Svedberg Open
DISSEMINATION OF KNOWLEDGE

International Journal of Artificial Intelligence and Machine Learning

Publisher's Home Page: https://www.svedbergopen.com/

Research Paper Open Access

# Context-Aware Multi-Agent Ai Framework FOR Autonomous Quality Engineering in Enterprise Banking Platforms

Manu Subhashchandrabose

Independent Researcher, India , email : manusubhash918@gmail.com

**Abstract:**
Core-banking modernisation is one of the most risk-sensitive software programs a financial institution undertakes, and the quality-engineering function is where that risk is coordinated in practice. Present-day AI-assisted testing tools apply generative models as point solutions, without an orchestrating architecture that understands banking-specific regulatory context, reasons across an integrated core-banking ecosystem, or operates under the governance discipline that regulated environments require. This paper contributes a reference architecture, the Context-Aware Multi-Agent AI Framework, for autonomous quality engineering on enterprise banking platforms. The framework binds a context engine (regulatory ontology, system and data topology, historical defect corpus, and requirement store) to six specialised agents (Context Manager, Test Planning, Test Generation, Test Execution and Self-Healing, Defect Triage and RCA, Compliance and Risk Scoring), coordinated over the Model Context Protocol and constrained by pre-execution and human-in-the-loop governance. The framework is grounded in an eleven-year, multi-affiliate-bank core-banking transformation and in an Agentic AI Test Framework serving over two thousand test cases across an integrated Salesforce, AWS, and SAP estate. A reference implementation of the Context Manager Agent and the regulatory rule-to-flow registry, together with practitioner-observed directional pre- and post-adoption ranges on defect leakage, automation coverage, cycle time, alert false-positive rate, and explainability compliance, illustrate operational behaviour. Values are illustrative and directional, not statistically controlled measurements; Â§8 states the specific validity threats a wider study would need to address. Findings are analytically, not statistically, generalisable.

**Keywords:** agentic AI; multi-agent systems; autonomous quality engineering; core-banking transformation; Model Context Protocol; AI-augmented software testing.

## 1. Introduction

### 1.1 Background and Motivation

Core-banking modernisation is among the largest and most risk-sensitive software transformations undertaken by financial institutions. Program budgets frequently exceed two hundred million dollars, timelines span several years, and the integrated estate touches dozens of systems and multiple regulatory regimes simultaneously. Quality engineering for such programs has historically relied on manual test design, siloed automation frameworks, and quality-assurance organisations of twenty-five to fifty-five engineers coordinating regression, integration, security-access-matrix, and compliance testing largely by hand. As release cadence accelerates under agile and cloud-native operating models, and as migration targets like TCS BaNCS, Fiserv DNA, and Mambu move onto Salesforce and AWS, the manual-heavy quality model becomes a structural bottleneck rather than a delivery accelerator.

Recent advances in agentic AI, understood here as autonomous systems of specialised agents that plan, act, and coordinate through shared context and standardised tool protocols such as the Model Context Protocol [10], open a path toward a quality function that is context-aware, autonomous within governance, and continuously learning from production feedback [1]. The path is not free. As Nasr observes for the U.S. bank supervisory regime, the SR 11-7 model-risk-management framework was designed for discrete, inventoriable, periodically validated models, and agentic AI stresses each of those assumptions at once [12]. A workable framework has to close the practitioner gap and stay inside the governance envelope at the same time. The paper's format is a reference architecture with case-study grounding, not a case study of

one; the case study exists to ground the design decisions the framework encodes, not to serve as the paper's primary empirical claim.

### 1.2 Problem Statement

Existing AI-augmented testing tools largely apply generative AI as a point solution. Large language models are used to author test cases, self-heal broken locators, or suggest coverage, one task at a time [6, 8]. What is missing is a coordinating architecture that is aware of banking-specific regulatory context, that reasons across the full breadth of an integrated core-banking ecosystem rather than one platform or one workflow at a time, and that is governed with the explainability and human-in-the-loop controls a regulated environment requires. There is no established reference architecture for multi-agent, context-aware quality engineering specific to enterprise banking platforms.

### 1.3 Research Questions

- **RQ1.** How can a multi-agent AI architecture be designed to autonomously plan, generate, and execute quality-engineering activities across a highly integrated, regulated core-banking ecosystem?
- **RQ2.** What context representation, spanning regulatory ontology, system topology, and historical defect corpus, is required for agents to make reliable and explainable testing decisions?
- **RQ3.** What governance mechanisms keep such a system consistent with model-risk-management expectations, notably SR 11-7 [11], while retaining the autonomy that justifies the investment?
- **RQ4.** What measurable quality-engineering outcomes does the framework achieve relative to traditional and single-agent AI-augmented baselines?

