Where should customer complexity live?

What history shows about scaling a standardized product as enterprise customers demand more complexity.

Enterprise customers create exceptions. The question is not whether those exceptions exist, because they always do. It is where the work created by them ends up. **Complexity does not disappear when a company holds its architecture. It gets assigned somewhere, sometimes without anyone deciding.**

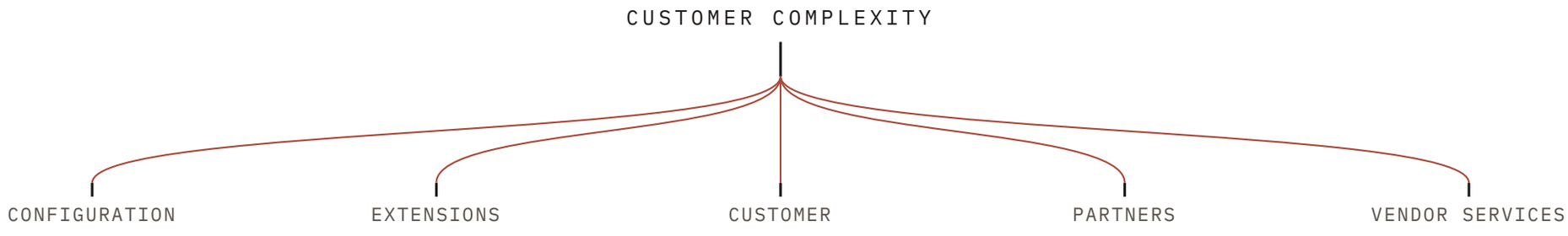
THE EXECUTIVE QUESTION

Where did the last customer's complexity go?

THE FIVE DESTINATIONS

Follow the work

A requirement that does not fit the product has to be absorbed by someone. There are five places it can go, and each produces a different company.



DESTINATION	WHAT IT MEANS	WHO CARRIES THE WORK	WHERE THE COST APPEARS	SCALING RISK
Product configuration	The platform already has a model for the requirement. The customer sets values inside a schema the vendor owns.	The vendor builds and maintains the abstraction; customers reuse it.	Mostly in product and platform R&D, with relatively low incremental cost per customer.	The schema stops fitting, and requirements start arriving that have no place to go.
Controlled extensions	Customers or partners add behavior inside a runtime and a contract the vendor defines and versions.	The customer or an app developer, inside the vendor's boundary.	In platform engineering, and in the vendor's testing surface.	The contract has no version or no deprecation path, so the extension becomes permanent.
Customer environment	Custom code, workflows, and integrations accumulate in the customer's own instance or estate.	The customer's IT organization.	Nowhere in the vendor's accounts. In the customer's cost of change.	Customer-specific code can accumulate outside the vendor's visibility while the vendor's own release metrics remain healthy.
Implementation partners	Systems integrators configure, integrate, and migrate, often billing the customer directly.	The partner's consultants.	On the partner's income statement and the customer's budget.	The vendor loses sight of how its product is actually deployed, and inherits partner quality.
Vendor services	The vendor's own people configure, integrate, migrate, and support each enterprise customer.	The vendor's services organization.	Immediately, in services gross margin.	Growth becomes headcount-bound, and delivery capacity gates new logos.

WHY IT MATTERS FINANCIALLY

Software margins do not tell you what complexity costs

Two companies, two structures, and in both cases the software economics and the deployment economics sit far apart inside the same business.

GUIDEWIRE · FY2025

70% Subscription and support gross margin

13% Services gross margin

The software and the work required to deploy it can have completely different economics inside one company, in one year, across one customer base.

ZUORA · FY2019

77% Subscription gross margin

55% Blended gross margin

25% Professional services as a share of revenue

When the vendor carries implementation itself, the cost surfaces in its own accounts. Zuora disclosed a quarter in which services cost more to deliver than they generated.

You cannot read the cost of customer complexity from software margins alone.

The same underlying difficulty can produce a high services line, a low one, or none at all, depending on how delivery was structured. If implementation moves to systems integrators, the vendor's statements can improve even when substantial deployment work still remains outside the vendor. Everything in the accounts stays accurate. The picture is still incomplete. **You have to follow the work.**

WHAT HISTORY SHOWS

Four things the record actually supports

Each rests on a company's own filings, changelogs, or documentation rather than on commentary about them.

Build the abstraction before the variations arrive

ADYEN

Adyen supports more than 100 payment methods on one platform, and those methods behave differently from each other in flow, failure, and settlement. Its own documentation shows a payment method being requested and configured per store, with bulk actions for adding several at once.

The hundredth payment method entered a model that already existed. A genuinely novel requirement would require the model itself to change.

DIAGNOSTIC

When a large customer asks for something unusual, are you adding another instance to an existing model, or inventing a new model?

Know what you can take back

SALESFORCE

Salesforce retires old API versions on a published schedule, and retired calls fail outright. Its own retirement notice states that the retirement does not include Apex classes, Apex triggers, Visualforce pages, or versioned metadata components in managed packages.

Old tenant-resident code is not swept up by that mechanism. Salesforce holds a hard boundary at the integration layer and declines to enforce one inside the customer's instance.

DIAGNOSTIC

Which customer-specific behaviors could you actually retire, and which have effectively become permanent?

