International Journal of Artificial Intelligence, Data Science, and Machine Learning
Grace Horizon Publication | Volume 7, Issue 3, 71-82, 2026
ISSN: 3050-9262 | https://doi.org/10.63282/3050-9262.IJAIDSML-V7I3P110

Original Article

Enterprise Digital Transformation in Practice: Modernizing Mission-Critical Business Systems Through Enterprise Architecture

Ashok Apati
Independent researcher, USA.

Received On: 17/06/2026 ***Revised On: 20/07/2026*** ***Accepted On: 27/07/2026*** ***Published On: 04/08/2026***

Abstract - *Large consumer-brand enterprises rarely fail at digital transformation in a single visible event. They accumulate cost quietly, through duplicate customer records, slow onboarding, manual support work, integration defects, and elevated risk every time a release touches a mission-critical system. This paper reports a practitioner case study of a multi-year modernization program at a large consumer-brand enterprise whose customer-facing estate spanned CRM, marketing automation, B2B and direct-to-consumer commerce, customer identity, order management, payments, and ERP. The systems were connected. They were not unified, and the customer journey they collectively served had never been designed as one system. The paper describes the architectural approach applied across eight named workstreams, including identity modernization on OKTA, a registration redesign, direct-to-consumer commerce modernization, order management and order-to-cash enablement, payment modernization across CyberSource and Adyen, a B2B commerce upgrade, an ERP transformation, and cloud modernization using Azure services. The central contribution is a reusable five-step method: start from the business capability rather than the application, map upstream and downstream dependencies, classify each integration as real-time or scheduled, design API-first and integration-first, and treat operational readiness as part of the architecture rather than a phase that follows it. Reported outcomes include an estate serving a large customer base with a comparable annual order volume, several dozen enterprise APIs, registration processing improved to sub-second, and meaningful operational cost savings from cloud modernization. Several of the most consequential outcomes are expressed as risk reduction rather than as improvement metrics. The paper also scopes, honestly, what artificial intelligence readiness did and did not mean in this program.*

Keywords - *Enterprise Architecture, Digital Transformation, Systems Integration, Identity and Access Management, Legacy Modernization, Cloud Migration, Practitioner Case Study.*

1. Introduction

Digital transformation is often discussed as a strategy problem and delivered as a development problem. The gap between those two framings is where most of the difficulty lives. A leadership team commits to a better customer experience; a delivery team receives a backlog of screens, endpoints, and platform upgrades. What sits between them, the architecture of how business capabilities are decomposed across systems and how those systems exchange data, tends to be assumed rather than designed.

This paper argues, from a single extended practitioner case, that transformation in a large enterprise is primarily an architecture problem. The claim is not that architecture matters more than delivery. It is narrower and more specific: in an estate where seven or eight platforms each own a fragment of one customer journey, the decisions that determine whether modernization succeeds are decisions about capability boundaries, dependency direction, integration classification, identity ownership, and data integrity. Those decisions are made before code is written, and they are expensive to reverse afterward.

The setting is a large consumer-brand enterprise operating both B2B commerce and direct-to-consumer digital experiences. Over years, the estate accumulated the way estates usually do, one business need at a time. Salesforce CRM served relationship management. Salesforce Marketing Cloud served engagement. Separate B2B and D2C commerce platforms served ordering. An identity layer served authentication. An order management system served fulfillment. An ERP served pricing, product, and financial truth. Payment platforms served transactions. Each of those decisions was defensible in isolation. The aggregate result was a customer journey assembled from parts, none of which had been designed with the whole journey in view.

The symptoms of that condition are familiar to practitioners and are worth stating plainly, because they are how the problem actually presents. Onboarding a customer takes longer than it should. The same customer exists more

than once, in slightly different forms, across systems. Support teams do manual reconciliation that the architecture should have made unnecessary. Order processing stalls at handoffs. Integration defects appear at the seams rather than inside any one platform. Launching a feature requires coordination across teams who do not share a model of the estate. And every upgrade or release carries operational risk out of proportion to its technical size, because nobody is confident about what depends on what.

Three contributions are offered. First, the paper documents a reusable five-step modernization method that was applied consistently across workstreams of very different technical character, from identity to ERP to payments. Second, it reports the trade-off decisions made at each major fork, including point-to-point versus governed middleware, real-time versus batch, buy versus build, and modernize-now versus stabilize-first, with the reasoning stated rather than only the outcome. Third, it scopes the role of artificial intelligence in this program honestly. The program did not deploy production AI. What it did was build the modernized, connected, governed, and data-ready foundation that AI-enabled capability depends on, and evaluate practical use cases against that foundation. Treating that distinction carefully is, the paper argues, more useful to practitioners than a broader claim would be.

The paper is organized as follows. Section 2 positions the work against enterprise architecture, integration, identity, and cloud migration literature. Section 3 describes the case context and scale. Section 4 presents the five-step method, which is the core contribution. Section 5 covers integration design and identity modernization. Section 6 covers reliability, governance, and data integrity. Section 7 reports outcomes. Section 8 discusses trade-offs, AI readiness, lessons, and limitations. Section 9 concludes.

2. Related Work

2.1. Enterprise architecture as a foundation for execution

The framing this paper adopts, that architecture is the mechanism by which strategy becomes executable, follows Ross, Weill, and Robertson's argument that enterprises benefit less from any particular technology choice than from a deliberately designed foundation for business execution [1]. Their central observation, that firms which standardize and integrate their core processes gain optionality they cannot otherwise buy, matches what this case found in practice: the value of the modernization was not any single platform, but the reduction in coupling between platforms.

