COMPARATIVE ANALYSIS OF CASE STUDIES ON LEGACY SYSTEM MIGRATION IN BANKING INDUSTRY

by

H. Emre Hayretci

M.S., Computer Engineering, Boğaziçi University, 2021

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computer Engineering

Boğaziçi University

2021

ABSTRACT

Comparative Analysis of Case Studies on Legacy System Migration in Banking Industry

Advances in technology, changing customer requirements, and pressure from business goals are the main drivers for innovation in the banking industry. Legacy architectures with monolithic structures prevent banks from implementing new generation banking models. To stay competitive, banks migrate to modular and scalable architectures. This migration has a significant impact on banks' technical and organizational infrastructures, so it is crucial to devise an end-to-end migration strategy and plan the transformation. This thesis reports our observations on the legacy system migration of three large retail banks between 2014 and 2020, focusing on the evaluation and prioritization criteria for their application portfolio to be migrated. We compare and contrast the motivations, migration strategies, and migration prioritization methods and discuss key takeaways from these high scale migration projects. The existing legacy architecture, most urging issues of the current operations and industry trends at the time of the transformation program, define the target architecture for legacy migration. The culture and availability of strong governance structures such as architecture boards, solution architecture teams, and target architecture patterns play an essential role in how migration planning is conducted. Regardless of the planning approach selected, value-driven program management practice would provide the most optimized prioritization. A robust migration project management is required to ensure the continuity, progress, and quality of the transformation. As transformation programs migrate the applications to the new platform, migration evaluation and prioritization criteria are subject to change. Continuous alignment of these criteria with program objectives is essential to succeed in the application migration during legacy modernization programs.

ÖZET

Eski Sistem Geçişlerinin Bankacılık Endüstrisinde Karşılaştırmalı Vaka Analizi

Yaşanan teknolojik gelişmeler, müşteri ihtiyaçlarındaki sürekli değişim ve iş hedeflerinden doğan baskılar bankacılık endüstrisindeki inovasyonun temel itici gücü olmaktadır. Monolitik yapıdaki eski nesil mimariler bankaların yeni nesil iş modellerini uygulamalarının önünde engel oluşturmaktadır. Bankalar rekabetçi kalabilmek için daha modüler ve genişleyebilir mimarilere göç etmek durumundadırlar. Bu tür bir göç, bankaların sahip olduğu mevcut teknik ve organizasyonel yapıya ciddi bir etki yaratmaktadır, bu sebeple böyle bir dönüşüm için uçtan uca bir göç stratejisinin oluşturulması ve planlanması kaçınılmazdır. Bu tez 2014 ve 2020 yılları arasında eski nesil sistem mimarilerinden böyle bir dönüşüm yolculuğuna çıkmış olan üç büyük perakende bankanın taşınma süreclerini değerlendirme kriterlerini, taşınacak olan uygulama portföylerini nasıl önceliklendirdiklerine yönelik gözlemlerimizi karşılaştırmalı olarak raporlamaktadır. Çalışmamızda bankaların motivasyonları, göç stratejlerini ve göç önceliklendirme metodlarını karşılaştırıyor ve bu tür büyük çaplı dönüşüm projeleri için önemli olacak unsurları tartışıyoruz. Eski nesil teknoloji mimarileri, iş operasyonlarının en öncelikli sorunları ve dönüşüm programlarının başladığı dönemlerdeki teknoloji akımları yeni nesil mimarilerin şeçimini etkilemektedir. Kurum kültürü ve mimari karar otoriteleri, çözüm mimarisi ekipleri gibi kuvvetli yönetişim yapıları uygulama göçlerinin yapılış şekillerini belirlemektedir. Uygulama taşıma planları ne şekilde yapılırsa yapılsın, değer odaklı program yönetişim yapıları en etkin önceliklendirmelerin yapılmasını sağlayan metod olarak ortaya çıkmaktadır. Dönüşüm programları uygulamaları yeni mimariye taşıdıkça, taşıma ve önceliklendirme kriterleri dinamik olarak değişmektedir. Bu kriterlerin program motivasyonları ve hedefleri ile sürekli olarak hizalanması programlar açısından önemli bir başarı kriteri olarak çıkmaktadır.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS/ABBREVIATIONS

