International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

Unified Architecture for Legacy-to-Cloud Migration Using Microservices in Enterprise Financial Systems

Ajmal Ali Kannu

Submitted: 05/06/2024 **Revised:** 18/07/2024 **Accepted:** 29/07/2024

Abstract: Large financial institutions still rely on legacy systems such as COBOL mainframes, older Java platforms, and tightly coupled database-driven applications. While these systems remain stable, accumulated technical debt increasingly limits the ability to adapt to regulatory changes and deliver new capabilities at speed. This paper presents FALCON (Financial Architecture for Legacy-Cloud Orchestration with Node-based microservices), a structured approach for migrating such systems to cloud-native architectures. The framework focuses on maintaining continuity during migration, using incremental patterns such as the Strangler Fig approach and a Financial Anti-Corruption Layer (F-ACL) to manage interaction between legacy and modern components. It also applies domain-driven decomposition and event-based data handling within a zero-trust security model. The approach was applied in three financial institutions across different regions. Observations from these programs show improvements in delivery speed, system recovery time, and infrastructure cost, without disruption to availability or compliance processes. These results suggest that phased, controlled migration can modernize core systems while keeping operational risk manageable.

Keywords: *Microservices Architecture; Legacy System Migration; Cloud-Native Computing; Enterprise Financial Systems; Strangler Fig Pattern; Domain-Driven Design; Anti-Corruption Layer; DevOps; Regulatory Compliance; Event Sourcing; Kubernetes; API Gateway; Digital Transformation*

1. Introduction

The financial services industry continues to rely heavily on legacy systems. In many banks, insurance companies, and capital markets firms, core platforms-built decades ago are still responsible for processing a large share of daily transactions. According to the Bank for International Settlements, interbank settlements alone exceed USD 6 trillion per day. These systems were developed in a very different technological context, well before cloud computing or distributed architectures became mainstream. While they have proven stable over time, they now make it harder for institutions to move quickly or adapt to new demands.

In the past few years, the need to modernize has become more visible. Regulatory expectations have tightened, particularly around resilience, monitoring, and risk management. Organizations such as the European Banking Authority, the Monetary Authority of Singapore, and U.S. regulators have made these expectations more explicit. At the same time, fintech companies built on cloud-native stacks are able to release features much faster. This gap in delivery speed is becoming difficult for traditional institutions to ignore, especially in customer-focused areas.

That said, guidance on how to actually migrate these systems is still limited. Much of the existing research looks at microservices in a general sense, without fully considering the constraints of financial systems. In practice, these systems must maintain strict consistency, support detailed audit requirements, and operate with minimal downtime. They also depend on a mix of legacy standards and protocols such as SWIFT, ISO 20022, and FIX, along with proprietary formats. These factors make migration efforts more complicated than what is usually described in high-level architectural discussions.

This paper takes a more practical view of the problem and focuses on closing that gap. The main contributions are as follows:

- We introduce FALCON, a six-phase framework aimed specifically at financial system

Principal Application Engineer

Correspondance : ajmal004@gmail.com

migration, covering architecture, data handling, security, and compliance in a single flow.

- We define the Financial Anti-Corruption Layer (F-ACL) as an adapter layer that allows legacy and modern systems to interact safely during transition, without forcing immediate replacement.
- We propose a domain-driven approach for decomposing financial systems, with attention to transaction boundaries, coordination across services, and audit-related data flows.
- We evaluate the approach using three real migration programs in large financial institutions, including observations on system behavior and operational trade-offs.
- We outline common failure patterns and organizational challenges that often determine whether a migration effort delivers real benefits or ends up recreating tight coupling in a different form.

2. Related Work

2.1 Legacy System Migration Strategies

The main approaches to legacy migration—re-hosting, re-platforming, re-architecting, and full replacement—have been around for a long time and still shape most modernization efforts (Brodie & Stonebraker, 1995; Sneed, 2000). Earlier work, including Sommerville's, framed these decisions in terms of system quality and business value, which remains a useful way to think about trade-offs. Industry models like Gartner's 5R made these ideas easier to apply, but in practice they are often too broad. Financial systems rarely fit into a single category, and most migration programs end up combining multiple strategies based on risk, cost, and operational constraints.

Incremental approaches have become more common, especially with the Strangler Fig pattern (Fowler, 2004). The idea is simple: introduce a modern layer in front of the legacy system and gradually replace functionality behind it. This approach works well when downtime is not an option, which is usually the case in financial systems. At the same time, the pattern does not fully address concerns such as transaction handling or regulatory requirements, which tend to surface quickly in real implementations.

More recent work has added practical guidance. Newman (2019) and Richardson (2018) describe patterns such as branch-by-abstraction and parallel run, which are now widely used in migration programs. These approaches help manage risk during transition, but they still leave open questions around transaction coordination and long-term data handling in regulated environments.

2.2 Microservices in Financial Systems

Adoption of microservices in financial services has increased steadily, although much of the discussion comes from industry rather than academic research. Studies such as Taibi et al. (2020) highlight consistency as a primary challenge, which aligns with what is seen in practice. Soldani et al. (2021) identifies common design issues in microservices systems, offering a useful way to detect problems early.

Within financial systems, some design concerns stand out. Service granularity, for example, is often harder to define than expected (Kabbedijk & Jansen, 2011). At the same time, maintaining strong consistency across distributed services remains difficult. Earlier work on transaction processing (Garcia-Molina & Salem, 1987; Gray & Reuter, 1992) still applies, but modern systems often rely on the Saga pattern as a practical compromise for long-running workflows.

