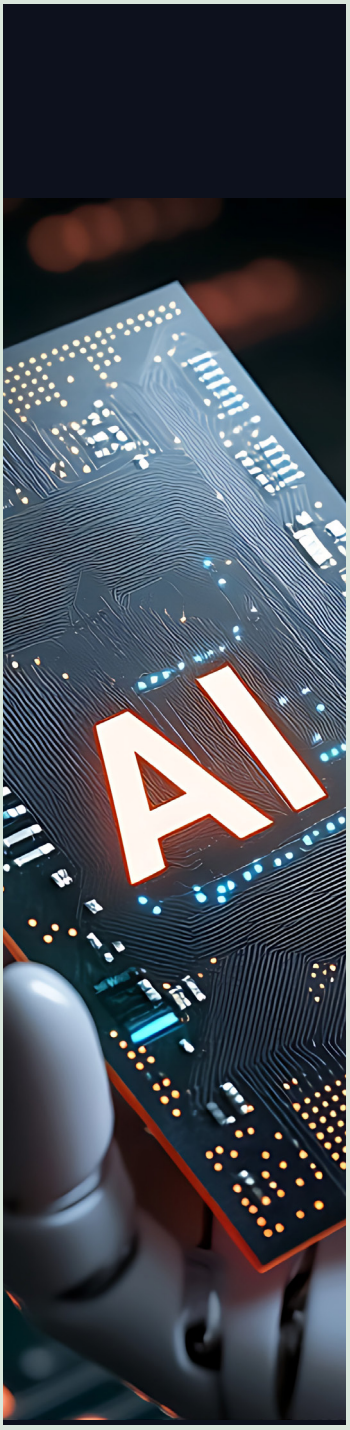
Infosys topaz

WHITE PAPER



A COMPARATIVE ANALYSIS OF LEGACY MODERNIZATION APPROACHES: AUTO-CONVERSION TOOLS VERSUS AI-AGENT-ASSISTED REWRITE

Abstract

Mainframe applications built on technologies such as Common Business-Oriented Language (COBOL), Customer Information Control System (CICS), and IBM Database (DB2) continue to support mission-critical operations across industries. Although these systems remain reliable and performant, they increasingly constrain agility and limit integration with modern digital platforms. They rely on skills that are becoming harder to sustain. Rising operational costs and accumulated technical debt have made modernization a strategic imperative.

This white paper examines two primary modernization approaches. The first, tool-based automated code conversion, prioritizes speed and cost efficiency by translating legacy applications into modern runtimes with minimal change to existing business logic. The second, AI-assisted business rule extraction followed by manual rewrite, enables deeper architectural transformation and stronger long-term maintainability. However, it requires greater upfront investment and domain involvement.

The paper compares these approaches across productivity, code quality, maintainability, modernization depth, risk, and required skill sets. It also provides guidance on aligning modernization strategy with business timelines and architectural objectives. Additionally, it outlines a hybrid model that combines accelerated migration for stable workloads with targeted transformation of strategically critical systems.

Infosys
Navigate your next

Introduction

Common Business-Oriented Language (COBOL) continues to underpin mission-critical systems across banking, insurance, government, and large enterprises. Systems in these sectors process large volumes of transactions with high reliability even decades after COBOL were introduced. The programming language remains deeply embedded in core financial infrastructure and legacy applications because of its scalability and strong data-processing capabilities. As a result, organizations find it difficult to retire these systems outright, despite the emergence of newer technologies.

However, these legacy systems often operate on aging mainframe infrastructure that is costly and complex to maintain. As experienced COBOL developers retire with few replacements entering the field, the talent pool for maintaining and evolving this codebase is shrinking. With rising costs, operational complexity, and workforce challenges, modernization has shifted from a technical choice to a strategic imperative for organizations seeking to reduce operational risk, improve agility, and integrate core systems with cloud-native architectures.

Legacy Modernization Landscape: The Two Approaches

Organizations modernizing legacy mainframe systems typically evaluate two primary approaches. Each reflects a different balance between speed, risk, and long-term architectural goals.

1. Automated code conversion

Automated code conversion focuses on rapid migration with minimal functional disruption. The primary objective is to move applications off the mainframe quickly while preserving existing business logic and operational behavior.