API	Application Programming Interface
ATM	Automatic Teller Machine
BI	Business Intelligence
BPM	Business Process Manager
CD	Continuous Development
CI	Continuous Integration
CIO	Chief Information Officer
CICS	Customer Information Control System
CTO	Chief Technology Officer
COTS	Commercial off-the-shelf
ESB	Enterprise Service Bus
IMS	Information Management System
MIPS	Million Instruction Per Second
KPI	Key Performance Indicator
IT	Information Technology
SaaS	Software as a Service
SOA	Service Oriented Architecture

1. INTRODUCTION

Rapidly changing customer expectations, advances in technology, and pressure from the business side force banks to constantly assess their current IT infrastructures and innovate. The customers require that banking systems seamlessly integrate into their daily activities. The technology provides tools for multi-device, anytime, anywhere access. The business demands lower costs for ever-high quality of service. All these drivers for change demand new models for banking infrastructures, and banks with legacy architecture struggle to implement these new models [1].

Legacy banking platforms are monolithic structures that are too complex. Applications for these platforms get too complicated over time to maintain. Such applications require development teams to split by functions - user interface, application, middleware, database, etc. Another disadvantage of these applications is their fragility, for a single bug quickly brings the entire application down. To overcome these disadvantages, banks replace their legacy systems with inherently resilient scalable modular architectures following modern design principles [2].

The success of the migration from the legacy system directly affects the competitiveness of a bank; therefore it requires careful planning and execution. This thesis reports our observations on the legacy system migration of three large retail banks between 2014 and 2020.

In this comparative study, we examined the following research questions:

R1: Which are the *key motivations* for banks to start their transforming from legacy systems towards next generation architectures?
R2: What is the *strategy* for application migration?
R3: How are the migrations of individual applications *prioritized*?
R4: How to *measure success* for the transformation programs?

This thesis is structured as follows. Chapter 2 provides background information on legacy architectures and migration to modern architectures. Chapter 3 outline our research method. Chapter 4 explains details on the migration of three retail banks and discusses the similarities and differences in these cases. Chapter 5 includes the related work and Chapter 6 concludes the thesis.

2. BACKGROUND

This section provides background information on legacy systems in the banking industry and an overview of legacy migration. The legacy system in banking is typically called core banking system as a back-end system that processes daily banking transactions and posts updates to accounts and other financial records. Core banking functions typically include transaction accounts, loans, mortgages, and payments. Banks make these services available across multiple channels like automated teller machines, Internet banking, mobile banking, and branches. Core banking systems or legacy systems for a bank underpin nearly every major banking process.

2.1. Legacy Architectures in Banking Industry

Considering competition from non-banks such as telecommunication service providers, online retailers, or sole payment providers, cost pressures, and increased product environments, banks need to continuously evolve their operating models. They need to increase their investments in new core banking systems to overcome constraints in their existing environments and upgrade their products and services. Therefore, banks are searching for adopting an agile and effective way to clarify their architecture to respond more effectively to continually shifting market conditions. However, this search for evolution is not unusual. There has always been a continuous change and transformation need in banking architecture since the banking industry existed like all other industries.

The first core banking system appeared in the 1970s [3, 4]. Legacy core banking systems became product–centric structures that have separations for each major product line and developed in silos within a decade. Core banking with product–centric structures is a banking model with the objective of acquiring customers through products. An example of such separation is a product specialization on payment processes for the banking ecosystem.

In 1990s the first Internet banking applications appeared. Also, the product–centric model evolved to the customer–centric model with multi-channel integration and service-oriented architecture [5]. This evolution facilitated to increase of customer–centricity in distribution way in the banking industry, which is called segment–centric architecture. Core banking application with segment–centric architecture groups customers in segments and serves them with shared functions across products that create synergies, cost efficiency, and increased sales opportunities.

Banks traditionally have embarked on industrialization programs to move from product–centric models to manufacturing–centric models, adopting the automotive industry's practices. The main objective is to have one production house/kitchen that provides requirements for many restaurants. An example is providing financial solutions to an ecosystem. With the everyday use of internet access, the banking industry has witnessed an increase in the number of channels with multi-channel platforms facilitating multi-channel convergence. As a result, online banking became mainstream in the industry, and the structure evolved to customer–centricity.

