

Managed Evolution of a Legacy Core Banking System: Case Study from a European Bank

Philipp Brune^{1,2}[0000–0002–2570–7462] and Kay Helbig³

¹ Neu-Ulm University of Applied Sciences, Wileystraße 1, 89231 Neu Ulm, Germany

² SQ Solutiosn GmbH, Platz der Einheit 2, 60327 Frankfurt/Main, Germany

Philipp.Brune@hnu.de

³ EGC Eurogroup Consulting AG, Thurn und Taxis Platz 6, 60313 Frankfurt/Main, Germany

Kay.Helbig@eurogroupconsulting.de

Abstract. Legacy systems continue to remain crucial for the core business operations of most financial service providers around the globe. However, increasing operational risk due to knowledge loss and lack of skills because of demographic change, rising maintenance and run costs, as well as changing business requirements, put organizations under pressure to modernize them. By applying the case study research methodology, this paper derives the core elements of a socio-technical system design methodology for the stepwise managed evolution of a large, mainframe-based and historically grown legacy application. It demonstrates why this approach could be superior to frequently favored "lift-and-shift" or replace strategies, with respect to cost efficiency, risk mitigation and solution quality. In addition, a system architecture and implementation strategy are proposed and evaluated to demonstrate, how this is achieved.

Keywords: Legacy Modernization · Socio-Technical Systems · Core Banking System · Re-Engineering · COBOL · Java

1 Introduction

The modernization of software systems, particularly the migration of monolithic legacy mainframe applications, represents a significant challenge in software engineering. As organizations seek to transition away from outdated technologies like COBOL and PL/1, along with the corresponding legacy databases (e.g. IMS-DB) or transaction processing monitors (like IMS-TM or CICS) [1], [2], they must overcome various technical and organizational barriers.

One of the main difficulties arises from the declining availability of expertise in these legacy technologies [3]. Many legacy applications were developed several decades ago, and the original developers have since retired. This demographic shift has resulted in a growing skill gap, with respect to maintaining these systems [4], [5].

Moreover, as usually multiple legacy systems coexist, understanding their interdependencies becomes a significant task [6].

Another significant complication is the often insufficient documentation of legacy systems. The lack of clear and comprehensive documentation limits the ability of current development teams to decipher existing code [7], [8].

The usual tight coupling of monolithic architectures, which are often intricately integrated at the database and module call levels, makes it challenging to isolate and transfer individual components to more flexible architectures [9].

The absence of automated testing poses a further challenge. Historically, these systems were designed without contemporary testing protocols in mind, increasing the testing efforts massively, including the risk of errors during system alterations [8], [10].

Decoupling monolithic structures into services-style components can mitigate such risks, as service-oriented architectures promote modularity and independent deployments, thus simplifying testing and error recovery processes [11].

Furthermore, the effort and cost necessary for refactoring legacy systems cannot be understated. This transition is not merely a technical migration, but requires a broader change management strategy, involving stakeholder engagement and potential restructuring of team dynamics [6], [7].

Due to these reasons, while different modernization approaches have been proposed and applied in research and practice, the legacy modernization challenge remains unsolved to a large extent until today [1], [2].

Therefore, in this paper, results from a case study of an ongoing modernization project for large-scale COBOL applications in a big European bank are presented, which uses a step-wise, managed evolution approach for the transition from COBOL to Java and to containerized services.

As such modernization projects could be understood as complex socio-technical systems [12], the strategy adopted in this case study serves as a blueprint for the core elements of an evolving socio-technical systems design (STSD) methodology [13] for evolutionary legacy modernization. STSD is an approach to design combining technological, organizational, human and social factors [13].

The rest of this paper is organized as follows: In section 2, the related work is analyzed in more detail, while the research design used is presented in section 3. Section 4 describes the approach used and the related design artifacts. The evaluation and results are presented in section 5. We conclude with a summary of our findings.

2 Related Work

The concept of understanding organizations as socio-technical systems was originally introduced by Emery in 1959 [12], [15], and over the years has been widely applied to various kinds of organizational problems [15], [16].

Here, we adopt the conceptual structure of socio-technical systems proposed by Leavitt [14], as it combines the various aspects of social-technical systems into four main subsystems, which allow for a more pragmatic application during the case study. Figure 1 illustrates these subsystems and their mutual interactions.