Build the replacement before removing the old surface

SHOPIFY

Checkout extensibility became available on all Shopify Plus stores on 30 January 2023. Fourteen days later Shopify announced that checkout.liquid was deprecated and would stop working for in-checkout pages on 13 August 2024.

Merchants had a supported destination in market before the deprecation was announced, then roughly eighteen months before the first enforcement date.

DIAGNOSTIC

If you removed your hardest-to-maintain customization surface, what would customers migrate to?

Decide who carries the implementation labor

NCINO · GUIDEWIRE · ZUORA

nCino's filings describe systems integrators performing most implementation for enterprise and larger regional institutions, billing those customers directly, while nCino delivers services itself at the small end. Guidewire cut a full-year revenue outlook in March 2024 because partners were leading cloud engagements faster than expected. Zuora kept the work in-house.

Implementation complexity can be moved. It cannot be declared gone.

DIAGNOSTIC

Whose employees do the work for your largest customers, and whose invoice carries it?

THE DECISION

Where should the complexity go?

None of these is superior to the others. They carry different costs, and the cost of choosing by accident is higher than the cost of choosing any one of them deliberately.

Put it in the product when	Put it in an extension layer when	Put it with partners when	Keep it in vendor services when
The requirement repeats across customers	Customer variation is legitimate and specific	Implementation labor is substantial	The implementation knowledge is strategically valuable
It fits a general abstraction you can name	You cannot predict every use case	Deployment needs domain expertise you lack	The product is still learning from deployments
You can own and version it	Behavior can sit inside a stable contract	You want to preserve software economics	The work cannot yet be standardized
Each new instance should get cheaper	You know how that contract will evolve	The work can be done without fragmenting the product	You are knowingly accepting the margin trade

Services intensity is not automatically failure. Guidewire has run the last two of these together, taking capped-fee implementations to accelerate cloud migration while shifting engagements to partners. Adyen absorbed enormous variation on one platform for two decades. Workday combined a single codeline with a customer extension platform for years. The pattern is deliberate placement rather than elimination.

THE FAILURE MODE

The dangerous state is accidental complexity

The dangerous state is not customization itself. It is customer-specific work accumulating somewhere without anyone deliberately deciding that is where it should live.

- Engineers repeatedly add one-off logic for a named account
- Customer-created behavior has quietly become impossible to remove
- Services hours rise while the work is still called configuration
- Implementation moved to partners and nobody measures what customers now pay
- New requirements keep needing new concepts rather than new values
- Old behavior cannot be retired because there is no supported migration path

Each of these is survivable alone. What makes them costly together is that none of them arrives as a decision. They surface later, as a deployment timeline nobody can explain.

TRIAGE

How to route a new requirement

Four questions, then one test that matters more than the route you took to reach it.

01 **Does a general model already exist?**

`Yes → configuration. Add the instance and move on.`

02 **Will the requirement repeat?**

`Yes → expand the product abstraction now, while there is one instance to design against.`

03 **Can it live inside a controlled extension boundary?**

`Yes → extension layer, on a versioned contract you can later change.`

04 **If not, who should carry the work?**

`Vendor services · Partner · Customer`

> Then the test: is that destination intentional, measured, and priced accordingly?

- Know what the product **absorbs**.
- Know what customers can **create**.
- Know what can be **removed**.
- Know who carries the remaining **labor**.

SELF-ASSESSMENT

Five questions for your own company

Answerable this week by people who already know. Mark each honestly.

Question one

When the last three large customers asked for something unusual, did engineers add an instance to a model that already existed, or invent a new one?

WE KNOW | PARTLY | DON’T KNOW

Question two

Which customer-specific behaviors could you actually remove, and on what notice?

WE KNOW | PARTLY | DON’T KNOW

Question three

If you eliminated your most problematic customization surface, what would customers move to?

WE KNOW | PARTLY | DON’T KNOW

Question four

Who performs the majority of implementation work for your largest customers, and on whose invoice does it appear?

WE KNOW | PARTLY | DON’T KNOW

Question five

Are implementation economics improving as you scale, or moving off your financial statements?

WE KNOW | PARTLY | DON’T KNOW

Answer the five above. This is not a maturity score. It measures how much of your customer complexity currently has a destination somebody can name.

UNRESOLVED COMPLEXITY

A company can maintain one codebase, one release cadence, and one version of its product long after customer complexity has moved somewhere the architecture diagram does not show.

It may be sitting in customer code, in an implementation partner’s delivery practice, in a services organization, or in a product abstraction that absorbed it successfully years ago and never made a sound.

So ask where the last customer’s complexity went. And whether that is where you intended it to go.

SOURCES

Guidewire — Q4 and full year fiscal 2025 results, 4 September 2025; fiscal 2025 annual report; Q2 fiscal 2024 results, 7 March 2024.

Zuora — Form 10-Q for the quarter ended 30 April 2019; analyst day presentation filed as an 8-K exhibit, fiscal 2021.

nCino — annual reports on Form 10-K, fiscal 2023 and fiscal 2026.

Adyen — developer documentation on payment method configuration; annual reports 2024 and 2025.

Shopify — changelog, 30 January 2023; developer changelog, 13 February 2023; Help Center upgrade guidance.

Salesforce — platform API version retirement notices and developer end-of-life documentation.

Workday — company blog on one version and one codeline, June 2015; Workday Extend general availability, 2020.

Figures are as reported by each company, including non-GAAP measures where the company presented them that way. Claims that could only be traced to consultancies or vendors selling remediation services were excluded.