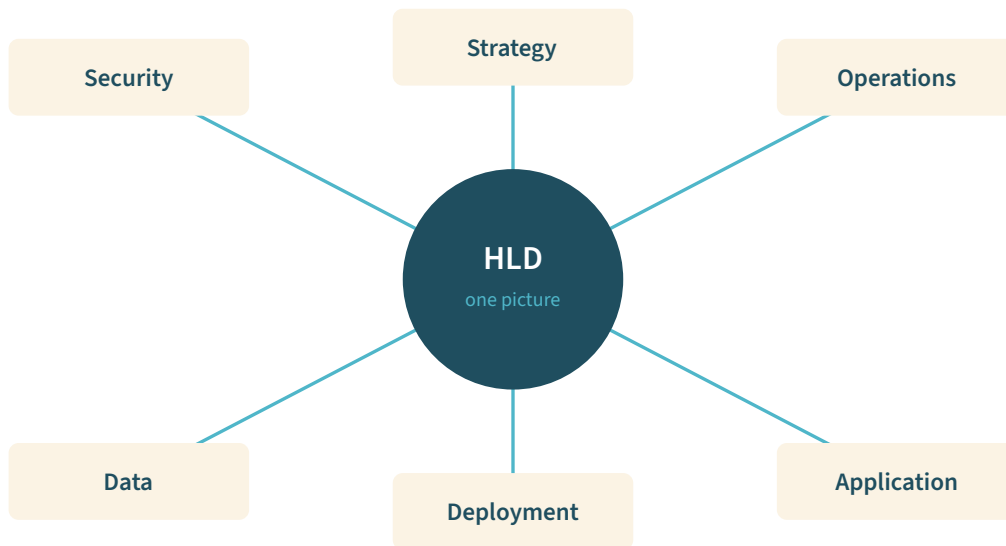


FIGURE 1 The high-level design is the single point where every domain meets. **When it drifts, it does not fail in one place. It becomes unreliable across all of them at once.**

That is exactly why drift in the high-level design is more dangerous than drift anywhere else. When a low-level detail falls out of date, one team is inconvenienced. When the high-level design falls out of date, the single source of truth for how the whole system works becomes unreliable across every domain at the same time. Security believes one thing. Operations believes another. Strategy is planning against a picture that no longer holds. The document that was meant to hold the system together is the one that quietly stops doing its job.

So the high-level design cannot be allowed to drift. But it also cannot be frozen, because the system it describes will not hold still. The document has to become live. The real question is not whether it evolves. It is whether the evolution is governed.

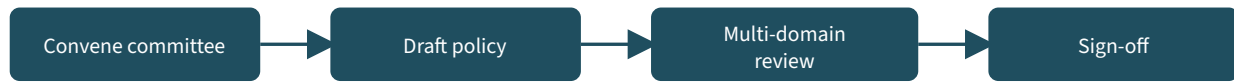
The trap of governing the governance

Here is where enterprises get stuck, and it is worth naming plainly because it is the most avoidable failure in the whole picture.

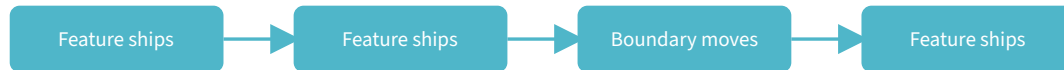
An organisation decides it needs to take AI and governance seriously. Reasonable. The first thing it does is convene a committee. The committee is tasked with producing a governance policy. A high-level statement of how the organisation will govern agent-assisted work. The policy has to be reviewed by legal, by security, by compliance, by architecture. It has to be agreed across domains. Getting people to agree takes cycles. Depending on the organisation, the whole exercise can take a

month, or several.

Governance track



Codebase, in parallel



The policy is obsolete on arrival. It describes a way of working the organisation has already left behind.

FIGURE 2 The recursive trap. The slow governance machine is applied to the design of the governance itself, and produces the same result it produces everywhere else. **A document that describes an intention rather than a reality.**

By the time the policy is signed off, the ground has moved. The agents did not wait for the committee. They shipped features while the policy was in review. The document that was meant to govern the work describes a way of working the organisation has already left behind. The governance is obsolete on arrival.

This is the recursive version of the same problem. The organisation applies its slow governance machine to the design of its own governance, and the machine produces the same result it produces everywhere else. A document that describes an intention rather than a reality. A binder that no one reads because it was out of date before it was published.

The instinct behind this is not wrong. Someone does have to be legally accountable. But there has to be a line. A level of acceptable risk that lets work proceed at pace, rather than a standard of certainty that slows everything down for everyone in the name of a safety it does not actually deliver. A governance policy that takes three months to write and is wrong when it lands is not safer than moving quickly with substance captured as you go. It only looks safer.

A governance policy that takes three months to write and is wrong when it lands is not safer than moving quickly with substance captured as you go. It only looks safer.

