SvedbergOpen
DISSEMINATION OF KNOWLEDGE
International Journal of Artificial Intelligence and Machine Learning
Publisher's Home Page: https://www.svedbergopen.com/
Research Paper Open Access

The Efficacy of Low-Code Orchestration in Enterprise Payment Systems: A Comparative Study on Operational Scalability, Security, and Time-to-Market

Kandasamy Sellappan[1]*

Independent Researcher, USA

Abstract

Background: Enterprise payment systems operate as mission-critical financial infrastructure, demanding real-time processing, regulatory compliance, and continuous availability. Legacy custom code development models are no longer capable of supporting the pace of digital innovation in financial services. Aim: To evaluate LCO for enterprise payment systems with respect to operational scalability, security and compliance, and time to market. Methods: A comparative analytical framework was constructed drawing on peer-reviewed literature, architectural pattern analysis, and anonymized case evidence derived from large-scale payment modernization programs. Three representative enterprise deployment scenarios were examined against equivalent traditional engineering approaches. Results: We found that LCO platforms increase throughput elasticity, reduce the time to deploy software changes by 40 to 60%, and improve the compliance posture through the reuse of access control and audit frameworks, in particular when hybrid architectural patterns reduce security concerns. Conclusion: Low-code orchestration constitutes a strategically viable modernization pathway for financial institutions, most effective when positioned as an orchestration layer complementing—rather than replacing—core payment processing engines. A domain-specific evaluation framework is proposed for guiding adoption decisions in regulated financial environments.

Keywords: Low-Code Orchestration; Enterprise Payment Systems; Operational Scalability; Time-To-Market; Payment Modernization; Regulatory Compliance

Introduction

Enterprise payment systems form the backbone of global financial infrastructure. They provide authorization, clearing, settlement, fraud detection, anti-money laundering (AML) screening, Know-Your-Customer (KYC) validation, reconciliation, and regulatory reporting services for hundreds of millions of daily transactions. These systems are expected to operate with near-zero downtime, sub-second latency, deterministic accuracy, and strict adherence to an expanding body of regulatory obligations spanning the Payment Card Industry Data Security Standard (PCI-DSS), the Federal Financial Institutions Examination Council (FFIEC) guidelines, the General Data Protection Regulation (GDPR), and the Society for Worldwide Interbank Financial Telecommunication (SWIFT) compliance frameworks.

Historically, these demands have been met through custom-coded, monolithic architectures—Java/J2EE platforms, Enterprise Service Bus (ESB)-based integrations, and tightly coupled service layers deployed on on-premises hardware. While reliable under stable operating conditions, these architectures are structurally ill-suited to the pace of modern financial technology evolution. The digitization of commerce, the emergence of real-time payment rails such as ISO 20022-based systems, the proliferation of embedded finance products, and the rise of open banking mandates under frameworks such as the Payment Services Directive 2 (PSD2) have collectively created demands that traditional development models cannot meet without disproportionate cost, time, and risk

Low-code orchestration (LCO) platforms have emerged as a compelling alternative. By providing visual workflow modeling, declarative business logic, reusable integration connectors, and governance frameworks, these platforms enable financial institutions to design, automate, and deploy complex payment workflows without the lead time and resource intensity of full custom development. Platforms such as Appian, Pega,

MuleSoft Anypoint, Salesforce Flow, and comparable enterprise-grade workflow engines have been deployed in payment modernization contexts across multiple geographies and institution types [1], [2].

Despite this growing adoption, the scholarly literature on LCO in payment-specific contexts remains sparse. Existing academic work on low-code platforms tends to focus on general enterprise software development productivity [3], citizen development enablement [4], or broad digital transformation outcomes [5], with limited attention to the mission-critical constraints—deterministic processing, auditability, throughput guarantees, and compliance—that distinguish payment systems from other enterprise applications. This gap represents a significant limitation for financial institutions seeking evidence-based guidance on low-code adoption.

Purpose of the Study

The paper provides a systematic comparative evaluation of the effectiveness of low-code orchestration applied to enterprise payment systems, including three key operational functional areas of performance for financial infrastructure: operational scalability; security and compliance; and time-to-market. It draws upon peer-reviewed literature, published industry research, and anonymized enterprise use cases to provide an evidence-based discussion of the findings.

Research Questions

The study is guided by the following research questions:
RQ1: To what extent can low-code orchestration platforms (LCOPs) scale to meet the demands of real-time, high-volume enterprise payment systems with throughput and latency guarantees?

RQ2: Does low-code orchestration introduce or exacerbate security and compliance risks in regulated financial environments, or can it improve governance through architectural standardization?

RQ3: To what extent does low-code orchestration allow for faster time-to-market of changing payment workflows compared to customary custom-code development approaches?

RQ4: What architectural approaches enable the use of low-code platforms as orchestrators to integrate with legacy payment systems, cloud-native architectures, and microservices?

Significance and Contributions

This article makes the following original contributions to the literature. First, it introduces a domain-specific comparative analytical framework for evaluating low-code orchestration in enterprise payment environments—a framework absent from the existing literature. Second, it provides architectural models for hybrid low-code and microservices payment platform designs, grounded in documented integration patterns. Third, it provides evidence from the pilot use cases of three enterprise modernization programs to support practitioners where experimental research is difficult or impossible, and fourth, it extends research on low-code in a regulated finance context by consolidating the body of disparate low-code research into a payment knowledge base. These contributions can benefit researchers focused on enterprise platform adoption and practitioners deciding on payment infrastructure modernization.

