



BEYOND THE PLUMBING MINDSET

The API Economy and the Integration Programme That Never Ends

A Practitioner's Perspective on Why Integration Complexity Persists in Enterprise Programme Delivery

Giovanni Leonardi

November 2016

Contents

1. Executive Summary	3
2. The Promise, and the Disappointment That Follows It	3
3. A Pattern That Predates Its Latest Name	4
4. Why the Pattern Persists: Four Structural Forces	5
5. The Gap Between Intent and Reality	6
6. The Economy Beyond the Firewall	7
7. Toward a More Honest Frame	8

The API Economy and the Integration Programme That Never Ends

Programme · No. 169

Originally written November 2016 | Re-edited and re-rendered for the WhereBeyond Edition, 2025

"The complexity did not leave. It changed address."

1. Executive Summary

Every few years the enterprise is offered a new answer to the same old question: how do we get our systems to talk to each other without the effort growing faster than the value? The current answer is the API — small, well-described, independently deployable interfaces, and the API economy they are said to unlock. It is a good answer, better than most that came before it. And yet the integration programmes built to realise it are, if anything, larger, longer, and more troubled than the ones they replaced.

This essay is an attempt to explain that apparent contradiction. My argument is that integration complexity is not a technical problem that each new architectural generation gets closer to solving. It is a structural property of organisations that build software across boundaries they did not design — and every wave of technology, the API being only the latest, relocates that complexity rather than removing it. The point-to-point era hid it in the wiring. The service-bus era concentrated it in the middle. The API era distributes it back out to the edges and, crucially, into time — as integration debt that accrues quietly and is serviced by whoever is unlucky enough to be holding the programme when the interest comes due.

If that diagnosis is right, then the persistent failure of integration programmes to deliver on their promise is not a failure of execution. It is a failure of framing. We keep treating integration as plumbing — a cost to be minimised and delegated — when it is in fact the load-bearing structure of the modern enterprise, and deserves to be owned, resourced, and governed as such.

2. The Promise, and the Disappointment That Follows It

The pitch for the API economy is genuinely compelling, and I want to be fair to it before I take it apart. The idea is that if every capability in the enterprise — and beyond it — exposes a clean, stable, well-documented interface, then building new things becomes a matter of composition rather than construction. You assemble rather than integrate. Internal teams consume each other's services the way they consume a payments provider or a mapping service on the open market: through a contract, without needing to know or care what lies behind it. The organisation becomes, in the fashionable

phrase, a platform.

I have watched several organisations set out toward this vision with real conviction, and I have watched what happens next. The first few APIs are a triumph. They are built by strong teams, for real consumers, with the enthusiasm of the new. Then the programme scales, and the character of the work changes. What began as an architecture initiative becomes an integration programme, and integration programmes have a way of behaving like the ones we all remember from a decade ago: sprawling dependency maps, environments that are never quite aligned, a release calendar held hostage by whichever interface is least ready, and a steadily rising proportion of effort spent not on building capability but on keeping the connections between capabilities from breaking.

The disappointment is rarely that the technology failed. The APIs usually work. The disappointment is that the programme — the coordinated, multi-team, multi-quarter effort to knit the estate together — turned out to be just as hard as it always was. The tools got better. The problem did not get smaller. That gap, between the elegance of the unit and the difficulty of the whole, is the thing worth understanding.

3. A Pattern That Predates Its Latest Name

To see why, it helps to recognise that we have been here before, more than once, and that each visit followed the same arc.

In the earliest era, systems were integrated point to point. When system A needed something from system B, someone built a direct connection between them. This is intuitive, fast for the first few connections, and catastrophic at scale: the number of possible connections grows with the square of the number of systems, and before long the estate is a tangle that no single person understands — the integration "spaghetti" that became a term of art precisely because everyone had a plateful.

The answer to spaghetti was to put something in the middle. The enterprise service bus, and the broader service-oriented architecture around it, proposed that every system should connect once — to the bus — and the bus would handle translation, routing, and orchestration. Elegant in principle. In practice, the bus became the most complex, most contended, most politically fraught component in the enterprise. It concentrated integration logic in a single place owned by a single team, which meant that team became a bottleneck through which all change had to pass, and the "logic in the middle" grew into a second application as large and as brittle as any it connected.