Substance over theatre

The distinction that resolves this is the distinction between governance that is documented and governance that changes behaviour.

It is possible to have governance that is complete on paper and hollow in practice. The policy exists. The committee met. The design was signed off. The audit trail is populated. Every box is ticked. And none of it changes what an agent does at the moment it makes a decision. The gap between a process that is documented, configured and logged, and a process that demonstrably changes behaviour at the point of action, is the gap that matters. It is the difference between governance you can show an auditor and governance that actually governs.

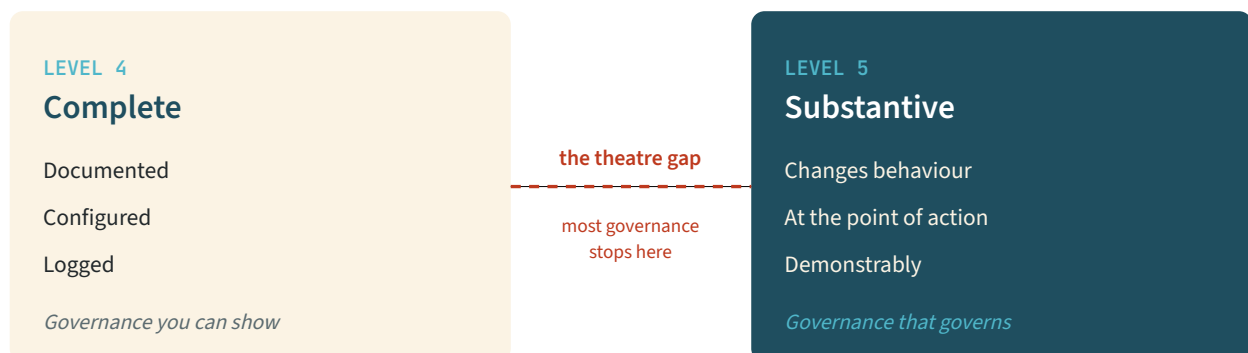


FIGURE 3 The theatre gap. Being complete on paper and changing behaviour at the point of action are not the same thing. **Most enterprise architecture governance lives on the left and never crosses to the right.**

Most enterprise architecture governance lives on the documented side of that gap. There is a design. There is a review board. There is a change process. And still the code drifts, because none of it

operates at the point where the decision is made. It operates weeks later, in a room, on a version of the system that has already changed.

Capability without governance is speed without control. But governance that does not change behaviour is not control either. It is the appearance of control. Closing the gap does not mean adding more documents or more committees. It means moving the governance to where the work happens.

Governed evolution: moving the rigour, not removing it

The answer is not to abandon the gates. It is to move the rigour from before the work to during and after it.

Under the old model, the rigour sits in front of the change. Nothing happens until the design is approved. That is what makes it slow, and at agent speed it is what makes it useless, because approval cannot keep pace with the rate of change.

The alternative puts the rigour in three different places. Detection at the point of authoring, so a material change is caught as it is made rather than discovered weeks later. Evidence captured as the change happens, so the reasoning, the alternatives considered and the provenance of the decision are recorded at the moment they exist and are freshest. And review after the fact, fast and focused, not on reconstructing what changed but on judging whether it was the right call.

Old model · rigour in front of the work



New model · rigour in three lighter places

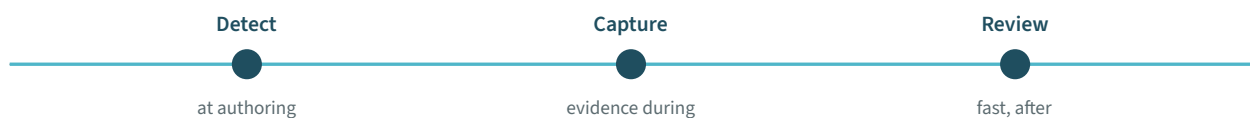


FIGURE 4 The rigour does not disappear. It moves. **From a single heavy gate that blocks the work to three lighter points that travel with it.**

This changes what a review board actually does. Under the old model, the board spends its time working out what changed and why, from documents that are already stale. Under this model, the change and its reasoning are already recorded, tied to the code and the moment they happened. The board is not doing archaeology. It is making a judgement. That is a short conversation, not a month-long cycle.

The building blocks for this already exist in the industry. They just exist separately.

THREE PATTERNS THAT ALREADY WORK, IN ISOLATION

Architecture decision records give you an append-only log of decisions. You never edit history. You write a new record that supersedes the old one, preserving why the design moved and what was considered at the time.

Drift detection tells you when the code and the documented architecture have diverged, so divergence is surfaced as a signal rather than discovered as a surprise.

Versioned contracts let interfaces evolve without constant re-engineering, so change at a boundary does not force a rewrite on both sides of it.