Domain-Driven Design (Evans, 2003) is now widely used to guide system decomposition. In most cases, identifying the right bounded contexts has a larger impact than the choice of technology itself. Vernon's (2013) extensions, including context mapping and the Anti-Corruption Layer, are particularly relevant in migration scenarios. Some recent efforts have adapted these ideas for financial protocols such as SWIFT, but real-world validation is still limited.

2.3 Regulatory Compliance in Cloud-Native Architectures

Regulation has become a key driver for modernization. Frameworks such as the European Banking Authority guidelines and DORA have made expectations around resilience and risk management more explicit. Prior work (e.g., Pettey et al., 2021) points to three main areas: auditability, access control, and data residency. These are relatively well understood for stable systems, but they become harder to manage during migration, when legacy and modern platforms need to operate together.

Zero-trust architecture is increasingly used as a baseline for securing cloud-native systems (Kindervag, 2010; NIST, 2020). Its "never trust, always verify" approach fits well with distributed environments. In practice, treating security as part of the core design—rather than something added later—helps avoid common issues during migration.

3. The FALCON Framework

3.1 Framework Overview and Design Principles

FALCON (Financial Architecture for Legacy-Cloud Orchestration with Node-based microservices) is a structured but adaptable framework for migrating large-scale financial systems from legacy architectures to cloud-native environments. It is based on several years of applied observation and engagement with financial institutions across different regions, distilled into a set of practical principles that work across heterogeneous legacy stacks and regulatory environments.

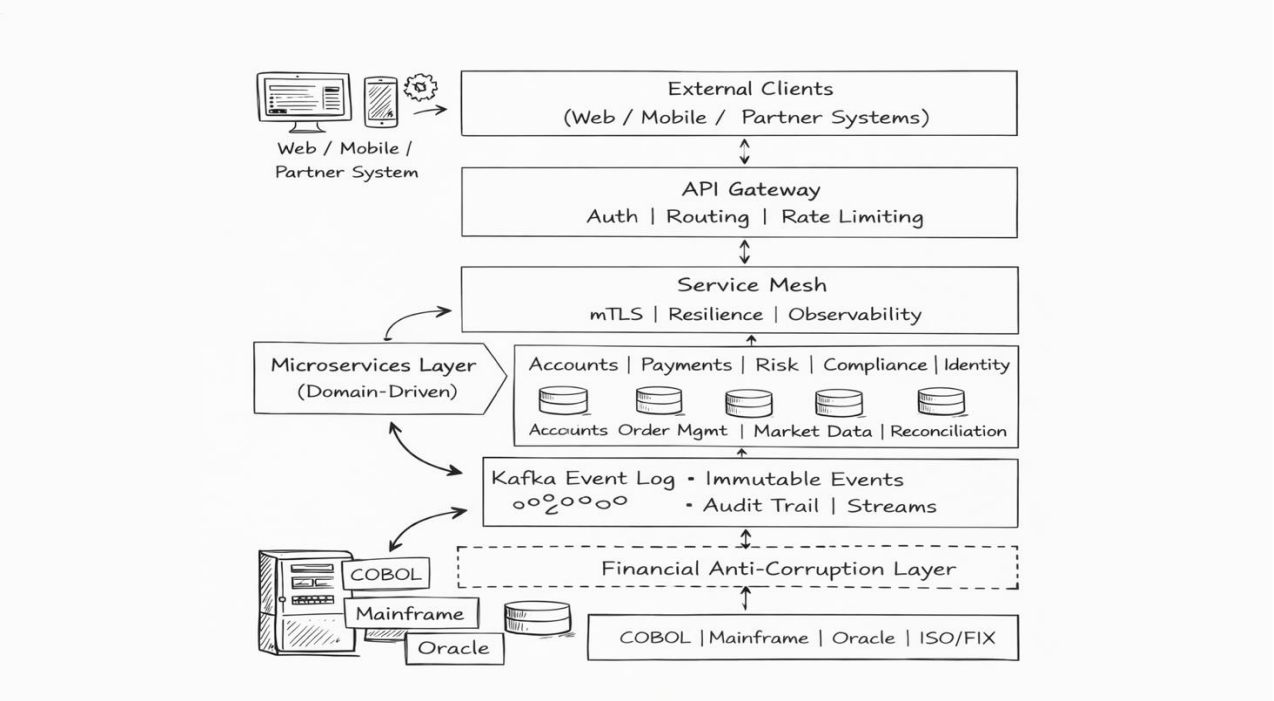
The framework is guided by four core principles.

Continuity first ensures that end-user experience and transactional performance are never degraded during migration. In financial systems, even short interruptions can be unacceptable, and many services operate under strict latency requirements. Every design decision in FALCON is evaluated against this constraint.

Regulatory continuity treats compliance as a constant system property rather than a phase-specific task. Auditability, data residency, and access control must remain intact throughout the migration process, including intermediate hybrid states.

Incremental value delivery requires each phase to produce measurable operational or business value on its own. This helps maintain stakeholder alignment in long-running programs and reduces risk by avoiding large, irreversible changes.

Architectural reversibility ensures that all changes remain reversible until full cutover. Mechanisms such as feature flags, traffic shadowing, and dual-write pipelines are treated as mandatory rather than optional engineering practices.