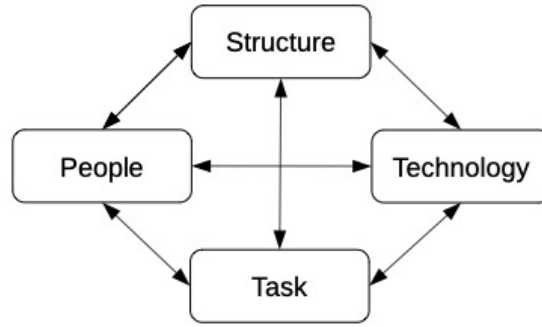


Fig. 1. Main subsystems of a socio-technical system and their mutual interactions as proposed by Leavitt [14].

Introducing a socio-technical perspective to the field of software development and system design is considered beneficial to improve project outcomes and reduce risks [13]. Consequently, such an approach would then lead to STSD methodologies [13].

Nevertheless, in many software development projects the technical side is still dominating, so Baxter and Sommerville argue that “a pragmatic approach to the engineering of socio-technical systems based on the gradual introduction of socio-technical considerations into existing software procurement and development processes” is required [13].

This holds in particular true for software re-engineering and legacy application modernization projects, where in addition to the general tasks of software engineering, one typically has to deal with a lack of skills, documentation and knowledge [17].

However, the design of a STSD method for legacy application modernization has rarely been discussed so far. While the modernization of legacy mainframe applications [18] has been a key focus of both research and practical applications over the years [17], [19]–[22], the majority of the existing literature addresses the technical side only.

Traditional methods for modernizing legacy systems are generally grouped into the following strategies:

Rehosting and Replatforming involve migrating legacy applications to new platforms, which can improve their performance and accessibility [1], [20], [23]. This strategy is often combined with wrapping techniques that encapsulate legacy systems, enabling them to interact with more modern applications [19], [21], [22], [24].

Rearchitecting focuses on redesigning legacy programs to run on modern architectures and programming languages. The goal is to enhance the maintainability and scalability of these applications in a gradual, evolutionary manner, ensuring they align better with current business requirements [22], [24], [25].

Redevelopment refers to the complete redevelopment of the legacy system, which can be resource-intensive but may provide substantial long-term advantages. This approach is often necessary when the existing system is too outdated to be effectively modernized [17], [18], [25], [26].

Replacement entails replacing the legacy system entirely with an "off-the-shelf" solution, offering immediate functionality without requiring substantial redevelopment efforts [17].

Refactoring makes use of **transpilers** — tools that formally translate one programming language into another, facilitating modernization by converting legacy code into more contemporary languages, thereby extending the life and utility of the existing application [27], [28].

However, such translations typically only result in the original code being written in a new syntax without altering the underlying architecture or structure. In the case of COBOL-to-Java migrations, this often leads to the creation of "Jobol" or "Jabol" code, which can be challenging for native Java developers to understand and maintain.

While other current approaches try to overcome the drawbacks of transpiler solutions by applying generative AI (GenAI)[2], in this study we investigate the general approach of a partial, step-wise redevelopment of legacy systems, enabling a risk-minimized managed evolution instead of the full migration usually studied in the literature. This has only rarely been studied in research so far.

Nevertheless, our approach might be combined with GenAI techniques in a second step as well.

Therefore, in this paper the research question is addressed, how the core elements of a STSD methodology for legacy application modernization with respect to technology (software architecture, tools), task (development and operations processes), people and structure (change management, organizational structure) need to be designed, to enable a step-wise, managed evolution of legacy systems from COBOL to Java, and finally to containerized services.

3 Research Design

This paper adopts a case study research design, following the guidelines described by Yin [29].

This approach was chosen, as a case study "investigates a contemporary phenomenon within its real-life context, especially when the boundaries between phenomenon and context are not clearly evident" [29, p. 13], which corresponds best to the scenario given in the respective project under investigation.

3.1 Project Context

The main original driver for the described project was a substantial increase in mainframe run costs by the bank's current mainframe outsourcing provider.

