PRINCIPAL TECHNICAL PRODUCT LEADERSHIP CASE STUDY

From Fragmented Systems to a Reusable Platform

An anonymized case study in defining shared integrations, data contracts, and delivery mechanisms for an enterprise cloud transformation.

Prepared by Mandie Morgan | Technical Product & Platform Leadership

Executive story. I led a confidential proof of concept that established the feasibility of a new cloud platform and shared integration layer, then became the full-time Sales/New Business Lead for implementation. I defined the roadmap, scope, service and interface expectations, requirements, acceptance path, and executive decision context. The operating mechanisms I introduced improved vendor quality, accelerated feature delivery, and created a repeatable foundation for extending the platform to additional business capabilities.

Impact at a glance

3X+	15-18 TO 2-3	3-4 WKS.	6 TEAMS
Feature completion: 4 to 12+ monthly	**Open vendor-delivery defects**	**POC development avoided**	**9 engineers at peak**

The platform problem

A large enterprise faced declining support for a legacy policy administration platform. The target was a stronger vendor-operated cloud platform with a shared integration layer that could connect systems, normalize data, and support reusable capabilities rather than reproduce isolated point solutions.

The fragmentation was visible to customers and operations: lines of business could send redundant communications, while policy and claims systems duplicated information because data was represented and consumed differently. The product challenge was therefore broader than a system migration - it required clearer service boundaries, consistent data contracts, and coordinated adoption across teams.

Role progression

Phase	My mandate	Evidence and outcome
Aug-Dec 2025 Proof of concept	POC development lead for five months at 70% allocation while also serving as Business Architect on another initiative	Set roadmap and scope, owned requirements and stubbing decisions, led executive readouts, and coordinated six teams with nine engineers at peak.
Jan 2026 onward Implementation	Full-time Sales/New Business Lead on a newly formed implementation team after the POC team disbanded	Own integration and API expectations, vendor and UI/technical/business requirements, design and functionality acceptance, executive strategy, and the plan to extend the platform to Service.

Define the right service boundaries

Two decisions show how I connect customer outcomes, platform architecture, and delivery constraints to define reusable technical work.

Decision 1 | Isolate the feasibility question

Constraint. The proof of concept was confidential, so the team could not connect to the billing and payments suite. Some engineering leaders initially favored a more complete end-to-end build despite that limitation.

Product judgment. The architecture needed to prove that the new platform could orchestrate the quote flow, connect to rating, return accurate coverage, and produce a reliable customer price. Payment collection was not the critical feasibility question.

Decision. I challenged the initial direction, sought alternative technical views, and recommended stubbing the unavailable services after validating that the approach would preserve the integrity of the test and its interfaces.

Outcome. The decision avoided approximately three to four weeks of development, concentrated engineering capacity on the highest-value service path, and helped leadership approve the transition from feasibility to implementation.

Reusable lesson. A strong platform test isolates the capability that must be proven, defines the surrounding interfaces, and avoids confusing end-to-end completeness with architectural evidence.

Decision 2 | Make customer-facing data the contract

Gap identified. The legacy administration system represented UI selections differently from the new customer experience, but the vendor had received requirements tied to the legacy representation. Implementing them as written would return the wrong data.

Technical analysis. I mapped the customer-visible data points, the internal API's responsibilities, the vendor responses we should receive, and the path through the experience. I verified payloads and behavior with Splunk, browser developer tools, and Dynatrace.

Action and result. I redirected requirements toward the data shown to customers and translated the findings into business, technical, UI, interface, and acceptance expectations for the vendor, engineers, and Experience Design. This prevented a legacy representation from becoming the new cross-system contract.

Principal product mechanisms I owned

Strategy	Set the roadmap, scope boundaries, capability sequence, dependencies, and executive decision path.
Service definition	Translate workflows into API responsibilities, interface expectations, data behavior, acceptance criteria, and reusable capability boundaries.
Validation	Review Figma prototypes, accept functionality, and use observability and AI-assisted repository analysis to verify behavior.
Alignment	Connect customer needs, vendor delivery, architecture, engineering capacity, and leadership decisions across teams without formal authority.

Build reliability into the delivery system

For roughly two months, vendor deliveries repeatedly generated defects. Rather than managing the tickets as unrelated symptoms, I reviewed the root cause of every defect. Most traced to missing product-configuration data: the vendor had restructured configuration but had not integrated the new structure into its requirements-consumption and backend-delivery process.

Root cause, not symptom management. The code and the configuration required to make the feature work were traveling through separate delivery assumptions. Reliability required a shared handoff standard, not faster defect triage.

The lightweight quality and governance gate

Vendor pre-check	Before handoff, the vendor verifies the product-configuration changes associated with its service or feature delivery.
Product validation	The lead architect and I compare available configuration, release notes, interface expectations, and expected behavior.
Engineering handoff	I document the validated changes and give the development team an implementation-ready view of the work.

Align the organization to the platform need

The handoff control addressed quality, but the delivery model still lacked the right UI depth. As the work became more interface-heavy, I recommended adding engineers from the UI team that owned the affected experience. Leadership agreed, and three UI engineers joined the implementation team, strengthening domain knowledge and delivery confidence for a group that had been primarily API-focused.

Measured change and responsible attribution

Signal	Before	After the intervention
Open vendor-related defects	Approximately 15-18	2-3 in the current backlog after the quality gate
Feature completion	Approximately 4 per month	More than 12 per month after the process and staffing changes
UI capability	Implementation team primarily API-focused	Three UI engineers added from the team that owned the experience

The defect figures compare backlog snapshots, not normalized defect rates. The throughput improvement followed the combined quality-gate and staffing changes and is not attributed to one action alone.

Scale through reuse and adoption

With New Business operating as a more stable program, I was given responsibility for planning how to bring Service into the transformation. The expansion reuses the same product principles: customer-centered requirements, explicit service and data contracts, validated interfaces, targeted governance, and decision-ready leadership context.

Principal product pattern. Start with customer and operational outcomes, identify the capability that should be shared, define durable boundaries and acceptance criteria, make tradeoffs explicit, and build the governance required for adoption across products.