New technologies and innovations such as big data analytics and cloud-based platforms brought new perspectives to the banking industry to reach their customers and respond quickly to their requests during the first decade of the 2000s. In the early 2010s, banks realized they are required to be more than banks for their customers via mobile banking burst and transformed rapidly.

Fig. 2.1 depicts the legacy information technology (IT) architecture of a typical bank installed after 2000s. It is composed of an online application layer that provides self-servicing functions to its customers running on top of a core banking system that transacts and store all banking transactions. The core system is monolithic architecture. While all the online transactions run over the core, separate siloed and mostly batch-based business intelligence (BI) systems are evolved as a side system for serving the bank's reporting and analytic requirements.

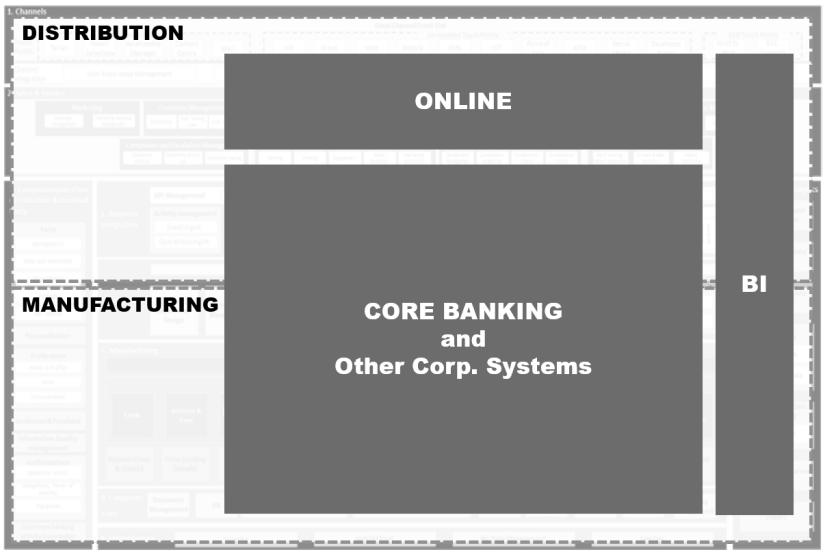

Figure 2.1. A typical IT architecture of banks after 2000s

The legacy core banking systems have a monolithic architecture comprised of tightly coupled components relying on shared resources such as a single code base, databases, and servers. While these systems provided a robust and secure architecture for systems transactions, they hurt modularity, scalability, and flexibility [4]. To meet the increasing and changing needs of the market, monolithic core banking systems have been modified excessively and deviated from the intended architecture over time. Insufficient documentation, risk of a single point of failure, inconvenience of deployment, complexity, and unscalability have prevented banks from releasing new services and new features in their existing services in a short time and triggered migration from the legacy systems.

2.2. Overview of Legacy Migration

Banks typically start to consider on migrating legacy applications to new technology architectures, either to private cloud or cloud-ready on-premises platforms, after the identification of target reference architecture. Shifting to a next-generation banking architecture starts with identification of required architectural capabilities and building the infrastructure and platform to enable these capabilities. After providing the framework and environment to develop applications in target architecture, banks

initiate transformation program for the migration of legacy to the new.

Based on the observations on several transformation programs executed by banks, the migration process commonly encompasses four phases as seen in Fig 2.2. Initially banks determine the migration strategy based on their target reference architecture, the complexity of the legacy architectures, and co-existence of legacy and new during the transition stage. Secondly, banks determine the approach to develop migration plans. Governance structures responsible for architecture influences the way this planning activity is conducted. In the next phase, current legacy applications are assessed, the inter-dependencies are identified, and domains/ applications are prioritized according to the enterprises' objectives. This evaluation study leads banks to plan on how to migrate each of the domains/ applications in their portfolio and in which order. This is a continuous activity that should take place throughout the transformation. The final phase, migration execution, could be generalized as plan the migration of domain/ application, design the solution blueprint, implement the application based on new solution standards, and migrate.

