World Journal of Advanced Research and Reviews
eISSN: 2581-9615 CODEN (USA): WJARAI
Cross Ref DOI: 10.30574/wjarr
Journal homepage: https://wjarr.com/

(REVIEW ARTICLE)

Check for updates

A structured approach to legacy banking system modernization at enterprise scale

Ashmitha Nagraj *

Principal Full Stack Engineer.

World Journal of Advanced Research and Reviews, 2022, 15(01), 890-897

Publication history: Received on 18 June 2022; revised on 24 July 2022; accepted on 29 July 2022

Article DOI: https://doi.org/10.30574/wjarr.2022.15.1.0736

Abstract

Legacy Banking Systems must be upgraded. Large banks today rely upon core systems developed decades ago. Those systems are expensive to support and difficult to change. This means that banks need to modernize their legacy banking systems so they can be agile, resilient and cost-effective. In this paper, a systematic approach to migrating large legacy systems has been identified. Drivers have been defined with both the structure and operation of legacy systems and have surveyed migration methods and have developed a quantifiable cost model to associate migration design with expense. This paper also provides decision-making matrices and tables to guide in the selection of strategy and identify "best practices" for implementing such strategies such as automation of development and operations, cost management of finance, testing and governance. The analysis includes specific enterprise examples and references to peer-reviewed research and industry reports. Key takeaways are to perform modernization in a physical and use detailed financial models to measure the potential value of migration.

Keywords: Legacy Modernization; Banking IT; Cloud Migration; Microservices; FinOps; Hybrid Cloud; Compliance; Cost Model.

1. Introduction

Legacy banking applications typically use decades old core systems which tend to be expensive to operate and are highly rigid. The cost associated with maintaining older systems consumes approximately 70 – 75 percent of an organizations IT spending [1] and thus limits the availability of capital and personnel to support innovation. On the other hand, customers now demand and expect new and modern digital customer experiences while regulatory bodies are increasingly requiring banks to have well-designed and maintained technology systems.

Therefore, the need for large scale legacy modernization within enterprises has never been greater. This paper will provide a holistic view of how to carry out legacy modernization by describing the architectural and motivating factors behind large scale banking legacy IT. In addition, the paper will describe and compare different types of migration approaches and develop a quantitative cost model that will allow organizations to make optimal investment choices during the modernization process. Finally, this paper will include decision-making tools that will assist organizations in making strategic decisions during the modernization process and present best practices for implementing legacy modernization e.g., DevOps/FinOps, testing, data migration, and compliance.

2. Background and Related Work

Legacy systems are vital to businesses, however they are often old-fashioned, written in an older language, poorly documented and have inflexible architectures.[2] Many organizations have legacy systems that account for the majority

* Corresponding author: Ashmitha Nagraj

of their IT expenditures; Approximately seventy-five percent of bank IT spending is used to maintain legacy systems.[1] Legacy systems hinder an organization's ability to innovate and move quickly.[3]

In addition to outlining traditional methods for modernizing legacy systems, recent research has focused on using cloud computing and microservices. Breaking up monolithic applications into multiple small, loosely coupled services can result in increased scalability and improved team velocity [6][7] The concepts mentioned above are echoed throughout industry publications. Migrating application workloads away from legacy on-premises systems into the cloud could lead to "better use of resources" due to economies of scale.[8] While regulators are warning that legacy system inadequacy creates operational risk,[9] many organizations within the financial service sector are using cloud computing for cost reduction and as a way to provide greater resiliency.[10][11] However, all uses of cloud computing must be governed with a strong set of controls.

Building upon this body of knowledge, this study will focus on large banking environments that integrate both the economic and regulatory perspectives as well as the architectural perspective of legacy systems.

2.1. Legacy System Characteristics at Enterprise Scale

Large financial enterprises typically have a mix of legacy core platforms and newer services. Common elements include:

- **Monolithic Core Systems:** Most of the mainframe/old server-based applications for core banking, payments, and trading were built using older programming languages like COBOL, PL/I, etc. Monolithic applications encapsulate entire business domains within a single code base. While monolithic applications can provide stable and reliable performance during continuous, steady workloads, they cannot be scaled and/or modified in an easy manner.
- **Centralized Databases:** Legacy applications utilize a few very large databases, which have complex data schema's that are tightly coupled. When one application is updated, it typically causes some form of significant data migration.
- **High-Availability Infrastructure:** Due to their dependence upon on-premises data centers and use of redundant hardware, operations teams have created high-availability infrastructure. In addition, many organizations still process overnight batches of data. Even though outages of core systems are extremely rare, they can be extremely expensive when they do occur.
- **Vendor and Package Lock-In:** Most of the core banking and trading software is developed by only a handful of vendors. Due to heavy customization of these packages, many organizations find themselves locked into a specific vendor roadmap for future updates.
- **Critical Skill Shortages:** There are skill shortages in legacy platforms (mainframes, COBOL), since the experts in those areas are nearing retirement age and/or younger engineers are focusing on developing cloud native skills. As a result, there is pressure on organizations to modernize their systems so that they will be able to leverage existing engineering talent.