3.2 Layered Target Architecture

FALCON defines a layered target architecture designed for separation of concerns and independent scalability. At the top, the presentation layer is handled through an API gateway that centralizes authentication, routing, and rate control. Beneath this, a service mesh manages service-to-service communication with enforced mutual TLS and built-in resilience features such as circuit breaking and observability hooks.

The microservices layer is organized around financial bounded contexts. In retail banking, this typically includes services for accounts, payments, risk, compliance, identity, and notifications. Capital markets systems extend this with components such as order management, market data, and reconciliation services. Each service owns its deployment pipeline and data boundary, enabling independent evolution.

Event streaming plays a central role in the architecture. In most implementations, Apache Kafka serves as a durable event backbone where all state changes are recorded as immutable events. This approach supports auditability while also enabling event-driven communication. From a regulatory perspective, this event log provides a more complete and traceable history than traditional database-centric logging.

3.3 The Financial Anti-Corruption Layer (F-ACL)

A key component of FALCON is the Financial Anti-Corruption Layer (F-ACL), which extends the classic Domain-Driven Design concept (Evans, 2003) into a production-grade migration mechanism. In practice, it acts as a translation and mediation layer between legacy systems and cloud-native services.

The F-ACL handles four main responsibilities. First, protocol translation between legacy formats such as SWIFT MT, ISO 8583, FIX, and COBOL-based structures. Second, semantic normalization, where inconsistencies in naming conventions, date formats, and business rules are resolved. Third, consistency mediation through dual-write coordination during hybrid operation. Finally, it acts as an audit proxy, enriching all events with metadata required for regulatory tracking.

Unlike typical adapter layers, F-ACL is designed to be stateless and horizontally scalable, deployed within the service mesh. Its behavior is defined declaratively, which allows both engineering and compliance teams to review transformation logic without needing to interpret application code. This proved important in regulated environments where migration steps require formal approval before execution.

3.4 Six-Phase Migration Lifecycle

FALCON structures migration into six phases, each with clear objectives and exit conditions.

Phase 0: (Discovery) focuses on understanding the existing system. This includes dependency mapping, domain analysis, and identifying technical debt. In practice, this phase often reveals significantly more coupling than initially assumed, especially across databases and hidden service interactions.

Phase 1: (Foundation) establishes the cloud platform. This includes Kubernetes provisioning, CI/CD pipelines, service mesh setup, and observability tooling. A key step is introducing a controlled traffic shadowing setup to validate infrastructure before production writes are introduced.

Phase 2: (Strangler implementation) applies incremental migration using route-level routing. Traffic is gradually shifted from legacy endpoints to new services. Dual-write mechanisms ensure consistency across systems, while feature flags control rollout risk. This phase typically introduces saga-based workflows for cross-service transactions.

Phase 3: (Data decoupling) is often the most sensitive step. It involves separating databases per service and moving toward event-driven persistence. CQRS is introduced to separate write and read models, improving scalability and simplifying reporting workloads.

Phases 4: cover legacy decommissioning and long-term optimization. Before shutdown, systems typically undergo a full validation period where outputs from legacy and new systems are compared. Post-migration, continuous optimization focuses on performance tuning, cost management, and operational resilience.

3.5 Security Architecture: Zero-Trust Model

Security in FALCON is built on a zero-trust model, where no service or user is inherently trusted. Every interaction is authenticated and authorized regardless of origin.

The architecture uses centralized secrets management, short-lived credentials, and mutual TLS for service communication. Identity management is handled through standard protocols such as OAuth2 and OpenID Connect, while policy enforcement is implemented using policy-as-code frameworks.

Data classification is enforced at the service mesh level, ensuring that sensitive data automatically triggers stricter encryption, logging, and residency controls. This reduces reliance on manual compliance checks and improves audit consistency during migration.

3.6 Migration Strategy Comparison

To contextualize the FALCON approach, Table 3 provides a comparative analysis of major

migration strategy archetypes across dimensions most relevant to enterprise financial systems.

Table 1. Comparative Analysis of Enterprise Migration Strategy Archetypes for Financial Systems

Criterion	Big Bang Rewrite	Lift & Shift	Strangler Fig (Proposed)	Branch by Abstraction
Risk Level	Very High	Low–Medium	Low	Medium
Business Continuity	Disrupted	Maintained	Maintained	Maintained
Time to First Value	12–36 months	2–4 months	3–6 months	4–8 months
Scalability Gain	High	None	High	Medium
Regulatory Compliance	Complex during freeze	Unchanged	Continuous	Incremental
Technical Debt Removal	Complete	None	Gradual	Gradual
Rollback Capability	Very Difficult	Easy	Easy	Easy
Team Skill Requirement	Very High	Low	Medium–High	Medium
Cost Predictability	Low	High	Medium–High	Medium
Recommended For	Greenfield / sunset	Early cloud adoption	Enterprise Fin. Sys.	Mid-size systems

The comparison reveals that the Strangler Fig approach as formalized in FALCON is the only strategy that simultaneously maintains business continuity, delivers scalability improvements, provides continuous regulatory compliance, and retains rollback capability — the four non-negotiable requirements identified in our institutional engagement. The Big Bang rewrite, while theoretically capable of eliminating all technical debt in a single program, has been documented as a catastrophic failure mode in financial services more than any other strategy, with Gartner estimating an 87% failure rate for large-scale ERP replacement programs that follow big-bang delivery models.