Each of these is a real and proven pattern. What almost no one has done is unify them into a single governance model, where architectural documents follow the same lifecycle as code, with the same traceability and the same review discipline.

That is the move. Treat the high-level design, the low-level design and the decision records as governed artefacts, not as separate documentation that lives outside the process. Give architecture the discipline that code already has. Version control. Diffs. A record of who changed what and why. Review at the point of change. The ability to roll back. Code has had this for decades. Architecture has not, and it is the reason architecture drifts while code stays honest.

Architecture as a governed artefact

What does it look like in practice to put the high-level design through the same lifecycle as code? The same phases a feature passes through apply to a material change in the architecture. The difference is

what counts as evidence at each gate.

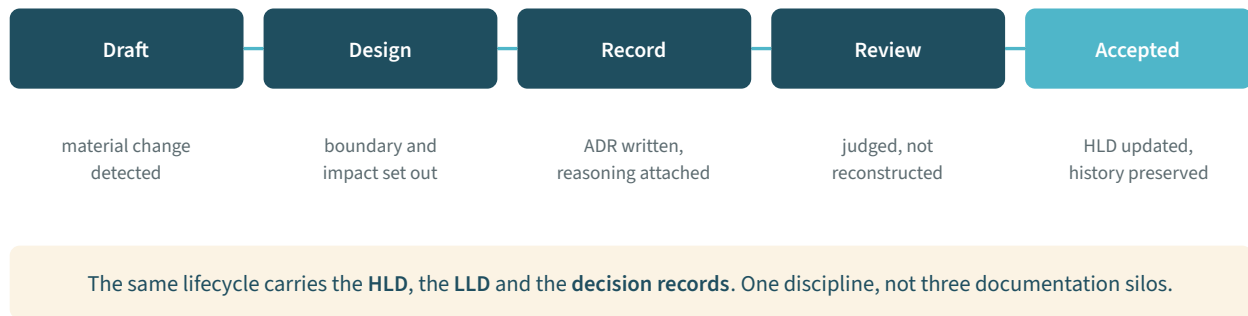


FIGURE 5 A material architecture change moving through a governed lifecycle. **The document is never frozen and never drifting. It is versioned, reasoned and current, with the history of how it got there attached.**

The high-level design becomes what it was always meant to be. A living, honest picture of the system across every domain, kept current because keeping it current is part of the work rather than a separate approval cycle. Not a bible that describes the past. A record that reflects the present, with the reasoning for how it got there attached. The low-level design and the decision records sit under it in the same discipline. When someone asks why a boundary moved, the answer is a record, not a memory.

What the platforms will not tell you

It is worth being clear about why this gap has stayed open, because the reason is not that it is technically hard.

The large platform vendors have their own interests. They sell products, and their products benefit from a particular framing of the problem. If the answer to architecture governance is a platform you buy, then the vendor has every reason to describe the problem as a tooling problem. The confusion is useful. It leaves room for the sale.

But the real gap here is not technical. It is conceptual. No one is saying plainly that your architecture documents need to be governed the same way your code is, and here is how. The moment that is said clearly, you can no longer sell a platform as the answer, because the answer is an operating model, not a product. It is a way of working that an organisation can adopt on its own terms, using tools it already has.

That is why so much of this remains unsaid. Not because it is unsolvable, but because the people with the loudest voices in the conversation have little reason to solve it in the open. The organisations living the problem know it is real. They are simply waiting for someone to name it in a way that does not come with a licence attached.

What this asks of the enterprise

None of this requires a perfect governance policy written upfront. That is the trap. You do not need a committee to spend a quarter producing a document that will be wrong when it lands.

What you need is a lightweight operating model that captures substance in real time. Governance that lives in how the work is done, not in a policy that has been signed off and shelved. Detection as changes are made. Reasoning recorded as decisions are taken. Review that is fast because the evidence is already there. An audit trail that is a by-product of working properly rather than a separate exercise bolted on afterwards.

This is not less accountability. It is more, because it operates where the old model only pretended to. It is auditable and fast at the same time, which the old model insisted was impossible. It was only impossible because the rigour was in the wrong place.

The frameworks that shaped enterprise architecture were built for a world where change was rare. That world is gone. Agents have made change the constant, not the exception. The discipline does not have to be abandoned to keep up. It has to be moved. From approval before the fact to evidence during it. From certainty to acceptable risk held to account. From a document that sits still to one that keeps pace.

Agility in a world of lightning-fast thought is not the absence of governance. It is governance that has finally learned to move as quickly as the work it is meant to govern.

This paper draws on the Cognitive Delivery Framework, an operating model for governing software delivery where humans and AI agents share responsibility across the lifecycle. The framework treats architectural artefacts as governed specifications rather than static documentation, and grades governance by whether it changes behaviour at the point of action rather than by whether it has been documented.

Jason Cruz Falzon · jason@cognitivedelivery.co.uk ·
cognitivedelivery.co.uk