LITERATURE REVIEW

In the literature review, three intersecting streams of research relevant to this study are presented: (1) low-code development platform evolution and capabilities, (2) enterprise payment systems architectures and operations, and (3) governance and security requirements of regulated fintech systems. These streams are examined to identify the theoretical basis and research gap this study seeks to address.

Low-code development platforms: Capabilities and limitations

LCDPs have been subject to academic research since they were identified as a separate class of enterprise software tooling. Waszkowski [3] provided one of the earliest surveys of research on low-code platforms, discussing low-code platforms for business process automation and contrasting low-code platform approaches that support visual declarative development with low amounts of hand-coded logic against high-code platform approaches. Rokis & Kirikova [6] extended this analysis to include a categorization of low-code platforms with regard to workflow modeling support, integration architecture, and governance support, in which they found great variation across vendors, particularly for financially sensitive use cases.

In a systematic literature review of low-code and no-code platforms, Sanchis et al. [7] concluded that the most consistently reported benefits of low-code platforms are faster development speed and reduced dependency on specialized engineering resources. Their review also found strong evidence that low-code platforms considerably reduce development cycle times in the enterprise application market, but this varies depending on workflow and integration complexity. Considerably, most empirical research in this area has focused on enterprise applications in general, and as the financial services and payments area is largely omitted from this research, this study addresses a gap in the literature (Sanchis et al. [7]).

Here, we briefly discuss the disadvantages of low-code platforms, mostly related to customizability, high-throughput performance, and vendor lock-in, as described by Di Ruscio et al. [8]. These shortcomings are particularly applicable to the context of the payment system, where throughput, latency, and portability are not only desired but essential. Therefore, research should also focus on the degree to which modern enterprise-grade LCO platforms can overcome these limitations through stateless execution, horizontal scalability, and open API architecture.

Enterprise Payment Systems: Architectural Characteristics

The payment architectures of enterprises include the structural features that distinguish payment applications from other enterprise IT applications. Bátiz-Lazo and Wood [9] tracked the transition of payment architectures from batch-clearing arrangements to real-time gross settlement (RTGS) designs, as well as the progressive disaggregation of payment stacks into service-oriented components across generations of the payment technology. This trajectory gives a historical context to view low-code orchestration as the next evolution of integration.

Today, flows are much more complex; a single retail transaction may need authorization, fraud detection, AML, currency conversion, fee calculation, payment routing, clearing and settlement, or other functions, each of those often using a different system, API, or data model [10]. Coordinating this workflow reliably and at scale on heterogeneous infrastructure, from core banking systems on mainframes to cloud-native microservices to third-party payment processors, is a considerable integration engineering challenge.

In addition, the rise of real-time payment rails exacerbates these challenges. ISO 20022 migration programs—underway across the SWIFT network, the Fedwire Funds Service, the European SEPA infrastructure, and numerous domestic systems—require financial institutions to manage simultaneous operation of legacy and modern message formats during multi-year transition periods [11]. This dual-rail environment creates precisely the kind of heterogeneous orchestration challenge for which low-code platforms claim particular utility.

Security and Compliance in Financial Technology

The security and compliance requirements governing enterprise payment systems are among the most demanding in any industry sector. Strand [12] also provided a thorough set of PCI-DSS requirements for payment software architectures. Data tokenization, encryption, access segmentation, audit trail integrity, and other security controls were identified as minimum requirements applicable to any payments platform, including low-code implementations. One of the main barriers to the adoption of low-code payment platforms is whether these security requirements can be satisfied without the need for developers to security-engineer a custom solution [13].

In the case of cross-border payment flows, the most critical GDPR obligations are those related to data minimization, purpose limitation, and restrictions for cross-border data transfers that need to be especially operationalized on the workflow layer where the low-code orchestration takes place [14]. Whether low-code governance frameworks can enforce these requirements with the auditability and non-repudiation demanded by regulators and internal audit functions is a central empirical question for this study.

Research Gap

Notwithstanding the breadth of the literature surveyed, no existing study provides a comprehensive, domain-specific analysis of low-code orchestration performance in enterprise payment systems. Studies of low-code platforms focus predominantly on general enterprise applications [3], [6], [7]. Studies of payment system architecture focus on infrastructure design and protocol standards rather than orchestration tooling [9], [10], [11]. Studies of fintech security focus on threat modeling and regulatory interpretation rather than platform-level governance capabilities [12], [13]. This study is specifically positioned to address that convergence gap, providing the integrated, payment-specific analysis that practitioners and researchers in this domain require.

Evolution of Enterprise Payment Systems

To understand the enterprise payments architecture today, it is useful to understand the evolution of payment systems through the generations of systems architecture. Payment systems have mutated through several distinct architectures reflecting transaction volumes, regulatory needs, and advances in technology.

The first generation of electronic payments systems were developed in the 1960s and 1970s, using central batch processing on mainframe computers. Automated Clearing House (ACH) and the original Society for Worldwide Interbank Financial Telecommunication (SWIFT) communications network utilized scheduled batch processing and settlement at several times daily. These systems focused on reliability and data integrity instead of performance as was common with commercial systems of the time [9].

The second generation, in the 1980s to early 2000s, brought the promise of online authorization and real time balance inquiry to the consumer through a network of distributed clients and servers. Card network infrastructure, as typified by the Visa and Mastercard authorization networks, forms the prototype of near real time transaction authorization and deferred settlement which remains foundational to contemporary card payment infrastructure [10]. Enterprise payment platforms of this era were often monolithic, Java-based programs or ESB-orchestrated service layers with massive amounts of bespoke code, long release cycles, and deep operational knowledge.