Empirical work has since tested how enterprise architecture actually produces benefit. Tamm, Seddon, and Shanks find that architecture delivers value indirectly, through improved project delivery, better-informed decisions, and increased organizational agility, rather than through direct technical effect [2]. That indirection is visible in the present case, where several of the strongest outcomes are hard to express as a metric and are better stated as continuity preserved during high-risk change. The TOGAF standard provides the vocabulary for capability-oriented decomposition used here, particularly the separation of business capability from application realization [3], though the program applied that separation pragmatically rather than adopting the full method.

2.2. What digital transformation means, and what it costs

Vial's review consolidates a fragmented literature into a model in which digital technologies trigger structural change, and in which organizational barriers, not technical ones, are the dominant constraint [4]. This matches the case directly. The hardest problems were not connections or code. They were understanding how business process, customer experience, enterprise data, integration, security, and operations interacted, and who owned each.

Warner and Wager treat transformation as ongoing strategic renewal rather than a program with an end date [5]. That framing is adopted here explicitly, and it shapes one of the paper's stated limitations: the architecture described is not finished, and was not intended to reach a finished state.

2.3. Integration patterns and the cost of point-to-point growth

Hohpe and Woolf's catalog of enterprise integration patterns remains the reference vocabulary for the messaging, routing, and transformation decisions described in Section 5 [6]. Their treatment of the trade-off between synchronous request-response and asynchronous messaging is the conceptual basis for the classification step in the method presented here.

Themistocleous and Irani's benchmarking of application integration benefits and barriers documented what practitioners observe repeatedly, that point-to-point connections are cheap at low counts and become the dominant maintenance burden as the estate grows [7]. The present case reached that threshold in a specific way: with several dozen enterprise APIs and 25 active integration processes spanning commerce, ERP, OMS, identity, payments, and marketing, ungoverned direct connections would have made dependency analysis impossible before any change.

2.4. Modernization strategy: incremental over big-bang

Newman's treatment of monolith-to-microservices migration argues for evolutionary, incremental decomposition with the ability to run old and new paths in parallel [8]. The phased migration and parallel validation approach used in this program follows the same logic, though the program was not a microservices decomposition and should not be read as one.

Hasan and colleagues review legacy-to-cloud migration from an architectural perspective and identify dependency opacity and data consistency as recurring failure modes [9]. Di Francesco, Lago, and Malavolta's systematic mapping of microservice architecting, and Fritzsch and colleagues' industrial study of migration intentions and challenges, both report that organizations underestimate the operational and governance work relative to the decomposition work [10],

[11]. Fritzsch's finding that practitioners struggle most with data management and the synchronous-versus-asynchronous boundary is consistent with what this program encountered at the commerce-to-ERP and commerce-to-OMS seams.

2.5. Identity and access management

Glockler and colleagues' systematic review of enterprise identity and access management requirements establishes that IAM is simultaneously a security control, a compliance instrument, and an experience layer, and that treating it as only one of those produces predictable gaps [12]. The identity workstream described in Section 5.3 was designed against all three, and the review's framing is useful for explaining why identity could not be modernized independently of registration, consent, and CRM.

The NIST digital identity guidelines on federation and assertions inform the token-based access and federated single sign-on patterns applied, including the receivables platform SSO integration [13]. These are used as a standards reference rather than as an evaluated framework.

2.6. Omnichannel expectation and integration quality

Verhoef, Kannan, and Inman describe the shift from multi-channel to omni-channel retailing, in which channels stop being separate and customers expect one continuous relationship [14]. That expectation is the business driver behind the entire case; a customer who registers, browses, orders, pays, and receives service does not experience seven systems, and does not accept seven versions of themselves.

Hossain and colleagues reconceptualize integration quality as the mechanism that produces perceived seamlessness in omnichannel marketing [15]. Their argument, that the integration layer is not plumbing behind the experience but a constituent of it, supports this paper's position that integration architecture is the backbone of transformation rather than a supporting concern.

2.7. Master data and cloud migration

Vilminko-Heikkinen and Pekkola's study of master data management implementation finds that MDM fails as a technical project and succeeds only as an organizational one, with ownership and stewardship as the determining factors [16]. This is directly relevant to the duplicate-customer-record problem in the case, and to the decision to treat data integrity as a first-class architectural requirement rather than a data-team concern.

Jamshidi, Ahmad, and Pahl's systematic review of cloud migration research provides the taxonomy of migration types used to reason about the Azure workstreams [17]. Khajeh-Hosseini, Greenwood, and Sommerville's case study of migrating an enterprise IT system to infrastructure-as-a-service is cited here for two reasons: for its cost and continuity findings, and as methodological precedent, since it demonstrates that a single well-documented enterprise migration case can contribute usefully to the literature even without controlled comparison [18]. The present paper follows that precedent and inherits its limitations.

2.8. Gap addressed

The literature is strong on architectural principle [1], [2], [3], on integration pattern vocabulary [6], and on what goes wrong in migration [9], [11]. It is thinner on documented accounts of how a single practitioner applies one consistent method across workstreams as heterogeneous as identity, commerce, ERP, payments, and cloud, in one estate, under continuous business operation. This paper contributes that account.

3. Case Context and Scale

3.1. The organization and its estate

The setting is a large enterprise consumer-brand organization. The employer is not named. The organization operates B2B commerce for trade and channel customers alongside direct-to-consumer digital experiences, and supports customer registration, customer identity, marketing engagement, ERP operations, order management, payment workflows, and enterprise integrations.

The platform estate includes Salesforce CRM and Salesforce Marketing Cloud, separate B2B and D2C commerce platforms, OKTA for customer identity, an ERP system, an order management system, CyberSource and Adyen for payments, MuleSoft and other integration services, Azure cloud and integration services including Azure Data Factory, consent-management platforms, and custom customer registration applications.

3.2. Scale

The modernized platforms serve a large customer base with a comparable annual order volume. Business operations running on this estate are associated with a large North American revenue base. The integration estate comprises several dozen enterprise APIs, 15 real-time integrations, and 10 scheduled integration processes.

