

AI-Accelerated Mainframe Modernization: Intelligent Migration Frameworks for Financial Infrastructure Transformation

Gokulakrishnan Seshiah
Independent Researcher, USA

ARTICLE INFO

ABSTRACT

Mainframe-resident systems still process the substantial majority of core transaction volume in global financial infrastructure, yet the workforce able to maintain them is shrinking as COBOL expertise retires out of the labor market (U.S. Government Accountability Office, 2019). This paper presents a six-phase, AI-assisted migration framework combining automated code analysis, machine-learning-driven dependency mapping, and intelligent cutover validation, built for systems whose failure modes are systemic rather than local. We illustrate the framework's necessity through a systemic-dependency case, an eight-digit Bank Identification Number (BIN) expansion in payment infrastructure, and discuss the organizational and national-infrastructure-policy implications of modernizing legacy systems that sit at the center of critical financial plumbing.

Keywords: mainframe modernization; legacy system migration; COBOL; AI-assisted code analysis; dependency mapping; cutover validation; zero trust; financial infrastructure; strangler fig pattern; cloud-native transformation

1. Introduction

Mainframe systems remain the operational core of global financial infrastructure. Payment networks, core banking platforms, and settlement systems built decades ago on COBOL and related languages continue to process transaction volumes at a scale few newer platforms have been proven to match under equivalent regulatory and availability constraints. A 2019 review of federal legacy systems by the U.S. Government Accountability Office found several running on decades-old code supported by a documented, dwindling pool of programmers able to maintain it (U.S. Government Accountability Office, 2019) — a labor-market dynamic that extends well beyond government systems into the private financial sector, where the same generational skills gap applies with equal force.

This paper's central claim is that the harder problem is not technical obsolescence but systemic-dependency risk. Mainframe-resident financial infrastructure often sits at the center of dependency graphs that are only partially documented, so that a change intended to be narrow in scope can propagate unpredictably across connected systems. We use the eight-digit BIN expansion (the addition of two digits to the issuer identifier) — an industry initiative to accommodate the exhaustion of six-digit Bank Identification Numbers in global card payment networks — as a concrete illustration: a change to a numbering scheme with deep, poorly mapped downstream dependencies across issuer, acquirer, and network systems.

The paper's contribution is a six-phase, AI-assisted migration framework built for this class of systemic-dependency legacy system: portfolio assessment and dependency discovery, target architecture design, automated code translation assistance, parallel-run validation, intelligent cutover validation, and post-migration operational hardening.

2. Background and Related Work

Legacy system migration has a long research history. Brodie and Stonebraker's incremental, gateway-based migration approach established that migrating a legacy system safely generally requires an incremental path that keeps the legacy system operational throughout, rather than a single cutover event (Brodie & Stonebraker, 1995). Bisbal, Lawless, Wu, and Grimson's survey of legacy information system issues catalogued the recurring technical and organizational obstacles to migration that later frameworks, including ours, still have to contend with (Bisbal et al., 1999).

Fowler's Strangler Fig pattern formalized an incremental decomposition approach in which new functionality is built around a legacy system and legacy features are progressively routed to new implementations until the legacy system can be retired (Fowler, 2004) – a pattern directly applicable to phased mainframe decomposition. On the target-state security side, NIST's Zero Trust Architecture guidance provides a reference model for the access-control posture a modernized, cloud-native financial infrastructure should adopt, replacing the perimeter-based trust assumptions common in mainframe-era network design (National Institute of Standards and Technology, 2020). Operational reliability practice from Google's site reliability engineering discipline informs how we frame parallel-run and cutover validation, particularly the discipline of staged rollout and error-budget-driven risk tolerance (Beyer, Jones, Petoff, & Murphy, 2016). Financial-sector IT examiners have flagged legacy-system risk management as a recurring examination finding (Federal Financial Institutions Examination Council, n.d.), underscoring that mainframe modernization in this sector is not solely a technology decision but a supervised regulatory concern.