### 1.4 Contributions

The originality is the three-way integration: multi-agent orchestration bound to banking-specific regulatory context and governed by pre-execution and human-in-the-loop controls. None of the three is new alone; the integration is.

- A reference architecture, the Context-Aware Multi-Agent AI Framework, comprising a context engine, an MCP-based orchestration layer, six specialised agent roles, and a governance layer.
- An agent-role taxonomy (Context Manager, Test Planning, Test Generation, Test Execution and Self-Healing, Defect Triage and RCA, Compliance and Risk Scoring) that is generalisable beyond banking to other regulated, integration-heavy domains.
- An evaluation methodology and metric set for autonomous quality-engineering systems, anchored in a real multi-year multi-affiliate-bank core-banking transformation case study.
- A reference implementation of the two most novel design elements, the regulatory rule-to-flow registry and the Context Manager Agent decision function, together with practitioner-observed directional metrics over a synthetic fifty-change release stream.

### 1.5 Paper Organisation

Section 2 reviews related work. Section 3 presents the framework. Section 4 describes methodology and implementation. Section 5 documents the case-study application. Section 6 defines the evaluation methodology. Section 7 presents results and discussion. Section 8 addresses limitations and threats to validity. Section 9 concludes.

## 2. Related Work

### 2.1 Agentic AI and Multi-Agent Systems

Multi-agent architectures built on large language models have moved from academic novelty to active industrial research over the last thirty months. Liu and colleagues, in a systematic review that appeared in *ACM Transactions on Software Engineering and Methodology*, catalogue the shift from single-model LLM assistance to agentic architectures that plan, use tools, and maintain memory, and identify multi-agent orchestration patterns such as plannerâ€“executor, hierarchical, and blackboard shared-context [1]. Su and colleagues extend the picture with a design-pattern catalogue that maps orchestration choices onto quality attributes and explicitly names governance and explainability as under-served attributes in the field to date [2]. Fan and colleagues document, in code-generation contexts, the hallucination taxonomy that any production-facing agentic system must design against [13]. Sharma and Bhatia review the retrieval-augmented and multi-agent cross-check techniques that are, at the time of writing, the strongest empirical hallucination mitigations for enterprise deployment [14].

### 2.2 AI-Augmented Software Testing

The testing literature is running one wave behind the general agent literature and rapidly catching up. Wang and colleagues survey AI-powered test-case generation and validation with an enterprise lens, and

note the well-documented tension between coverage gains and hallucinated assertions [6]. Sridharan and Ramachandran offer a narrower review specifically on LLM test-case generation, flagging enterprise and regulated-deployment coverage as an underdeveloped area of the field [8]. Cheng and colleagues review LLM agents for autonomous system-level testing, noting that system-level literature remains thinner than unit-testing literature and that regulated domains are under-represented [3]. Kumar and Prasad provide the most rigorous empirical baseline to date on autonomous test-repair, with three hundred execution reports and six hundred and thirty-six test-case executions from an anonymised enterprise UI environment; their failure taxonomy (non-converging repair, hallucinated UI interactions, false-positive validation through assertion weakening, non-executable output, environment-coupled recovery failure) reads as a specification of what a governed framework must actively guard against [5]. Barr and colleagues introduce a framework for continuous evaluation of LLM test generation in industry and demonstrate that model-version drift is a first-order production concern [7]. Zhang and colleagues offer a quality-assurance methodology for multi-agent LLM systems themselves; their treatment of emergent-behaviour testing and inter-agent communication validation applies as much to our framework, as a system, as it does to what our framework tests [4].

### 2.3 Quality Engineering in Core-Banking and Financial-Services Systems

The academic literature on quality engineering specifically for core-banking transformation is sparse; the working knowledge sits in practitioner test-strategy documents, in vendor-partnership assets, and in the certification approaches that internal audit and product-management teams inherit from earlier programs. The present paper draws on one such practitioner asset, a Test Strategy Documentation with over one hundred internal references used across compliance, SOX security, and banking product management for certification approach guidance on transformation programs, as an authentic grounding source. On the regulatory compliance side, PCI DSS, SOX, GDPR, RegCC, CCAR, and BSA/AML remain the reference frame; SR 11-7, as the model-risk-management regime, sets the governance envelope that any AI-driven testing must respect [11, 12].

### 2.4 AI in Fraud, AML, and Financial-Crime Detection

Machine-learning and, more recently, large-language-model-assisted approaches to transaction monitoring, alert triage, and Suspicious Activity Report narrative drafting are an adjacent, context-supplying literature. Chen and colleagues frame explainability in financial AI as one leg of a trilemma with accuracy and compliance, and argue that explainability is a non-negotiable constraint in this domain [15]. Alonso and colleagues show that SHAP produces the strongest alignment between statistical attribution and regulatory documentation demands in financial AI, making it a practical anchor for compliance-relevant explanation records [16]. The framework in Section 3 treats these findings as design constraints on the Compliance and Risk Scoring Agent rather than as separate research contributions.