The above features indicate the need for a high level of continued support for these products. It is common for "the increasing costs of maintaining" older technology, and the inability to evolve an architecture to easily adapt to changing business needs [2]. Practically speaking, many large organizations have hundreds of applications. Therefore, the scale of modernization must enable gradual separation of legacy components instead of replacing them in whole.

Figure 1 Incremental modernization architecture: legacy core systems coexist with new microservices via an integration layer, supporting modern digital channels

An example of how an organization would implement an incremental approach to modernizing their system is illustrated in Figure 1. The legacy systems are operational on the left side of the figure and provide access to new domain services through an integration layer to modern digital channels on the right side. This "strangler" approach will allow both legacy and new pieces of the system to operate together throughout the transitional process.

3. Drivers for Modernization

Multiple factors push enterprise banks to modernize legacy IT. The most significant trends affecting banking today are digital competitiveness, cost efficiency, regulation and risk, and workforce and skill development; all four are impacted by the increasing use of legacy systems.

- **Legacy Systems Impact Digital Competitiveness:** The ability of banks to rapidly implement innovative products and services using newer architectures is limited when those same banks have many legacy systems to support. Financial institutions with modern architecture can respond faster than traditional ones because their systems were built from the ground up to accommodate the needs of a digital world. One of the greatest constraints on the financial industry's ability to optimize customer service is its reliance on legacy systems.[12] As such, incumbent banks will need to create the flexibility to compete and innovate through the adoption of modern platforms.
- **Legacy Systems Impact Cost Efficiency:** Most banks' IT budgets are spent on "run the bank" activities, i.e., maintaining existing systems rather than developing new ones. This spending has been estimated at as much as 70-80 percent of banks' IT expenses [1]. Banks that rationally apply software applications can realize significant reductions in costs [13]. Additionally, banks can reduce their capital expenses and the amount of work needed to operate their IT systems using cloud-based architecture and/or managed services.
- **Legacy Systems Impact Regulation and Risk:** There are growing concerns regarding the ability of banks to demonstrate to regulators that their IT systems can provide adequate data governance and resiliency. Legacy systems are often unable to provide the level of detail required to enable regulators to monitor the effectiveness of a bank's operational resiliency efforts. Banks with outdated IT systems may be "insufficiently adaptable" to address emerging risks [9]. Cloud-based and micro-services-based architectures are able to provide improved fail-over, disaster recovery, and real-time monitoring capabilities which help to ensure compliance with regulations related to operational resiliency.
- **Legacy Systems Impact Workforce and Skill Development:** As COBOL and other legacy programming languages become less commonly used, it is becoming increasingly difficult for banks to find qualified developers who are proficient in these areas [4]. Further, there is a trend among developers to utilize cloud-native stacks and therefore banks that continue to rely on legacy platforms are facing increasing challenges in terms of attracting and retaining skilled employees. By modernizing their technology, banks can ensure alignment with available technical skills and make it easier to recruit and train employees.

In summary, modernization provides solutions to both push factors, the high cost and risk associated with legacy systems and pull factors like the business agility and cost savings provided by cloud-based architectures. Therefore, modernization is no longer optional for financial firms, but rather an imperative: firms that do not modernize run the risk of being left behind in product development, operating efficiency and regulatory compliance.