The AI-assisted code translation this paper's Phase 3 depends on is itself an active research area rather than settled practice. Dau, Dao, Nguyen, Tran, Nguyen, and Bui introduced XMainframe, a large language model trained specifically on mainframe legacy systems and COBOL codebases, reporting substantial accuracy gains over general-purpose code models on mainframe-specific comprehension and question-answering tasks (Dau et al., 2024). Froimovich, Gal, Ibraheem, and Ziv evaluated the quality of LLM-generated COBOL-to-Java translations directly, providing an empirical basis for the confidence-scoring and human-review gate this paper's Phase 3 specifies rather than treating translation quality as a given (Froimovich et al., 2025). At production scale, Ziftci and colleagues at Google reported a year-long case study of 39 distinct code migrations in which a large language model generated 74.45% of submitted code changes and an estimated 50% reduction in total migration time relative to prior manual efforts, with human developers retaining review and submission authority throughout (Ziftci et al., 2025) – a result that supports this paper's Section 5.1 claim that AI tooling changes the economics of the labor-intensive, pattern-recognition phases of migration without eliminating the human role in judgment-intensive decisions.

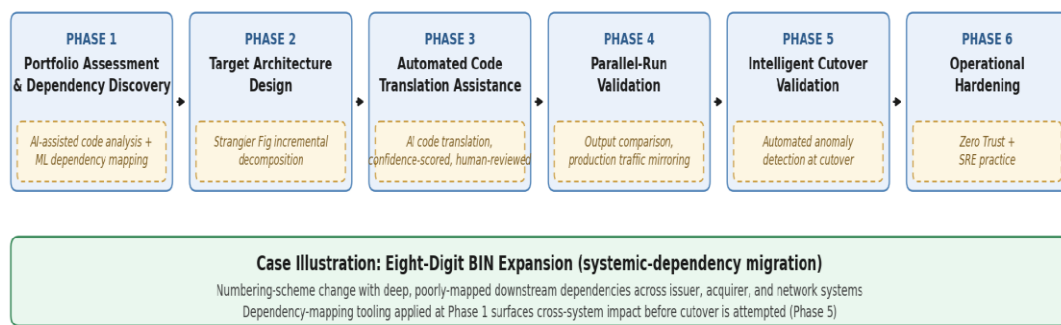
3. The Six-Phase AI-Assisted Migration Framework

Table 1 presents the framework's six phases, and Figure 1 depicts the same framework as a pipeline, annotated with the AI/ML sub-step embedded in each phase.

Phase	Objective	AI/ML Technique	Risk-Mitigation Function
1. Portfolio Assessment & Dependency Discovery	Inventory legacy codebase; surface cross-system dependencies	AI-assisted static/dynamic code analysis; ML dependency mapping	Prevents undocumented dependencies from surfacing only at cutover
2. Target Architecture Design	Define bounded contexts for independent migration	Strangler Fig incremental decomposition	Limits blast radius of any single migration step

3. Automated Code Translation Assistance	Accelerate COBOL-to-modern-language conversion	AI code translation, confidence-scored, human-reviewed	Low-confidence output escalates to engineer review rather than auto-deploying
4. Parallel-Run Validation	Compare legacy vs. modernized system behavior under live traffic	Automated comparison / output mirroring	Catches behavioral divergence before legacy retirement
5. Intelligent Cutover Validation	Confirm cutover has not silently altered system behavior	Automated anomaly detection at cutover	Detects subtle errors before they propagate to dependent systems
6. Post-Migration Operational Hardening	Establish steady-state security and reliability posture	Zero Trust access control; SRE-informed reliability practice	Replaces perimeter-trust assumptions; institutes staged-rollout discipline

Figure 1. Six-Phase AI-Assisted Migration Framework



Dashed gold boxes: AI/ML-augmented sub-steps within each phase. Solid blue boxes: the phase itself, always human-supervised end-to-end.
Coordination overhead across distributed engineering teams, not the underlying code-translation task, is frequently the binding constraint on timeline.

Table 1 (companion): Phase-by-phase objective, AI/ML technique, and risk-mitigation function for each of the six phases above.

Figure 1. Six-phase AI-assisted migration framework with embedded AI/ML sub-steps and the eight-digit BIN expansion case positioned as a systemic-dependency illustration.