The API era is, in large part, a reaction against the bus. Push the intelligence back to the endpoints; keep the network dumb; let each service own its own interface and its own logic; connect through lightweight, standard protocols rather than a heavyweight central broker. And this is genuinely better. But notice what it does with the complexity: it takes the integration logic that the bus concentrated in the centre and redistributes it across dozens or hundreds of independently owned services. The complexity did not leave. It changed address.

Era	Where the complexity lived	The characteristic failure
Point-to-point	In the wiring between systems	Combinatorial tangle; nobody owns the whole
Service bus	Concentrated in the centre	Central bottleneck; the middle becomes a monolith
API / microservices	Distributed to the edges and into time	Runtime coupling; integration debt nobody is funding

Three generations, three architectures, one constant: the total integration complexity of the estate stayed roughly proportional to the number of boundaries the business asked its software to span. What changed each time was where the complexity was made to live — and therefore who felt it, how visibly, and whether anyone was accountable for it.

4. Why the Pattern Persists: Four Structural Forces

If integration complexity were merely a matter of tooling, better tools would have retired it by now. That they have not tells us the forces sustaining it are structural. I see four, and they compound.

- Conway's Law is not a slogan; it is the mechanism. Organisations produce systems whose structure mirrors their communication structure. When you decompose a monolith into services, you are not only drawing lines in the software — you are drawing them along the seams of your organisation, because that is where the knowledge and the ownership already sit. The interfaces between services therefore inherit every awkwardness of the interfaces between teams: the misaligned priorities, the different release cadences, the meeting that never quite happens. Integration complexity is, to a large degree, organisational complexity wearing a technical costume. You cannot refactor it away in the code because its root is not in the code.
- Decomposition converts static coupling into runtime coupling — and hides it. Inside a single deployable system, two modules that depend on each other are coupled, but the coupling is visible and it is checked: the thing will not compile, or will not ship, if the dependency is broken. Split those modules into two services calling each other over the network, and the coupling does not disappear. It becomes a runtime relationship that no compiler checks, that fails in production rather than at build time, and that is now mediated by the network, with all the latency, partial failure, and versioning drift that implies. We often describe microservices as "loosely coupled." Frequently they are merely invisibly coupled — which is worse, because you no longer see the dependency until it breaks.
- Integration debt accrues silently and is serviced by strangers. When a team ships an API to hit a date, it makes a hundred small compromises: a field that should have been an enumeration left as free text, a versioning strategy deferred, an error contract left vague, a backward-incompatible change waved through because the only consumer today is a friendly team down the corridor. Each compromise is individually reasonable and collectively corrosive. Unlike financial debt, nobody

records it, and the interest is paid not by the team that took it on but by every future consumer of that interface, and by the programme that eventually has to reconcile a hundred such interfaces into something coherent.

- APIs are cheap to publish and expensive to keep as promises. An interface is a contract, and a contract has a lifecycle: it must be versioned, deprecated with notice, documented as it actually behaves rather than as it was first imagined, and supported for as long as anyone depends on it. The economics of this are treated as an afterthought. Publishing an API is a sprint's work and gets celebrated; maintaining it as a stable promise to consumers you may never meet is an open-ended commitment that appears on no team's roadmap. So the commitment is made implicitly, funded by nobody, and the gap between the promise the API represents and the resourcing behind it becomes one more form of debt.

The deepest problem is one of ownership. In a point-to-point world the connections were nobody's, and it showed. The bus made them somebody's, and that somebody became a bottleneck. The API era has quietly returned integration to the state of being nobody's job in particular — except that now there are far more connections, and they fail at runtime.

5. The Gap Between Intent and Reality

Here is where the structural forces meet the way we actually run programmes, and the gap opens.

The intent behind an API programme is almost always framed in the language of agility and reuse: faster delivery, less duplication, an estate that composes. The reality is governed by how the programme is funded, staffed, and measured — and those mechanisms are almost always inherited from an era that thought of integration as plumbing. Integration work is scoped as a horizontal enabler, which is programme-speak for underfunded and unloved. It has no product owner because it is not seen as a product. Its quality is invisible on any dashboard the steering committee reads, because dashboards measure features shipped, not contracts honoured. And so the very thing that determines whether the platform vision succeeds or fails — the stability, discoverability, and stewardship of the interfaces between everything — is the thing no one is explicitly accountable for.