4. Migration Strategies

Legacy applications can be migrated using different approaches:

- **Rehosting (Lift-and-Shift):** Application is migrated to cloud-based virtual machines or containers with limited to no changes to the existing application's source code. The cost and risk associated with rehosting are very low which enables rapid cloud on ramps, but there will remain a monolithic architecture in place. The benefits of being in the cloud will be relatively modest, while there may be slight improvements to management processes [5]. Simple migration of an enterprise workload from an on-premises environment to AWS IaaS resulted in a reduction in the total cost of ownership of ~37% over five years [14] yet the original monolithic architecture remained in place.
- **Re-platforming / Refactoring:** Targeted architectural modifications made to the existing monolithic application. While the effort associated with re-platforming/refactoring is medium, the risk to the organization remains moderate. This approach has been demonstrated to have potential to deliver some degree of scalability and maintenance savings. For example, converting a batch job to a serverless function or implementing a cloud-based caching strategy could potentially improve the performance of the application, but would require considerable time and effort to implement.
- **Re-architecture (Microservices**): The existing monolithic application is decomposed into multiple independent micro-services. The effort and risk associated with re-architecting an application as a collection of micro-services is extremely high, essentially a complete rewrite of the application, as many changes to both the data and the code of the application will need to occur. While the benefits of re-

architecting an application as a collection of micro-services are the highest, each micro-service can be independently developed, deployed, and scaled; each micro-service can be maintained by a small team, this level of benefit cannot be realized until the re-architecture process is complete.

- **Strangler Fig (Incremental):** A façade is wrapped around the legacy system, and the functionality of the legacy system is incrementally replaced with new functionality. New features are implemented as separate services that call the legacy system and "strangle" the legacy logic out of the primary execution path. This method of replacing legacy functionality with new functionality allows the effort to be spread across multiple iterations and provides a lower risk to the organization than a single, large re-architecture project [15]. As each new feature is successfully implemented, business value is accrued, and rolling back to previous versions of the application is easier when issues arise during development.
- **Replace/Retire:** The legacy system is eliminated via replacement with a SaaS or COTS solution, or it is retired altogether. When a legacy system is replaced or retired, all legacy code is removed from the organization. This approach can be cost-effective for non-differentiating functionality but typically involves a great deal of work to migrate data and integrate with other systems [16]. In banking, compliance or reporting systems may be swapped out for third-party services, but core transaction systems are rarely completely replaced.

Table 1 Compares these strategies qualitatively.

Strategy	Effort	Risk	Cost	Comments
Rehosting (Lift and Shift)	Low	Low–Moderate	Moderate	Fast migration with minimal code change. Limited cloud optimization benefits.
Refactoring / Re-platforming	Medium	Moderate	Moderate	Selective modernization such as containerization. Balanced effort and benefit.
Re-architect (Microservices)	High	High	High	Full redesign delivering maximum agility, scalability, and resilience.
Strangler (Incremental)	Medium	Lower	Phased	Facade-driven gradual replacement that spreads risk and investment.
Replace / Retire	Variable	Moderate	Variable	Best for commodity or low-value systems; avoids ongoing legacy maintenance.

For example, lift-and-shift has low effort but only moderate cost benefit, whereas re-architecting has high effort and risk but can yield high agility. Strategic applications often use strangler or refactoring, while commodity apps suit replacement. The appropriate strategy depends on the application's role. Core "systems of record" usually undergo incremental transformation, while non-critical or customer-facing services can be built anew or replaced.

5. Architecture and Design Patterns

Certain architectural patterns facilitate migration:

- **Domain-Driven Decomposition:** Break down application into business domains and map each domain into its own service. In this way each domain becomes its own "micro-service". Micro-services have their own database and therefore do not share data across domains. Therefore, the coupling of one function to another is reduced.
- **API Gateway / Integration Bus:** Create an interface that provides a single-entry point into the application. This interface directs incoming requests to either the legacy systems or the newly developed micro-services. In this way, an "anti-corruption layer" can be created that separates the legacy system from newer systems and allows both to run simultaneously.
- **Event-Driven Communication:** Utilize an Event Driven Architecture (EDA), such as using a message bus like Kafka, to allow components to communicate without being coupled together. The legacy system can send events based on business changes and then have new systems to listen to those events. As new systems take over as the source of truth for specific data elements, they will send updated events, and legacy system can subscribe to receive those updates to support a "data migration".

- **Containerization:** Allow each of the services to be packaged as a container. Containers provide portability. Even legacy systems can be packaged as containers and deployed on Cloud Platforms. For Example: a COBOL application in a Linux container.
- **CI/CD Pipelines:** Implement automated build/test/deployment pipelines for all services. Automated regression testing verifies that the output of new services matches what legacy services produce. A mature pipeline supports rapid development through continuous deployment [16].
- **Observability:** Instrument all the services with logging, metrics and distributed tracing. Legacy systems typically do not include instrumentation, so instrumenting them or adding monitoring capabilities is necessary [17]. Distributed tracing provides visibility into call flows from legacy systems to new systems. Observability supports detecting errors in the new services quickly and ensures that new services meet Service Level Agreements.

