Pontil

Why agent projects stall.

A briefing on the structural reason enterprise agent projects break down, and the realistic options for closing the gap. Drawing on integration research from Gartner, MuleSoft, and Postman, it names the surface gap between what a SaaS product can do for a person and what it can do for an agent.

Agent integration, the API-to-UI surface gap, and what to do about it.

Contents

Agent projects across enterprise SaaS are stalling at predictable rates. Gartner forecasts that over 40% will be cancelled by the end of 2027.[1] The reasons usually cited are cost, value, and governance, but the integration research from MuleSoft, Postman, and others points at a more specific problem underneath.

SaaS products have spent two decades being built for human users in the UI. The APIs that exposed those products to developers never kept pace with what the UI could do. Now agents are arriving, and they need the UI's capability set but can only reach the API's. The result is a structural gap between what a SaaS product can do for a person and what it can do for an agent.

This briefing pulls together the third-party research, names the structural cause, and outlines the realistic options for the teams already in front of it.

The big picture

In June 2025, Gartner predicted that over 40% of agentic AI projects will be cancelled by the end of 2027.[1] The stated reasons were escalating costs, unclear business value, and inadequate risk controls.

40% of agentic AI projects forecast to be cancelled by the end of 2027 — Gartner

A reasonable response is to ask whether the technology is the problem. Models have improved every six months for three years. Frameworks have matured. Tooling is no longer scarce. The leaders running these projects are not naive. So why do the projects stall?

Gartner's reasons describe the symptoms, what gets named when a project is cancelled. The integration research describes the substrate, the operational reality of where projects actually break.

MuleSoft's 2026 Connectivity Benchmark Report surveyed 1,050 IT leaders worldwide.[2] 82% cited data integration as one of the biggest challenges their organisation faces when using AI. 86% said that without proper integration, AI agents add more complexity than value. 96% agreed that agent success depends on integration. Half of all agents in production are already operating in isolated silos.

The pattern shows up in adjacent research too. Postman's 2025 State of the API report surveyed over 5,700 developers, architects, and executives.[3] 93% said their teams struggle with API collaboration. 43% admitted to rebuilding functionality that already exists because they couldn't discover or access existing APIs.

The picture is consistent across studies. Integration is not a peripheral concern. It is the operational reality of where agent projects actually break. The model is not the problem. The framework is not the problem. The agent works perfectly until it has to do something, and the something requires reaching into a product whose API was never built for what an agent needs to do.

That is where the projects stall.

What the research is telling us

Three threads run through the research worth pulling apart.

The cost is large and growing

MuleSoft's 2025 report puts the average cost of integration challenges at $6.8 million per year per organisation, in lost productivity and delayed projects.[4] The 2026 report adds a related figure: 26% of IT projects, on average, were not delivered on time in the last 12 months.[2] The friction has a number, and the number is not small.

But the financial cost is the visible part of the iceberg. Underneath, the same report shows that IT teams spend 36% of their time designing, building, and testing new connectivity.[2] More than a third of available engineering capacity is consumed by the work of making systems talk to each other, work that does not compound, work that produces no end-customer feature, work that exists because the architecture forces it.

This is the cost basis that an agent project lands on top of. Before the team writes a line of agent code, a third of their capacity is already absorbed.

Documentation is the proximate friction

Postman's 2024 survey of 5,600 developers found that 39% cited inconsistent documentation as the single biggest roadblock to API collaboration.[5] 44% said they still dig through source code to understand APIs because the docs are not enough.

These are developers integrating with APIs they have access to. They have the credentials, the endpoint, the SDK. They are still digging through source to figure out what the API actually does.

For human developers, this is annoying friction. For an agent, a system that has to generate calls from a description of available capabilities, it is a wall. The agent cannot dig through source. It cannot ask a teammate. It can only act on what the documentation tells it the API can do.

The agent layer is the new pressure point

The 2026 MuleSoft data shows what happens when the integration friction meets the agent push.[2] 88% of organisations report partial or full agent deployment underway. The average enterprise now runs 12 agents in production. By 2027, that number is projected to grow 67%.

Onto a substrate that already loses 36% of capacity to integration work, and where 82% of leaders cite integration as the biggest AI challenge, the industry is now deploying agents at scale. The pressure goes somewhere. The MuleSoft data shows where: 50% of agents are operating in isolated silos, 64% of IT leaders are concerned about meeting near-term AI goals, and 86% agree that without integration the agents add more complexity than value.[2]

The pattern

The research is describing the same phenomenon from three angles. The financial pressure shows up as $6.8M and 26% late delivery. The technical pressure shows up as 36% of developer time, inconsistent documentation, and developers digging through source. The agent pressure shows up as silos, cancellations, and concern about AI goals.