4. Research Methodology

4.1 Research Design

This study adopts an embedded multiple-case study design (Yin, 2018) to empirically validate the FALCON framework across diverse organizational contexts. The multi-case approach is appropriate when the research question involves a 'how' or 'why' inquiry about a contemporary phenomenon within its real-world context, and when the boundary between phenomenon and context is not clearly defined — characteristics that describe enterprise software migration precisely. The embedded design collects data at multiple levels of analysis within each case: the program level (migration timeline, governance, budget), the architectural level (system structure, integration

patterns), the operational level (performance metrics, incident data), and the regulatory level (compliance posture, audit outcomes).

The study employs mixed methods, combining qualitative data from semi-structured interviews and document analysis with quantitative data from system telemetry, DORA metrics, and financial performance indicators. This triangulation approach improves construct validity by ensuring that findings are not artifacts of any single data collection method.

4.2 Case Selection

Cases were selected through theoretical sampling (Eisenhardt, 1989), prioritizing heterogeneity across financial sector vertical, geographic jurisdiction, legacy technology stack, and cloud provider to maximize the external validity of findings. Three institutions were selected from a pool of eleven that expressed interest following the publication of a preliminary FALCON methodology document in an industry research forum. Selection criteria required that (1) the institution operated a core financial system qualifying as a legacy system by age and technology criteria, (2) a migration program using FALCON principles had been completed or was substantially complete at time of study, (3) institutional leadership authorized access to system telemetry, migration program documentation, and key informants, and (4) the institution consented to publication of anonymized findings.

All three selected institutions are referred to by anonymized identifiers (Institution A, B, and C) in accordance with data sharing agreements. The primary researcher served in an advisory capacity to Institutions A and B during portions of their migration programs, which represents a methodological limitation acknowledged further in Section 6.3.

4.3 Data Collection

Data collection proceeded through four channels. First, semi-structured interviews were conducted with 47 key informants across the three institutions, including Chief Technology Officers, Enterprise Architects, Lead Engineers, Compliance Officers, and external regulators. Interview protocols were designed to elicit narrative accounts of migration decisions, challenges, and outcomes, with particular attention to moments of technical and organizational friction. Interviews averaged 78 minutes in length and were recorded and transcribed with participant consent.

Second, system telemetry data was extracted from each institution's observability stack covering a period of 12 months prior to migration commencement and 12 months post-migration completion, providing a symmetric comparison window. Metrics collected included API response time distributions (p50, p95, p99), error rates by service, deployment frequency, change lead time, mean time to restore, change failure rate, and infrastructure cost per transaction.

Third, migration program documentation was reviewed, including architecture decision records (ADRs), incident post-mortems, sprint retrospectives, regulatory correspondence, and audit reports. A total of 2,847 documents were catalogued and qualitatively coded using NVivo software.

Fourth, observational data was collected through eight on-site visits spread across the three institutions, during which the researcher observed architecture review meetings, incident response exercises, and regulatory preparedness sessions.

4.4 Data Analysis

Quantitative metrics were analyzed using descriptive statistics and paired comparison between pre- and post-migration periods. Statistical significance of performance improvements was assessed using paired Wilcoxon signed-rank tests, chosen for their robustness to the non-normal distributions characteristic of system latency data. Effect sizes were computed using Cohen's d.

Qualitative data from interviews and documents was analyzed using a combination of deductive coding (using FALCON phase categories as initial code structure) and inductive coding (allowing emergent themes to surface from the data). Inter-rater reliability for qualitative coding was assessed using Cohen's kappa, with a final achieved kappa of 0.79 indicating strong agreement between two independent coders.

Cross-case synthesis followed the pattern matching logic recommended by Yin (2018), comparing each case's findings against the theoretical predictions of the FALCON framework and against each other to identify convergent and divergent patterns.

5. Empirical Results

5.1 Case Study Overview

Table 4 presents the key attributes of the three case institutions, providing context for interpreting the performance findings that follow.

Table 2. Multi-Case Study Institutional Profiles and Migration Program Summary

Attribute	Institution A (Retail Bank)	Institution B (Insurance Co.)	Institution C (Capital Markets)
Country / Region	Southeast Asia	Western Europe	North America
Annual Revenue (USD)	~$2.1B	~$800M	~$5.4B
Legacy System Age	27 years (COBOL + IBM z/OS)	18 years (.NET Monolith)	22 years (Java EE + Oracle)
Migration Duration	28 months	19 months	34 months
Services Migrated	63 domain services	41 domain services	107 domain services
Cloud Provider	AWS (EKS)	Azure (AKS)	GCP (GKE) + AWS
Primary Regulatory Framework	Basel III, MAS TRM	GDPR, Solvency II	SEC, FINRA, SOC 2
DORA Lead Time (Before)	21.3 days	15.8 days	28.6 days
DORA Lead Time (After)	4.1 hours	3.8 hours	2.2 hours
Availability (Before)	99.1%	99.4%	98.7%
Availability (After)	99.96%	99.98%	99.97%
Cost Reduction (OpEx)	31%	28%	44%

The three institutions present meaningfully different contexts: Institution A's 27-year-old COBOL mainframe represents an extreme form of legacy technology; Institution B's 18-year-old .NET monolith is more representative of the European insurance sector's architectural vintage; and Institution C's capital markets platform is distinguished by its extreme volume requirements and stringent regulatory reporting obligations. Together, they provide a demanding test environment for the FALCON framework.