This approach relies on automated modernization tools such as Amazon Web Services (AWS) Blu Age and IBM Watson Code Assistant, Micro Focus AMT, and Heirloom. These tools convert legacy COBOL applications into Java-based batch processes and transform Customer Information Control System (CICS) screens into web-based interfaces. Job Control Language (JCL) and associated procedures (PROC) are translated into Groovy-based scripts for job orchestration. IBM Database (Db2) structures are mapped to Db2 Linux, UNIX, and Windows (LUW), resulting in like-for-like modernized code that can run on modern infrastructure. Figure 1 illustrates a representative end-to-end automated code conversion flow using the AWS Blu Age tool.

Tool-converted End-to-end Flow

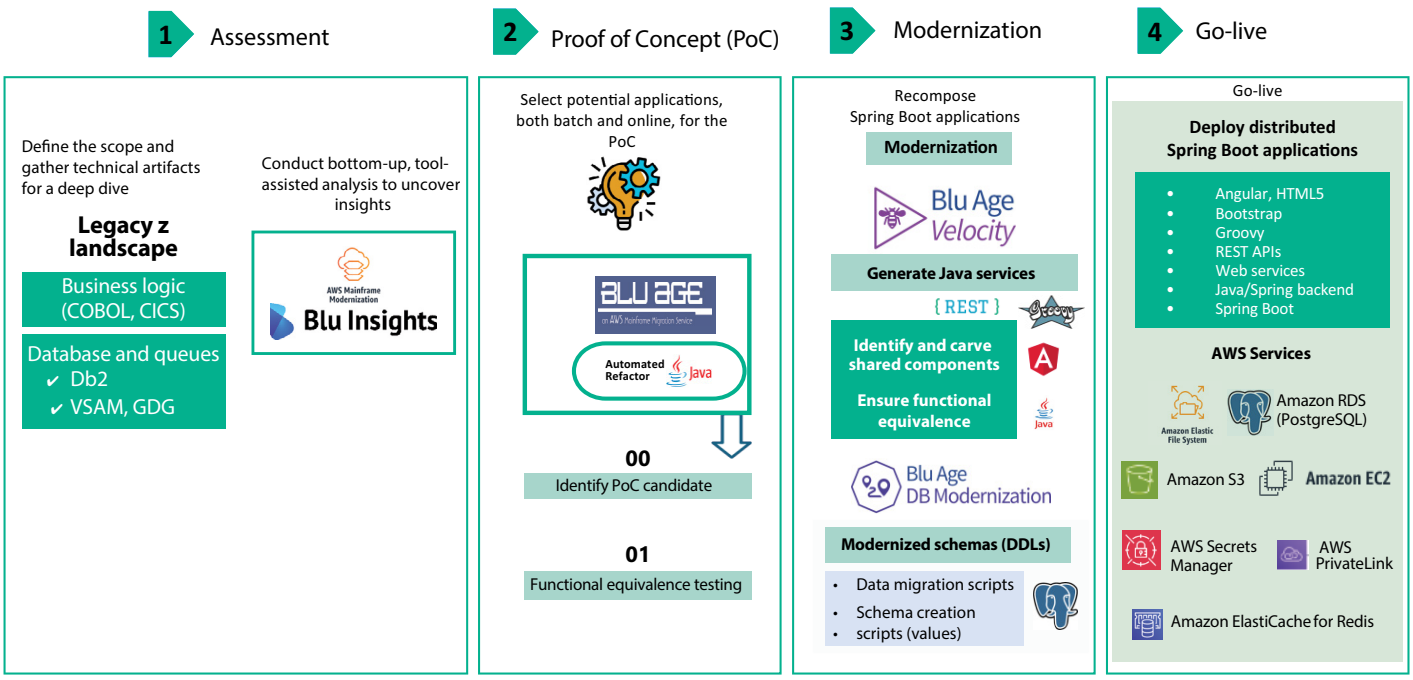


Fig 1: Automated tool-based legacy modernization workflow

Key characteristics

- **Refactoring and code correction**: Generic exception handling must be replaced with standard Java error management. Database calls and batch processing logic typically require post-conversion optimization.
- **Runtime behavior**: COBOL loops are converted literally, resulting in inefficient iterations. Caching and concurrency handling are not introduced by default, while batch jobs may require redesign using Spring Batch.
- **Maintainability impact:** Multiple Java classes may be generated to implement equivalent functionality. Cyclomatic complexity can increase due to entity and service layering. Code changes often require tool knowledge, and auto-generated code may be less intuitive
- **Skill set requirements**: COBOL subject matter experts (SMEs) are required for validation along with Java developers with intermediate proficiency and mainframe familiarity; tool specialists for installation and configuration; Db2 experts for migration; and DevOps engineers for deployment.