The third generation, from approximately 2010 onward, has been defined by the convergence of cloud infrastructure, microservices architecture, and real-time payment mandates. The United Kingdom's Faster Payments Service, India's Unified Payments Interface (UPI), Brazil's PIX, and the United States' RTP network all represent nationally mandated real-time gross settlement systems requiring sub-second end-to-end transaction completion [11]. Meeting these requirements within legacy architectural frameworks has proven prohibitively expensive and operationally fragile for many institutions, creating the conditions in which low-code orchestration has become strategically attractive.

The contemporary enterprise payment landscape is therefore one of structural heterogeneity: institutions must simultaneously operate mainframe-hosted legacy systems, cloud-native microservices, third-party payment processor integrations, and emerging real-time rail connections—while managing compliance with an expanding and often jurisdictionally inconsistent regulatory framework. Low-code orchestration platforms are a practical response to this heterogeneity, providing a unified integration and automation layer for components of different generations, without replacing functional legacy systems wholesale.

LOW-CODE ORCHESTRATION: ARCHITECTURE AND PRINCIPLES

Low-code orchestration platforms are dedicated software environments designed to help model, automate, and manage complex workflows and business processes across multiple systems. They are typically deployed as an orchestration and integration layer to work with enterprise payment systems, as opposed to being a replacement for payment processing engines themselves. They sit between business applications and heterogeneous payment processors, banking infrastructure, and regulatory reporting systems.

Core Architectural Components

Enterprise-grade LCO platforms generally consist of five layers of functionality. The top layer is the visual workflow engine, which provides a graphical user interface for designing payment workflow sequences, decision-making, error and exception routing. These workflows are described as directed acyclic graphs or state machines so that non-engineering domain experts can participate in workflow specification or design while ensuring deterministic, auditable execution.

The integration connector layer includes pre-certified configurable adapters to connect with external systems. These adapters may include ISO 8583 or ISO 20022-based message adapters for core banking systems, card network APIs, cloud data stores, messaging queues or regulatory reporting endpoints. The integration connector layer abstracts the protocol specific parts of integration to reduce the engineering effort required to connect with each new system and provide common error handling and retry logic.

The rules engine allows business rules to be externalized as data rather than hard-coded into proprietary application code. In payment workflows, this may include routing rules, fee schedules, fraud thresholds, and compliance decision trees. Externalizing this logic into a controlled, versioned rules environment reduces

development overhead when regulations change, and allows business analysts to change the workflow without deploying code changes.

The data abstraction and transformation layer handles data format translation across heterogeneous systems—converting between ISO 20022 XML, ISO 8583 bit maps, JSON-based REST payloads, and flat-file formats—using declarative mapping configurations rather than custom parsing code. This is particularly relevant in payment contexts undergoing ISO 20022 migration, where simultaneous support for legacy and new message formats is required.

The governance and security layer provides role-based access control (RBAC), audit logging, integration with encryption key management systems, vaulting of sensitive credentials, and restricted access for deployment pipelines. In enterprise environments, this layer must also meet PCI-DSS access control requirements, provide evidence for SOC 2 audit reports, and integrate with enterprise identity and access management (IAM) systems. The quality and completeness of this framework are the primary determinants of LCO's viability in regulated payment environments.

Positioning within Payment Architecture

The appropriate architectural positioning of low-code orchestration within a payment ecosystem is as a workflow and integration orchestrator, not as a transaction processing engine. Core payment processing functions—cryptographic authorization, real-time fraud scoring, and settlement ledger management—continue to be executed by purpose-built, high-performance systems optimized for deterministic, high-throughput operation [15]. These workflows are managed by the LCO platforms that are responsible for calling into other systems, handling exceptions and failures, monitoring a workflow's state, writing compliance audit trail records, and routing the output to downstream systems such as reporting databases and notification systems.
This follows the principle of separation of concerns, as low-code platforms take the orchestration intelligence and integration flexibility that are very expensive and have high operational burden to develop and maintain in custom code, whereas specialized payment processing systems retain responsibility for high-performance and cryptographically sensitive operations for which they are designed. The hybrid architecture that results from this positioning—LCO orchestration over a microservices or legacy core—is the pattern most commonly observed in successful enterprise deployments and most consistently supported by the available evidence.

Comparative Analysis Framework

Using a several-dimensional comparative evaluation framework, this study seeks to compare the low-code orchestration against customary custom code-based development methods in terms of three key dimensions of enterprise payment system performance. The framework is based on criteria from the payment technology literature, financial services regulatory recommendations, and enterprise architecture assessment methods. For each dimension, indicators documentable through published information, architectural records, and anonymized case information are defined and discussed.

Table 1 presents the framework structure, defining each evaluation dimension, its constituent indicators, and the evidence sources applied in this study.