The core strategy for cost reduction is to successively shift the workload from the monolithic COBOL code to a modern, clean Java implementation to

increasingly utilize the mainframe’s so-called z Information Integration Processors (zIIP) (dedicated CPUs for Java workloads), while additionally modernizing the software architecture.

zIIPs are licensed on a per-CPU basis instead consumption-based, therefore being much cheaper than the general-purpose CPU (CPC) required by COBOL code.

The first phase of the project’s implementation focuses on short-term cost optimization and preparing for platform neutrality. This involves transferring COBOL to Java under CICS, which already results in initial cost savings through the use of zIIPs. The initial focus of this phase is on the top 10 applications in the securities system that have the highest CPU consumption.

3.2 Units of Analysis

The units of analysis cover the main outcomes of the project’s scope:

The project is centered around a hybrid cloud target platform architecture, complemented by a developed software architecture that integrates COBOL and Java.

This combination supports the managed evolution of the solution, ensuring both legacy compatibility and modern scalability. A structured development process is required, together with appropriate tools to enhance productivity and maintain quality.

In parallel, the implementation of organizational change and skill development initiatives ensures the workforce is equipped to adapt to new technologies and methodologies.

A comprehensive risk mitigation strategy has been defined to proactively address potential challenges, while the overall project is guided by a robust project management approach to ensure timely and successful delivery.

3.3 Evaluation Criteria

The evaluation criteria for this initiative focus on several key factors to ensure both operational success and long-term sustainability.

A primary consideration is the reduction of run costs, aiming to optimize resources while maintaining system efficiency.

In addition, the solution must deliver performance, scalability, and security that are on par with existing standards, ensuring the system can handle growth without compromising reliability or data protection.

Future maintainability is also crucial, with an emphasis on ease of updates and long-term system health. The approach supports a stepwise implementation, allowing for gradual transitions and minimizing disruption. It is essential that the solution fits well within the organization’s existing structure, promoting seamless integration and adoption.

Furthermore, alignment with both business requirements and IT strategy is key, ensuring that the solution serves the company’s objectives while also leveraging current technological priorities. Finally, a strong focus on risk reduction

ensures that potential challenges are proactively addressed, safeguarding the project's success.

4 Designed Artifacts and Data Collection

4.1 Platform Architecture Enabling Managed Evolution

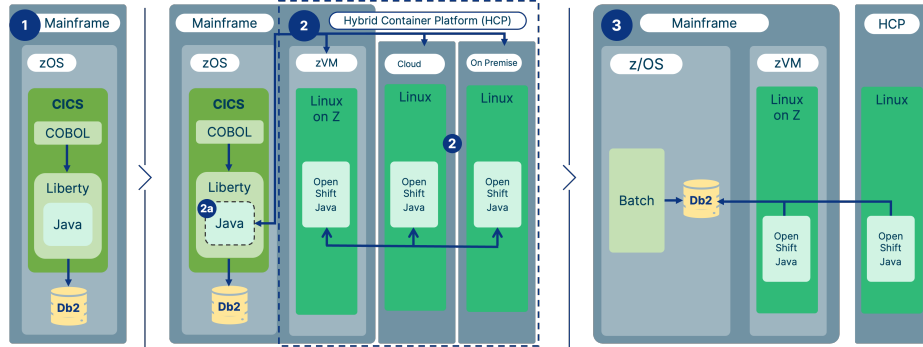


Fig. 2. Proposed phases of the managed evolution of the platform architecture.

To enable the managed evolution of the existing mainframe platform, a three-phased approach was designed, as illustrated in figure 2. It allows for a step-by-step modernization of the platform, moving from a traditional mainframe-centric setup towards a hybrid, cloud-ready environment:

1. Phase focuses on short-term cost optimization and preparation for platform neutrality. It involves the selective transfer of COBOL services to Java. These Java services will run under CICS on the mainframe, using the Liberty⁴ application server. The initial focus is on the top 10 applications in the securities system with respect to highest CPU consumption. This phase maintains the core mainframe infrastructure but introduces modern Java applications integrated with the existing COBOL framework.
2. Phase represents the further development towards a cloud-ready architecture (platform neutrality). The design introduces the operation of Java applications using container technology (OpenShift). This includes the option for a hybrid architecture, where Java applications can run on both the mainframe (specifically under z/VM Linux on Z with OpenShift) and cloud container platforms. Migration steps to container platforms are possible during this phase.