These patterns provide with a structured approach to ensuring that modernizing the application does not introduce additional complexity but instead makes overall application simpler. Using an API Gateway to manage traffic to services means that not to hard-code calls to new services in every client. Containerization and CI/CD enable faster and less error prone deployments, a common theme in DevOps literature [6].

6. Decision-Support: Tables and Frameworks

To aid portfolio decisions, additional artifacts are proposed:

Application Rationalization Framework: Classify applications by business value and technical state. For example, strategic differentiators are worth heavy investment to modernize, while commodity or redundant systems can be retired or moved to SaaS [13].

Table 2 Outlines typical categories and actions. This aligns with official guidance that rationalization "eliminates redundancies, lowers costs, and maximizes efficiency

Category	Description	Recommended Action
Strategic Differentiator	Systems central to competitive advantage and revenue generation (e.g., proprietary pricing models, payment engines, risk platforms)	Incremental modernization using refactoring or microservices. Continue investing in innovation and feature development.
Key Operational	Essential systems that support daily operations but provide limited differentiation (e.g., core ledger, reporting platforms)	Re-platform to managed/cloud services or partially rewrite to reduce operational overhead.
Commodity	Standard business capabilities (e.g., CRM, HR systems, collaboration tools)	Replace with SaaS or commercial off-the-shelf solutions. Minimize custom development.
Redundant / Legacy	Obsolete, duplicated, or low-value systems with limited business justification	Retire, archive, or consolidate. Avoid further investment and maintenance costs.

- **Migration Decision Matrix:** Table I already provided a qualitative comparison of strategies. Together, these tables help stakeholders score and prioritize the application portfolio, balancing business urgency against technical cost.

7. Implementation Considerations

Modernization of success hinges on operational practices:

- **Automated Development through DevOps**: Implement automated development pipelines that include continuous integration / deployment (CI/CD) for each change. Implement automation for building, testing and deploying. High frequency testing via continuous integration will assist in ensuring newly developed microservices along with modified legacy components are tested early and often. Banks using high-performing teams employing DevOps have proven to be able to deploy faster while achieving lower failure rates. Banks can also automate many of their repeatable processes which will assist banks in reducing both the number of errors they experience and the amount of time it takes them to perform these functions.

- **Fintech Operations (FinOps) / Cloud Spend Governance**: Provide an organization wide view into cloud spend. Have a cross functional group provide governance around how cloud spend is tracked by application and budgeted for. Monitors spend on applications and/or departments. As an example, banks should measure costs by transactions or cost by environment and automatically scale down resources when not being used [18]. The FinOps team should produce reports on cloud spend KPIs on a regular basis.
- **Testing and Rollback:** Test both old and new versions in parallel to ensure that each works well. Consider using "feature flags" or "API versioning" to shift users from the old to the new version in stages. Make sure that each version of rollout has a rollback plan: for any mission-critical service, there should be an option to turn off or roll back a new version of a component if there is something seriously wrong with it [17]. To find any small discrepancies, use automated regression testing and synthetic transactions.
- **Data Migration:** Plan carefully how to move with the data. There are two general strategies: all at once, or in phases by domain or date range. After migrating the data, use data comparison tools to verify that the outputs from the legacy system and the new system match. As part of the migration process, perform data cleanup to improve the quality of the data in the new system [16].
- **Governance and Skills:** Organize a cross-functional team to oversee modernization. Obtain senior leadership buy-in and budget allocation for a multi-year effort. Educate new teams in cloud platforms and container orchestration. The legacy development staff should educate the new teams on the knowledge they have acquired. Change management practices must be adopted in the new environment as they were in the legacy environment.

8. Risk Management and Compliance

Banks must manage risks introduced by modernization:

