AI-Assisted Legacy Modernization in a Brazilian Bank

Bruno Pereira Soares
Banco Bradesco S.A.
Osasco, Brasil
bruno.p.soares@bradesco.com.br

Manoel Valerio da Silveira Neto
Banco Bradesco S.A.
Osasco, Brasil
Pontifícia Universidade Católica do Paraná (PUCPR)
Curitiba, Brasil
manoel.silveira@pucpr.edu.br

Abstract

Legacy software modernization in financial institutions involves critical systems, complex dependencies, distributed tacit knowledge, heterogeneous technologies, and strong governance requirements. This paper presents an industry experience report on the governed use of Artificial Intelligence (AI) to support a large-scale modernization program in a Brazilian bank. The initiative focuses on business services derived from an internal BIAN-based capability structure and involves multiple business units, internal and external teams, and modernization waves. The AI strategy prioritized three fronts: reverse engineering, dependency mapping, and modernization of legacy technologies. Instead of treating AI as an unrestricted automation mechanism, the initiative combined AI agents, human review, technical validation, guardrails, and organizational governance. The report also describes the role of an AI-Driven Coding Engineering squad, a corporate catalog of AI agents, and an internal platform to manage, evaluate, approve, and communicate AI-supported solutions. The experience indicates that AI can support knowledge-intensive modernization activities, especially when combined with traceability, human oversight, and governance mechanisms.

Keywords

Artificial Intelligence, Software Engineering, Legacy Modernization, Reverse Engineering, AI Governance, AI Agents

1 Introduction

Legacy software modernization remains a central challenge in large organizations. In financial institutions, this challenge is amplified by the operational criticality of systems, coexistence between old and modern platforms, the high volume of dependencies, and the need to preserve business rules frequently embedded in legacy code, incomplete documentation, and tacit knowledge.

The literature indicates that legacy systems are perceived as critical, reliable, and strongly connected to the business, but also associated with high maintenance costs, difficulty of evolution, and scarcity of technical knowledge [10]. At the same time, language models and machine learning techniques applied to code have created opportunities to support summarization, documentation, dependency analysis, translation between languages, refactoring, test generation, and technology upgrading [2, 8, 12]. However, in critical contexts, AI cannot be treated as unrestricted automation: semantic preservation, security, traceability, human validation, and artifact quality remain determining factors [13, 15].

This paper presents an industry experience report on the governed application of AI in a large-scale legacy modernization program in a Brazilian bank. The program involves 252 Business Services organized from an internal BIAN-derived layer [3]. Initial evidence includes 21,780 hours saved in reverse engineering and 11,646 hours in Angular-related technology upgrading, totaling 33,426 hours.

The report also describes the governance structure based on an AI-Driven Coding Engineering squad, a corporate agent catalog, and internal governance software. It extends a previous experience in the same banking context by examining AI support for reverse engineering, dependency mapping, and technology upgrading [16].

2 Industrial Problem and Context

The modernization program was structured around 252 Business Services defined from an internal BIAN-derived layer. This organization treats modernization as a business capability-oriented transformation rather than only as the replacement of applications or technical components.

The scope comprises approximately 1.4 million development hours, about 1,000 internal and external collaborators, and seven business units: Wealth & Personal Banking, Cards & International Banking, Digital Business, Credit, Retail & SME, Corporate Functions, and Corporate Daily Banking.

The scale of the program creates technical and organizational challenges: understanding legacy systems, identifying dependencies, impact analysis, extracting business rules, architecture documentation, technology upgrading, and functional equivalence validation. From an organizational perspective, it requires alignment among business, architecture, engineering, security, operations, vendors, and platform teams.

In this context, AI was prioritized in three fronts:

Reverse engineering: understanding legacy systems, extracting business rules, and generating technical documentation.

Dependency mapping: identifying relationships among applications, components, services, databases, integrations, and business flows.

Technology modernization: supporting upgrades, library migration, assisted refactoring, and repetitive code transformations.

The program builds on a previous pilot involving mainframe and cloud-native squads, which revealed decision inconsistencies, high reverse-engineering effort, and the need for standardized architectural evidence [16]. These lessons motivated the adoption of phases, templates, transition architectures, and systematic AI support.