These figures describe the environment the architecture had to support. They are stated as association, not as attribution. Revenue at this scale is produced by a business, not by an integration layer; the relevant point is that the systems described carry transactions of that magnitude, which is what makes them mission-critical and what constrains every modernization decision that follows.

3.3. The before state

Each platform in the estate supported a different part of the customer journey. CRM held the relationship. Marketing Cloud held engagement. Commerce held browsing and ordering. Identity held authentication. Order management held fulfillment lifecycle. ERP held pricing, product, and transaction truth. Payment platforms held authorization and settlement. The journey itself, the continuous path a customer takes from registration through ordering to service, was not owned by any of them.

As shown in Figure 1, the result was a set of systems that were connected but not unified, with customer-facing capability distributed across platform boundaries that had

been drawn for reasons of procurement history rather than journey design.

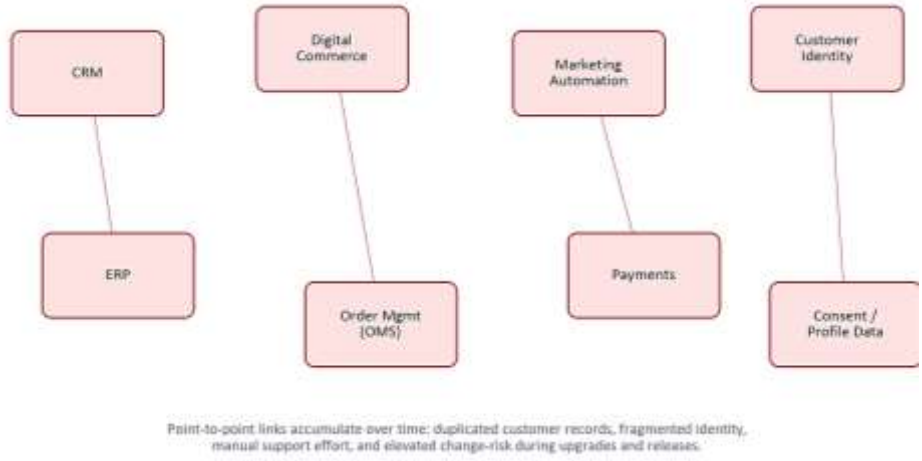

Fig 1: Before-State Architecture of Siloed Customer-Facing Systems

The consequences did not present as an outage. They presented as friction: slower customer onboarding, duplicate customer data across systems, manual support effort to reconcile records, inconsistent customer records depending on which system was consulted, delays in order processing at system handoffs, integration defects concentrated at the seams, difficulty launching new features because each launch touched several platforms, and elevated operational risk during upgrades and releases because dependency knowledge was distributed across people rather than documented.

This condition is common across large consumer brands, and the reason is structural rather than a matter of any one firm's decisions. Most run a mix of legacy platforms, cloud systems, ERP dependencies, and third-party tools, under customer expectations that rise faster than the estate can be rebuilt [14], [15].

3.4. Workstreams

The modernization was delivered as eight named programs rather than a single initiative: the customer identity modernization workstream, Registration Redesign, D2C Commerce Modernization, OMS and Order-to-Cash Enablement, Payment Modernization, the B2B commerce platform upgrade, the enterprise ERP transformation programme, and cloud modernization. Table 1 maps each to its architectural contribution.

Table 1: Modernization Programs and Architectural Contribution

Program	Primary systems touched	Architectural contribution
the customer identity modernization workstream	OKTA, commerce, CRM, registration, consent	Centralized, token-based customer identity replacing fragmented legacy identity
Registration Redesign	Registration applications, OKTA, CRM, Marketing Cloud, consent	Redesigned the registration capability end to end, not the registration screen
D2C Commerce Modernization	D2C commerce, OMS, payments, identity	Direct-to-consumer journey aligned to the unified identity and order model
OMS and Order-to-Cash Enablement	Commerce, OMS, ERP	Strengthened commerce-to-order-management integration across the order lifecycle
Payment Modernization	CyberSource, Adyen, commerce, ERP	Improved payment integration architecture, settlement processing, refund workflows
the B2B commerce platform upgrade	B2B commerce	Reduced platform complexity; improved deployment reliability and operational governance
the enterprise ERP transformation programme	ERP, commerce, OMS, integrations	Data integrity and continuity through transformation of the system of financial record
Cloud modernization	Azure services, Azure Data Factory, B2B-to-ERP integrations	Cloud-aligned data movement; improved maintainability and scalability

4. Architectural Approach

This section presents the core contribution: a five-step method applied consistently across all eight workstreams. The method is deliberately simple. Its value is not novelty in any single step, but the discipline of applying all five before delivery begins, and of refusing to skip step five.

As shown in Figure 2, the target state reorganizes the estate into layers, with experience-facing systems separated from a governed integration layer, which is separated in turn from systems of record.

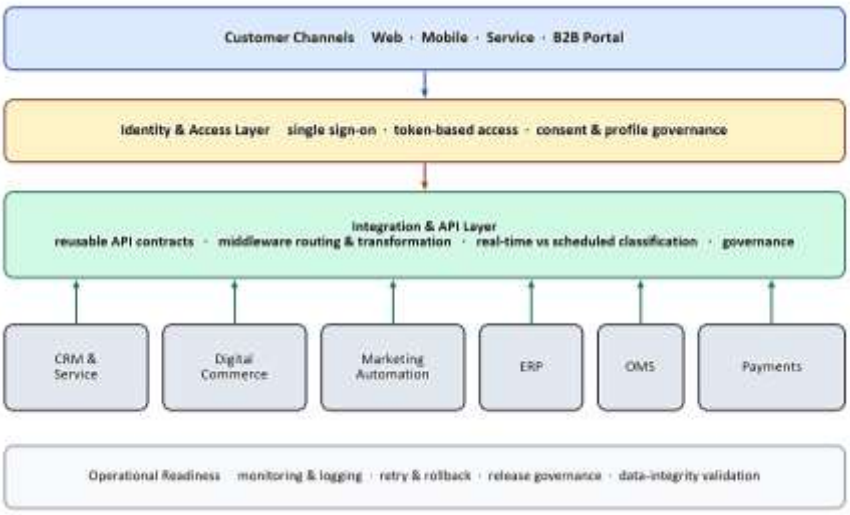

Fig 2: Target-State Layered Integration Architecture for Customer-Facing Systems

4.1. Step 1: Start from the business capability, not the application

The most consequential reframing in the program was refusing to accept application boundaries as problem boundaries.