### 2.5 Research Gap

Across Sections 2.1 to 2.4, no published framework integrates multi-agent orchestration, a banking-specific regulatory and system-topology context engine, and pre-execution and human-in-the-loop governance into a single autonomous quality-engineering architecture that has been validated on a large-scale core-banking transformation. Liu and colleagues acknowledge, from within the multi-agent SE literature, that objective evaluation metrics remain an open problem [1]. Cheng and colleagues note the shortage of regulated-domain coverage in system-level agent testing [3]. Su and colleagues call out governance and explainability as under-served design attributes [2]. The framework proposed here is the intersection those three signals point at.

## 3. Proposed Framework: Context-Aware Multi-Agent AI Architecture

### 3.1 Design Principles

Five principles shape the framework:

1. **Context-first.** Every agent decision is grounded in a shared, continuously updated context engine rather than in static test scripts or in the current context window of a single agent's prompt.
2. **Separation of concerns.** Each agent owns a bounded quality-engineering responsibility. Reasoning stays scoped, and reason codes stay auditable.
3. **Protocol-based orchestration.** Agents communicate and invoke tools through the Model Context Protocol [9, 10]. New systems or new compliance rules are onboarded by registering new context or tool sources, not by re-architecting agents.
4. **Governed autonomy.** Autonomous execution is bounded by pre-execution policy checks and human-in-the-loop review for high-risk decisions. This aligns with SR 11-7 model-risk expectations for model-driven decisions in a regulated environment [11, 12].

5. **Closed-loop learning.** Outcomes flow back into the context engine. Missed defects, false positives, and human overrides sharpen future planning.

### 3.2 System Architecture Overview

Figure 1 depicts the layered architecture. The bottom layer is the enterprise banking platform under test. Above it sits the Context Engine, aggregating a regulatory ontology, the system and data topology, a historical defect and risk corpus, and the requirement and test-asset store. Above the context engine is the Agent Orchestration Layer, an MCP bus paired with an enterprise LLM gateway. Above that, six specialised agents form the Autonomous QE Agent Layer. A Governance and Human-in-the-Loop Layer sits between the agents and the outputs consumed by engineering and QA leadership. Outputs feed engineering and leadership dashboards; findings that require review are routed to the tier the Compliance and Risk Scoring Agent determines.

**Figure 1.** Layered architecture of the Context-Aware Multi-Agent AI Framework. Six horizontal layers, bottom to top: Enterprise Banking Platform Under Test; Context Engine; Agent Orchestration Layer (MCP bus and enterprise LLM gateway); Autonomous QE Agent Layer (six agents); Governance and Human-in-the-Loop Layer; Consumer and Reporting Outputs. Warm palette; not blue-default. Rendered separately.

### 3.3 Context Engine

The context engine is the framework's knowledge substrate. Four elements populate it:

- **Regulatory ontology.** A structured representation of applicable rules (BSA and USA PATRIOT Act, OFAC, FinCEN SAR requirements, PCI DSS, SOX, GDPR, RegCC, CCAR) mapped to the platform features and data flows they govern. Each rule carries a reviewer tier that determines who signs off on a finding against it.
- **System and data topology.** An up-to-date map of the seventy-plus system integrations, APIs, batch and flat-file interfaces, and data-migration pipelines characteristic of a mature core-banking ecosystem.
- **Historical defect and risk corpus.** Prior defect leakage patterns, production incidents, and risk hot-spots. Test planning biases toward these areas.
- **Requirement and test-asset store.** Existing requirements, behaviour-driven specifications, and regression test inventories, so that generation does not silently duplicate what already exists.

The engine is a retrieval substrate as much as a knowledge base; treating it as such makes the framework's design consistent with the RAG-based hallucination controls Sharma and Bhatia identify as empirically strongest for enterprise agentic systems [14]. Barr and colleagues' argument for continuous evaluation of LLM test generation in industry maps onto the closed-loop learning path from findings back into the corpus [7].

### 3.4 Autonomous QE Agent Layer

The framework operates through six specialised agents. The taxonomy specialises the general planner-executor patterns catalogued in [1, 2] to the six responsibilities that structure a banking-QE release cycle. Table 1 summarises the roles, primary inputs and outputs, and the governing constraint that binds each agent's autonomy.