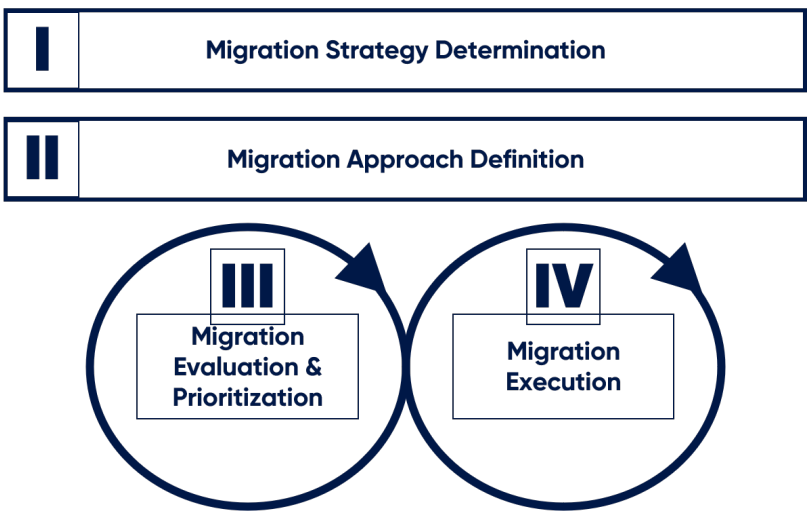

Figure 2.2. Overview of Legacy Migration

In the following sections, we conduct an in-depth analysis of the application migration planning approach by outlining the considerations of different strategies on application migration and depicting the evaluation process on how to prioritize the

migration.

2.3. Migration Strategies

Every migration journey needs to define a clear migration strategy for every application that is based on a holistic application analysis. This analysis should consider not only technical aspects but also business, organization, security, and compliance. The selected strategy fundamentally affects the expected migration effort, the potential benefits, and possible long term cost savings of the new operations model. Each strategy indicates a clear outcome for a transformed application, but not necessarily the actual migration steps. Commonly used migration strategies of next-generation architectures and how they typically conducted are as follows;

- *Retain*: Application with modern applications or newly upgraded software packages are generally not prioritized during the application migration planning. Banks only required to migrate what makes sense for the business; yet, as the gravity of the application portfolio changes from on-premises to the cloud, there will be fewer reasons to retain.
  - Remediate the application by addressing pain points
  - Implement CI/CD (Continuous Integration/Continuous Development) capabilities
  - Run application(s) to end of life where they have a finite lifespan to save wasted cost to achieve
  - Move Dev/Test environments to off-host/cloud infrastructure where possible to reduce cost and increase agility
- *Replace*: Moving to an commercial off-the-shelf product. More suited to applications that provide non-differentiating functionality
  - Identify a managed service/ application that can provide required functionality
  - Implementation of a COTS or SaaS package to replace existing, custom-built mainframe application(s).

  - Extract and migrate data to new system
- *Re-Host*: Also known as "lift and shift". In a large legacy migration scenario where the organization is looking to scale its migration quickly to meet a business case, it is applicable for most applications to be re-hosted. More suited to more modern architectures than mainframe.
  - Moving an application as-is to a cheaper location, utilizing the same hardware platform, making no changes to the application code.
  - Utilize 3rd party mainframe hosting to externally host applications and data.
  - Delivers some cost benefits while avoiding the risk of changing the programming language
- *Re-Platform*: To port an application from one technical platform to another by mostly or fully reusing the code asset in the target platform.
  - Move an application without changing the programming language to another platform / Operating System
  - Use a solution to re-compile COBOL to run on off-host hardware.
  - Use solutions to migrate application binaries (e.g. where no source code)
  - Allows running legacy applications in the Cloud – public, private and hybrid
  - Delivers cost benefits while avoiding the risk of changing language
- *Re-Factor*: Using migration toolkits to transform from legacy to modern programming languages code.
  - Fully automated tooling yields partially refactored applications at great speed/low cost.
  - Semi automated tooling yields fully refactored application at greater duration/cost
  - Migrate the risk by removing the dependencies on legacy skills.
- *Re-Architect*: Transforming to a modular and scalable architecture with cloud-native features and adopting new methods like agile and DevOps. Suitable if the Business is re-imagined.
  - Rewrite the application using Cloud-based architectures based on newly developed requirements, domain driven design
  - Resultant system is not based on current system capabilities, thereby en-

abling not only technology modernization, but the modernization of outdated business processes
- Enables the highest usage of cloud native services, tools, development patterns, and future innovation.