- **Data Governance:** Data about customers and transactions on the Cloud must adhere to laws regarding Privacy & Residency. To protect customer data, utilize Encrypted Storage & Strict Identity Access Management Policies. Any new Services integrated into the Bank's Identity Management System must also have auditing capabilities. Any access must be logged. Non-compliance with these laws could result in fines.
- **Auditability:** Legacy Systems may not have complete audit trails. New Systems developed as part of this modernization must generate complete logs and audit records. For example, if a Transaction Detail was modified by a Service, then that Event must be logged and linked back to the original Record. Auditors will verify that all aspects of the End-to-End Flow can be audited.
- **Operational Risk:** While the Bank seeks to modernize its processes, it must NOT introduce additional operational risks. Change Management must continue to follow a Documented Approval Process for all Code Changes. Wherever practically automate Policy Checks and clearly define Ownership of each Service. If a new Micro-Service fails, there MUST be a well-tested Failback Plan.
- **Vendor Risk:** As part of the supply chain the bank will now utilize Third-Party vendors. The Bank must perform due diligence on the Security and Resilience of those Third-Party vendors. Contracts with Third-Party vendors should include Service Level Agreements (SLA) for availability and incident response plans.
- **Resiliency Testing:** Banks are often required to demonstrate their ability to withstand various types of disruptions to their supervisory bodies. Conduct Chaos Testing and regularly conduct disaster recovery drills in the cloud. Establish backup and restore procedures. Critical systems must have Cross-Zone/Cross-Region deployments so that the failure of one instance does not disrupt service.

While the risk and compliance requirements for banks are nothing new, the way they will be implemented using a modern stack of technology will require new approaches to how the bank implements them. For example, while an Agile Pipeline may allow the bank to automatically enforce Code Quality and Security Checks, there may be instances where modernization will make compliance easier. Regardless, all changes made to the bank's technology environment must be reviewed by both the bank's risk team and audit team.

9. Discussion

Modernizing legacy systems at enterprise level is a difficult balance in terms of the potential risks, the total cost of ownership and the value delivered over time. The results of the study suggest that the most likely approach to achieving this balance is through a phased or incremental strategy, to allow a bank to remove its legacy components incrementally, so that new functionality can be made available to customers and users early in the process while continuing to run the legacy system for those applications which are critical to the bank's operations. However, banks need to beware of the trap of "false economies", where the only aspect of the application refactored is the visible part and there remain many

other aspects of the application which still depend on the old system and therefore "shadow IT" can continue to exist and reduce the benefits of the modernized system.

The cost model developed in this paper shows that modernization programs typically involve large upfront investments which are followed by savings over time. For example, to achieve success, organizations need to be able to maintain their momentum throughout the entire program and not stop once they have achieved some "lift and shift" in the short term; otherwise, they will not recover the costs of the program. The recommendation is using portfolio rationalization to obtain the first set of savings and then using these savings to fund additional levels of modernization within the organization. There are two limitations to the current research. First, because of the constraints imposed by publication scope, no proprietary financial data was provided from real-world organizations. Second, in addition to costs, modernization also has intangible advantages which may be important but difficult to quantify. Future studies could create multi-criteria decision-making models that take advantage of these types of advantages in addition to economic ones [19].

From an organizational perspective, the modernization program requires effective governance. The paper also recommends establishing a permanent modernization steering committee which would provide metrics to measure the program. These recommendations align with the best practices identified by researchers in DevOps who emphasized the importance of metrics and sponsorship by senior management to ensure that the program receives sufficient resources to succeed. In general, the modernization program should be considered as a multi-year program, rather than a single project. It should begin with "low hanging fruit" to gain confidence in the ability to execute the program. The organization should establish clear link to the business strategy and not simply treat it as another IT project.

Recommendations

- Establish a dedicated modernization office combining business, IT, risk, and finance leadership to govern the program.
- Integrate compliance and risk teams into planning, using an agile, metrics-driven approach to track progress and value delivery.
- Maintain strong executive sponsorship, setting clear success metrics and monitoring them to ensure the program meets its objectives.
- These steps, combined with the architectural and cost analyses above, prepare an institution to modernize its legacy estate effectively.

10. Conclusion

Modernizing large-scale legacy banking systems is imperative for all the large financial institutions out there. The paper has provided both a theoretical framework and practical advice on how to do this. Some key takeaways from this paper: Legacy banking systems are reliable but limit the ability of the organization to be innovative and very expensive [1][12]. A phased modernization plan will help manage risk. The paper has also developed a cost model which ties strategy selections to actual budgetary impacts so that organizations can make data driven decisions. Lastly, successful implementation, disciplined finance operations, disciplined Testing and compliance are important to achieving success.

Banks should:

- Rationalize their Application Portfolio to Eliminate Redundancy.
- Implement a Domain Driven Incremental Modernization Strategy for Core Systems.
- Use the Cost Model to Evaluate Return on Investment.
- Build the Organizational Capabilities to sustain the new architecture.

Once these capabilities are in place, Enterprise Financial Institutions can convert their expensive legacy portfolios into Agile efficient platforms that support future growth.

Compliance with ethical standards

Disclosure of conflict of interest

No conflict of interest to be disclosed.