⁴ See <https://openliberty.io>

3. Phase depicts target state of the architecture. In this state, the host (mainframe under z/OS) is primarily developed into a data storage and provision platform for a heterogeneous application environment. It is mainly used for high-volume batch processing. Applications are migrated to a platform-independent architecture based on their respective requirements. Online transaction processing takes place in the hybrid cloud (using OpenShift containers on Linux on Z and other platforms).

The managed evolution through these phases was chosen to achieve several key objectives:

- **Significant cost reduction:** The primary driver is the substantial increase in prices for outsourcing services, necessitating urgent action to address rising operating costs, particularly affecting the securities system. Migrating COBOL services to Java allows the project to increasingly shift workload to cheaper zIIP processors. The operational transition of applications immediately after implementation enables a continuous reduction of operating costs for a quick effect.
- **Future-proofing and platform neutrality:** Introducing Java and container technology (OpenShift) ensures platform neutrality and establishes cloud-readiness. This provides flexibility for future deployment choices across different platforms.
- **Combining modernization with short-term effects:** Phase 1, focusing on the top CPU consumers and migrating them to Java on the mainframe, combines modernization with short-term savings effects. This allows the organization to start realizing cost benefits quickly while building the foundation for future architectural changes.
- **Managed transition and skill building:** The phased approach allows for a stepwise conversion of COBOL programs. It helps the bank to manage the changes associated with the new technology, including building necessary skills and processes. The project plans for the training of internal employees by integrating them into implementation teams and providing defined training measures.

4.2 Software Architecture

The core business logic of the project is strategically designed to reside within a modern Java architecture, with the intention that this business logic should be implemented only once to promote efficiency and maintainability.

The Java codebase is constructed to function as a plug-in replacement for existing COBOL modules, thereby facilitating a seamless transition and integration of contemporary technologies into legacy systems.

In this design, as illustrated in figure 3, mainframe-specific aspects are carefully encapsulated within the Java code, ensuring that legacy constraints are effectively managed without compromising the advantages of modernization.

The results and impacts of this Java reference architecture design are substantial: the business-specific Java code is implemented as a container-ready

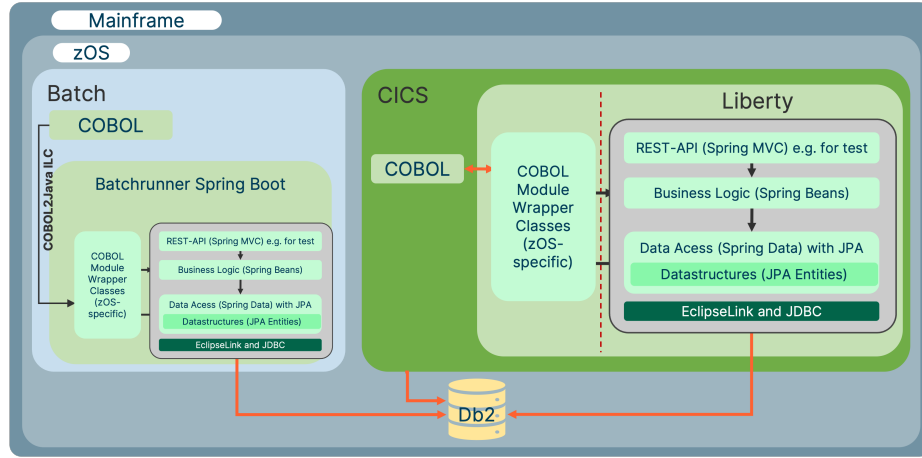


Fig. 3. Software architecture for embedding modern Java components using a cloud-ready implementation within the existing COBOL codebase, enabling a step-wise evolutionary replacement of COBOL with Java.

Spring Boot application, which is then integrated into the mainframe environment through a COBOL wrapper. This integration adheres to the established Java architecture guidelines, including Java 17, Spring Boot 3.2.x, and Spring Data, thereby ensuring compliance with contemporary industry standards.

Moreover, the architecture has been optimized to run under Liberty within the CICS environment, as well as in batch processing scenarios, enhancing its versatility.