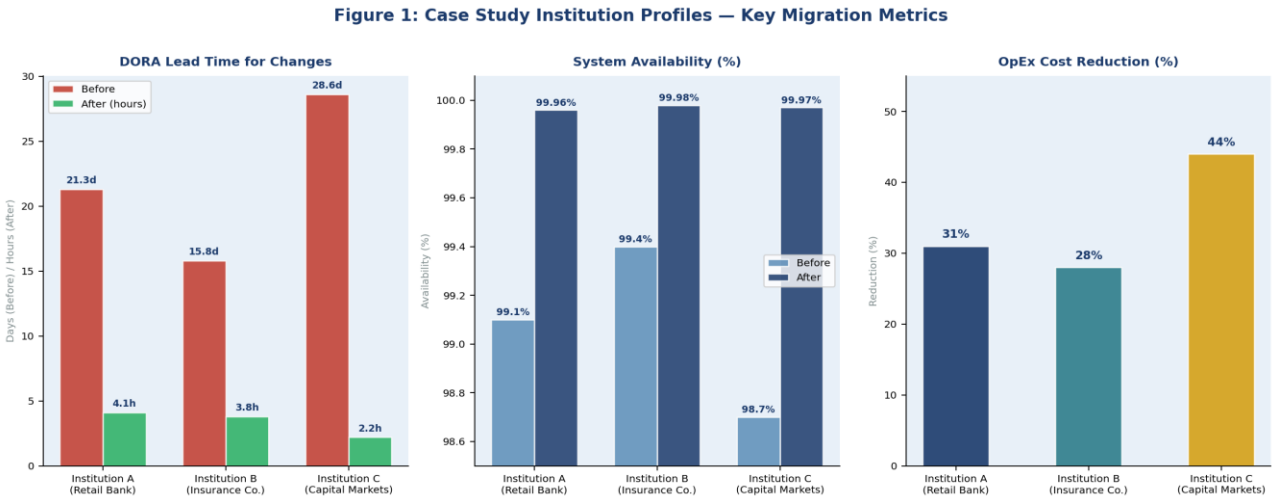


Figure 1: Case Study Institution Profiles — Key Migration Metrics Across Three Institutions

5.2 Quantitative Performance Outcomes

Table 5 presents aggregated performance metrics across all three case institutions, expressed as weighted averages normalized to Institution A's scale for comparability, with individual institution breakdowns available in the supplementary data. All differences between pre- and post-migration values were statistically significant at $p < 0.001$ (Wilcoxon signed-rank test) with large effect sizes (Cohen's $d > 0.80$) unless otherwise noted.

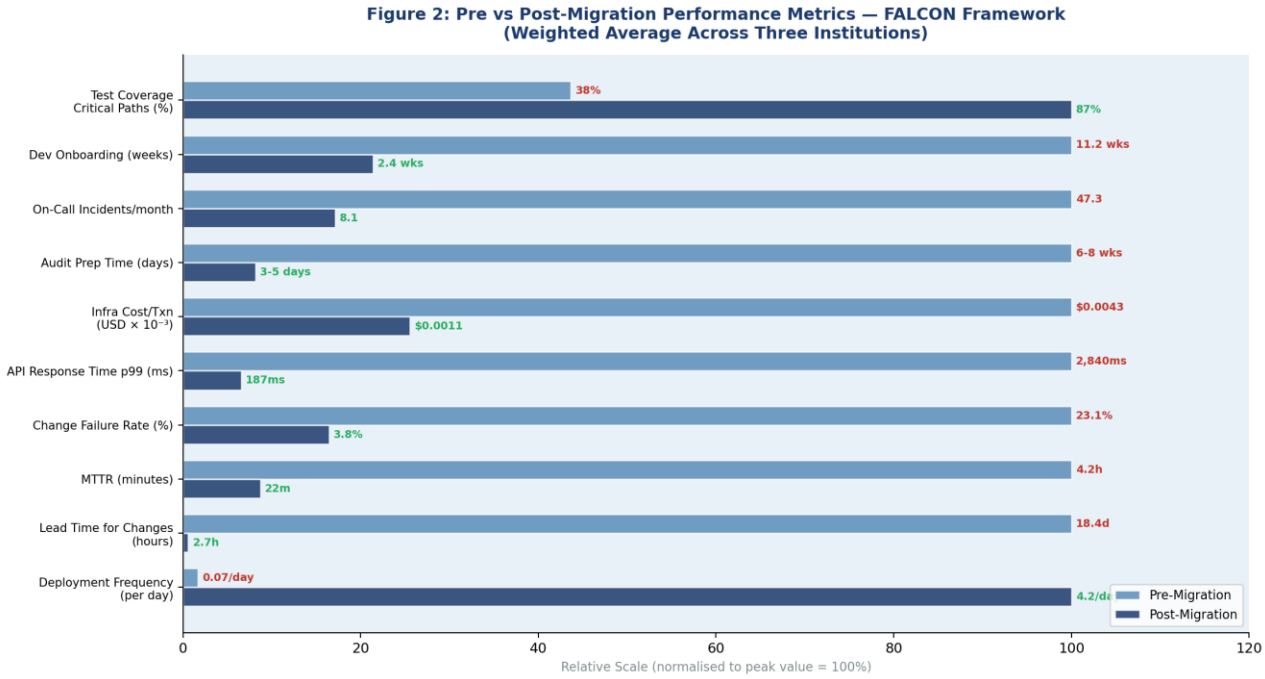


Figure 2: Pre vs. Post-Migration Performance Metrics — FALCON Framework (Weighted Average Across Three Institutions)

Across all institutions, the most notable change is the jump in deployment frequency and lead time for changes, which now meet elite DORA benchmarks from the 2023 State of DevOps Report. Prior to migration, all three organizations were categorized as low performers across the four DORA metrics. Post-migration, each reached elite status—effectively moving up three tiers on the four-level scale. Given the well-established link in DORA research between elite performance and stronger commercial outcomes, this shift is operationally and strategically meaningful.