Customer registration illustrates this. Treated as an application, registration is a form with fields, validation, and a submit action, and modernizing it means rebuilding that form. Treated as a business capability, registration means identity creation, profile data capture, consent capture and storage, CRM record creation, marketing communication enrollment, commerce access provisioning, and every downstream process that assumes a customer exists. The screen is the smallest part of it.

The practical test applied was a set of questions asked before any workstream started. What business process does this support? What data does it own, as distinct from data it merely displays? What depends on it? What happens if it fails? Who is affected, in business terms rather than system terms? Capability-oriented decomposition of this kind is standard in the architecture literature [1], [3], but the difficulty in practice is not knowing the principle. It is holding to it when a delivery timeline makes the application-shaped version of the problem look considerably cheaper.

Figure 4 summarizes the method itself.

Fig 3: Five-Step Enterprise Modernization Framework for Reusable System Integration

4.2. Step 2: Map all upstream and downstream dependencies

Before changing anything, the program mapped what each capability depended on and what depended on it.

In this estate, commerce depends on ERP for pricing and product data, on OMS for order lifecycle state, on the payment platforms for transaction processing, on identity for authentication, and on CRM and Marketing Cloud for

engagement continuity. Each of those is a different kind of dependency with a different failure mode, and treating them as one class was a mistake the mapping exercise was designed to prevent.

The mapping was the precondition for everything else. Dependency opacity is one of the most frequently reported causes of migration failure [9], [11], and the reason is not mysterious. When a system's dependents are unknown, the blast radius of a change cannot be estimated, so either the change is over-tested at high cost or under-tested at high risk. In a large enterprise, a technically small change can carry large business impact precisely when the dependency graph is not understood.

4.3. Step 3: Separate real-time from scheduled

Every integration in the estate was classified explicitly as real-time or scheduled before it was designed. The classification was treated as an architectural decision requiring justification, not a default inherited from whatever the previous integration did.

Real-time was reserved for processes where the business requires an immediate response: authentication, checkout, order confirmation, customer profile access, and payment authorization. Scheduled or batch was applied where data can synchronize at intervals without harming the customer or the process: large data synchronization, product and pricing updates, customer data alignment, reporting feeds, and back-office movement.

The two classes get different designs, and that is the point of separating them. Real-time integrations are designed for speed, security, and reliability. Scheduled integrations are designed for accuracy, monitoring, failure handling, and reconciliation. Figure 4 shows the decision applied in practice.

The synchronous-versus-asynchronous boundary is well documented as a source of difficulty in industrial migration [10], [11], and the mechanism is straightforward: a real-time call made where a batch would serve creates a runtime coupling that turns any downstream slowdown into a customer-visible failure, while a batch used where real-time is required produces stale data at exactly the moment a customer is making a decision.

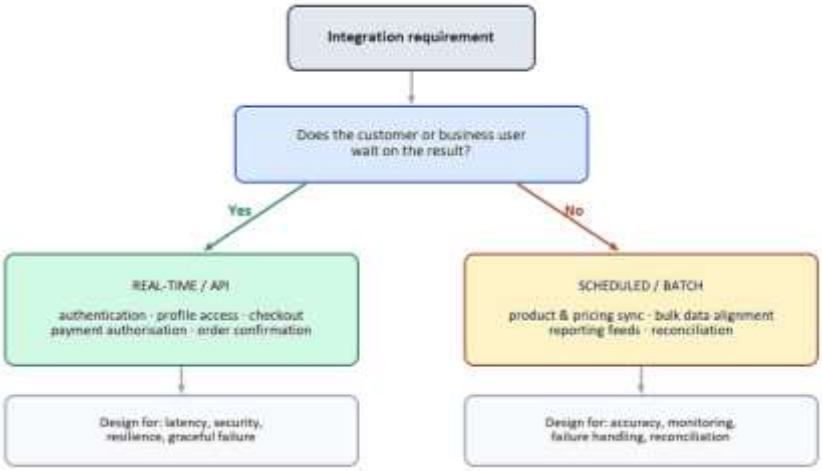

Fig 4: Decision Framework for Integration Pattern Classification

4.4. Step 4: Design API-first and integration-first

Rather than connecting systems directly to each other as needs arose, the program exposed business capabilities as reusable APIs under controlled contracts: identity, profile, order, payment status, product, and consent.

The distinction is between an API that exposes a system and an API that exposes a capability. An API that exposes a system leaks that system's internal model to every consumer, so replacing the system breaks every consumer. An API that exposes a capability under a stable contract allows the system behind it to change. This is what made phased migration feasible in later workstreams.

Middleware, primarily MuleSoft alongside Azure integration services, managed system-to-system communication, transformation, routing, and governance. The pattern vocabulary is Hohpe and Woolf's [6]. The economic argument is Themistocleous and Irani's [7]: point-to-point connections scale in cost faster than they scale in count, because each new connection multiplies the coordination surface rather than adding to it.