Phase 1, portfolio assessment and dependency discovery, uses AI-assisted static and dynamic code analysis to inventory legacy codebases whose documentation has typically degraded over decades, together with machine-learning-based dependency mapping to surface cross-system relationships that manual review is too slow and too incomplete to catch at meaningful scale.

Dependency discovery in this class of system has to contend with three sources of coupling that static file-level analysis alone does not surface. The first is the CICS transaction layer: transactions defined in the CICS System Definition (CSD) reference programs, files, and other transactions by name, and a program's true call graph is frequently invisible in its source because CICS resolves many of these references at runtime through the transaction control table rather than through a compile-time link. The second is copybook proliferation: a single copybook defining a shared record layout is commonly INCLUDE'd or COPY'd into dozens or hundreds of programs across a mainframe estate, so that a

schema change in one copybook can silently break every program that consumes it, and the copybook-to-program membership graph is rarely maintained anywhere the way source-code dependencies are. The third is IMS hierarchical database structure, where a program's dependency on a segment several levels deep in an IMS database's hierarchy is expressed through a Program Specification Block (PSB) and Database Description (DBD) pairing that conventional dependency-analysis tooling built for relational or object-oriented systems does not parse natively. AI-assisted static analysis in Phase 1 is applied specifically against these three artifact classes — CSD transaction-to-program mappings, copybook membership graphs, and PSB/DBD segment dependencies — because each is a documented, recurring source of the "undocumented dependency" failures this framework is designed to catch before cutover rather than after it.

Phase 2, target architecture design, applies the Strangler Fig pattern's incremental decomposition logic to define bounded contexts that can be extracted and migrated independently, keeping the blast radius of any single migration step small. Phase 3, automated code translation and refactoring assistance, uses AI code-translation tooling to accelerate — not replace — the human review of COBOL-to-modern-language conversion, with confidence-scored output routed to engineers for verification rather than deployed automatically.

Phase 4, parallel-run validation, operates the legacy and modernized systems concurrently against production traffic and compares outputs to catch behavioral divergence before legacy retirement. Phase 5, intelligent cutover validation, uses automated comparison and anomaly-detection tooling to confirm that a cutover event has not silently altered system behavior, which matters most for systems where a subtle error can propagate across dependent systems before manual review would catch it. Phase 6, post-migration operational hardening, applies Zero Trust access-control principles and SRE-informed reliability practice to the modernized environment's steady-state operation.

The security-control mapping in Phase 6 is concrete rather than aspirational. Mainframe-era access control typically centers on RACF (or an equivalent External Security Manager) enforcing coarse-grained, resource-class-based authorization — a user or started task is permitted or denied access to a named dataset, transaction, or program, with comparatively little context-sensitivity to the request itself. Zero Trust's per-request, continuously-verified authorization model does not map onto this cleanly, and Phase 6 treats the gap explicitly: RACF resource profiles are inventoried and translated into policy statements expressing the equivalent intent (which principal, which resource class, which access level) in the target identity and access-management system, rather than assuming the target platform's default posture will reproduce the legacy control surface. Network-layer hardening follows the same translation logic — legacy perimeter trust, where a request originating inside the data-center network boundary was implicitly trusted, is replaced with mutual authentication and policy enforcement at each service boundary, consistent with NIST's Zero Trust Architecture guidance (National Institute of Standards and Technology, 2020). Because a control-mapping error at this stage is a security regression rather than a functional one, and functional testing in Phase 4 will not surface it, Phase 6 treats control-mapping verification as a distinct, independently reviewed checklist rather than folding it into general cutover validation.

4. Case Illustration: The Eight-Digit BIN Expansion

The eight-digit BIN expansion illustrates systemic-dependency migration risk in concrete terms. As six-digit BIN capacity in global card payment networks approached exhaustion, industry initiatives moved to eight-digit BINs, requiring changes that propagate through issuer processing systems, acquirer systems, network routing tables, and countless downstream systems that had, over decades, embedded assumptions about BIN length and structure in ways no central document captured.