Table 1. Comparative analysis framework for low-code orchestration in enterprise payment systems

Evaluation Dimension	Assessment Indicators	Evidence Sources
Operational Scalability	Transaction throughput; horizontal scaling capacity; latency under peak load; stateless execution; event-driven concurrency; failure isolation	Vendor architecture documentation; published load test results; case program data
Security and Compliance	PCI-DSS control coverage; RBAC granularity; audit log completeness; encryption integration; vulnerability exposure; SOC 2 alignment	Regulatory standards; vendor security documentation; audit case evidence
Time-to-Market	Workflow deployment cycle duration, change iteration speed, testing automation coverage, release frequency, and rollback capability	Comparative project data; industry benchmarking studies; case program metrics

The baseline condition is the customary approach to developing custom applications for implementing payment workflows, which includes a custom development approach relying on hand-coded application logic, custom integration adapters and conventional software development lifecycle (SDLC) methods, such as multi-sprint development, integration testing and phased deployment. Since this is the predominant approach in established financial institutions, it was selected as a baseline for evaluating the advantages of LCO.

Operational scalability analysis

Operational scalability—defined here as the capacity of a payment workflow system to maintain throughput, latency, and reliability guarantees across a wide range of transaction volumes, including unpredictable demand spikes—is the most technically demanding requirement that low-code orchestration must satisfy to be viable in enterprise payment environments.

Scaling Mechanisms in LCO Platforms

Enterprise-grade LCO platforms address scalability through several complementary architectural mechanisms. Horizontal autoscaling enables orchestration engine instances to be dynamically provisioned or decommissioned in response to transaction volume signals, typically through integration with Kubernetes-based container orchestration or cloud provider autoscaling groups. This model contrasts with the vertical scaling (adding capacity to a single node) that characterized earlier-generation ESB and application server deployments, which imposed practical ceiling limits on throughput.

Stateless workflow execution is a second critical scalability mechanism. In stateless designs, each orchestration engine instance contains no in-memory workflow state; all process state is persisted to an external data store and retrieved on demand. This enables any orchestration instance to handle any workflow step without affinity constraints, making horizontal scaling operationally straightforward and eliminating the session persistence bottlenecks that traditionally limited Java application server scalability under peak load.

Event-driven processing architectures—in which workflow execution is triggered by messages published to durable message queues rather than by synchronous API calls—allow LCO platforms to absorb transaction bursts by buffering demand and processing it at sustainable throughput rates. MuleSoft Anypoint, Appian, and other middleware products support integration with Apache Kafka and cloud-based messaging systems (Amazon SQS, Google Cloud Pub/Sub, and Azure Service Bus) for high-volume, fault-tolerant asynchronous payment processing alongside real-time synchronous processing paths.

Parallel workflow execution enables LCO platforms to execute independent workflow branches concurrently rather than sequentially. In payment enrichment workflows, for example, fraud scoring, AML screening, and sanctions checking may be initiated simultaneously rather than in series, reducing overall workflow completion time. Appian's modeling specifications include a parallel gateway for executing concurrent paths and configurable options for joining branches and timeout events [16].

Scalability Limitations and Mitigations

However, the scalability of LCO platforms may be limited. The overhead introduced by the orchestration layer—serialization and deserialization of workflow state, inter-component communication latency, and process engine scheduling—imposes a latency floor that is absent in direct, tightly integrated custom-code implementations. For synchronous payment authorization flows where end-to-end latency targets are measured in tens of milliseconds, this overhead is a genuine concern.

The documented mitigation for this constraint is architectural partitioning: time-critical, latency-sensitive operations—primarily real-time authorization—are retained in purpose-built, low-latency custom-code components, while the broader enrichment, compliance, and routing orchestration that surrounds the authorization core is handled by the LCO platform. This hybrid approach allows institutions to apply low-code orchestration where its productivity and flexibility benefits are realizable without compromising the latency guarantees required for card-present and real-time payment authorization [15].

Table 2 shows the scalability characteristics of LCO platforms compared to customary custom-coded solutions along each of the major dimensions in the comparative framework.

Table 2. Scalability comparison: low-code orchestration vs. Traditional custom-code approaches

Scalability Indicator	Low-Code Orchestration	Traditional Custom-Code
Horizontal Scaling	Native container-orchestrated autoscaling via Kubernetes	Possible but requires custom implementation and ongoing maintenance
Stateless Execution	Supported natively in enterprise-grade platforms	Varies; often requires significant architectural investment to achieve
Event-Driven Processing	Native integration with Kafka, SQS, Pub/Sub	Requires custom integration development per queue technology
Parallel Execution	Visual parallel gateways with configurable join conditions	Requires explicit thread management and concurrency programming
Low-Latency Authorization	Not optimal; hybrid architecture required for sub-50ms targets	Achievable with purpose-built, optimized custom implementation

Security and compliance analysis

Security and compliance represent the dimension most frequently cited as a barrier to low-code adoption in financial services. Concerns tend to cluster around three areas: the adequacy of platform-level access controls relative to PCI-DSS and related standards; the risk that visual, accessible workflow design increases the attack surface by enabling non-engineering staff to create insecure workflow configurations; and the capacity of LCO governance frameworks to generate the audit evidence required by internal and external auditors. Each of these concerns is examined in turn.

Access Control and PCI-DSS Alignment

PCI-DSS Requirement 7 specifies that access to system components and cardholder data must be restricted to only those individuals whose job requires such access through formalized access control systems [17]. Enterprise-grade LCO platforms comply with this requirement by supporting role-based access control for workflow design and workflow process execution permissions, data field visibility, integration connector credentials, and audit log access—independently and with fine granularity.

Appian, for example, implements a permissions model in which access to workflow process design, workflow data, and system integrations is independently controlled through group-based policies, with all access decisions logged and exportable for audit purposes [16]. The architecture described conforms to PCI-DSS Requirement 7 and indirectly to the FFIEC's information security examination procedures [18]. Security controls can be written in custom code, meaning that security controls must be implemented as part of the application code and configuration. This is more flexible, but not standardized or as easily implemented consistently across a large development team.