At the end of the transformation, an assessment on each functional domain, who owns lines of application, is necessary to figure out the applications in the IT portfolio that are no longer useful and can simply be turned off. These savings can boost the business case, direct your team's scarce attention to the things that people use, and lessen the surface area you have to secure.

These strategies build upon the amazon's cloud migration strategies. Banks typically consider multiple strategies while building their migration plans.

2.4. Migration Planning Approaches

There are numerous alternatives for migrating applications to target reference architecture; best fitting alternative varies between banks according to enterprise routines, organizational culture, and banking architecture.

Centralized Approach. The nature of centralized approach considers business priorities and requests on the pipeline regarding current application inventory in terms of interface, data model, integration etc. This approach requires comprehensive and complete planning of migration at the beginning of the transformation lifecycle. The migration planning activities are led by a centralized team, commonly IT architecture team. The centralized team has a leading role to coordinate business and technical stakeholders and drive the execution of the activities. In planning phase, all legacy applications and domains are evaluated, target application mapping is developed, dependencies are overseen closely, and co-existence scenarios are considered. The migration plan is prepared based on prioritization that relies on expert judgement. The central-

ized team creates guidelines and documentation that will be followed throughout the transformation.

The main advantage of the centralized approach is that the time required to complete evaluation process is relatively less compared to de-centralized approaches as they require a series of workshops. Overall plan and prioritization will be ready much earlier when decided by the architecture team. In addition, less resources are needed in terms of effort and dedication.

Applying centralized approach have also negative aspects. Generally, domain specific needs could not be captured on planning stage and may lead to misidentification of common components. As the prioritization is solely based on expert judgement and previous experience, transformation programs tend to lean on technical perspective and fall behind the realization of business value.

Domain–driven Approach. The domain driven approach indoctrinates a migration based on target application architecture model and a logical ranking according to component usage, interdependencies, scalability etc. Each domain defines its migration plan in cooperation with architecture team and a solution is created for each domain separately. Migration tasks and activities are planned at domain level for each application following individual workshops with each domain. This approach obviously requires a series of workshops to discuss and define the domain specific applications with participation of representatives from a single domain. Both evaluation and execution of different domains take place separately and the teams work in parallel.

The main advantage of the domain driven approach is that domain specific requirements can be identified at early stage. As the domains involve in the transformation from an early stage, it is easier to cascade the know-how to the project teams. In addition, a more detailed domain implementation plans are created with this approach.

As domain driven approach gives architectural compliance responsibility to the domain teams, the solution designed may lead to inconsistence with technology architecture principles, and outputs may differentiate among domains. A misalignment among domains may also occur due to different level of knowledge on new architectural components. Resource limitations in terms of domain support is another negative aspect of the approach that causes additional effort of IT architecture team.

Hybrid Approach. In this approach, evaluation and prioritization is selected based on the domain specific requirements and dependencies. Migration tasks, activities and common components are defined and planned by a series of workshops with participation of representatives from every domain. Even the execution is domain driven, yet a centralized governance team ensures the compliance to architectural standards and assists the coordination. The identified common components include processes, products (e.g. product factory, pricing, accounts, commission etc.), and both internal and external services.

Combining the two different approaches have various benefits. Firstly, company-wide defined common components reduce replicated functionalities and avoid double effort. Also, domains do not confront same challenges as there will be unified compliance to functional and technical design principles. It provides ability to define priorities and plan in line with transformation goals. With the hybrid approach, it is easier to identify and manage dependencies among applications. Collective approach also has drawbacks. It may not meet every domain specific need and there is still a requirement for additional effort to clarify details at domain level as there is limited responsiveness to product level requirements. Establishing team and workshop execution may be difficult to organize and require additional time. Domain solution blueprint definition may be started after the identification of common components and prioritization of domains are possible to be deferred after common components' definition.