4.5. Step 5: Operational readiness is part of the architecture

The step most often deferred, and the one the program treated as non-negotiable, is that operational readiness belongs inside the architecture rather than after it.

Concretely, this meant that monitoring, rollback planning, release governance, support documentation, production validation, and long-term maintainability were design inputs, not launch-week activities. An integration without error logging is not a finished integration. A

migration without a rollback path is not a plan. A capability without support documentation transfers operational risk onto people who were not part of the design conversation.

This is the step where the empirical enterprise architecture literature and practitioner experience converge. Tamm and colleagues find that architecture produces benefit through delivery quality and decision quality rather than through direct technical effect [2]. Operational readiness is where that mechanism becomes visible, because it is what determines whether a technically correct change can be made safely in a production environment carrying live commerce.

4.6. The method as a whole

Stated as a sequence, modernization should begin with ecosystem mapping, proceed through business capability alignment, integration classification, and identity and data governance, and deliver through phased execution with operational readiness built in. The underlying position is that transformation is an architecture problem rather than a development task. Development is how the architecture is realized; it is not where the outcome is determined.

5. Integration Design and Identity Modernization

5.1. The end-to-end flow

The integration design is best understood by following one customer through the estate. As shown in Figure 3, a customer registers, an identity is created and validated, profile and consent data synchronize to CRM and marketing systems, commerce access is enabled, an order flows through commerce into order management, ERP supplies pricing and transaction data, the payment platform processes the transaction, and marketing and service systems operate on governed customer data thereafter.

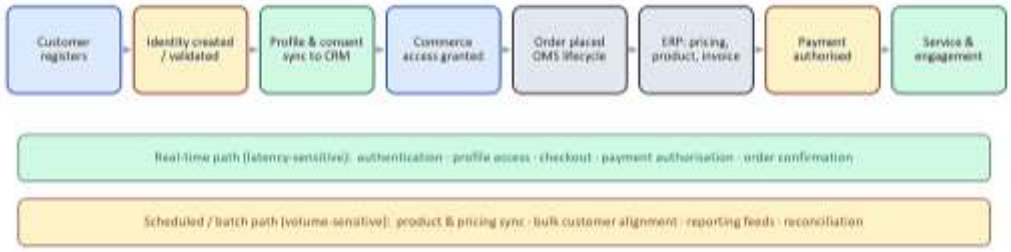

Fig 5: End-to-End Customer and Order Journey Across the Integrated Enterprise System

Read as a diagram this looks linear. It is not. Each arrow is a contract with its own latency expectation, failure mode, retry semantics, and data-ownership question, and the work of the integration design was making those explicit rather than emergent.

5.2. Real-time and scheduled in practice

Real-time integrations carried authentication, customer profile access, order actions, payment authorization, and selected commerce-to-backend calls. These were designed with attention to response time, secure transport and token handling, and reliability under concurrent load, since each of them sits directly in a customer's path.

Scheduled integrations carried large data synchronization, product and pricing updates, customer data alignment, reporting feeds, and back-office data movement. These were designed with attention to accuracy, monitoring, failure handling, and reconciliation, because a batch process that fails silently produces a data-integrity problem that surfaces later, in a different system, usually in front of a customer or an auditor.

Azure Data Factory was used to redesign legacy B2B-to-ERP integrations for cloud-aligned data movement. This is a representative case of the classification working as intended: high-volume, non-interactive, reconciliation-sensitive movement between commerce and the system of financial record was a batch problem that had partly been solved with real-time-shaped legacy mechanisms, and correcting the classification simplified both the design and its operational profile.

5.3. Reusable APIs over point-to-point

Several dozen enterprise APIs expose capabilities rather than systems. The consent API is a useful example of why the distinction matters. Consent is captured at registration, is legally meaningful, is consumed by marketing, and constrains what CRM and service systems may do. Exposing consent as a governed capability with a stable contract means that changing the consent-management platform does not require changing every consumer of consent state. Exposing it as a direct connection to the consent platform would have meant the opposite.

Middleware handled transformation, routing, and governance between systems. The governance element matters as much as the technical one: a middleware layer without contract ownership and change control becomes a centralized point-to-point estate, which is the original problem with a shorter cable.

5.4. Identity modernization

The the customer identity modernization workstream replaced fragmented and legacy identity handling with a scalable, secure customer identity ecosystem spanning B2B and digital commerce.

The design balanced three priorities simultaneously, which is the specific difficulty of identity work and the reason it cannot be delegated to any one function [12].

Security. Centralized and governed authentication, token-based access, and controls over user access, replacing arrangements where authentication behavior varied by platform. Federation and assertion patterns follow established guidance [13], and the receivables platform SSO integration is an instance of that pattern applied to a specific enterprise system.

Compliance and governance. Identity could not be separated from profile data, consent management, communication preferences, and customer records. A customer's identity is the key under which consent is recorded and enforced, so an identity change is a data-governance change whether or not it is planned as one.

Customer experience. Reduced login friction and a consistent experience across registration, login, profile management, ordering, and communications. Customers do not distinguish between an identity failure and a commerce failure; they experience both as the brand not working.

Because of these three couplings, identity modernization was deliberately delivered together with registration redesign, commerce access, consent management, CRM, and marketing rather than as an isolated platform swap. This was the single most important sequencing decision in the program. Success was not measured by whether login worked. It was measured by whether identity supported a secure, scalable, and smoother customer journey, which is a materially harder standard and one that only holds if the adjacent capabilities move with it.

6. Reliability, Governance, and Data Integrity

Mission-critical systems constrain how modernization can proceed. The estate described here processes live commerce continuously, so every change had to be made without interrupting business operation. Four practices carried that requirement.

6.1. Understand dependencies before changing anything

This restates step two of the method, and it is restated because it is the practice most often compressed under schedule pressure. Dependency understanding is the input to blast-radius estimation, and without it, risk assessment becomes assertion.