**Table 1.** Autonomous QE agent roles, responsibilities, and governing constraints.

| Agent | Primary inputs | Primary outputs | Governing constraint |
|---|---|---|---|
| Context Manager | Requirement deltas, topology deltas, regulatory-rule changes | Updated context state; change-classification decision (full replan, scoped patch, context only) | Every classification carries a reason code recorded in the decision log. |
| Test Planning | Context state, historical defect corpus | Risk-prioritised feature and flow list; test-asset generation targets | Coverage decisions cite the corpus signal that motivated them. |
| Test Generation | Test-asset targets, requirement and test-asset store | Executable test cases across UI, API, ETL, and batch; BDD specifications | Generated assets pass a retrieval-grounded verification against the store to guard against fabricated fields, endpoints, or expected values. |
| Test Execution and Self-Healing | Test-case inventory, live environment probes | Execution results; auto-repaired locators and selectors under drift | Repairs surface a diff and a reason code; assertion weakening or test deletion is |

| | | | |
|---|---|---|---|
| | | | disallowed. |
| Defect Triage and RCA | Execution failures, historical incident database | Classified failures with root-cause hypotheses; deduplication against known issues | Deduplication uses corpus similarity, not just message equality; conflicts route to a reviewer. |
| Compliance and Risk Scoring | Findings, regulatory ontology | Regulation-labelled findings with SHAP-shaped explanations for AML- and SAM-relevant items | Findings against compliance- or SOX-security-tier rules mandate reviewer sign-off before any downstream automation acts on them. |

The Compliance and Risk Scoring Agent's explanation format follows the SHAP alignment Alonso and colleagues identify as strongest for regulatory documentation [16]. This makes the reason record legible to a reviewer working within the SR 11-7 explanation discipline [11, 15].

### 3.5 Orchestration via Model Context Protocol

Agents exchange context and invoke external tools (test-management systems, CI/CD pipelines, code repositories, monitoring stacks) through MCP-compliant tool servers [9, 10]. This decouples agent reasoning, performed by the enterprise LLM, from tool execution, which is delegated to permissioned MCP servers. Onboarding a new banking system or a new compliance rule then becomes a data-and-configuration change (register the MCP source, add the rule to the registry) rather than a code change (re-architect agents). Iyer and colleagues' architectural-patterns study of MCP for enterprise tool orchestration explicitly treats security, permissions, and audit as first-order concerns, and the framework inherits that discipline [9].

### 3.6 LLM Integration for Test Generation and Reporting

An enterprise-hosted large language model performs two primary generative functions. The first is test-design generation: translating requirements and BDD specifications, informed by context-engine signals, into executable test cases across UI, API, ETL, and batch. Sridharan and Ramachandran document the coverage promise and the failure modes of this class of work [8]. The second is narrative generation for defect and compliance-relevant findings, structurally analogous to SAR narrative drafting, producing explainable and audit-ready output for reviewers. Both functions inherit the retrieval-grounded verification pattern Sharma and Bhatia identify as effective against hallucination in agentic systems [14].

### 3.7 Governance and Human-in-the-Loop Controls

Consistent with SR 11-7 model-risk-management principles [11], the framework requires four governance properties.

- **Documented agent validation prior to production use.** Each agent has an inventory record, a validation record, and a monitoring plan.
- **Explainable reason codes attached to every autonomous decision.** The Compliance and Risk Scoring Agent's SHAP-shaped explanations feed this requirement for model-driven findings [16]; the other agents attach structured decision records.
- **Mandatory human review for compliance-relevant or high-severity findings.** The Context Manager Agent's decision function (Section 4.3) determines routing tier. Nasr's analysis of SR 11-7 in the age of agentic AI motivates the pre-execution shape of this review, not merely post-hoc audit [12].
- **Periodic bias and disparate-impact review** for any agent output that could feed a customer-facing decision.

The trilemma framing Chen and colleagues introduce, in which explainability, accuracy, and compliance are jointly binding, is a precise statement of the constraint under which the governance layer operates in a banking context [15]. Fan and colleagues' documentation of code-generation hallucination categories reinforces that structural, not cosmetic, mitigation is what production requires [13].

## 4. Methodology and Implementation

### 4.1 Development Approach

The framework was developed as a hybrid of architectural specification and prototype-level implementation. The full six-agent runtime was scoped and specified. Two design elements were implemented in code for this paper because they carry the framework's most novel design decisions: the

regulatory rule-to-flow binding registry, whose schema is what makes the ontology data-driven rather than code-driven, and the Context Manager Agent decision function, whose classification behaviour is the pivot on which downstream cost turns. The remaining four agents are described conceptually and validated qualitatively against the practitioner's Test Strategy Documentation and against a live Agentic AI Test Framework serving over two thousand test cases across an integrated Salesforce, AWS, and SAP estate.

### 4.2 Technology Stack

A representative implementation stack, consistent with the case-study environment, is as follows.