2.5. Application Migration Prioritization

The migration prioritization is based on assessing business impacts, technological dependencies, regulatory requirements, and value trapped by technical debt. A holistic evaluation of every aspect is required for the right prioritization.

- Reflecting on new business models in technology architecture
- Rapidly changing customers' needs and continuous demands
- Respond quickly to new entrants both in terms of defense and collaboration
- Compliance with regulatory requirements with challenging deadlines
- Unlocking the trapped value with the elimination of technical debt
- Readiness of technological capabilities and technical dependencies to common services

The prioritization activity is performed by a central authority, mainly a team composed of enterprise and solution architects, at the beginning of the transformation program for each application. The enterprise acknowledges this prioritization assessment as the guideline to initiate the execution stage. However, in non-centralized approaches, the prioritization assessment is executed by each domain per application. Architects only provide a checklist, a content to conduct a survey, to support domains. The main difference between the domain-driven and hybrid approach is on the central governing body, which oversees technical dependencies and schedules the execution plan.

Applications in the portfolio need to be assessed, and a clear roadmap needs to be produced. This process is typically performed in four phases as seen Fig 2.3.

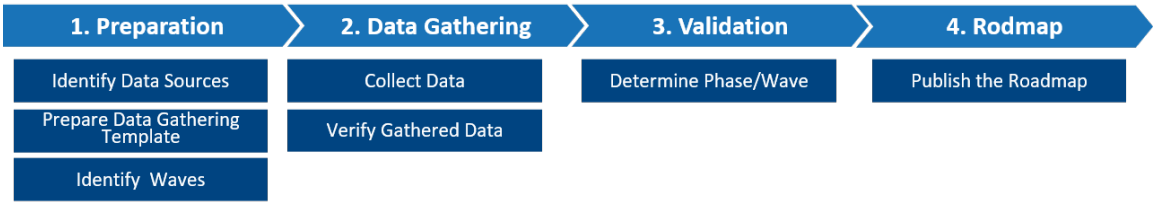

Figure 2.3. Application Migration Assessment Process

Preparation. In this phase, data sources to be used during prioritization are identified for each application. A template for data gathering template is prepared to collect data in a structured manner. Projects/Application lists, Service lists, functional models, application models, effort estimates are typical work products of this phase. Depending on the migration planning approach, enterprise architects, domain architects, technical architects are involved in this phase. For each application, an evaluation criterion is set. Business value, technical feasibility, inter-dependencies, application life cycle, and application priorities are typical criteria for identifying a score for each application. This score is calculated by giving weight to each application. The release period of the application is determined by the scope and duration of the application and the number of projects that can be started simultaneously. In this way, related applications are clustered, and the migration strategy for each application is identified. Application development duration is specified at a high level (i.e. short-term less than 3months, medium-term between 3 and 12 months, long term more than 12 months).

Data Gathering. Applications data are collected and verified in this phase. Domain architects provide the data through the templates. The work product of this phase is the list of applications and projects. Enterprise architects and technical architects support this process and verify the gathered data through cross-checks.

Validation. Enterprise architects determine the wave based on the data gathered for each project/application. Some applications/projects may span multiple waves. This phase also ensures that dependencies are handled properly. Domain Architects detect dependencies among the projects. Existing and potential dependencies are identified and challenged.

Roadmap. In the last phase, a roadmap is developed and presented to the committees to align all stakeholders. The roadmap represents the waves in which the

application is migrated, dependencies among the applications, and a high-level time plan.

A sample roadmap after conduction this prioritization may look like in Fig.2.4.