Because architectural patterns are largely preserved, modernization depth is limited. While this approach accelerates mainframe exit and reduces upfront cost, it often defers deeper modernization and maintainability improvements to later phases.

2. AI-assisted business rules extraction and manual rewrite

AI-assisted modernization prioritizes architectural transformation over direct code preservation. Its objective is to extract business intent from legacy systems and reimplement it using modern, modular, and service-oriented architecture.

In this approach, artificial intelligence (AI) agents powered by large language models (LLMs), analyze COBOL batch programs and CICS modules to identify and extract business rules. SMEs validate these rules, which can then be reimplemented using modern technology stacks. Java Spring Boot is typically used for services, Java Spring Batch for batch processing, and Angular or NodeJS for user interfaces (UIs). Data layers are migrated from Db2 to DB2 LUW or Oracle, enabling adoption of modern patterns such as representational state transfer application programming interfaces (REST APIs) and microservices.

Key characteristics

- **Refactoring and design optimization**: UI refactoring is incorporated during redevelopment. JVM tuning and caching mechanisms are configured for high-volume transactional workloads.
- **Performance engineering**: It includes UI performance tuning for Angular or NodeJS database indexing and query optimization. It is also characterized by garbage collection tuning for batch-intensive and real-time workloads.
- **Maintainability impact**: Modular, well-structured code simplifies issue isolation and feature extension. Reduced structural complexity limits future rework. Clear organization also accelerates developer onboarding.
- **Skill set requirements**: Java architects are a key asset. So are backend developers skilled in microservices, object-oriented design, and Spring Boot. Also required are Angular or NodeJS UI engineers; cloud engineers across AWS, Microsoft Azure, or Google Cloud Platform; database architects; test automation engineers; and business analysts for validation of extracted rules.

Figure 2 below shows an AI-assisted pipeline for business rule extraction, validation, and refinement. Next, Figure 3 illustrates the process from rule extraction to modernization and go-live. Finally, Figure 4 presents structural code conversion from z/OS batch systems to a Java-based architecture.