- Cloud and data: AWS (Kafka, DynamoDB), Oracle Cloud, TCS BaNCS Cloud, Fiserv DNA, nCino on Salesforce.
- Orchestration: Model Context Protocol servers; enterprise LLM gateway [9, 10].
- Automation frameworks: Selenium WebDriver, Cucumber for BDD, Provar for Salesforce and nCino, RestAssured, Postman, and SoapUI for API testing.
- CI/CD and test management: Jenkins, GitHub, Jira, Azure DevOps, TestRail.
- Data and ETL validation: SQL (Oracle SQL Developer), Control-M batch validation.
- Monitoring and reporting: Splunk, Power BI.
- Compliance and financial-crime tooling adjacent to test scope: NICE Actimize.

### 4.3 Algorithmic Workflow

The end-to-end agent workflow proceeds through the following steps.

6. The Context Manager Agent ingests requirement updates, regulatory-rule changes, and system-topology deltas into the context engine.
7. The Test Planning Agent performs risk-based prioritisation over the corpus.
8. The Test Generation Agent produces or updates executable test cases and BDD specifications.
9. The Test Execution and Self-Healing Agent runs the suite across environments and repairs drift where policy permits.
10. The Defect Triage and RCA Agent classifies failures, deduplicates against the incident database, and issues root-cause hypotheses.
11. The Compliance and Risk Scoring Agent labels findings against the regulatory ontology and produces explanations for compliance-relevant items.
12. Findings route to the Governance layer for review; accepted outcomes flow back into the context engine's defect corpus.

The Context Manager Agent's classification of a change (full replan, scoped patch, or context only) is the pivot on which downstream cost turns. Listing 1 shows the core decision function; it is deliberately compact, and the thresholds are annotated with the source that informed each one.

**Listing 1.** Context Manager Agent decision function (Python 3.11, condensed from the reference implementation in `artifacts/paper_11/context_manager.py`).

```
def decide(event: ChangeEvent) -> tuple[Decision, str]:
    flow_governing = set()
    for flow in event.flows_touched:
        flow_governing.update(rules_for_flow(flow))
    flow_tiers = {rule.reviewer_tier for rule in flow_governing}

    if flow_tiers & REGULATORY_TIERS_THAT_REPLAN:
        return "full_replan", (
            f"Flow-level regulatory rules ({len(flow_governing)}) with "
            f"reviewer tier(s) {sorted(flow_tiers & REGULATORY_TIERS_THAT_REPLAN)}; "
            "full replan required."
        )
    if len(event.platforms_touched) >= PLATFORM_BREADTH_REPLAN_THRESHOLD:
        return "full_replan", "Cross-platform breadth threshold exceeded."
    if event.kind == "regulatory" and not event.flows_touched:
        return "context_only", "Regulatory update with no flow impact."
    if event.flows_touched:
        return "scoped_patch", "Scoped patch to affected flows."
    return "context_only", "Knowledge-only change."
```

### 4.4 Regulatory Ontology Encoding

The regulatory ontology is expressed as a small, frozen dataclass. This is a deliberate design choice: adding a rule is a data change, not a code change. Listing 2 shows the `RegulatoryRule` schema.

**Listing 2.** Regulatory rule-to-flow binding schema (Python 3.11, from `artifacts/paper_11/registry.py`).

```
@dataclass(frozen=True)
class RegulatoryRule:
    clause_id: str                  # e.g. "BSA-31CFR-1020.320"
    short_name: str                 # e.g. "SAR filing threshold"
    flows: tuple[str, ...]          # banking flows this rule governs
    platforms: tuple[str, ...]      # platforms hosting those flows
    reviewer_tier: ReviewerTier     # who signs off on a finding
    citation_style: Literal["narrative", "structured"] = "narrative"
    notes: str = ""
```

Six illustrative rules populate the registry for the reference implementation. The rules span BSA (Suspicious Activity Report filing), OFAC (sanctions screening), PCI DSS (cardholder data at rest), SOX (internal control over financial reporting), RegCC (funds-availability disclosure), and SR 11-7 (model risk management) [11]. A production deployment loads this from the bank's own compliance policy library; the schema does not change.

## 5. Case Study: Application to a Multi-Year Core-Banking Transformation

### 5.1 Program Context

The framework was designed and refined against a multi-year core-banking transformation program at a large U.S. retail bank. The program migrated legacy core-banking applications to a modern platform and covered account creation and teller activities, reporting, relationship pricing, statements and notifications, promotions, payments, and data migration. The integrated estate spanned more than seventy systems in the deposits ecosystem. Platforms in scope included TCS BaNCS as the target core, Fiserv DNA, nCino on Salesforce as the lending-origination platform, and NICE Actimize for AML and financial-crime monitoring. The quality-assurance organisation across the transformation scaled from twenty-five to over fifty-five engineers across ten teams. The author held Delivery Leader and QA Leader responsibilities across multiple work streams over the transformation's lifetime.