Notably, Java-COBOL interoperability is well-supported through IBM LE ILC, available from Compiler Version 6.4 onwards, which further facilitates the smooth coexistence of Java and COBOL in the enterprise system landscape.

4.3 Development Process and Tools

Jira and Confluence are key tools used in the project across the main areas of development and project management:

Here, Confluence is used for organization of the team, maintaining the meeting calendar (including workshops, etc.), managing the vacation calendar (to be maintained by team members), handling planning and providing an overview list of milestones with status, and tracking an open points list for topics that are not managed via Jira tickets. In addition, it is used to store meeting minutes, result and status reports, and thereby supporting risk management.

On the other hand, Jira is used for structuring and tracking work items based on the development steps up to deployment. This includes the phases of development, test, deployment, and operations.

Within the project’s meeting structures, Jira tickets will be reviewed if necessary. Open points that are not captured as tickets will also be documented and tracked.

In summary, Confluence serves as a central repository for project documentation, planning, meeting information, and non-ticketed open points, while Jira is used for the structured tracking and management of tasks and work items throughout the development lifecycle and for general organizational purposes.

The standard Java development toolchain, including tools like Git, Jenkins, etc., is used, with deployment to z/OS and container platforms using the company’s respective, well-established custom tools.

4.4 Project Management Approach and Risk Mitigation

To manage the project, we rely on a combination of agile project management and traditional methods to achieve the best possible results. This hybrid approach allows us to react flexibly to changes while maintaining a stable and clear structure.

The project team consists of external employees from the service provider (including the authors) and a team from the customer. The external team focuses on architecture, development, project control and management communication.

On the customer side, the team includes the developers responsible for the COBOL application, which will be successively converted to Java as part of the modernization.

Project management is a shared responsibility between the service provider and the customer. The main role of the customer project manager is to manage the customer organization to ensure that the technical and organizational framework is in place for the project to run smoothly. In particular, the internal project manager provides access to resources that are not directly assigned to the project, such as setting up permissions or the necessary tool chain to deploy the Java modules in the infrastructure (development, test and production).

The technical conversion of the COBOL modules to Java follows an agile approach with defined sprints for each module. Each module is implemented in three sprints of three weeks each. Topics such as documentation, integration and acceptance tests are managed according to the classic project management approach.

A key component of our project management is a clear governance structure, which ensures that all project participants are fully aware of their roles and responsibilities.

In addition, the established meeting structure ensures continuous dialogue through weekly internal and joint status meetings for each sprint.

The actual implementation takes place in blocks for each COBOL module to be converted. Each module is then implemented in three sprints of three weeks each. After the last sprint, the integration test phase begins, followed by a four-week test block in the acceptance environment to ensure that the new Java modules are ready for use. The entire testing phase runs in parallel with the subsequent implementation phase of the COBOL modules.

At the beginning of each implementation block, the backlog is planned and divided into sprints. The first sprint is planned very precisely, while the remaining sprints are planned more roughly. Further sprint planning takes place before each sprint as part of the defined governance structures to plan the following sprint in detail.

At the end of the last sprint, a retrospective is conducted. This allows us to learn from the experiences of previous sprints and to continuously improve our processes.

Independent of the planned retrospectives, any shortcomings identified during the sprints are addressed immediately.

During the conversion of the modules from COBOL to Java, the COBOL developers were simultaneously trained in Java (on-the-job training) to ensure the future operation and further development of the converted application.

4.5 Data Collection

For the empirical evaluation of the described processes and artifacts designed for the project, a multifaceted approach to gathering feedback was adopted to ensure a comprehensive assessment of their efficacy and impact.

This evaluation utilized various sources of feedback information, which included the minutes from sprint retrospectives, wherein team members reflected on the successes and challenges encountered during the sprint cycle.

In table 1, the experts participating in the qualitative study by contributing their feedback and insights are described in detail.

Additionally, the insights garnered from code reviews played a crucial role in identifying areas of strength and opportunities for improvement in the codebase.

Performance measurements derived from load tests offered quantitative data regarding the system’s responsiveness and stability under different conditions, serving as a critical indicator of operational capabilities.

Furthermore, documented reports from collaboration tools such as Confluence and Jira provided structured insights into task progress, project timelines, and bottlenecks that were identified throughout the project lifecycle.