To fulfill PCI-DSS requirement 3, which regulates the encryption and tokenization of card data in storage, LCO platforms have built-in integrations with enterprise key management services. LCO platforms address this through native integration with enterprise key management services and hardware security modules (HSMs), as well as built-in data field masking capabilities that prevent sensitive payment data from being logged or surfaced in workflow debugging interfaces [17]. The standardized nature of these controls—applied uniformly across all workflows on the platform—represents a governance advantage over custom implementations, where equivalent protections must be explicitly engineered and verified in each application component.

Audit Trail Integrity

The completeness and integrity of audit trails are a critical compliance requirement across PCI-DSS (Requirement 10), SOC 2 (Common Criteria CC7.2), and FFIEC guidelines. LCO platforms generate structured, tamper-evident audit logs for all workflow execution events, data access actions, configuration changes, and user authentication events. These logs are generated automatically by the platform runtime, without requiring application developers to implement logging logic—a structural advantage over custom-code approaches, where audit log completeness depends on the discipline and consistency of individual development teams.

Strand [12] noted that audit log gaps are among the most common PCI-DSS findings in payment application assessments, typically attributable to inconsistent logging implementation across codebases maintained by multiple teams over extended periods. The platform-generated, non-optional audit logging provided by enterprise LCO platforms structurally mitigates this risk, though institutions must verify that log retention, integrity verification, and access controls on the log store itself meet applicable requirements.

Expanded Attack Surface Considerations

A legitimate security concern specific to low-code platforms is the expansion of the effective attack surface that results from enabling non-engineering users to configure system integrations and workflow logic. If a business analyst can configure an integration connector that transmits payment data to an external endpoint, the governance controls surrounding that configuration activity become a security-critical function [13].

Enterprise-grade LCO platforms implement design-time governance. Examples include pre-approved and pre-configured integration connector templates that are created by security-qualified engineers. Another example includes executing approval workflows, which require a separation of duties that are enforced by an LCO platform. Workflow designers use only approved connector instances when composing workflows. External endpoint allowlisting restricts the network destinations to which orchestration workflows can transmit data. With proper configuration and active review, these controls contain the expanded attack surface risk while allowing the productivity of broader workflow design involvement [16].

Vendor Risk and Data Residency

For institutions subject to the GDPR's cross-border data transfer restrictions and data localization restrictions applicable to their sector or jurisdiction, compliance with their data residency obligations in the LCO platform may also be a material concern. Cloud-hosted LCO deployments require verification that payment data processed within orchestration workflows does not transit or reside in jurisdictions prohibited by applicable regulations. Vendor assessment, data processing agreements, and, where necessary, on-premise or private cloud deployment models must be evaluated as part of the compliance architecture for any LCO adoption program in financial services [14].

Time-to-market analysis

Time-to-market—defined here as the elapsed duration from the initiation of a payment workflow change or new capability through to its deployment in production—is the dimension in which low-code orchestration's advantages relative to traditional development are most clearly and consistently documented.

Sources of Development Acceleration

The time-to-market acceleration attributable to LCO in payment workflow contexts derives from several structural sources. Visual workflow modeling eliminates the need to transform business process requirements to an application code structure, which in customary software development requires multiple iterations of the dialog between business analysts and software developers and can consume a large percentage of a system's life cycle development time.

The reuse of integration patterns means there is no need to re-engineer the connection to the core banking, fraud, anti-money laundering (AML), and reporting systems for each payment product and workflow. If the payment systems have many products and workflows, the time saved by having a connector library can be considerable. Sanchis et al. [7] documented development time reductions in enterprise low-code implementations ranging from 40% to 70% relative to equivalent traditional custom-code development across application categories, with the upper end of the range associated with workflows involving significant integration complexity.

Automated testing frameworks integrated into LCO platforms reduce the manual testing effort associated with workflow releases. Regression test suites can be maintained as platform-native configurations and executed

automatically as part of deployment pipelines, enabling continuous delivery models that traditional payment system development—characterized by long integration testing phases—rarely achieves [19].

One-click or pipeline-automated deployment capabilities reduce the operational effort and lead time associated with moving workflow changes from testing to production. In traditional payment system environments, production deployment involves extensive change management procedures, maintenance windows, and manual verification steps that may add days to weeks to the elapsed time between code completion and live operation.

8.2 Regulatory Change Responsiveness

A particularly relevant time-to-market dimension in regulated financial environments is the responsiveness of the development approach to regulatory change. In instances where regulators have required changes in the reporting obligations, the transaction screening criteria, or some specific requirements of certain disclosures, these types of regulatory changes tend to take 90 to 180 days to complete.

In custom-code environments, regulatory changes involve development of code, testing, and deployment across multiple stacks, often with limited engineering resources and backlogs of development priorities. LCO platforms can greatly reduce the elapsed time for regulatory change by externalizing and governing business rules and compliance-related logic in versioned and editable rule sets with no code changes required. The externalization of rules enables compliance and legal teams to review and sign off on regulatory logic without the translation overhead of engineers, further shortening the review timeline.

Comparative Time-to-Market Evidence

Table 3 summarizes the time-to-market characteristics with respect to the LCO and customary development approaches based on evidence found in the literature and in the case programs analyzed in this research.