“We say we are building a platform. We fund a collection of projects. The distance between those two sentences is where most API programmes quietly fail.”

This is why so many of these programmes deliver a large number of working APIs and, at the same time, fail to deliver the economy — the fluid, low-friction composition that was the entire point. An economy needs more than goods; it needs a marketplace, a currency, rules of exchange, and someone maintaining the institutions that make trade trustworthy. A pile of APIs with no catalogue, no consistent contract discipline, no shared conventions, and no owner for the whole is not an economy. It is a warehouse. And a warehouse that no one is curating is, given a little time, just a more modern kind of spaghetti — the connections are cleaner and better described, but they are just as numerous, just as entangled, and just as unowned.

6. The Economy Beyond the Firewall

There is a tell in the phrase API economy that is worth pausing on, because the organisations that coined it were not internal platform teams. They were the businesses whose entire product is an API — the payments providers, the messaging providers, the mapping and identity and infrastructure firms that sell capability through an interface and nothing else. It is instructive to compare how they treat their APIs with how the typical internal programme treats its own, because the contrast explains a great deal.

For a company whose revenue arrives through its API, the interface is not a by-product of the software; it is the software, and every discipline flows from that fact. The contract is sacred, because breaking it breaks customers who can leave. Versioning is meticulous and deprecation is glacial, because a partner who wrote against last year's version is still paying. Documentation is treated as a first-class product surface, written and maintained by people whose job that is. Backward compatibility is a promise defended for years. And crucially, there is unambiguous ownership: a named team, funded indefinitely, whose entire remit is the health of the interface and the trust of the people who depend on it.

The internal programme borrows the language of this world — "treat consumers as customers," "the API as a product" — while importing almost none of its economics. It wants the fluid composition that a Stripe or a Twilio enables, but it will not fund the stewardship that makes such composition trustworthy, because internally the consumer cannot leave and therefore exerts none of the market pressure that disciplines a real API business. The friendly team down the corridor will put up with a breaking change and a vague error contract, because what else can they do? That absence of exit is usually described as an advantage of internal platforms. It is closer to the opposite. The market discipline that forces external providers to honour their contracts is precisely the discipline internal programmes lack, and its absence is why internal interfaces decay in ways that commercial ones cannot afford to.

This is the deeper reason the API economy so rarely materialises inside the firewall. An economy is held together not by its goods but by trust in the rules of exchange, and trust is expensive to maintain. The firms that built the actual API economy paid that price because the market made them. The organisations trying to build a private version of it have to choose to pay it, deliberately, against no external pressure at all — and most, when the quarter tightens, choose not to.

7. Toward a More Honest Frame

If the diagnosis is that we misframe integration, the remedy is to reframe it — not to buy a different tool. Three shifts follow from the argument.

The first is to treat integration complexity as something you locate, not something you eliminate. The essential question about any architecture is not "how do we remove the coupling?" — you cannot, because the business genuinely needs its systems to depend on one another — but "where do we want the coupling to live, who feels it there, and is that the right place?" The point-to-point, bus, and API eras were each an answer to that question, whether or not their proponents knew they were answering it. Making the choice consciously is most of the battle.

The second is to treat the interface as a product with a lifecycle and an owner. An API that matters has consumers, and consumers deserve the things any product owes its users: a stable contract, honest documentation, versioning with fair notice, and someone whose job it is to care when it breaks. This is not gold-plating. It is the difference between an asset and a liability, and it costs real money that must be planned for rather than assumed.

The third, and hardest, is to govern integration as a first-class concern of the programme, not a horizontal afterthought. That means it appears on the dashboard the leadership actually reads. It means integration debt is named, tracked, and paid down deliberately, the way a serious engineering organisation tracks any other debt. And it means someone senior owns the coherence of the whole — the catalogue, the conventions, the marketplace — because as we have seen three times over now, when integration is nobody's job, its complexity does not vanish. It simply moves somewhere darker and waits.

The API is a better technology than what came before, and the API economy is a vision worth pursuing. But it will keep disappointing the organisations that chase it until they accept the thing the last three architectural generations have been trying to teach them: that the difficulty was never in the wiring, or the bus, or the interface. It was, and remains, in the boundaries — and boundaries are a matter of organisation and ownership long before they are a matter of technology.