Lastly, informal feedback collected from project members contributed qualitative insights, capturing the nuanced experiences and perceptions that may not be fully represented in formal documentation.

These different feedback sources were crucial in continuously evaluating the project’s processes and artifacts, thereby facilitating informed decision-making and continuous improvement.

5 Evaluation and Results

The platform and software architectures, as well as the development and deployment process and tools, were validated using a pilot implementation of a first, comparably simple business service. A second, more complex business service was re-implemented in Java immediately afterwards.

Table 1. Overview about the experts contributing to the qualitative analysis, the topics they gave feedback on, the artifacts used for analysis and the respective time period covered by the analysis. The experts consisted of project team members and stakeholders, grouped here into internal employees of the European bank and employees of the external project partner.

Internal Experts	Topics	Analyzed Artifacts	Analysis Period
Project Manager	Schedule, Cost, Quality	Retrospectives, Emails, Meeting Minutes, Oral Communication	02/2024 - 02/2025
Senior developers COBOL and Java 1 and 2	Software Architecture, Code, Tests, Build, Deployment, Performance	Code Reviews, Documentation, Test Results, Meeting Minutes, Emails, Jira Tickets, Retrospectives	07/2024 - 02/2025
Solution Manager	Correctness, Performance, Maintainability,	Meeting Minutes, Test Results, Retrospectives	07/2024 - 02/2025
Enterprise Java Architect	Software Architecture, Build, Deployment	Source Code, Documentation, Code Reviews	02/2024 - 06/2024
Mainframe Security Specialist	IT Security Configuraton	Emails, Retrospectives	02/2024 - 02/2025
Senior Mainframe Specialist	Architecture, Deployment, IT Security, Performance	Documentation, Meeting Minutes, Emails, Jira Tickets, Retrospectives	02/2024 - 12/2024
Test Management Team	Functional Correctness, Performance	Test Reports, Logfiles	01/2025 - 02/2025
Head of Mainframe Operations	Cost, Skills, Reliability,	Discussions, Emails	02/2024 - 02/2025
Head of Core Banking Systems	IT Strategy, Alignment, Cost	Discussions, Emails	02/2024 - 02/2025
External Experts			
Business Consultant (Project Office)	Schedule, Planning	Retrospectives, Emails, Meeting Minutes, Slides, Project Plans	02/2024 - 02/2025
Senior Mainframe Consultant	Architecture, Deployment, IT Security, Performance	Documentation, Meeting Minutes, Emails, Jira Tickets, Retrospectives	07/2024 - 02/2025
Senior Technical Consultant	Software Architecture, Code, Tests, Deployment, Performance	Code Reviews, Documentation, Meeting Minutes, Emails, Jira Tickets, Retrospectives	07/2024 - 02/2025
Senior CICS Consultant	Architecture, Deployment, IT Security, Performance	Documentation, Meeting Minutes, Emails, Jira Tickets, Retrospectives	02/2024 - 02/2025

The findings from the case study indicate how the core elements of a STSD methodology with respect to the four subsystems illustrated in figure 1 need to be designed:

5.1 Technology

The data collected during the evaluation successfully validated the architecture for migrating COBOL to Java. The qualitative evaluation of the architecture and its technical feasibility made clear that it fits well for the purpose, resulting in a clean and modern Java architecture.

The runtime analysis confirmed that approximately 30-50 percent of the processor load can be shifted to the less expensive zIIP.

Even with Java programs, some processing still occurs on the main CPC processor because the customers proprietary COBOL framework is still necessary, and the Java program is called from within this framework.

The comparison measurement between the original COBOL module and the newly written Java application showed that approximately 30% to 50% of the processor load can be shifted to the cheaper zIIP processors.

Specifically, for the service implemented first, containing little business logic, about 30% of the load could be shifted to zIPP. For the second service with much more business logic, about 50% of the load could be shifted.

The success of the runtime behaviour provides a valid decision basis and confirms the feasibility of shifting workload to zIIP processors, supporting the planned conversion of COBOL to Java.

5.2 People and Structure

The established project management structures promote transparency and facilitate communication within the operational team and with management through regular steering committees with formalized status reporting.

These regular meetings enable us to continuously monitor progress, identify issues early and implement improvements promptly.