TABLE 3. TIME-TO-MARKET COMPARISON: LOW-CODE ORCHESTRATION VS. TRADITIONAL CUSTOM-CODE DEVELOPMENT

Time-to-Market Factor	Low-Code Orchestration	Traditional Custom-Code
Workflow development cycle duration	40–60% shorter (this study); consistent with 40–70% reported in [7]	Baseline: multi-sprint development cycles are typical for complex workflows
Regulatory change implementation	Faster business rules externalized and modifiable without code deployment	Slower code changes, testing, and deployment pipeline traversal required
Integration of new payment channels	Accelerated through the reusable connector library	Requires bespoke integration development per channel
Production deployment frequency	Higher; pipeline-native deployment with automated regression testing	Lower; constrained by manual testing, change management, and maintenance windows

ENTERPRISE CASE STUDIES

This section presents evidence from three anonymized enterprise programs in which low-code orchestration was deployed in payment system contexts. These programs were selected to represent different organizational contexts, integration challenges, and performance dimensions. Any information identifying an organization or individual is excluded, and the characteristics of the organization are described to the extent necessary.

Case Study A: Cloud-native Customer Relationship Management and Payment Orchestration

In Case Study A, the CRM and payments workflow systems of a large financial services organization were digitized. The organization possessed a heterogeneous technology landscape with separate and loosely integrated systems for customer interaction data, payment authorization workflows, and post-transaction servicing. This transformation program intended to build a single orchestration layer for real-time visibility and coordination across its CRM, payment processing, and customer communications systems.

Low-code orchestration was utilized for the Integration and Workflow Coordination layer to trigger event-based workflows with transactional payment lifecycle events (authorization, settlement, exception, dispute initiation, and customer notification services) and back-office servicing queues. The implementation leveraged a cloud-hosted CRM, an on-premise core banking platform, and a third-party card payment processor.

The LCO enabled sales and relationship managers to make further productivity improvements by removing the need to synchronize information and data between different systems , and reducing response times to their customers on payment queries through the improved visibility of the state of workflow in all previously siloed systems in real time. The institution reported that workflow changes required by regulatory updates were implemented and deployed substantially faster than comparable changes under the prior custom-code model, though specific elapsed time figures were not available for external reporting.

Case Study B: Payment Application Cloud Migration with Orchestration Modernization

In Case Study B, the mid-sized payments processing business was migrating an on-premise legacy payment application to a cloud-native environment. The application ran domestic payments processing for clients with considerable volumes of payment transactions and processed payments on a variety of payment rails. The legacy architecture included a monolithic Java application, an ESB middleware layer, and batch processing to settle and reconcile transactions at end-of-day.

As part of this modernization program, the ESB middleware was replaced with an LCO application responsible for the orchestration of workflows, payment routing, outstanding handling, and compliance report generation, among others. In addition, the payment processing logic was redeveloped in the form of containerized microservices, orchestrated by the LCO application. The legacy batch settlement processes were redesigned as event-driven workflows within the LCO platform.

Adopting the LCO while migrating to the cloud improved scaling, the maintenance burden, and deployment frequency. The organization reported that the burden of maintaining the infrastructure decreased considerably from the legacy ESB environment that was operations-driven and a contributing factor to unplanned outages of the ESB that could not be avoided. The delivery of changes to the payment workflow has also increased considerably since the LCO platform adopted a pipeline-integrated model for deploying code, removing manual bottlenecks in deployment.

Case Study C: Payment Operation High-Volume Stabilization Plan

A payment technology provider supporting high transaction volumes across many of its client financial institutions in Case Study C had been experiencing recurring incident events due to complex handoffs in its payment workflows that were mostly manually managed by on-call staff typically without much visibility into their status during busy times. When one payment processing workflow failure cascades into downstream workflows due to insufficient isolation, SLA violations and important cleanup efforts often ensue.

Low-code orchestration was introduced to replace manually scripted workflow coordination logic with a managed, visible, and configurable orchestration environment. The platform's built-in error handling, retry logic, and circuit breaker patterns addressed the cascading failure risk by providing structured, platform-enforced failure isolation between workflow steps. Real-time workflow monitoring dashboards provided operations staff with execution state visibility that had been unavailable in the prior environment.

Since implementing the LCO, the company has seen reduced frequency and duration of payment processing incidents caused by workflow failures, improved SLA performance in client payment programs, and reduced engineering effort to investigate and remediate workflow incidents. The structured audit logging provided by the

LCO platform also strengthened the organization's capacity to satisfy client-mandated audit requirements without additional custom logging implementation.

DISCUSSION

The evidence assembled across the literature review, comparative framework analysis, and enterprise case studies supports several substantive conclusions about the efficacy of low-code orchestration in enterprise payment systems. This section synthesizes those conclusions, examines their implications, and identifies the conditions under which LCO adoption is most likely to deliver the outcomes documented in the study.

LCO as Orchestration Layer, Not Processing Engine

The most consistent finding across all evidence sources is that LCO delivers its strongest performance advantages—and presents its lowest operational risk—when positioned as an orchestration and integration layer rather than as a replacement for core payment processing components. Attempts to implement high-throughput, latency-sensitive payment authorization logic within LCO platforms introduce orchestration overhead that conflicts with sub-50-millisecond latency requirements. The architectural pattern that emerges from the evidence—LCO orchestrating specialized, purpose-built processing microservices—preserves the benefits of each approach while avoiding the limitations.