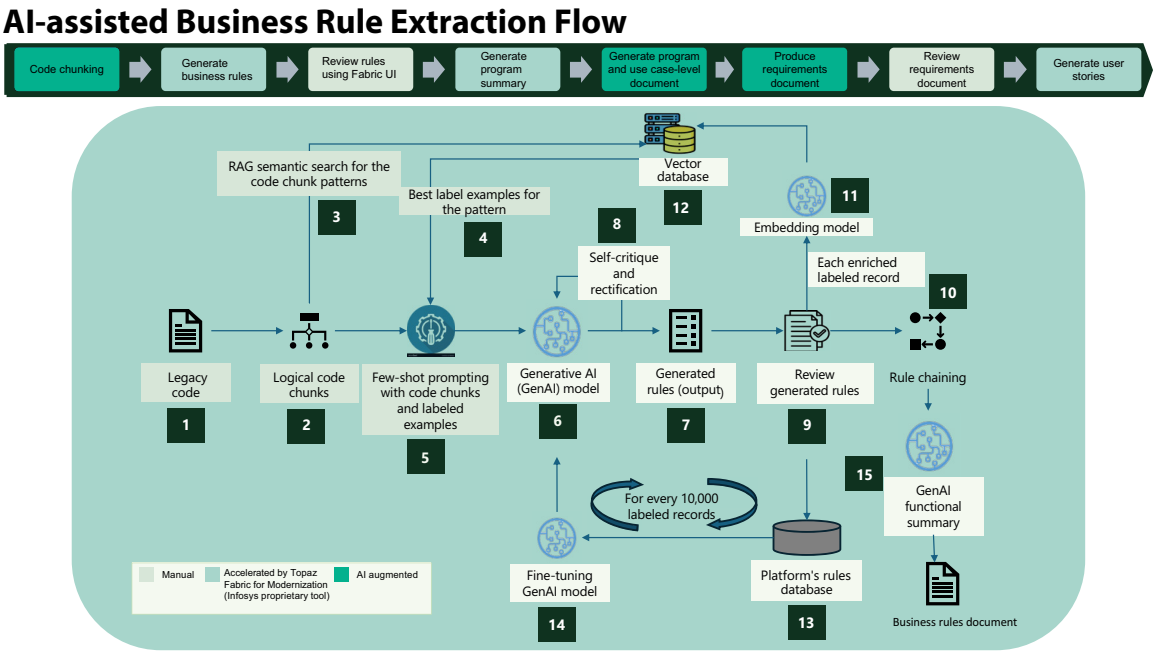


Fig 2: AI-assisted business rule extraction using Topaz Fabric for Modernization

AI-assisted End-to-end Flow

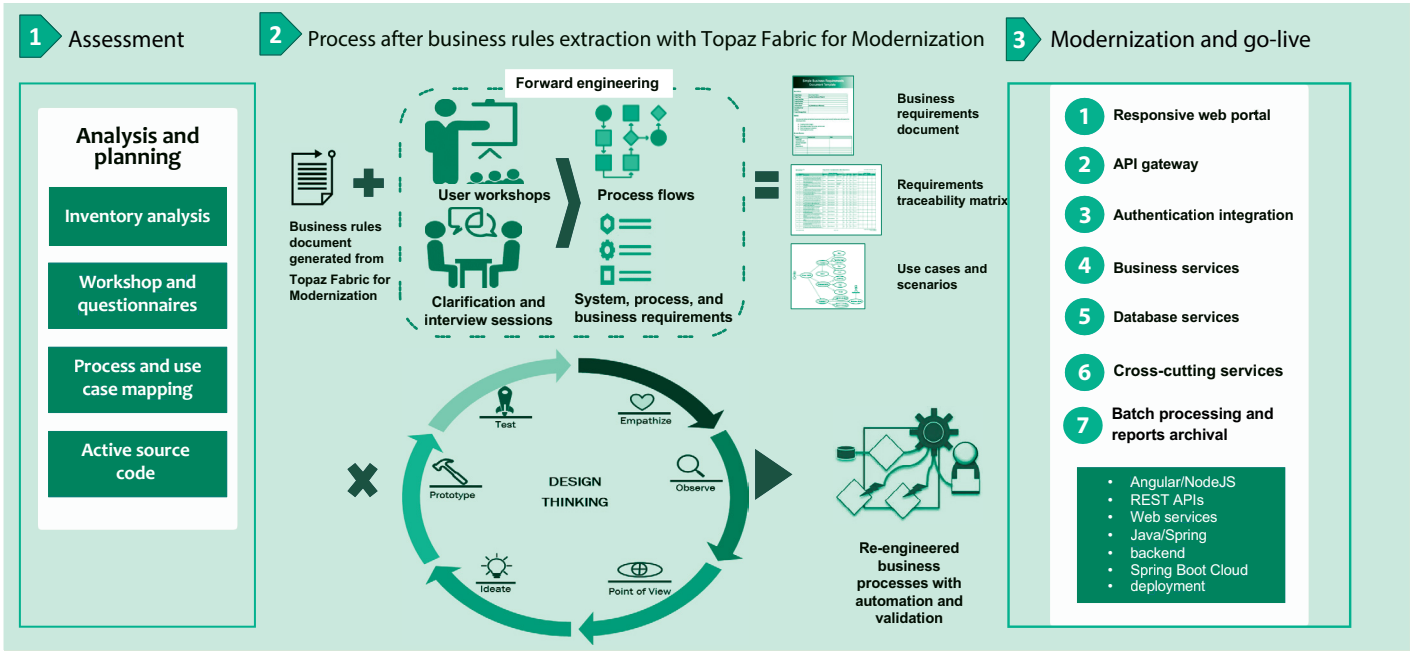


Fig 3: AI-assisted post–business rule extraction modernization flow

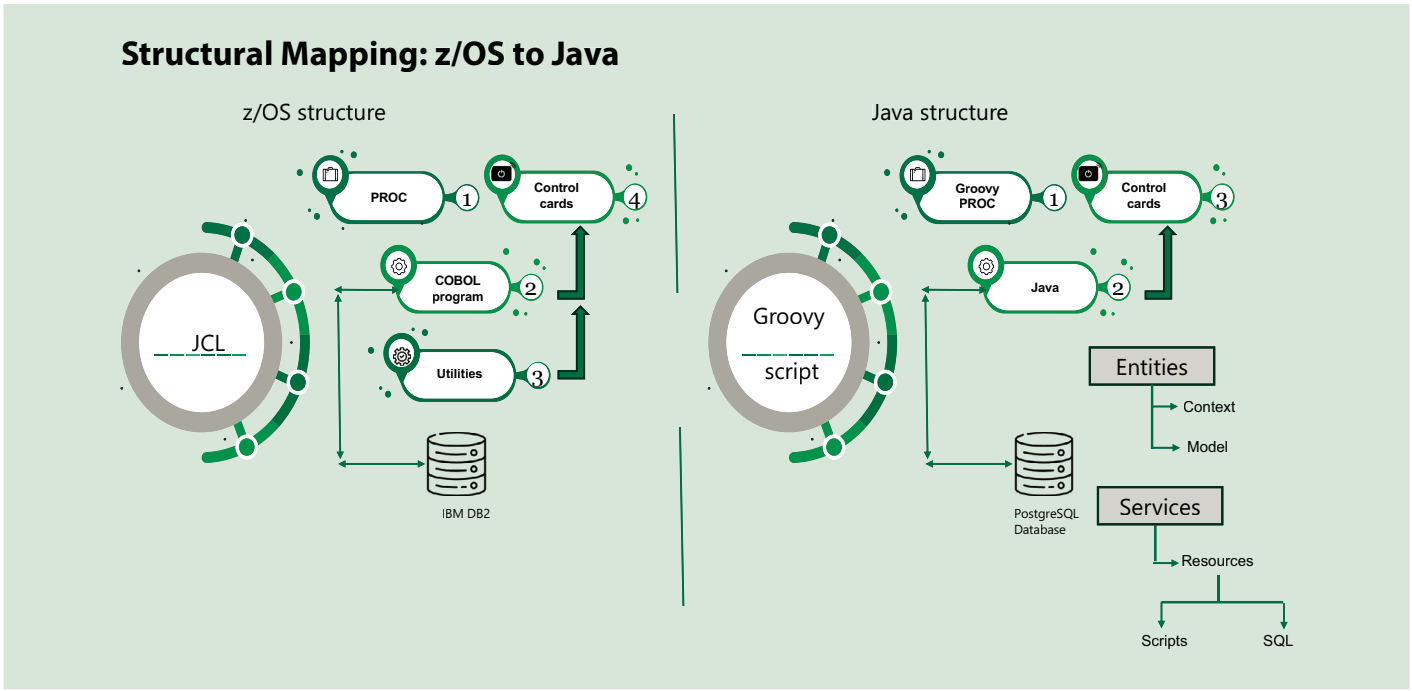


Fig 4: Automated architectural transformation from z/OS to Java platform

This approach requires higher upfront investment and a longer initial development timeline. However, it delivers long-term agility, deeper modernization, and lower operational complexity over time.

Table 1 offers a comparative perspective between automation code conversion and AI-assisted extraction and manual rewrite.

Table 1: Automation versus AI assistance

Aspect	Automated code conversion	AI-assisted extraction and manual rewrite
Code quality	Code closely mirrors COBOL logic; often verbose and procedural; difficult for new developers to maintain.	Clean, modular, object-oriented Java code; REST APIs and microservices ready
Modernization depth	Syntax-only modernization; logic and design largely unchanged	Full architectural modernization; redesigns workflows, reduces technical debt, and improves performance
Speed and cost	Fast with low upfront cost; high long-term maintenance cost	Slower and higher upfront cost; lower long-term operational cost
Risk	Low risk as logic remains unchanged; however, risk of low-quality code remains	Higher build-phase risk, but long-term functional and technical stability
Dependency on SME	Minimal SME involvement; logic preserved by tools	SME validation required for AI-extracted rules.

Choosing the Right Modernization Approach

The choice between automated code conversion and AI-assisted modernization is driven by migration urgency, budget availability, system complexity, and long-term architectural intent.