With mainframe modernization projects such as this, it is important to understand that it is not a purely technical project. The technical change touches the customer's organization and starts a major change process.

5.3 Task

New processes and other skills are required to manage the new infrastructure efficiently and effectively. As part of our project, the external project team not only converted the modules, but also worked with the customer to build the new tool chain, convert and define processes, and transfer knowledge to the customer's staff.

The hybrid project management approach allows for a balanced mix of flexibility and stability, which enables us to manage projects efficiently and successfully. By combining agile and traditional methods, we are able to make short-term adjustments while keeping an eye on long-term goals.

Overall, the results of this case study clearly show that the approach chosen is not only feasible, but addresses the specific requirements of the managed evolution of legacy systems towards a modern architecture in a successful way.

6 Conclusion

In conclusion, despite the various modernization strategies that have been proposed and implemented in both academic research and practical settings, the challenge of legacy modernization remains largely unresolved to this day.

This paper presents findings from a case study focused on an ongoing modernization initiative involving large-scale COBOL applications within a major European bank.

The project employs a stepwise, managed evolution approach to transition from COBOL to Java and subsequently to containerized services.

The results indicate, how the core elements of a socio-technical system design methodology for legacy application modernization need to be designed, and support the advantages of adopting a managed, stepwise evolution of core legacy mainframe applications over more commonly pursued strategies like "lift-and-shift" or complete replacement.

The presented approach is particularly effective in terms of cost efficiency, risk minimization, and solution quality.

Though further research is needed, the results indicate that the strategy adopted in this case study has the potential to serve as a valuable blueprint and best practice for similar modernization efforts.

References

- [1] P. Brune, "An open source approach for modernizing message-processing and transactional cobol applications by integration in java ee application servers," in *Web Information Systems and Technologies: 14th International Conference, WEBIST 2018, Seville, Spain, September 18–20, 2018, Revised Selected Papers 14*, Springer, 2019, pp. 244–261.
- [2] C. Kuck and P. Brune, "Genai-driven migration of legacy cobol applications to clean java code," in *22nd International Conference on Smart Business Technologies (ICSBT 2025)*, To be published, Springer, 2025.
- [3] R. Khadka, B. V. Batlajery, A. M. Saeidi, S. Jansen, and J. Hage, "How do professionals perceive legacy systems and software modernization?" In *Proceedings of the 36th International Conference on Software Engineering*, ACM, 2014, pp. 36–47.
- [4] A. Ganesan, T. Chithralekha, and M. Rajapandian, "A formal model for legacy system understanding," *International Journal of Intelligent Systems and Applications*, vol. 10, pp. 27–41, 10 2018. DOI: 10.5815/ijisa.2018.10.04.