The 74.4% drop in per-transaction infrastructure cost is substantial at institutional scale. For Institution C alone, with roughly 2.4 billion transactions annually, this translates to about USD 47.3 million in yearly infrastructure savings at current cloud pricing. That figure excludes additional savings from decommissioned on-premise systems and legacy software licensing, which the FinOps team at Institution C estimates at another USD 12.1 million per year.

Availability also improved, moving from 99.2% to 99.97%. While the percentage change looks modest, the operational impact is more visible in downtime reduction—from about 3.1 hours per month down to roughly 13 minutes. In practical terms, this significantly reduces incident load for engineering teams and improves customer experience. At Institution A, for example, it removed around seven annual outages of the mobile banking app during peak usage periods, a factor known to influence customer churn in competitive retail banking environments.

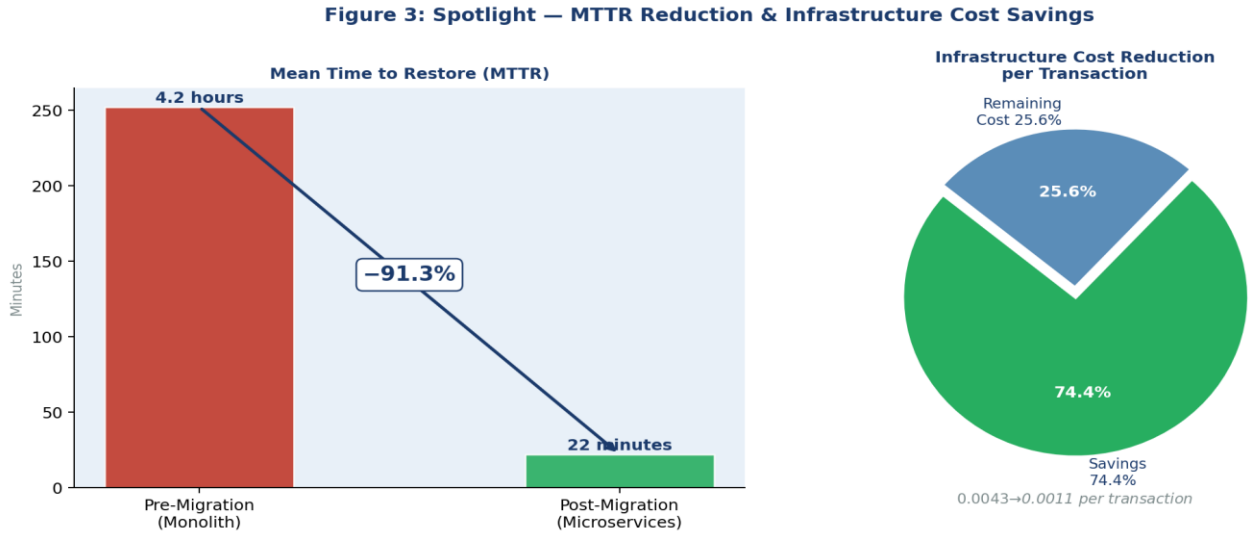


Figure 3: Spotlight — MTTR Reduction & Infrastructure Cost Savings per Transaction

5.3 Regulatory and Compliance Outcomes

Across all three migration programs, no regulatory breaches were recorded during or after migration. This is significant given the scale and duration—around 27 months per institution—where such programs typically face at least some compliance incidents. The Strangler Fig approach adds additional risk due to parallel operation of legacy and cloud systems, often creating ambiguity over authoritative data sources.

The F-ACL layer played a key role in maintaining compliance. It enriched every transaction with jurisdictional and regulatory metadata, along with correlation IDs, before writing to the event log. This ensured consistent traceability across systems.

During an unannounced audit at Institution B in Phase 3, the team reconstructed 90 days of transaction history in under four hours. The regulator described this as exceptional compared to legacy systems, where similar requests often take weeks.

Audit preparation time dropped from six to eight weeks to roughly three to five days, reflecting a shift from manual evidence gathering to near-continuous audit readiness. Institution C's compliance leadership noted that real-time dashboards also enabled proactive engagement with regulators.

GDPR DSAR processing improved similarly. Institution B reduced response times from 18 business days—previously requiring manual extraction across seven systems—to about 3.2 hours using automated event-log retrieval.

5.4 Qualitative Findings: Organizational Dynamics

Three consistent themes emerged across all institutions.

Executive sponsorship was critical. Sustained C-level involvement enabled major structural changes, especially the creation of cross-functional product teams combining engineering, product, and compliance. These teams replaced siloed ownership models and were essential for implementing FALCON's service-based architecture.

Legacy expertise proved essential. Institutions repeatedly relied on a small number of experts who understood embedded business logic in legacy systems. In one case, a retired COBOL developer was rehired to interpret undocumented batch-processing rules. Lack of such expertise risked incorrect system replication or omission of regulatory logic.