6.2. API and integration governance

Each integration carried a clear contract, defined error handling, logging, monitoring, and named ownership. Integrations were required to fail safely, provide visibility into what failed, and support quick recovery. Retry mechanisms and error logging were part of the delivered artifact rather than a subsequent hardening pass.

The distinction between failing and failing safely is worth stating precisely. A failing integration that returns an ambiguous state, writes a partial record, and logs nothing produces a data-integrity incident and an investigation. A failing integration that fails cleanly, logs the failure with correlation detail, and can be retried or rolled back produces an alert and a recovery. The difference is entirely design, not reliability.

6.3. Deployment governance and operational readiness

Release planning, production validation, rollback capability, and support readiness were treated as gates. Coordination spanned product, infrastructure, integration, commerce operations, and external partners, which in practice was as demanding as the technical work. Organizational coordination is repeatedly identified as the dominant transformation constraint [4], and this program did not find otherwise.

6.4. Data integrity as a first-class requirement

Data integrity was treated as an architectural requirement rather than a testing activity, and this became most visible during ERP modernization, customer identity migration, SubKey migration, and consent work.

Validation covered customer records, orders, invoices, pricing, products, credits, consent state, and transaction data, before and after each change. The reason for before-and-after validation rather than after-only is that a migration cannot be judged correct against an unknown baseline; without a pre-change measurement, a discrepancy discovered afterward cannot be distinguished from a discrepancy that already existed.

The duplicate-customer-record problem described in Section 3.3 is a master data problem, and the literature is clear that master data management succeeds or fails on ownership and stewardship rather than on tooling [16]. The program's response followed that finding: identity was established as the governed key for customer data, which is an ownership decision before it is a technical one.

6.5. Cloud readiness and security

Cloud modernization on Azure was delivered with a planned environment strategy, monitoring, security controls, and defined service ownership rather than as a lift of existing workloads. Migration type and its architectural consequences were reasoned about using the established taxonomy [17], and the architectural risks documented in the legacy-to-cloud literature, particularly dependency opacity and data consistency, were treated as the primary concerns [9]. Security was addressed through identity modernization, secure integration design, authentication, and access control rather than as a separate track.

7. Results

Outcomes are reported as observed by the program. Where an outcome is an association rather than a measured causal effect, it is stated as such. There is no control condition, no comparison enterprise, and no counterfactual, which is a limitation the paper accepts and discusses in

Section 8.6. Figure 6 summarizes the scale and the reported outcomes.

Fig 6: Scale and Outcomes of the Modernized Enterprise Integration Estate

7.1. Scale supported

The modernized platforms serve a large customer base with a comparable annual order volume, associated with a large North American revenue base. The integration estate comprises several dozen enterprise APIs, 15 real-time integrations, and 10 scheduled integration processes.

7.2. Registration modernization

The customer registration modernization supported a large volume of customer registrations. Registration processing time improved to sub-second, and page load performance improved to approximately one second.

These are the program's cleanest quantitative results, and the reason is that registration is a bounded capability with a measurable interaction. Most of the other workstreams do not have an equivalent single measurable moment.

7.3. Cloud and infrastructure modernization

Cloud and infrastructure modernization contributed meaningful operational cost savings while improving maintainability and scalability. Cost reduction as an outcome of infrastructure migration is consistent with case findings in the migration literature, though the same literature cautions that savings vary widely with workload profile and are not a general property of migration [18].

7.4. Commerce, identity, order management, and payments

- B2B commerce and Insite modernization reduced platform complexity and improved deployment reliability and operational governance.
- OKTA identity modernization established a more scalable customer identity ecosystem connecting authentication, profile management, consent, and engagement.
- OMS and order-to-cash enablement strengthened commerce-to-order-management integration across the order lifecycle.

Payment modernization covering CyberSource, Adyen, settlement processing, and refund workflows improved payment integration architecture and support processes.

7.5. Outcomes expressed as risk reduction

Several of the most significant results are not improvements in a metric. They are the absence of a failure that the change could plausibly have caused. Business continuity was maintained through ERP transformation, identity modernization, commerce upgrades, payment changes, and cloud migration, each of which touched a system carrying live revenue.

This category of outcome is real and is systematically undercounted, because it does not appear in a dashboard. An ERP transformation that does not disrupt order-to-cash, or an identity migration that does not lock customers out, produces no visible signal at all. The practitioner argument is that in mission-critical modernization this is frequently the primary deliverable, and that programs which cannot articulate it end up justified only by the metrics that happen to be easy to collect.

8. Discussion

8.1. Point-to-point versus middleware and reusable APIs

Point-to-point integration is faster to build for a small, well-understood need, and pretending otherwise is not useful. Its cost is deferred rather than avoided. As the estate grows, each additional direct connection increases the coordination surface, and dependency analysis degrades from an exercise into an investigation [7].

The decision applied here was that mission-critical flows spanning commerce, ERP, OMS, identity, payments, and marketing warrant governed integration with reusable APIs and middleware, and that the overhead is justified by what it buys: the ability to change one system without renegotiating with every consumer.

8.2. Real-time versus batch

Real-time was chosen where the customer is waiting and the business process requires immediate response. Batch was chosen where volume is high, urgency is lower, and reconciliation matters more than latency. The failure mode in both directions is worth naming. Over-using real-time creates runtime coupling and turns downstream slowness into customer-visible failure. Over-using batch produces stale data at decision points. The industrial evidence indicates this boundary is a persistent source of difficulty in practice [10], [11], which is why the method makes the classification an explicit, justified decision rather than a default.

8.3. Buy versus build

Identity was bought. OKTA was selected because authentication, security, and access management require governance, standards conformance, and scale characteristics that are expensive to build and more expensive to maintain correctly, and because the requirement set is well understood and not enterprise-specific [12], [13].