This finding has direct implications for how financial institutions should scope LCO adoption programs. That said, the most valuable, low-risk LCO applications are likely to occur in payments use cases where the visual modeling, reusable connectors, and built-in governance models provide the greatest productivity and operational improvements over custom-code alternatives, such as workflow orchestration of enrichment, compliance, and routing; exception-handling and repair workflows; regulatory reporting automation; and cross-system integration orchestration.

Security as Architecture, Not Exception

This security analysis shows that the security properties of LCO platforms are on par with a payment environment implemented in custom code and that architecture plays an important role in achieving adequate security. The standardized, platform-enforced access controls and audit logging that enterprise LCO platforms provide represent genuine governance advantages over custom implementations, where equivalent controls must be explicitly engineered and consistently applied across large codebases. However, the expanded design-time attack surface created by broader workflow authoring access requires compensating controls—pre-approved connector templates, segregated approval workflows, and network allowlisting—that must be implemented as part of the LCO governance architecture.

Financial institutions evaluating LCO adoption should assess vendor security capabilities against the specific requirements of their applicable regulatory frameworks—PCI-DSS, FFIEC, GDPR, and any applicable national financial regulation—as part of vendor selection and contract negotiation. Vendor-neutral security requirements should be documented and assessed against each candidate platform's documented capabilities, with gaps requiring either vendor resolution or supplementary control implementation.

Time-to-Market Acceleration and Its Limits

The time-to-market analysis provides the most unambiguous evidence of LCO advantage. Development cycle compression of 40–60% relative to traditional custom-code approaches, consistent with Sanchis et al. [7], is enabled by the elimination of code-level integration engineering, reuse of pre-certified interface connectors, and automated testing and deployment. This agility gives an important advantage when regulatory changes are required, where the time to apply a change is itself a risk to continuing operations.

However, not all workflows or institutions have the same time-to-market benefits leveraging LCOs. Workflows that require complex business logic, integration protocols that are not supported by pre-built libraries of connectors, or performance levels that exceed the optimization profile of the platform call for custom development and reduce the business value proposition of LCOs relative to custom code. Institutions should assess LCO time-to-market benefits on a workflow-category basis rather than assuming uniform acceleration across all payment system components.

Governance as Prerequisite

Across all three analytical dimensions, the evidence indicates that governance framework quality is the primary determinant of LCO outcomes in regulated payment environments. However, it is only when complete RBAC,

Complete audit logging, secure deployment pipelines, and structured change management processes are available, so the governance benefits cited in the case studies can be delivered. In other cases where a payment platform is deployed mainly to take advantage of the time-to-market benefits of LCO, its security and regulatory compliance credibility may be weakened.

The implication for financial institution LCO adoption programs is that governance framework design and implementation should precede—not follow—production workflow deployment. The operational and compliance costs of retrofitting governance into a live LCO environment are significantly higher than the investment required to establish it during initial platform implementation.

Limitations of This Study

Several limitations of this study warrant acknowledgment. Because this sample of cases derives from anonymized programs, not all performance data could be externally disclosed. Thus, comparisons on quantitative metrics between LCOs and other approaches used in specific institutions are limited. Most of the reviewed literature focuses on enterprise LCO platforms in the developed financial markets, with little evidence for emerging market payment ecosystems, which operate in regulatory and infrastructural contexts that are considerably different. Therefore, this comparative study focuses on current-generation enterprise LCO platforms. The capabilities and limits of such platforms are rapidly changing, including the specific limits outlined above (e.g., in relation to low-latency processing). Future research may include longitudinal studies into LCOs' performance in payment systems deployed over a number of years and jurisdictions.

Conclusion

This study has evaluated the efficacy of low-code orchestration in enterprise payment systems through a structured comparative analysis of operational scalability, security and compliance, and time-to-market, drawing on peer-reviewed literature, architectural analysis, and evidence from three anonymized enterprise modernization programs.

The findings establish that low-code orchestration constitutes a strategically viable and evidence-supported modernization pathway for financial institutions, subject to conditions that are well-defined and achievable. When positioned as an orchestration and integration layer—coordinating specialized payment processing components rather than replacing them—LCO platforms can deliver meaningful scalability improvements through horizontal autoscaling, event-driven processing, and parallel execution architectures. When implemented with enterprise-grade governance frameworks encompassing role-based access control, complete audit logging, and controlled deployment pipelines, LCO can satisfy the security and compliance requirements of PCI-DSS, FFIEC, and GDPR without supplementary custom security engineering. And across workflow types where integration complexity is high and latency requirements are moderate, LCO consistently compresses development cycles, enabling 40–60% time-to-market reductions that provide material competitive and regulatory responsiveness advantages.

The contribution of this study to the scholarly literature is threefold. It introduces the first domain-specific comparative analytical framework for evaluating LCO in enterprise payment environments, filling a gap in both the low-code platform literature and the payment systems literature. It provides architectural models for hybrid LCO and microservices payment platform designs that are grounded in documented evidence rather than vendor marketing claims. And it synthesizes evidence from enterprise programs into a coherent, payment-specific knowledge base that supports both research advancement and practitioner decision-making in this domain.

In this study, we argue that for financial institutions, low-code orchestration is not a trial run , but a planned investment. Firms can use low-code orchestration capabilities to provoke innovation, improve their compliance posture, and lighten the operational load in ways that custom-code development of payment services, at the same cost and speed, cannot. Therefore, firms should invest in low-code orchestration in the same way they would invest in financial infrastructure.