The disparate findings are not loose threads. They are three views of one thing: a generation of SaaS products that cannot be reached at the depth that human users have always been able to reach them, hitting a generation of agent projects that need that depth to function.

That structural mismatch is the subject of the next section.

The structural reason: UI-first, API-behind

Agent projects keep hitting a wall. The wall has a structural cause. It is worth naming.

The decade we just lived through

Most B2B SaaS products in the market today were built between 2010 and 2022. In that window, the dominant unit of value was the UI experience. Customers signed contracts because the product was good to use. Sales conversations turned on what the interface could do. Product roadmaps were calibrated to what users could see on screen.

The API got built too. But it was built for a narrower job: data sync, mobile clients, partner integrations, occasionally a developer ecosystem. It was a side surface, not the main one.

There are exceptions. Products built API-first, where the UI is a real client of the same surface developers and partners use, do not have this problem in the same way. Stripe, Linear, and similar spec-driven platforms designed for the API to be the contract from day one. They are the minority of the established B2B SaaS market, but they exist, and the gap is narrower for them. The rest of the market, UI-first SaaS and code-driven platforms, is where the structural problem accumulates.

For the rest, the narrower API was a rational architectural choice. APIs that exposed the full capability of a UI would have been more expensive to build, more expensive to keep stable, and more exposed to security and abuse problems. Building the API to cover the same surface area as the UI required a sustained investment that the commercial logic did not support. So the API surface stayed narrower than the UI surface, and the gap accumulated.

What 2025 measured

Postman's 2025 State of the API report asked 5,700 developers, architects, and executives a direct question: who are you designing your APIs for?[3]

60% said they design APIs primarily for humans, not for AI agents. 24% said they design with agents in mind. 13% said they design equally for both.

The number that matters is not the 24%, or even the 60%. It is the structural fact underneath: the API surface of the average B2B SaaS product was never designed for what agents need it to do.

What agents need that the API doesn’t have

The UI of a B2B SaaS product is rarely a thin layer over the API. The UI contains workflow logic. It contains permission checks. It contains validation. It contains multi-step transactions composed between the form and the data store, sometimes in a controller, sometimes in a service layer, sometimes split across both. None of this is exposed in the API by default.

When an agent tries to reach into the product, it cannot use the UI. It has to use the API. And the API is the narrower surface, the one that exposes data, not workflows, objects, not transactions, the things that were cheap to expose, not the things that the product actually does.

This is the gap. The same product, viewed from a UI session, can do what business users have spent years asking it to do. Viewed from an API call, it can do considerably less. The agent inherits the smaller surface.

The gap is wider than missing endpoints

The surface gap is wider than missing endpoints. In practice it shows up in four shapes.

- **Missing endpoints.** The simplest case. The action a user can take in the UI has no corresponding API call. The agent cannot reach it because it does not exist.
- **Wrong semantics.** The endpoint exists, but it does not behave the way an agent needs it to. The API returns objects when the agent needs transactions. It accepts updates one at a time when the workflow is fundamentally multi-step. It models the data, not the action.
- **Mismatched permission models.** The endpoint exists and behaves correctly, but the API's permission checks are coarser than the UI's. An agent operating on a user's behalf can reach things the user couldn't through the interface, or the inverse, where the API blocks an action the user is entitled to perform.
- **Invisible state preconditions.** The endpoint exists, behaves correctly, and the permission model is fine, but the call fails because it assumes state the UI builds invisibly. A user clicking through the UI passes through validation, context resolution, and session-derived state that the controller assembles before the final action. The API exposes the final action but not the path to it. An agent calling the endpoint directly has no way to know it needed to resolve a pricing context, load a configuration ruleset, or establish a session-scoped token first. The endpoint is reachable. The state machine the UI runs to make it usable is not.

These are not edge cases. They are where teams trying to close the gap actually spend the time they did not expect to spend. An assessment that only counts endpoints will miss them. An assessment that scores the API against what the UI can actually do, its semantics, its permission model, its invisible state machine, will find them.

The shorthand 'surface gap' still applies. The work of closing it is broader than the shorthand suggests.

Why this isn’t a documentation problem

It is tempting to read the documentation findings, 44% of developers digging through source, 39% citing inconsistent docs, and conclude that better documentation would close the gap.

It would not. Better documentation makes the existing API more useful. It does not make the API more capable. The agent could have a perfectly documented map of every endpoint and still be unable to do what a human user can do through the interface, because the endpoint does not exist.

This is the diagnostic move that matters. The agent stall is not a documentation problem, a model problem, a framework problem, or a tooling problem. It is a surface gap: the API surface is structurally smaller than the UI surface.