Custom development was applied where the workflow depends on enterprise-specific commerce, ERP, OMS, or customer-data requirements, since those requirements do not generalize and a product would have to be bent to fit them.

The heuristic is not buy-for-commodity and build-for-differentiator, which is too coarse. It is closer to: buy where the requirement is standardized and the cost of being wrong is a security or compliance failure; build where the requirement encodes how this specific business operates.

8.4. Modernization speed versus business continuity, and modernize versus stabilize

For mission-critical systems, phased migration with parallel validation, rollback planning, and operational readiness was preferred over a big-bang approach. Incremental modernization with the ability to run old and new paths concurrently is the pattern the modernization literature converges on [8], and the reason applies directly here: a phased change can be reversed, and a big-bang change can only be repaired.

A related question is whether to modernize now or stabilize first. Replacing everything at once creates risk without a corresponding gain, so the program sequenced by value and risk, modernizing highest-value or highest-risk areas first and stabilizing the remainder.

Table 2 summarizes the trade-off decisions.

Table 2: Architectural Trade-Off Decisions and Applied Reasoning

Decision	Option A	Option B	Applied position
Integration style	Point-to-point	Middleware and reusable APIs	Point-to-point acceptable for small isolated needs; governed integration required for mission-critical commerce, ERP, OMS, identity, payments, marketing
Data movement	Real-time	Scheduled or batch	Real-time where the customer waits and the process needs immediacy; batch for volume, lower urgency, reconciliation
Sourcing	Buy	Build	Buy a mature identity platform where governance, security, and scale dominate; build where the workflow encodes enterprise-specific commerce, ERP, OMS, or data requirements
Delivery approach	Big-bang	Phased with parallel validation	Phased, with rollback planning and operational readiness, for mission-critical systems
Sequencing	Modernize everything now	Stabilize first, modernize selectively	Sequence by value and risk; modernize highest-value or highest-risk areas first
Scope of change	Front-end only	Front-end plus identity, data, and back-office	Front-end-only modernization leaves the business problem intact and was explicitly avoided

8.5. What to avoid

The clearest anti-pattern encountered is modernizing only the front end. A redesigned interface over fragmented identity, ERP, OMS, payments, and customer data changes what the problem looks like without changing what it is. Duplicate customer records persist. Manual reconciliation persists. Order delays persist. The visible artifact improves and the business outcome does not, which is a difficult position to recover from because the program has already spent its credibility.

8.6. Artificial intelligence: scoping the contribution honestly

This section is deliberately narrow, and the narrowness is the point.

The work described in this paper does not include broad production AI deployment, and no such claim is made. The author's AI experience in this context consists of evaluating enterprise AI use cases, working with Copilot, identifying automation opportunities, exploring intelligent assistants and knowledge discovery, and building proof-of-concepts. Those activities are real and are stated at their actual scope.

The substantive position is that the value of this program to AI-enabled capability is foundational rather than applied. AI creates enterprise value only once the underlying systems are modernized, connected, governed, and data-ready. An organization with duplicate customer records across seven systems, undocumented dependencies, ungoverned integrations, and no consent lineage does not have an AI problem. It has a data and architecture problem that will

surface as an AI problem the moment a model is pointed at the estate. The modernization described here, capability-aligned decomposition, governed APIs, classified integrations, identity as the governed customer key, and data integrity validated through migration, is the precondition rather than the application.

A second observation is that a considerable amount of what enterprises describe as automation opportunity already exists in the architecture, unlabeled. Integration logic makes routing and transformation decisions. Automated data movement runs on schedules with failure handling. Order processing advances through states without manual intervention. Deployment governance and monitoring apply rules continuously. This is intelligent automation in the operational sense, and recognizing it matters because it locates the realistic near-term opportunity: extending and instrumenting mechanisms that already run, rather than introducing new ones on top of an estate that cannot yet supply reliable data.

Stated plainly, the contribution here is the foundation required for AI-enabled transformation, plus practical evaluation of use cases against that foundation. It is not production AI, and a paper that claimed otherwise would be less useful to the practitioners most likely to read it.

8.7. Lessons

The hardest part of this work was not writing code or establishing connections. It was understanding how business processes, customer experience, enterprise data, integrations, security, and operations interact, and holding all of those in view while making a decision in one of them. In a large enterprise, a small technical change can carry large business impact when the dependencies are not understood, which is the practical form of the argument that transformation is organizational before it is technical [4].

Three lessons generalize. Start with business capability mapping, asking what process a capability supports, what data it owns, what depends on it, what happens when it fails, and who is affected. Treat integration architecture as the backbone of transformation, since it is what customers actually experience as continuity [15]. Treat data integrity as a first-class requirement with named ownership, since master data problems are organizational problems that present as technical ones [16].

8.8. Limitations

Modernization is never finished. The objective was an architecture that continues to evolve, not a final state, and that is consistent with treating transformation as ongoing strategic renewal [5]. The architecture described here should be read as a position at a point in time.

Methodologically, this is a single-organization practitioner case study without a control condition, a comparison enterprise, or a counterfactual. Reported outcomes are what the program observed and are associations rather than demonstrated causal effects. The employer is not named, which limits independent verification. Some outcomes are qualitative by nature, particularly the continuity results in Section 7.5, and are not quantified. The paper follows established precedent for single-case enterprise migration reporting [18] and inherits the limits of that form; the intended contribution is transferable architectural reasoning, not statistical generalization.

In hindsight, several things would be done earlier. Architecture documentation would be produced earlier and maintained as a delivery artifact rather than reconstructed under pressure. Reusable integration patterns would be established before the integration count grew, since retrofitting patterns is more expensive than adopting them. Automated monitoring and data-quality dashboards would be built alongside the integrations rather than after. And an AI-readiness assessment would be conducted early, not to deploy AI, but to make explicit which data, governance, and integration gaps would eventually constrain it.