3 Background

Legacy software modernization involves encapsulation, rewriting, automated migration, reengineering, refactoring, and architectural transformation [4, 9, 17]. In COBOL environments, rule-based approaches have historically supported code restructuring and migration [7, 20]. However, complete rewrites may introduce high costs and functional-loss risks, while automated migrations may preserve behavior without improving architectural quality, making functional-equivalence validation essential [17, 18]. Program slicing and model-based approaches can also extract business rules embedded in COBOL code [5].

Machine learning techniques support source-code modeling, generation, translation, and analysis [2, 12]. Applications include code transformation [19], pseudocode generation [1], library upgrading [14], automated refactoring [13], and monolith decomposition [11]. Nevertheless, data quality, generalization, semantic preservation, output evaluation, and human validation remain challenges [12, 13].

4 Governance of AI Agents in Modernization

In this report, an AI agent is a tool, workflow, or automation that supports a Software Engineering or modernization activity through LLMs, RAG, static analysis, pipeline integration, IDE assistance, documentation tools, or combinations of these approaches. Its outputs are candidate artifacts subject to human review, technical validation, security criteria, traceability, and approval by technical owners.

Governance is coordinated by the ACE squad (AI-Driven Coding Engineering), which defines standards, guardrails, acceptance criteria, and communication practices. Other teams may propose agents, but catalog admission requires a defined purpose, usage context, owner, expected artifacts, risks, limitations, and supervision requirements.

Each agent follows a lifecycle recorded in the corporate catalog. Proposals remain *In Development* during validation and become *Active* after technical approval and demonstrated use. Deprioritized or unused agents move to *Backlog* or *Inactive*, preserving traceability and reducing fragmented automation. These mechanisms complement the AS-IS, Transition, and TO-BE views, C4 models, architectural evidence, and enabling-team practices established in the previous modernization experience [6, 16].

5 Applied Solution

The Card Payment Business Service illustrates the end-to-end workflow. First, reverse-engineering tools generated technical documentation, including C4, UML, and entity-relationship diagrams, from legacy COBOL code and identified critical business rules. Second, dependency-mapping agents identified integrations and coupling risks, supporting a C4-based transition architecture. Third, modernization agents assisted the transformation to Java Spring Boot. Human approval and functional-equivalence validation were required before accepting the generated changes.

The ACE squad and corporate catalog described in the previous section govern this workflow. Table 1 summarizes the prioritized fronts, their objectives, and examples of expected artifacts.

6 Initial Results

The results correspond to gains already measured in specific activities of the program. They do not yet represent the total schedule gain of modernization, since the final benefits will be assessed after the modernization of the business services is completed. Table 2 presents the main initial indicators of scope, planned effort, and hours saved.

To quantify the impact of AI, we established a baseline by comparing the estimated historical effort for reverse engineering and technology modernization activities against the actual time spent after the agents were implemented. The measurement was performed by tracking hours logged in project management systems (Jira/Confluence) by members of the involved squads, focusing specifically on tasks where the agents were invoked (e.g., technical documentation generation, assisted refactoring). The "saved" value represents the difference between the effort originally planned for these activities and the effort actually expended, isolating the contribution of automation. It is important to note that this metric is conservative: it does not include indirect gains such as reduced rework or improved code quality, as these factors are difficult to attribute exclusively to AI without introducing bias. The reported hours saved (21,780 for reverse engineering and 11,646 for Angular modernization) reflect only the direct, measurable impact on the schedule of the modernization waves as of the date of this report.

In reverse engineering, the hours saved are associated with effort reduction in understanding, summarization, technical documentation, knowledge reconstruction, and preparation of artifacts for modernization decisions. In Angular-related technology upgrading, the gains indicate the potential of AI to accelerate technical transformation tasks, identification of obsolete patterns, support for refactoring, and reduction of repetitive effort.

Together, the measured results total 33,426 hours saved. Considering 252 Business Services, this represents an approximate average of 132.6 hours per service. In a managerial conversion of 160 hours per person-month, the gain is equivalent to about 209 person-months. This conversion is approximate, since gains vary according to service complexity, technologies involved, dependencies, code volume, existing documentation, and availability of specialists.

Dependency mapping and the measurement of schedule gains in architecture activities are still under analysis. Total schedule gains will be assessed at the end of the modernizations, when it will be possible to compare planned effort, actual effort, delivery-cycle impacts, rework, and artifact quality.