The hardest part of this diagnostic is not naming the problem once you have found it. It is recognising it during the weeks when it presents as flaky agent behaviour, calls that succeed but produce the wrong state, returns that look stale, intermittent failures that point at prompt engineering or model reliability. The surface gap does not announce itself.

Naming the problem this way is the prerequisite for choosing the right response.

What to do about it

For a team facing this problem, there are realistic options. None of them is comfortable.

Rewrite the API layer

The technically correct answer. Build the API surface that the product should have had all along, the one that exposes everything the UI exposes, with the same permission model, the same workflow semantics, the same coverage.

This is a multi-year project for any established SaaS product, commonly two to five years, depending on starting architecture and team capacity. It competes with the existing roadmap. It does not produce new features customers can see. It does not move competitive positioning in the short term. Leadership rarely funds it.

If the business case can close, this is the durable answer. For most teams asking the question, it cannot.

Build bespoke connectors per product

If the full rewrite is too big, the alternative is to expose what the agents need, one piece at a time, on demand. Each agent use case gets a bespoke connector mapped to whatever combination of internal APIs, scraping, or workflow stitching is required.

This works for one product. It does not scale across a portfolio. The maintenance burden compounds with every new integration. The team becomes the bottleneck for every agent feature anyone wants to ship. Eventually the cost of maintaining the connectors exceeds the value of the agents.

Works short-term. Breaks at scale.

Wait for the foundation model providers

OpenAI, Anthropic, and others are building tool catalogues. Protocols like MCP are emerging too. Postman's 2025 survey found 70% of developers aware of MCP, and only 10% using it regularly.[3] But MCP is a description layer. It makes existing API surface more legible to agents. It does not create new surface. If you are stalled on missing endpoints, wrong semantics, or invisible state preconditions, MCP does not help you. The substantive question is who closes the surface gap.

Every integration platform that has come before has followed the same pattern: cover the top 20% of products that account for 80% of the value, Salesforce, HubSpot, Slack, and leave the long tail uncovered. The economics that drive that choice have not changed.

If the product is in the top 20%, this is a real option. If it is in the long tail, most vertical SaaS, most platforms built through acquisition, most products outside the headline list, waiting does not solve the problem. It defers it.

Close the gap deliberately

The fourth option is to treat the API-to-UI gap as a measurable thing and close it on purpose, with the same discipline applied to any other piece of infrastructure. Inventory the gap. Prioritise the workflows that matter most to agent use cases. Generate the missing surface from the codebase rather than hand-building it. Automate the maintenance.

This is the option Pontil exists to make practical. It is not the right answer in every situation. Where it is, the work moves from a multi-year architectural commitment to an incremental closing of a measurable gap.

How to scope the conversation internally

Before choosing, name the problem cleanly to leadership. The framing that lands:

> *Our product's API exposes a fraction of what the UI exposes. Agents can only use the API. Without closing that gap, the agent project cannot deliver on what we promised.*

That sentence converts a vague AI-strategy conversation into a specific architectural one. The next question, how to close the gap, at what cost, on what timeline, is the conversation worth having.

Sources and further reading

Primary sources

1. **Gartner.** *Gartner Predicts Over 40% of Agentic AI Projects Will Be Canceled by End of 2027.* Press release, 25 June 2025. Underlying report: *Emerging Tech: Avoid Agentic AI Failure: Build Success Using Right Use Cases.* Analyst: Anushree Verma. gartner.com/en/newsroom/press-releases/2025-06-25-gartner-predicts-over-40-percent-of-agentic-ai-projects-will-be-canceled-by-end-of-2027
2. **MuleSoft.** *2026 Connectivity Benchmark Report* (11th annual). Survey of 1,050 IT leaders worldwide, conducted in late 2025 with Deloitte. mulesoft.com/lp/reports/connectivity-benchmark
3. **Postman.** *2025 State of the API Report.* Survey of over 5,700 developers, architects, and executives. postman.com/state-of-api/2025
4. **MuleSoft.** *2025 Connectivity Benchmark Report* (10th annual). Survey of 1,050 IT leaders, October to November 2024, conducted with Vanson Bourne and Deloitte Digital. blogs.mulesoft.com/news/connectivity-benchmark-report
5. **Postman.** *2024 State of the API Report.* Survey of 5,600 developers. postman.com/state-of-api/2024

About Pontil

Every product reaches the point where it needs to connect — to the systems it runs alongside, the tools its users depend on, and the agents now acting on the work inside it. Most weren't built for that, and building it after the fact is slow and expensive to maintain. Pontil generates and maintains that connectivity layer from your existing codebase, without a rewrite — delivered across three modules: Pontil Tools, Pontil Headless, and Pontil Integrations.

Learn more at pontil.com