References

[1] Crotty, James & Horrocks, Ivan. (2016). Managing legacy system costs: A case study of a meta-assessment model to identify solutions in a large financial services company. Applied Computing and Informatics. 13. 10.1016/j.aci.2016.12.001.

[2] Bass, Len & Weber, Ingo & Zhu, Liming. (2015). DevOps: A Software Architect's Perspective.

[3] Newman, S. (2015) Building Microservices: Designing Fine-Grained Systems. O'Reilly Media.

[4] Dragoni, Nicola & Giallorenzo, Saverio & Lluch-Lafuente, Alberto & Mazzara, Manuel & Montesi, Fabrizio & Mustafin, Ruslan & Safina, Larisa. (2017). Microservices: yesterday, today, and tomorrow.

[5] Sneed, Harry & Verhoef, Chris. (2019). Re-implementing a Legacy System. Journal of Systems and Software. 155. 10.1016/j.jss.2019.05.012.

[6] Wolfart, Daniele & Assunção, Wesley & Silva, Ivonei & Domingos, Diogo & Schmeing, Ederson & Villaca, Guilherme & Paza, Diogo. (2021). Modernizing Legacy Systems with Microservices: A Roadmap. 149-159. 10.1145/3463274.3463334.

[7] P. Jamshidi, A. Ahmad and C. Pahl, "Cloud Migration Research: A Systematic Review," in *IEEE Transactions on Cloud Computing*, vol. 1, no. 2, pp. 142-157, July-December 2013, doi: 10.1109/TCC.2013.10

[8] Richardson, C. (2018) Microservices Patterns: With Examples in Java. Manning Publications.

[9] Nicola Dragoni, Saverio Giallorenzo, Alberto Lluch Lafuente, Manuel Mazzara, Fabrizio Montesi, Ruslan Mustafin, and Larisa Safina. Microservices: yesterday, today, and tomorrow. arXiv preprint arXiv:1606.04036, 2016

[10] Strauch, Steve & Andrikopoulos, Vasilios & Karastoyanova, Dimka & Leymann, Frank & Nachev, Nikolay & Staebler, Albrecht. (2014). Migrating Enterprise Applications to the Cloud: Methodology and Evaluation. International Journal of Big Data Intelligence. 1. 10.1504/IJBDI.2014.066319.

[11] Wang Y, Kadiyala H, and Rubin J Promises and challenges of microservices: an exploratory study *Empir. Softw. Eng.* 2021 26 4 1-44

[12] Althani, Bashair & Khaddaj, Souheil. (2017). Systematic Review of Legacy System Migration. 154-157. 10.1109/DCABES.2017.41.

[13] Krause-Glau, Alexander & Zirkelbach, Christian & Hasselbring, Wilhelm & Lenga, Stephan & Kroger, Dan. (2020). Microservice Decomposition via Static and Dynamic Analysis of the Monolith. 9-16. 10.1109/ICSA-C50368.2020.00011.

[14] Khajeh-Hosseini, Ali & Greenwood, David & Sommerville, Ian. (2010). Cloud Migration: A Case Study of Migrating an Enterprise IT System to IaaS. Proceedings - 2010 IEEE 3rd International Conference on Cloud Computing, CLOUD 2010. 10.1109/CLOUD.2010.37.

[15] Hayretci, Hasan & Aydemir, Fatma. (2021). A Multi Case Study on Legacy System Migration in the Banking Industry. 10.1007/978-3-030-79382-1_32.

[16] Auer, Florian & Felderer, Michael & Lenarduzzi, Valentina. (2018). Towards Defining a Microservice Migration Framework. 10.1145/3234152.3234197.

[17] Malherbe, Magali & Simon, Fanny. (2021). The Business of Platforms: Strategy in the Age of Digital Competition, Innovation, and Power, Michael A. Cusumano, Annabelle Gawer, David B. Yoffie. HarperCollins, 2019, 320p.. Management international. 25. 240. 10.7202/1088148ar.

[18] Shahin, Mojtaba & Ali Babar, Muhammad & Zhu, Liming. (2017). Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices. IEEE Access. PP. 10.1109/ACCESS.2017.2685629.

[19] Fritzsch, Jonas & Bogner, Justus & Zimmermann, Alfred & Wagner, Stefan. (2019). From Monolith to Microservices: A Classification of Refactoring Approaches: First International Workshop, DEVOPS 2018, Chateau de Villebrumier, France, March 5-6, 2018, Revised Selected Papers. 10.1007/978-3-030-06019-0_10.