The distribution of gains is not uniform across legacy technologies; it reflects both the complexity of the code and the level of maturity of the AI tools available for each domain. Most of the hours saved in reverse engineering (21,780) were concentrated in Mainframe/COBOL systems, where technical documentation was virtually nonexistent and agents focused on rule extraction and diagram generation had a transformative impact. In terms of technological modernization, the significant gain (11,646 hours) occurred predominantly in the Angular frontend, due to the massive need for version upgrades and migration from obsolete standards, which the agents successfully automated. In contrast, the dependency mapping effort is still under detailed analysis; although it shows

Table 1: Prioritized fronts for using AI in modernization.

Front	Objective	Expected artifacts
Reverse engineering	Reduce the effort required to understand legacy systems and support the reconstruction of technical and functional knowledge.	Technical summaries, component documentation, description of business rules, functional flows, behavioral hypotheses, attention points, and inputs for architecture.
Dependency mapping	Identify relationships among applications, components, integrations, databases, services, external calls, and business flows.	Dependency maps, impact matrices, relationship graphs, lists of consumers/providers, coupling risks, and inputs for transition planning.
Technology modernization	Support the upgrading of legacy technologies, assisted refactoring, and effort reduction in repetitive technical transformation tasks.	Upgrade recommendations, assisted code changes, compatibility analysis, support for library migration, identification of obsolete patterns, and review evidence.

Table 2: Initial measured results.

Indicator	Value
Business Services in the initial scope	252
Business units involved	7
Approximate planned effort	1.4 million hours
Internal and external collaborators	∼1,000
Hours saved with Reverse Engineering	21,780
Hours saved with Angular upgrading	11,646
Total hours already measured as saved	33,426
Approximate average saving per service	132.6 hours
Approximate equivalent in person-months	209 person–months

promise for reducing the time required for impact analysis, its quantitative gains will only be realized after the completion of the next waves of modernization.

At the reporting cutoff, the corporate catalog contained 134 registered agent entries. Backend was the largest category, with 83 entries, followed by Frontend with 25, Mainframe with 18, and Copilot with eight. No entries were classified under Migration or Quality at this stage. These categories represent the primary technical focus assigned to each catalog entry and do not directly correspond to the three modernization fronts, since an agent may support reverse engineering, dependency mapping, or technology modernization regardless of its technical category.

7 Discussion and Lessons Learned

The results suggest that AI can generate relevant gains when applied to knowledge-intensive activities, and not only to code generation. Savings in reverse engineering reinforce that one of the greatest opportunities lies in accelerating system understanding, knowledge recovery, and production of intermediate artifacts. This finding is consistent with the literature on AI for code, which points to applications in summarization, documentation, pattern inference, and support for software comprehension [2, 8, 12].

Savings in technology upgrading indicate that AI can support repetitive transformations, compatibility analysis, and assisted refactorings. The literature on library upgrading and deep learning-based refactoring points to similar opportunities, but also to limitations related to behavioral preservation, data quality, and human validation [13, 14].

The lessons summarized in Table 3 synthesize the main practical implications of the experience for software organizations facing large-scale legacy modernization. These lessons complement evidence previously reported in the same industrial context [16].

The adoption of AI in a brownfield context revealed that not all initial assumptions held up under the pressure of operational reality. Initially, we attempted to apply general-purpose agents to documentation tasks without contextual constraints; however, we observed that plausible but semantically incorrect outputs were often accepted by developers, compromising the quality of the artifacts. This led us to implement strict guardrails based on cross-checking with static analysis and mandatory human review for any changes to automatically generated code. Furthermore, we discovered that agents focused exclusively on code generation (without support for domain understanding) failed to handle complex business rules embedded in the legacy system; the solution was to redirect these efforts toward the extraction and validation of rules prior to syntactic transformation. These adaptations—from the introduction of quality checkpoints to the refinement of prompts based on continuous feedback—demonstrate that governance is not merely a static catalog, but a dynamic process of learning from day-to-day mistakes.

Table 3: Lessons learned from the AI-assisted modernization experience.