	Wave 1	Wave 2	Wave 3	Wave 4	Wave 5
OBJECTIVE	Core Applications & Quick Wins	High Business Value & Application Priority	Medium Business Value & Application Priority	Low Business Value & Application Priority	Remaining and Ongoing Applications
SELECTION CRITERIA					
Business Value	High	High	Medium	Low	All remaining
Technical Feasibility	Yes	Yes	No	No	All remaining
Inter-dependency	Provider	Consumer & Provider	All	All	All remaining
Application Lifecycle	New & High Age	New & High Age	Medium Age	Low Age	All remaining
Application Priority	MF CPU, Tech Upgrade	MF CPU, Tech Upgrade, Tech Obsolete	All	All	All remaining

Figure 2.4. Sample Roadmap after Application Prioritization

2.6. Application Migration Execution and Governance

After the strategy is set and prioritization is complete, banks start the projects aligned with the roadmap. This phase continues throughout the transformation program until the program is official closed. Several governance activities executed during the execution phase and success of migration heavily depends on successful execution of the governance processes.

Project Onboarding. Project onboarding is the preparation stage, where several planning activities are held before starting a wave. The governance office is the entity that coordinates activities among project teams and ensures that planned activities are executed appropriately. According to the wave plan, the governance office includes the assessed and sorted applications to the wave plan and consolidates the program backlog. Capacity planning and its requirements need to be completed based on a consolidated program backlog. Resource planning activities and deployment of the teams are carried out. Coordination of the necessary training before the deployment of the teams is also conducted at this stage.

Application Migration Kick-off. Each project is initiated by kick-off stage, presented by the project manager. Governance office ensures that all projects follow this process. Domains prepare project estimation and kick-off and evaluate the up to date resource requirements. The start and end date of the project is adjusted regarding all sort of dependencies. Domains clearly define the outputs of the project plan in order to show clear results. In the meanwhile governance office identifies possible risks and issues. They control the project plan and challenge the time plan feasibility, effort estimation, team structure and wave plan during this stage. They give guidance and ask domains to revisit parts of their plans if necessary until there is no blocker issues to give "go decision" to the project.

Design Compliance Management. During the solution development phase of each project, application architecture compliance is reviewed and ensured. This activity is typically conducted by an authority such as architecture board that is part of the program governance. Domain architects prepare the solution through solution blueprint document. Application architecture is documented as part of the solution blueprint Governance team tracks each project and ensure that this steps is executed appropriately. Common services of the overall architecture, dependencies among services and compliance to the overall architecture and principles are ensured through this process.

Dependency Management. The process for declaring, resolving, and monitoring the program level dependencies is required for program execution. Stray and Moe et al. [29] state that managing dependencies between teams and within teams is critical when running large-scale agile projects. Dikert et al. [6] emphasize the importance of dependency management for the success of large-scale agile transformation programs. This process starts by identifying dependencies, describing in detail, and clarifying the effect, and detecting related/owner parties. The process goes with the planning phase. The governance team records each dependency as milestones and gives a reference number for unique identification. It makes sure that dependency is linked to the rest

of the program and schedule so its' impact on them would be known. Dependency is typically linked to possible risks and issues. The team evaluates dependencies and prioritizes them based on impacts. After planning, an agreement is assured by contacting the associated teams. If there is no agreed solution for the dependency, it needs to be escalated to the upper committee based on escalation rules. During the escalation period, this dependency is accepted as a risk. Monitoring and control is the last phase of dependency management. The governance team regularly communicates with the teams if they are still on track to deliver the dependency on the agreed date. It applies the determined/planned series of dependency action(s) and manages the series of dependencies and their impact on the project. The Dependency Log needs to be regularly reviewed and updated by teams and reviewed in associated boards and meetings.

3. RESEARCH METHOD

We have used the case study methodology in our research. Case study methodology is used for several types software engineering research, as the objects of study are modern phenomena, which are challenging to study in isolation. There are many case study methodology handbooks available in social sciences [7], [8], [9] which literature also have been used in software engineering. Software engineering cases studies are different from social science case studies and also to some extend case studies in information systems. The subjects in software engineering case studies are 1) either private or public companies or units in these companies 2) project based rather than operations 3) the studied work is sophisticated engineering or architecture work conducted by highly educated people rather than routine work [10]. Our case study meets all these characteristics.