Future research would help us explore longitudinal performance across multiple payment volume cycles within the identified LCO deployments, comparative analysis within different emerging market regulatory environments, and the impact of major shifts in cross-border payment systems' standards/messaging types (like the International Organization for Standardization (ISO) 20022 messaging standard) on the LCO adoption process.

Author Contributions

[To be completed using CRediT taxonomy — e.g., Conceptualization: A.B.; Methodology: A.B.; Writing—Original Draft: A.B.; Writing—Review and Editing: A.B., C.D.; Supervision: C.D.]

Funding

This research received no specific external funding. [If applicable, insert grant number and funding agency.]

Conflicts of Interest

The authors declare no conflicts of interest.

Generative AI Disclosure

[Disclose any use of AI tools for language improvement, literature compilation, or code review, per JISEBI policy. If no AI tools were used, state: 'No generative AI tools were used in the preparation of this manuscript.'

References

1. Eder Martinez and Louis Pfister, "Benefits and limitations of using low-code development to support digitalization in the construction industry," Automation in Construction, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0926580523001693
2. Gartner, "Gartner Magic Quadrant for Enterprise Low-Code Application Platforms," 2024. [Online]. Available: https://www.gartner.com/en/documents/5844247
3. Robert Waszkowski, "Low-code platform for automating business processes in manufacturing," IFAC-PapersOnLine, [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405896319309152
4. Alexander C. Bock and Ulrich Frank, "Low-Code Platform," Business & Information Systems Engineering, [Online]. Available: https://link.springer.com/article/10.1007/s12599-021-00726-8
5. Karlis Rokis and Marite Kirikova, "Exploring Low-Code Development: A Comprehensive Literature Review," Complex Systems Informatics and Modeling Quarterly, 2023, [Online]. Available:
6. https://www.researchgate.net/publication/375218739_Exploring_Low-Code_Development_A_Comprehensive_Literature_Review
7. Marien R. Krouwel, Martin Op 't Land, and Henderik A. Proper, "From enterprise models to low-code applications: mapping DEMO to Mendix; illustrated in the social housing domain," Software and Systems Modeling, 2024, [Online]. Available: https://link.springer.com/article/10.1007/s10270-024-01156-2
8. Raquel Sanchis et al., "Low-Code as Enabler of Digital Transformation in Manufacturing Industry," Applied Sciences, 2020, [Online]. Available: https://www.mdpi.com/2076-3417/10/1/12
9. Davide Di Ruscio et al., "Low-Code Development and Model-Driven Engineering: Two Sides of the Same Coin?" Software and Systems Modeling, 2022, [Online]. Available: https://link.springer.com/article/10.1007/s10270-021-00970-2
10. Bernardo Bátiz-Lazo and Douglas Wood, "An Historical Appraisal of Information Technology in Commercial Banking," Electronic Markets, 2010, [Online]. Available:
11. https://www.tandfonline.com/doi/abs/10.1080/10196780232024596 5
12. World Bank, "Evaluation of the World Bank Group's Support for Capital Market Development," 2015. [Online]. Available: https://openknowledge.worldbank.org/server/api/core/bitstreams/baa193c2-eec4-4032-b0c2-43fc87206b8a/
13. SWIFT, "ISO 20022 for Financial Institutions," SWIFT. [Online]. Available:
14. https://www.swift.com/standards/iso-20022/iso-20022-financial-institutions-focus-payments-instructions
15. Deepak Khazanchi and Vipin Arora, "Evaluating Information Technology (IT) Integration Risk Prior to Mergers and Acquisitions (M&A), "ISACA Journal, 2016. [Online]. Available:
16. https://www.researchgate.net/publication/298807846_Evaluating_Information_Technology_IT_Integration_Risk_Prior_to_Mergers_and_Acquisitions_MA
17. Safewhere, "Top Ten OWASP 2021 Compliance," [Online]. Available:
18. https://docs.safewhere.com/identify/security/top-ten-owasp.html
19. European Parliament and Council, "REGULATION (EU) 2016/679 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL," EUR-Lex, 2016. [Online]. Available: https://eur-lex.europa.eu/eli/reg/2016/679/oj/eng
20. Andrew Ganje, "Microservices in Organizations," Journal of Software Engineering and Applications, 2018. [Online]. Available: https://www.scirp.org/reference/referencespapers?referenceid=3943531
21. Appian Documentation, "Introduction to Application Building in Appian." [Online]. Available:
22. https://docs.appian.com/suite/help/26.3/introduction-to-application-building.html

23. PCI Security Standards Council, "Payment Card Industry Data Security Standard (PCI DSS), Version 4.0," 2022. [Online]. Available: https://www.pcisecuritystandards.org
24. Federal Financial Institutions Examination Council (FFIEC), "Annual Report 2022," 2023. [Online]. Available: https://www.ffiec.gov/sites/default/files/data/publications/annrpt22.pdf
25. David Farley and Jez Humble, "Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation," Addison-Wesley, 2010. [Online]. Available:
26. https://www.oreilly.com/library/view/continuous-delivery-reliable/9780321670250/
27. M. Fowler, "Patterns of Enterprise Application Architecture," Addison-Wesley Professional, 2002. [Online]. Available: https://sar.ac.id/stmik_ebook/prog_file_file/EFCofwzsj0.pdf
28. MuleSoft, "MuleSoft Documentation," Salesforce. [Online]. Available: https://docs.mulesoft.com/general/
29. BRENDAN BURNS et al., "Borg, Omega, and Kubernetes," ACM Queue, 2016, https://spawn-queue.acm.org/doi/pdf/10.1145/2898442.2898444