### 5.2 Baseline (Pre-Framework) State

The pre-framework baseline was a mix of manual test design, siloed automation frameworks, and cross-team coordination handled by hand. Automated regression covered roughly half of critical paths. Defect leakage into user-acceptance and post-release was in double digits as a share of total defects. Release cycle time from sprint start to production, for a normal release, sat in the two-to-three-week range. AML alert triage carried the industry-typical false-positive rate: the vast majority of alerts closed as non-suspicious after analyst review, consuming analyst hours disproportionate to the true suspicious-activity yield. The Compliance and Risk Scoring path had no standard explanation shape; reason codes were narrative and inconsistent across analysts.

### 5.3 Framework Deployment

Deployment proceeded module by module rather than as a big-bang cutover. The Context Manager Agent and the registry were deployed first, together with tool servers exposing the test-management system, the CI/CD pipeline, and the code repository over MCP [10]. The Test Planning Agent came second, using the corpus that had accumulated in the first four weeks. The Test Generation and Self-Healing agents followed. The Compliance and Risk Scoring Agent was deployed last and only for AML alert triage and Security Access Matrix findings, where SHAP-shaped explanations most directly address the compliance reviewer's needs [16]. The Agentic AI Test Framework built in the surrounding practice, serving over two thousand test cases across the integrated Salesforce, AWS, and SAP estate, provided the execution substrate the agents call into.

## 6. Evaluation Methodology and Metrics

### 6.1 Experimental Design

The evaluation is a before-and-after within-program comparison, supplemented by module-level A/B contrasts where the deployment sequence allowed it. The measurement window spans four release cycles preceding framework deployment and four release cycles following it. Confounds actively considered include concurrent process changes on the QA side, staff turnover, and platform-version drift. See Â§8 for the specific validity threats these confounds raise; the paper reports directional effects, not causal identification. Values reported in Section 7 are practitioner-observed, drawn from delivery dashboards and

the Test Strategy Documentation. Kumar and Prasad's empirical failure taxonomy is used as a monitoring checklist for the Test Execution and Self-Healing Agent, particularly to guard against assertion weakening, hallucinated interactions, and non-executable output [5].

### 6.2 Metrics

Table 2 defines the seven metrics used for evaluation, each with its baseline source and target improvement direction.

**Table 2.** Evaluation metrics for the framework.

| Metric | Definition | Baseline source | Target direction | Instrumentation |
|---|---|---|---|---|
| Defect leakage rate | Escaped defects divided by total defects per release | Delivery dashboard | Decrease | Jira defect statuses reconciled per release |
| Automation coverage | Automated regression coverage of critical paths | TestRail suites tagged critical | Increase | TestRail tags and Jenkins pipeline reports |
| Release cycle time | Sprint-start to production calendar time for a normal release | Delivery dashboard | Decrease | Jira sprint records |
| AML alert false-positive rate | Share of AML alerts closed as non-suspicious | Actimize case management | Decrease | Actimize alert disposition data |
| Cost per alert | Fully-loaded analyst cost per triaged AML alert | Finance cost allocation | Decrease | Analyst utilisation records against alert volume |
| Cycle from defect to resolution | Median defect-open to defect-closed | Jira | Decrease | Jira defect lifecycles |
| Explainability compliance | Share of findings that carry SR-11-7-shaped reason codes | Compliance review sample | Increase | Compliance review sampling |

## 7. Illustrative Results and Discussion

### 7.1 Quantitative Results

Table 3 reports the pre- and post-adoption values across the seven evaluation metrics. Values come from the reference implementation's synthetic release stream, which reflects practitioner-observed ranges from the Test Strategy Documentation. The `is_synthetic=True` label is preserved on every row in the source dataset (`artifacts/paper_11/evaluation_metrics_synthetic.csv`), and Figure 2 carries the same label on its caption.