Automated code conversion is appropriate when a quick exit from the mainframe is the primary objective. It aligns well with constrained budgets, short modernization timelines, limited SME availability, and scenarios where functional equivalence is sufficient. In this model, tools are used to convert code, migrate data, and move applications to the cloud quickly. Once workloads stabilize, modernization can proceed incrementally through phased enhancements.

AI-assisted business rule extraction combined with manual rewrite is better suited to systems with high technical debt and complex COBOL logic. In such scenarios, long-term maintainability and architectural evolution are clear goals. This approach supports restructuring into service-based or microservices architectures before or during migration, including adoption of REST APIs and cloud-native design patterns. While it requires a longer upfront timeline, it enables a more comprehensive and future-ready transformation into a single modernization effort.

Recommended Approach: Hybrid Modernization Strategy

In most enterprise environments, a hybrid modernization strategy delivers the most balanced outcome. This approach combines the speed and cost advantages of automation with the architectural benefits of selective transformation.

Automated code conversion should be applied to stable, low-change, high-volume batch programs and non-differentiating business functions to accelerate migration. AI-assisted extraction followed by manual rewrite should be reserved for core business logic, high-change and high-value domains, and customer-facing CICS online transactions. These components are best rewritten using modern stacks such as Java-based services or Angular with NodeJS and REST APIs to support scalability and long-term maintainability.

Overall, the hybrid approach strikes a pragmatic balance across speed, cost efficiency, risk, and sustainable architectural value. Tactical use of automation accelerates migration for stable workloads, while strategic AI-assisted transformation ensures that critical systems align with future business and technology objectives.

Infosys Best Practice for Migration and Optimization

Legacy modernization at scale introduces challenges across refactoring, security, performance, and operational continuity. Infosys addresses these through an integrated, partner-driven and proprietary, tool-based approach that supports migration, optimization, and continuous improvement. This enables scale, consistency, and risk-managed transformation.

Refactoring code quality, and vulnerability identification

Infosys uses tools designed to analyze large codebases, detect issues, and guide post-conversion refactoring. These tools identify code quality concerns, uncover refactoring opportunities, and systematically detect code vulnerabilities. Architectural visualization helps teams pinpoint problem areas and plan safe, efficient refactoring of modernized applications.

Optimization and performance improvement

Optimization and performance improvement are supported through tools that enable proactive cost management and workload optimization without manual intervention. This ensures efficient resource utilization and reduces idle capacity

Documentation, mapping, testing, and continuous improvement

These capabilities are delivered through the Infosys Topaz Fabric for Modernization, part of Infosys Topaz Fabric. It enables AI-based extraction of business rules from legacy code and helps modernization teams self-assess data to identify gaps. It also enhances collaboration, improves debugging efficiency, reduces time spent on issue resolution, and supports both manual and automated testing for data validation. The platform is designed to support all key personas involved in modernization initiatives, including architects, developers, testers, and release teams.

Conclusion: Rethinking Legacy Modernization Strategies

Mainframe modernization is a strategic choice shaped by urgency, architectural ambition, and organizational capability. Automated code conversion enables rapid re-platforming with low functional risk. But it often defers deeper maintainability and architectural improvements. In contrast, AI-assisted business rule extraction with manual rewrite supports more fundamental transformation. It improves long-term maintainability and scalability at the cost of higher upfront effort.

The appropriate approach depends on system complexity, skill availability, timelines, and long-term objectives. In many cases, a hybrid strategy offers the most effective balance, combining accelerated migration for stable workloads with targeted architectural redesign for critical systems. Ultimately, successful modernization is defined not by mainframe exit alone, but by the sustainability and adaptability of the modernized platform.

About the Author

Abhishek Nigam is a Technology Architect with the Legacy Modernization Practice at Infosys. With over 20 years of experience in mainframe and related technologies, he has worked on numerous application migration initiatives. His key area of focus is helping organizations modernize their legacy application portfolios.

Infosys Topaz is an AI-first set of services, solutions and platforms using generative AI technologies. It amplifies the potential of humans, enterprises, and communities to create value. With 12,000+ AI assets, 150+ pre-trained AI models, 10+ AI platforms steered by AI-first specialists and data strategists, and a 'responsible by design' approach, Infosys Topaz helps enterprises accelerate growth, unlock efficiencies at scale and build connected ecosystems.

Connect with us at modernization@infosys.com

For more information, contact askus@infosys.com

Infosys®
Navigate your next