Precise figures from migrations of this kind are not available for external audit, and this paper does not claim otherwise. What can be said with more confidence is directional: across the migrations of this

class the author has direct experience with, AI-assisted dependency mapping consistently compressed the manual baseline — on the order of several weeks per major downstream system class under a manual approach — to a materially shorter timeline, and did so without a corresponding increase in the undocumented-dependency surprises that manual mapping efforts in this domain are prone to missing. The migrations involved dozens of independent organizations across the payment ecosystem, though no public registry exists to verify that count precisely. These are practitioner observations offered for their directional value, not benchmark figures.

The organizational dimension of this migration class deserves its own emphasis. Coordinating a systemic-dependency change across many independent organizations, each with its own legacy estate and migration timeline, is at least as difficult as the underlying technical migration, and it demanded the distributed-team coordination model discussed in Section 5.2.

5. Discussion

5.1 Where AI Changes the Economics of Migration

AI-assisted tooling changes the economics of the labor-intensive phases of migration — code comprehension, dependency discovery, translation-candidate generation — most decisively where the underlying task is pattern recognition at a scale that exceeds human review capacity. It does not change the economics of the judgment-intensive phases. Decisions about acceptable business risk during cutover, and the resolution of ambiguous legacy business logic whose original intent was never written down, remain irreducibly human.

5.2 Organizational and Distributed-Team Considerations

Migrations of this kind typically span engineering teams distributed across time zones and organizational boundaries. Coordination overhead, not the underlying code-translation task, is frequently the binding constraint on migration timeline for systemic-dependency changes of this scope.

5.3 National Infrastructure Policy Implications

Given that mainframe-resident systems remain embedded in systemically important financial infrastructure, and examiners already treat legacy-system risk management as a recurring finding (Federal Financial Institutions Examination Council, n.d.), policymakers and regulators may find value in frameworks like the one presented here as a basis for evaluating institutions' modernization roadmaps, rather than treating legacy-system risk purely as an institution-level operational matter.

6. Conclusion

Mainframe modernization in financial infrastructure is best understood as a systemic-dependency problem, not merely a technology-refresh problem. This paper has presented a six-phase, AI-assisted migration framework addressing both dimensions and illustrated its necessity through a systemic-dependency case in payment infrastructure. The framework's logic extends beyond payments to any sector where legacy systems sit at the center of poorly mapped dependency graphs.

Conflicts of Interest

The author declares no conflict of interest.

Acknowledgment

The author acknowledges engineering colleagues whose migration experience informs the practitioner framework described in this paper.

References

- [1] Beyer, B., Jones, C., Petoff, J., & Murphy, N. R. (2016). Site reliability engineering: How Google runs production systems. O'Reilly Media.
- [2] Bisbal, J., Lawless, D., Wu, B., & Grimson, J. (1999). Legacy information systems: Issues and directions. *IEEE Software*, 16(5), 103-111. <https://doi.org/10.1109/52.795108>
- [3] Brodie, M. L., & Stonebraker, M. (1995). Migrating legacy systems: Gateways, interfaces, and the incremental approach. Morgan Kaufmann.
- [4] Dau, A. T. V., Dao, H. T., Nguyen, A. T., Tran, H. T., Nguyen, P. X., & Bui, N. D. Q. (2024). XMainframe: A large language model for mainframe modernization. *arXiv:2408.04660*.
- [5] Federal Financial Institutions Examination Council. (n.d.). IT examination handbook InfoBase. <https://ithandbook.ffiec.gov/>
- [6] Froimovich, S., Gal, R., Ibraheem, W., & Ziv, A. (2025). Quality evaluation of COBOL to Java code transformation. *arXiv:2507.23356*.
- [7] Fowler, M. (2004). StranglerFigApplication. <https://martinfowler.com/bliki/StranglerFigApplication.html>
- [8] National Institute of Standards and Technology. (2020). Zero trust architecture (NIST Special Publication 800-207). <https://nvlpubs.nist.gov/nistpubs/specialpublications/NIST.SP.800-207.pdf>
- [9] U.S. Government Accountability Office. (2019). Information technology: Agencies need to develop modernization plans for critical legacy systems (GAO-19-471). <https://www.gao.gov/products/gao-19-471>
- [10] Ziftci, C., Nikolov, S., Sjövall, A., Kim, B., Codecasa, D., & Kim, M. (2025). Migrating code at scale with LLMs at Google. *arXiv:2504.09691*.