ID	Lesson learned
L1	**AI generates value before coding.** The main initial gains occurred in understanding, documentation, reverse engineering, and technology transformation.
L2	**Large-scale modernization requires business orientation.** Business Services helped connect technical decisions to organizational capabilities.
L3	**Governance is a condition for scaling AI.** Without a catalog, guardrails, and owners, agents tend to multiply in a fragmented way.
L4	**Intermediate metrics are necessary.** Since total schedule gains will only be known at the end of the modernizations, intermediate indicators allow impact monitoring during execution.
L5	**Human supervision remains essential.** Specialists remain necessary to validate business rules, dependencies, security, architecture, and functional equivalence.

The results complement evidence previously reported in the same industrial context. While the previous report highlighted an estimated 65% reduction in analysis effort through reverse-engineering and documentation automation, in addition to a prioritized scope of 199 business services across multiple waves [16], this work considers 252 Business Services with modernization starting in 2026 and specific indicators of hours saved in reverse engineering and technology upgrading.

8 Threats to Validity and Limitations

This report has limitations. The results are initial and correspond to activities already measured; total schedule gains will be measured at the end of the modernizations. The aggregated numbers do not detail variations by domain, technology, complexity, code volume, or dependency criticality. For confidentiality reasons, system names, source code, prompts, knowledge bases, proprietary

architecture, sensitive data, and restricted operational indicators are not presented.

There are also threats related to measuring hours saved, since effort savings depend on the baseline used, the complexity of the artifacts, and the participation of specialists. As this is an ongoing industry report, the results should not be automatically generalized to other contexts, although banks, insurers, public agencies, and large enterprises may share similar legacy challenges.

9 Conclusion

This paper presented an industry experience report on the governed use of AI in a large-scale legacy software modernization program in a Brazilian bank. The program involves 252 Business Services, approximately 1.4 million development hours, about 1,000 internal and external collaborators, and seven business units. The strategy prioritized reverse engineering, dependency mapping, and technology modernization.

The initial results indicate 33,426 hours saved, with greater impact in reverse engineering and technology upgrading. These results suggest that AI can generate value in knowledge-intensive and technical transformation activities, especially when applied with governance, traceability, and human supervision.

In addition to quantitative results, the paper presented the organizational governance structure based on the ACE squad, a corporate agent catalog, and internal software to manage, evaluate, approve, and communicate AI agents. As future work, we intend to expand the measurement of schedule gains in architecture, measure results at the end of the modernizations, and longitudinally evaluate the impact of AI on the modernization cycle of Business Services.

Artifact Availability

No public artifacts are available for this study. Due to confidentiality and organizational restrictions, source code, prompts, knowledge bases, internal architectural artifacts, operational data, and proprietary AI agents cannot be made publicly available.

Acknowledgments

The authors thank the teams involved in the modernization program.

This study was supported by the Coordination for the Improvement of Higher Education Personnel – Brazil (CAPES) – Finance Code 001.

References

[1] Abdulaziz Alhefdhi, Hoa Khanh Dam, Hideaki Hata, and Aditya Ghose. 2018. Generating Pseudo-Code from Source Code Using Deep Learning. In *2018 25th Australasian Software Engineering Conference (ASWEC)*. 21–25. doi:10.1109/ASWEC.2018.00011

[2] Miltiadis Allamanis, Earl T. Barr, Premkumar Devanbu, and Charles Sutton. 2018. A Survey of Machine Learning for Big Code and Naturalness. *ACM Comput. Surv.* 51, 4, Article 81 (July 2018), 37 pages. doi:10.1145/3212695

[3] BIAN. 2026. Banking Industry Architecture Network. https://bian.org/. Accessed: 2026.

[4] Chia-Chu Chiang and Coskun Bayrak. 2006. Legacy Software Modernization. In *2006 IEEE International Conference on Systems, Man and Cybernetics*, Vol. 2. 1304–1309. doi:10.1109/ICSMC.2006.384895

[5] Valerio Cosentino, Jordi Cabot, Patrick Albert, Philippe Bauquel, and Jacques Perronnet. 2013. Extracting business rules from COBOL: A model-based framework. In *2013 20th Working Conference on Reverse Engineering (WCRE)*. 409–416. doi:10.1109/WCRE.2013.6671316

[6] Manoel Valerio da Silveira Neto, Andreia Malucelli, and Sheila Reinehr. 2026. A Subway Map of Software Architecture: A Multivocal Review and Visual Framework. In *Proceedings of the 28th International Conference on Enterprise Information Systems - Volume 2: ICEIS*. INSTICC, SciTePress, 1431–1442. doi:10.5220/0014672700004018