8.9. Transferability

The specific platforms in this case are incidental to the method. The approach applies across consumer goods, retail, healthcare, manufacturing, financial services, and logistics, wherever an organization runs a mix of legacy systems, cloud platforms, ERP dependencies, third-party tools, and customer-facing digital experiences. Vendors change and the sequencing of capability alignment, dependency mapping, integration classification, identity and data governance, phased delivery, and operational readiness does not.

9. Conclusion and Future Work

This paper documented a multi-year modernization of mission-critical customer-facing systems at a large consumer-brand enterprise, covering identity, registration, B2B and D2C commerce, order management, payments, ERP, and cloud. The estate began connected but not unified, with a customer journey distributed across platforms that had never been designed together.

The contribution is a five-step method applied consistently across heterogeneous workstreams: start from the business capability rather than the application, map upstream and downstream dependencies before changing anything, classify integrations explicitly as real-time or scheduled, design API-first and integration-first, and treat operational readiness as part of the architecture. Reported outcomes include an estate serving a large customer base with a comparable annual order volume, several dozen enterprise APIs, registration processing improved to sub-second, meaningful cloud-related operational savings, and business continuity maintained through ERP, identity, commerce, payment, and cloud change.

The paper's position on AI is intentionally conservative. The program built the modernized, connected, governed, and data-ready foundation that AI-enabled capability requires, and evaluated practical use cases against it. It did not deploy production AI, and it does not claim to have.

Future work follows from the limitations. Longitudinal measurement would establish whether the architectural benefits observed here persist as the estate continues to change. Comparative case work across organizations would test whether the five-step method holds outside a single enterprise. Data-quality instrumentation, built as a first-class capability rather than a reporting layer, would make integrity measurable continuously rather than at migration checkpoints. And a structured AI-readiness assessment framework, grounded in the architectural preconditions described in Section 8.6, would give enterprises a way to establish whether their foundation can support AI before they commit to it.

References

[1] Ross, J. W., Weill, P., & Robertson, D. C. (2006). *Enterprise architecture as strategy: Creating a foundation for business execution*. Harvard Business School Press.

[2] Tamm, T., Seddon, P. B., & Shanks, G. (2022). How enterprise architecture leads to organisational benefits. *International Journal of Information Management, 67*, 102554. https://doi.org/10.1016/j.ijinfomgt.2022.102554

[3] The Open Group. (2022). *The TOGAF standard, 10th edition*. Van Haren Publishing.

[4] Vial, G. (2019). Understanding digital transformation: A review and a research agenda. *The Journal of Strategic Information Systems, 28*(2), 118–144. https://doi.org/10.1016/j.jsis.2019.01.003

[5] Warner, K. S. R., & Wäger, M. (2019). Building dynamic capabilities for digital transformation: An ongoing process of strategic renewal. *Long Range Planning, 52*(3), 326–349.

[6] Hohpe, G., & Woolf, B. (2003). *Enterprise integration patterns*. Addison-Wesley.

[7] Themistocleous, M., & Irani, Z. (2001). Benchmarking the benefits and barriers of application integration. *Benchmarking: An International Journal, 8*(4), 317–331.

[8] Newman, S. (2019). *Monolith to microservices: Evolutionary patterns to transform your monolith*. O'Reilly Media.

[9] Hasan, M. H., Osman, M. H., Admodisastro, N., & Muhammad, S. (2023). Legacy systems to cloud migration: A review from the architectural perspective. *Journal of Systems and Software, 202*, 111702. https://doi.org/10.1016/j.jss.2023.111702

[10] Di Francesco, P., Lago, P., & Malavolta, I. (2019). Architecting with microservices: A systematic mapping study. *Journal of Systems and Software, 150*, 77–97.

[11] Fritzsch, J., Bogner, J., Wagner, S., & Zimmermann, A. (2019). Microservices migration in industry: Intentions, strategies, and challenges. *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 481–490. https://doi.org/10.1109/ICSME.2019.00081

[12] Glockler, J., Sedlmeir, J., Frank, M., & Fridgen, G. (2024). A systematic review of identity and access management requirements in enterprises. *Business & Information Systems Engineering, 66*(4), 421–440. https://doi.org/10.1007/s12599-023-00830-x

[13] National Institute of Standards and Technology. (2025). *Digital identity guidelines: Federation and assertions* (NIST SP 800-63C-4). https://doi.org/10.6028/NIST.SP.800-63c-4

[14] Verhoef, P. C., Kannan, P. K., & Inman, J. J. (2015). From multi-channel retailing to omni-channel retailing. *Journal of Retailing, 91*(2), 174–181.

[15] Hossain, T. M. T., Akter, S., Kattiyapornpong, U., & Dwivedi, Y. (2020). Reconceptualizing integration quality dynamics for omnichannel marketing. *Industrial Marketing Management, 87*, 225–241.

[16] Vilminko-Heikkinen, R., & Pekkola, S. (2017). Master data management and its organizational implementation. *Journal of Enterprise Information Management, 30*(3), 454–475. https://doi.org/10.1108/JEIM-07-2015-0070

[17] Jamshidi, P., Ahmad, A., & Pahl, C. (2013). Cloud migration research: A systematic review. *IEEE Transactions on Cloud Computing, 1*(2), 142–157. https://doi.org/10.1109/TCC.2013.10

[18] Khajeh-Hosseini, A., Greenwood, D., & Sommerville, I. (2010). Cloud migration: A case study of migrating an enterprise IT system to IaaS. *2010 IEEE International Conference on Cloud Computing (CLOUD)*, 450–457. https://doi.org/10.1109/CLOUD.2010.37