- [5] A. De Lucia, R. Francese, G. Scanniello, and G. Tortora, "Developing legacy system migration methods and tools for technology transfer," *Software: Practice and Experience*, vol. 38, no. 13, pp. 1333–1364, 2008. DOI: <https://doi.org/10.1002/spe.870>. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/spe.870>. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.870>.
- [6] Z. Mushtaq, N. Saher, F. Shazad, S. Iqbal, and A. Qasim, "A review on transformation of monolithic applications towards microservices environment," *International Journal of Innovations in Science and Technology*, vol. 4, pp. 1–18, 1 2022. DOI: [10.33411/ijist/2022040101](https://doi.org/10.33411/ijist/2022040101).
- [7] M. Srinivas, G. Ramakrishna, K. R. Rao, and E. S. Babu, "Analysis of legacy system in software application development: A comparative survey," *International Journal of Electrical and Computer Engineering*, vol. 6, no. 1, p. 292, 2016.
- [8] J. Fritzsche, J. Bogner, A. Zimmermann, and S. Wagner, "From monolith to microservices: A classification of refactoring approaches," in *International Workshop on Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment*, Springer, 2018, pp. 128–141.
- [9] M. Kalske, N. Mäkitalo, and T. Mikkonen, "Challenges when moving from monolith to microservice architecture," in *International Conference on Web Engineering*, Springer, 2017, pp. 32–47.
- [10] M. Schnappinger and J. Streit, "Efficient platform migration of a mainframe legacy system using custom transpilation," in *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2021, pp. 545–554. DOI: [10.1109/ICSME52107.2021.00055](https://doi.org/10.1109/ICSME52107.2021.00055).
- [11] M. Seedat, Q. Abbas, and N. Ahmad, "Systematic mapping of monolithic applications to microservices architecture," 2022. DOI: [10.21203/rs.3.rs-1990400/v1](https://doi.org/10.21203/rs.3.rs-1990400/v1).
- [12] F. E. Emery, *Characteristics of Socio-technical Systems: a Critical Review of Theories and Facts...* Human Resource Centre, Tavistock Institute of Human Relations, 1959.
- [13] G. Baxter and I. Sommerville, "Socio-technical systems: From design methods to systems engineering," *Interacting with computers*, vol. 23, no. 1, pp. 4–17, 2011.
- [14] H. J. Leavitt, "Applied organizational change in industry: Structural, technological and humanistic approaches," in *Handbook of organizations (RLE: Organizations)*, Routledge, 2013, pp. 1144–1170.
- [15] E. L. Trist, *The evolution of socio-technical systems*. Ontario Quality of Working Life Centre Toronto, 1981, vol. 2.
- [16] H. Hendrick, "Sociotechnical systems theory: The sociotechnical systems model of work systems," *International encyclopedia of ergonomics and human factors*. 2nd ed. London: Taylor and Francis, pp. 2966–2968, 2006.
- [17] A. Mbaya, J. Laval, and N. Moalla, "An assessment conceptual framework for the modernization of legacy systems," in Jan. 2019, pp. 347–357, ISBN: 978-3-030-13692-5. DOI: [10.1007/978-3-030-13693-2_29](https://doi.org/10.1007/978-3-030-13693-2_29).
- [18] J. Bisbal, D. Lawless, B. Wu, *et al.*, "A survey of research into legacy system migration," *Technique report*, pp. 1–39, 1997.
- [19] H. M. Sneed, "Migration of procedurally oriented cobol programs in an object-oriented architecture," in *Software Maintenance, 1992. Proceedings., Conference on*, IEEE, 1992, pp. 105–116.

- [20] K. Talati and C. W. Lackie, *Virtual software machine for enabling cics application software to run on unix based computer systems*, US Patent 6,006,277, 1999.
- [21] M. Mossienko, “Automated cobol to java recycling,” in *Seventh European Conference on Software Maintenance and Reengineering, 2003. Proceedings.*, 2003, pp. 40–50. DOI: 10.1109/CSMR.2003.1192409.
- [22] H. M. Sneed, “Migrating from cobol to java,” in *2010 IEEE International Conference on Software Maintenance*, 2010, pp. 1–7. DOI: 10.1109/ICSM.2010.5609583.
- [23] E. Farmer, “The reality of rehosting: Understanding the value of your main-frame,” 2013.
- [24] H. M. Sneed and K. Erdoes, “Migrating as400-cobol to java: A report from the field,” in *2013 17th European Conference on Software Maintenance and Reengineering*, 2013, pp. 231–240. DOI: 10.1109/CSMR.2013.32.
- [25] D. Mazeika and R. Butleris, “Identifying security issues with mbse while rebuilding legacy software systems,” in *2020 IEEE 15th International Conference of System of Systems Engineering (SoSE)*, 2020, pp. 83–86. DOI: 10.1109/SoSE50414.2020.9130491.
- [26] H. Sneed and C. Verhoef, “Re-implementing a legacy system,” *Journal of Systems and Software*, vol. 155, pp. 162–184, 2019, ISSN: 0164-1212. DOI: <https://doi.org/10.1016/j.jss.2019.05.012>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0164121219301050>.
- [27] F. A. Bastidas and M. Pérez, “A systematic review on transpiler usage for transaction-oriented applications,” in *2018 IEEE Third Ecuador Technical Chapters Meeting (ETCM)*, IEEE, 2018, pp. 1–6.
- [28] B. Roziere, M.-A. Lachaux, L. Chanussot, and G. Lample, “Unsupervised translation of programming languages,” *Advances in neural information processing systems*, vol. 33, pp. 20 601–20 611, 2020.
- [29] R. K. Yin, *Case study research: Design and methods*. sage, 2009, vol. 5.