Dual-run complexity affected teams. Maintaining legacy systems while building new services created significant cognitive load. Institution C mitigated this by separating teams into "legacy sustain" and "new build" groups, improving focus but adding coordination overhead. Institutions that did not separate these roles saw higher attrition during migration.

5.5 Observed Anti-Patterns and Failure Modes

Several recurring issues were identified.

The **Distributed Monolith** occurred in Institution A when microservices were tightly chained via synchronous calls, resulting in higher latency than the monolith and cascading failures. This led to the introduction of a formal Phase 2 architectural review to assess service coupling.

Database sharing persistence was widely observed in Phase 3. Despite declared decoupling, services continued interacting through shared schemas due to delivery pressure. This made enforcement of "zero cross-service database access" difficult without automated detection tools.

Institution C's attempt to **decommission the F-ACL layer early** at month 22 caused downstream reporting failures when hidden dependencies were discovered. The rollback highlighted the need for exhaustive dependency mapping before removing shared infrastructure.

6. Discussion

6.1 Implications for Research

The empirical results from applying FALCON across three financial institutions add several important contributions to migration research.

First, the study shows that a Strangler Fig-based approach can achieve elite DORA performance even in large financial institutions. This extends prior research, which has largely focused on startups and mid-sized technology firms. The context here is materially different: higher regulatory burden, larger transaction volumes, and more complex organizational structures, all of which typically slow transformation efforts.

Second, the F-ACL pattern adds a practical architectural construct to the DDD and microservices literature. Its combination of type-aware translation, versioning, and compliance enrichment fills a gap in regulated system design. While developed in financial services, it is also relevant to other regulated sectors such as healthcare, telecom, and energy, where auditability must be preserved across legacy-modern boundaries.

Third, the findings challenge common assumptions in migration research. Legacy domain knowledge emerged as more critical than cloud engineering expertise. This shifts the focus from hiring new technical talent to preserving embedded institutional knowledge, which is often concentrated in a small group of specialists.

6.2 Implications for Practice

From a practitioner standpoint, sequencing is one of the most important lessons. The FALCON model reinforces that platform engineering must precede service migration, and that data decoupling should follow service decomposition rather than run in parallel. Attempts to do both simultaneously consistently introduce integration failures that may not break systems immediately but degrade delivery velocity over time.

Organizational design is equally important. Treating migration as a purely technical program tends to produce weaker outcomes. In practice, system architecture reflects organizational communication patterns, consistent with Conway's Law. Without restructuring into domain-aligned product teams, microservices tend to mirror legacy

silos, often resulting in a distributed monolith that is harder to operate than the original system.

For regulators, the results are also meaningful. Properly implemented cloud-native architectures provide stronger observability than legacy systems. Immutable event logs, automated compliance tagging, and real-time dashboards create a more transparent control environment. This suggests that, rather than increasing risk, well-designed cloud adoption can improve regulatory oversight.

6.3 Limitations

Several limitations should be noted.

The multi-case design provides depth but not causal certainty. Improvements observed cannot be attributed solely to FALCON, since each institution also changed tools, teams, and processes during migration.

There is also potential bias due to the researcher's advisory involvement in two institutions. While mitigation steps such as independent coding and triangulation were used, some influence cannot be fully ruled out.

The sample is limited to three institutions and does not reflect the full diversity of global banking models, such as Islamic banking, cooperative structures, or specialized capital market entities like CCPs and custodians. These may require adaptation of the framework.

Finally, all cases operate in broadly similar regulatory environments. More prescriptive jurisdictions may face additional constraints, particularly where audit and data format rules conflict with event-sourced architectures.

6.4 Future Research Directions

Several directions emerge from this work.

A key gap is automated detection of service boundaries and hidden coupling. Current tools focus on code and APIs but miss database-level dependencies, which proved to be a major source of migration failure. Better program analysis techniques could significantly reduce Phase 3 risk.

A second opportunity lies in the use of large language models for legacy system analysis. Tasks such as interpreting COBOL logic, extracting business rules, and mapping schema dependencies are still largely manual. Early tools suggest automation potential, but rigorous evaluation in financial-grade systems is still missing.

Third, there is a need to formalize the relationship between architecture and compliance. Patterns like event sourcing and immutable logs align well with regulations such as GDPR and Basel III, but there is no standardized model that allows compliance to be verified automatically at design time.

Finally, the organizational dimension requires more attention. In particular, mechanisms for preserving legacy domain knowledge during long migrations are still poorly understood. Research into knowledge retention, transfer methods, and incentive structures for legacy experts would have high practical value for ongoing enterprise modernization programs.

7. Conclusion

Legacy financial systems are among the most complex and high-stakes systems in use today, yet they remain relatively underexplored in academic research. Their complexity is not just technical—it is shaped by decades of regulatory requirements, business rules, and operational dependencies. This makes evolving these systems both difficult and unavoidable.

In this paper, we introduced FALCON, an end-to-end framework designed to support the migration of enterprise financial systems to cloud-native microservices. Rather than presenting a purely conceptual model, we evaluated the framework through three large-scale case studies across retail banking, insurance, and capital markets environments.