[7] T. J. Harmer, P. J. McParland, and J. M. Boyle. 1996. Using Knowledge-Based Transformations to Reverse-Engineer COBOL Programs. In *Proceedings of the 11th Knowledge-Based Software Engineering Conference*. IEEE Computer Society Press, 114–123. doi:10.1109/KBSE.1996.552829

[8] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. 2024. Large Language Models for Software Engineering: A Systematic Literature Review. *ACM Trans. Softw. Eng. Methodol.* 33, 8, Article 220 (Dec. 2024), 79 pages. doi:10.1145/3695988

[9] M. Jha and P. Maheshwari. 2005. Reusing Code for Modernization of Legacy Systems. In *13th IEEE International Workshop on Software Technology and Engineering Practice*. IEEE, 57–66. doi:10.1109/STEP.2005.21

[10] Ravi Khadka, Bennie V. Batlajery, Amir M. Saeidi, Slinger Jansen, and Jurriaan Hage. 2014. How Do Professionals Perceive Legacy Systems and Software Modernization?. In *Proceedings of the 36th International Conference on Software Engineering (ICSE 2014)*. ACM, 36–47. doi:10.1145/2568225.2568318

[11] Ramesh Krishnan, Pradeep Murali, Rijurekha Sen, Vineet Chaoji, Ashish Shukla, Saurabh Bansal, and Manish Tiwari. 2020. Mono2Micro: An AI-Based Toolchain for Evolving Monolithic Enterprise Applications to a Microservice Architecture. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2020)*. ACM, 1606–1610. doi:10.1145/3368089.3417933

[12] Triet H. M. Le, Hao Chen, and Muhammad Ali Babar. 2020. Deep Learning for Source Code Modeling and Generation: Models, Applications, and Challenges. *Comput. Surveys* 53, 3 (2020), 1–38. doi:10.1145/3383458

[13] Pranav Naik, Sai Nelaballi, V. S. Pusuluri, and Doo-Kwon Kim. 2023. Deep Learning-Based Code Refactoring: A Review of Current Knowledge. *Journal of Computer Information Systems* 64, 1 (2023), 1–16. doi:10.1080/08874417.2023.2203088

[14] Phuong T. Nguyen, Juri Di Rocco, Riccardo Rubei, Claudio Di Sipio, and Davide Di Ruscio. 2022. DeepLib: Machine Translation Techniques to Recommend Upgrades for Third-Party Libraries. *Expert Systems with Applications* 202 (2022), 117267. doi:10.1016/j.eswa.2022.117267

[15] Hammond Pearce, Baleegh Ahmad, Benjamin Tan, Brendan Dolan-Gavitt, and Ramesh Karri. 2022. Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions. In *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE, 754–768. doi:10.1109/SP46214.2022.9833571

[16] Manoel Valerio da Silveira Neto and Bruno Pereira Soares. 2026. Large-Scale Legacy Systems Modernization: An Experience Report on Transition Architectures and AI in a Brazilian Bank. Presented at the 27th International Conference on Agile Software Development, São Paulo, Brazil. doi:10.13140/RG.2.2.11020.07041 Available online; not published in the conference proceedings.

[17] Harry M. Sneed. 2010. Migrating from COBOL to Java. In *2010 IEEE International Conference on Software Maintenance*. IEEE, 1–7. doi:10.1109/ICSM.2010.5609583

[18] Harry M. Sneed and K. Erdoes. 2013. Migrating AS400-COBOL to Java: A Report from the Field. In *2013 17th European Conference on Software Maintenance and Reengineering*. IEEE, 231–240. doi:10.1109/CSMR.2013.32

[19] Norbert Somogyi and Gábor Kővesdán. 2021. Software Modernization Using Machine Learning Techniques. In *2021 IEEE 19th World Symposium on Applied Machine Intelligence and Informatics (SAMI)*. IEEE, 361–366. doi:10.1109/SAMI50585.2021.9378659

[20] N. Veerman. 2003. Revitalizing Modifiability of Legacy Assets. In *Seventh European Conference on Software Maintenance and Reengineering*. IEEE Computer Society, 19–29. doi:10.1109/CSMR.2003.1192407