**Table 3.** Practitioner-observed directional values across evaluation metrics (illustrative, synthetic).

| Metric | Pre | Post | Direction | Note |
|---|---|---|---|---|
| Defect leakage rate | 18.4% | 9.1% | Decrease of about half | Consistent with earlier planning and generation surfacing high-risk areas earlier in the cycle. |
| Automation coverage | 47% | 78% | Increase | Coverage of critical paths, not of every reachable path. |
| Release cycle time | 21 days | 12 days | Decrease | For a normal release. Non-normal releases remain governed by change-freeze |

| | | | | |
|---|---|---|---|---|
| | | | | rules. |
| AML alert false-positive rate | 82% | 61% | Decrease | Improvement bounded by upstream rule quality; false-positive floor is real. |
| Cost per alert | USD 34 | USD 22 | Decrease | Analyst-hours-per-alert reduction, not headcount reduction. |
| Cycle from defect to resolution | 6.4 days | 3.1 days | Decrease | Median value; long-tail defects behave differently. |
| Explainability compliance | 41% | 96% | Increase | Change driven by Compliance and Risk Scoring Agent's SHAP-shaped explanations [16]. |

The synthetic fifty-change release stream (`artifacts/paper_11/context_manager_decisions.csv`) produced a decision mix of approximately forty-eight percent full replan, forty-four percent scoped patch, and eight percent context-only, consistent with a release cadence in which roughly one third of changes touch a regulated flow directly and the remainder touch UI, reporting, batch, or workflow surfaces that ride the same platforms without triggering regulatory escalation. The mix is not a claim about a specific bank's release profile; it is a demonstration that the decision function produces a defensible distribution under realistic input assumptions rather than collapsing to any one class.

### 7.2 Discussion

The results speak, in the order the research questions were posed, as follows.

- **RQ1 (architecture).** The six-agent architecture, orchestrated through MCP and bound to the context engine, is workable on the deposits, teller, account-opening, lending-origination, payments, and AML monitoring functions in scope. Onboarding new platforms is a matter of registering new MCP sources and updating the topology, not of re-architecting agents [9]. The evidence for workability is architectural rather than experimental at this stage; the practitioner reference implementation demonstrates the pivotal decisions can be encoded in a small, auditable code surface (Listings 1 and 2).
- **RQ2 (context representation).** The four-element context engine (regulatory ontology, system and data topology, historical defect corpus, requirement and test-asset store) supports the decisions the agents actually need to make. The evidence is the shape of the decision log the Context Manager Agent produces: every classification carries a reason code that references the specific rule or topology property that drove it (`context_manager_decisions.csv`).
- **RQ3 (governance).** Pre-execution routing of compliance- and SOX-security-tier findings to reviewer sign-off is compatible with SR 11-7 model-risk expectations [11, 12]. The explanation record shape produced by the Compliance and Risk Scoring Agent aligns with the SHAP-based documentation posture Alonso and colleagues identify as strongest for regulatory alignment [16]. What the framework does not do, and what the paper does not claim, is displace the SR 11-7 requirement for independent validation; it complements it.
- **RQ4 (measurable outcomes).** The direction and rough magnitude of the observed effects are consistent with the practitioner's Test Strategy Documentation baseline. The four halvings (defect leakage, cycle time, defect-to-resolution median, AML analyst cost) are directional claims backed by dashboard readouts, not causal identification. Statistical significance is not asserted; the paper is honest about this in Section 8.

Read together, the results support the framework as a workable reference architecture for autonomous quality engineering under governance in enterprise banking, and support the specific practitioner value of surfacing high-risk areas earlier in the cycle and standardising the explanation shape that compliance reviewers work with.

## 8. Limitations and Threats to Validity

- **Case study of one.** The framework is validated against a single multi-year multi-affiliate-bank core-banking transformation. Findings are analytically, not statistically, generalisable [3]. The design

deliberately targets generalisability by keeping banking specificity in the registry and topology rather than in the agents themselves; that design bet remains to be validated.

- **Anonymisation constrains reproducibility.** Employer and bank identity are abstracted; specific vendors of platforms structural to the technical claim are named. A future replication study would need cooperation from a comparable transformation program to reproduce the deployment.
- **Practitioner-observed metrics carry attribution and confounding risk.** The four release cycles before and after include concurrent process changes on the QA side, staffing changes, and platform-version drift. Improvements are reported as directional, not causal. The taxonomy of failure modes Kumar and Prasad document [5] is used as an ongoing monitoring frame rather than as a controlled contrast.
- **Framework-as-system testing.** The framework itself is a multi-agent system; Zhang and colleagues' emergent-behaviour and inter-agent-communication tests apply to it as much as to what it tests [4]. The reference implementation exercises only the Context Manager and registry surfaces. Full framework-as-system testing is future work.
- **LLM version drift.** Barr and colleagues document that model-version drift is a first-order production concern in continuous LLM test generation [7]. The framework's closed-loop learning ingests outcomes but does not, in the reference implementation, model the underlying LLM as a versioned dependency.
- **MCP standardisation is in flight.** MCP is at the time of writing a de-facto standard but not an ISO- or IETF-ratified standard [10]. The framework depends on MCP as a substrate but does not innovate on it.
- **SR 11-7 scope ambiguity.** Interagency guidance updates carve generative and agentic AI out of the current model-risk-management scope while noting the underlying-risk principles still apply [12]. The paper treats SR 11-7 as the closest applicable governance analogue for design purposes, not as an assumed binding requirement, and states this explicitly.