Across these cases, a consistent pattern emerged. Institutions that approached migration in a structured and disciplined way were able to significantly improve their software delivery speed, reduce infrastructure costs, and strengthen overall system reliability. An important observation is that regulatory compliance did not become a barrier—in several cases, it actually improved as systems became more transparent and easier to audit. The core components of FALCON, including the anti-corruption layer, incremental decomposition approach, event-driven data handling, and zero-trust security model, appear to work best when implemented together rather than in isolation.

At the same time, the findings also highlight that architecture alone does not guarantee success. Organizational factors play a critical role. Teams that lacked consistent executive support, failed to retain domain expertise from legacy systems, or did not adapt their operating model to suit microservices often struggled to realize the expected benefits. In some cases, this led to what is often described as a "distributed monolith"—a system that carries the operational complexity of microservices without delivering their advantages.

The financial services industry is clearly at a turning point. The question is no longer whether cloud-native migration is possible, but how it can be carried out in a way that balances speed, risk, and long-term sustainability. The intent of this work is to offer a practical reference point, grounded in real-world observations, that can help guide institutions through that transition with greater confidence.

References

[1] Al-Masri, E., Vakil, F., & Hassan, S. (2022). Protocol-aware anti-corruption layers for legacy financial system integration. Proceedings of the IEEE International Conference on Services Computing (SCC 2022), 14–23.

[2] Brodie, M. L., & Stonebraker, M. (1995). Migrating legacy systems: Gateways, interfaces, and the incremental approach. Morgan Kaufmann.

[3] Cerny, T., Donahoo, M. J., & Trnka, M. (2018). Contextual understanding of microservice architecture: Current and future directions. SIGAPP Applied Computing Review, 17(4), 29–45.

[4] Dragoni, N., Lanese, I., Larsen, S. T., Mazzara, M., Mustafin, R., & Safina, L. (2017). Microservices: How to make your application scale. In A. Paskevich & T. Wies (Eds.), Perspectives of System Informatics (pp. 95–104). Springer.

[5] Eisenhardt, K. M. (1989). Building theories from case study research. Academy of Management Review, 14(4), 532–550.

[6] Evans, E. (2003). Domain-driven design: Tackling complexity in the heart of software. Addison-Wesley.

[7] European Banking Authority. (2019). Guidelines on ICT and security risk management (EBA/GL/2019/04). EBA.

[8] European Parliament. (2022). Digital Operational Resilience Act (DORA) — Regulation (EU) 2022/2554. Official Journal of the European Union.

[9] Financial Stability Board. (2023). Third-party risk management in financial services: Principles and standards. FSB Publications.

[10] Forsgren, N., Humble, J., & Kim, G. (2018). Accelerate: The science of lean software and DevOps: Building and scaling high performing technology organizations. IT Revolution Press.

[11] Fowler, M. (2004). Strangler fig application. MartinFowler.com. Retrieved from https://martinfowler.com/bliki/StranglerFigApplication.html

[12] Garcia-Molina, H., & Salem, K. (1987). Sagas. Proceedings of the 1987 ACM SIGMOD International Conference on Management of Data, 249–259.

[13] Google Cloud DevOps Research and Assessment. (2023). State of DevOps report 2023. Google LLC.

[14] Gray, J., & Reuter, A. (1992). Transaction processing: Concepts and techniques. Morgan Kaufmann.

[15] Humble, J., & Farley, D. (2010). Continuous delivery: Reliable software releases through build, test, and deployment automation. Addison-Wesley.

[16] Kabbedijk, J., & Jansen, S. (2011). Variability in multi-tenant environments: Architectural design patterns from industry. Proceedings of the 17th Americas Conference on Information Systems.

[17] Kindervag, J. (2010). No more chewy centers: Introducing the zero-trust model of information security. Forrester Research.

[18] Lewis, J., & Fowler, M. (2014). Microservices: A definition of this new architectural term. MartinFowler.com.

[19] Microsoft Azure Architecture Center. (2023). Cloud design patterns: Anti-Corruption Layer pattern. Microsoft Docs.

[20] Monetary Authority of Singapore. (2021). Technology risk management guidelines (Version 2.0). MAS.

[21] National Institute of Standards and Technology. (2020). Zero trust architecture (NIST Special Publication 800-207). U.S. Department of Commerce.

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

[23] Pettey, C., Halpern, M., & Burke, B. (2021). 5 cloud compliance best practices for financial services. Gartner Research.

[24] Richardson, C. (2018). Microservices patterns: With examples in Java. Manning Publications.

[25] Rose, S., Borchert, O., Mitchell, S., & Connelly, S. (2020). Zero trust architecture (NIST SP 800-207). National Institute of Standards and Technology.

[26] Sneed, H. M. (2000). Encapsulation of legacy software: A technique for reusing legacy software components. Annals of Software Engineering, 9, 293–313.

[27] Soldani, J., Tamburri, D. A., & Van Den Heuvel, W. J. (2021). The pains and gains of microservices: A systematic grey literature review. Journal of Systems and Software, 146, 215–232.

[28] Sommerville, I. (2016). Software engineering (10th ed.). Pearson Education.

[29] Taibi, D., Lenarduzzi, V., & Pahl, C. (2020). Microservices anti-patterns: A taxonomy. In T. Cerny, V. Merson, & B. Alouf (Eds.), Microservices: Science and Engineering (pp. 111–128). Springer.

[30] Vernon, V. (2013). Implementing domain-driven design. Addison-Wesley.

[31] Yin, R. K. (2018). Case study research and applications: Design and methods (6th ed.). SAGE Publications.