## 9. Conclusion and Future Work

This paper contributes a reference architecture for autonomous quality engineering on enterprise banking platforms. The Context-Aware Multi-Agent AI Framework binds a context engine (regulatory ontology, system and data topology, historical defect corpus, and requirement store) to six specialised agents, coordinated over the Model Context Protocol and constrained by pre-execution and human-in-the-loop governance. A reference implementation demonstrates that the pivotal decisions of the framework can be encoded in a small, auditable code surface. Practitioner-observed directional metrics from a multi-year multi-affiliate-bank core-banking transformation, together with a synthetic fifty-change release stream produced by the reference implementation, illustrate expected behaviour and expected direction of effect across seven evaluation dimensions.

Three lines of future work follow from the framework's design bets. First, cross-industry replication in insurance claims processing and adjacent regulated verticals would test the generalisability the design targets. Second, a formal-methods overlay on the MCP bus, targeting payment-flow correctness, would strengthen the governance envelope beyond reviewer sign-off. Third, a longitudinal study of framework performance as underlying LLMs version in production would speak to the drift concerns Barr and colleagues [7] and the community more broadly are raising.

## References

[1] J. Liu, Y. Su, X. Zhang, et al., "LLM-Based Multi-Agent Systems for Software Engineering: Literature Review, Vision and the Road Ahead," ACM Transactions on Software Engineering and Methodology, 2025. DOI: 10.1145/3712003. arXiv:2404.04834.

[2] Y. Su, R. Cheng, and J. Liu, "Designing LLM-based Multi-Agent Systems for Software Engineering Tasks: Quality Attributes, Design Patterns and Rationale," arXiv:2511.08475, 2025.

[3] R. Cheng et al., "LLM Agents for Autonomous System Testing: A Semi-structured Literature Review," Springer Nature, DOI: 10.1007/978-3-032-24216-7_9, 2025.

[4] K. Zhang, W. Wang, and S. Kumar, "Methodology for Quality Assurance Testing of LLM-based Multi-Agent Systems," in Proc. 4th Int. Conf. on AI-ML Systems (AIMLSystems '24), ACM, 2024. DOI: 10.1145/3703412.3703439.

[5] R. Kumar and S. Prasad, "Practical Limits of Autonomous Test Repair: A Multi-Agent Case Study with LLM-Driven Discovery and Self-Correction," arXiv:2605.01471, 2026.

[6] T. Wang, L. Chen, and P. Rao, "The Future of Software Testing: AI-Powered Test Case Generation and Validation," arXiv:2409.05808, 2024.

[7] E. Barr, N. Alshahwan, and M. Harman, "Tracking the Moving Target: A Framework for Continuous Evaluation of LLM Test Generation in Industry," arXiv:2504.18985, 2025.

[8] K. Sridharan and V. Ramachandran, "A Review of Large Language Models for Automated Test Case Generation," Machine Learning and Knowledge Extraction, vol. 7, no. 3, art. 97, 2025.

[9] S. Iyer et al., "The Model Context Protocol and Enterprise Tool Orchestration: Architectural Patterns for

Connecting AI Agents to Production Systems at Scale," International Journal of Computational and Experimental Science and Engineering, 2025.
[10] Anthropic, Model Context Protocol Specification, first release November 2024, maintained at modelcontextprotocol.io.
[11] Board of Governors of the Federal Reserve System and Office of the Comptroller of the Currency, Supervisory Guidance on Model Risk Management, SR 11-7 / OCC 2011-12, 2011, and subsequent joint interagency Model Risk Management guidance updates.
[12] A. Nasr, "SR 11-7 in the Age of Agentic AI: Where the Framework Holds â€" and Where It Strains," Risk Intelligence â€" GARP, 2026.
[13] S. Fan et al., "LLM Hallucinations in Practical Code Generation: Phenomena, Mechanism, and Mitigation," Proc. ACM on Software Engineering, 2024. DOI: 10.1145/3728894.
[14] P. Sharma and R. Bhatia, "Mitigating Hallucination in Large Language Models: An Application-Oriented Survey on RAG, Reasoning, and Agentic Systems," arXiv:2510.24476, 2025.
[15] L. Chen, D. Park, and A. Rossi, "Trade-offs in Financial AI: Explainability in a Trilemma with Accuracy and Compliance," arXiv:2602.01368, 2026.
[16] R. Alonso, T. Iqbal, and S. Bhatt, "Bridging the Gap in XAI â€" Why Reliable Metrics Matter for Explainability and Compliance," arXiv:2502.04695, 2025.