Our research has descriptive and exploratory characteristics, as explained by Runeson and Höst [10]. We first portray the situation for each case by collecting information about them. We used questionnaires and surveys for collecting the information. Chief information officers, enterprise architects, solution architects, and application domain leaders and software engineers of the banks are the key stakeholders of our research. We tried to find out what has happened during these transformation programs, sought insights, and generated ideas and hypotheses for new research. We have conducted 19 one to one meetings, out of which 8 were with C-Level executives. We have also conducted more than 260 hours of workshops (wshop) to understand the situation; stakeholders' key decisions and their relationship during the transformation program have been observed and analyzed. Program managers, enterprise architects, domain architects, and software engineers attended these workshops.

We have used two sets of questions to gather data for our case studies. First set of question focus the why, what and how aspects of the transformations. We have collected the data through one to one interviews and workshop where specific topics

Table 3.1. Interview Schedule

	BankA	**BankB**	**BankC**
CIO	2 x 1-to-1 and $\geq$ $4hrs.ofws$	2 x 1-to-1 and $\geq$ 8hrs of ws	$\geq$ 6hrs of ws
CTO	2 x 1-to-1 and $\geq$ hrs of ws	2 x 1-to-1 and $\geq$ 12 hrs of ws	N/A
Trans. Prog. Mgr	N/A	2 x 1-to-1 meetings	2 x 1-to-1 and $\geq$ 20 hrs of ws
Enterprise Architect	1 x 1-to-1 meeting	N/A	N/A
Lead Architect	N/A	$\geq$ 4 hrs of ws	$\geq$ 4 hrs of 1-to-1 meeting
Domain Architects	2 x 1-to-1 meeting	4 x 1-to-1 meeting	1 x 1-to-1 meeting
Software Engineers (SE)	6 SE, $\geq$ 40 hrs of ws	4 SE, $\geq$ 40 hrs of ws	10 SE, $\geq$ 40 hrs of ws
Total	7 x 1-to-1 and $\geq$ 40 hrs of ws	10 x 1-to-1 and $\geq$ 120 hrs of ws	3 x 1-to-1 and $\geq$ 100 hrs of ws

related to our interview questions were being discussed as part of their transformation program. Table 3.1 outline the summary of interviews and workshops conducted for gathering the inside we have compiled in our research. Our initial set of questions were comprehensive. Questions were addressing further details of the execution of the transformation program where we did not cover in the scope of this research. Answer to some of these questions reveals follow-up research areas of this study. This set of general questions were directed to the C-Level and program management level executives of the transformation program.

Interview Questions.

- Background of Interviewee (e.g., your position, your responsibilities, career in in the organization)
- Do you see a need for a change in the banking industry? Why?
- High level overview of the transformation (e.g., objective of the transformation, key metrics used for measuring success)
- Approach and methods used during the transformation (i.e. Principles, Strategies, delivery methods, Product design and development methods or practices)
- Design principles used before during shaping and execution of the program.
- Challenges and resolution (e.g., key challenges of the transformation, risk and issues and mitigation actions)
- Key achievements and successes and obstacles (e.g., benefits achieved, results delivered to date, issues of methods implemented)
- Communication and collaboration (e.g. division of work, inter-team communication and collaboration, collaboration with other departments, knowledge sharing, challenges in communication and collaboration
- Which are the next phases (e.g., plans for the next phase, what else need to be done or considered in the scope)
- Any general final comment (e.g., anything you would like to add)

It was necessary to understand the dynamics behind the decisions made during

the programs. In order to understand these dynamics, we have directed questions to software engineers and domain architects who are making decisions related to their products or domains. These questions are addressing mostly what and how aspects of the transformations.

Questions for Architects and Software Engineers.

- Who are you and what is your experience in Software Engineering?
- What was your role in the programme?
- How would you explain the overall programme?
- How do you describe your legacy architecture?
- How did you describe your to be architecture?
- How do you describe your banking model?
- Who are the sponsors and other stakeholders?
- What type of architects played role in the program? Who were they?
- What is the decision processes?
- How did you make agreements and decisions?
- Which are the architecture governance bodies? How did you select the members of these entities?
- How does the architecture requirements collected?
- Which key architectural decisions were made?
- How was architecture introduce to the organization?
- What type of documentations exists for the architecture? Which documents did you produce?
- Which are the key architecture decisions that impact the program most?
- How did you prioritize requirements? Who decide on the prioritization?
- How did software engineers utilize architecture?
- Were the change in the requirements that impact architecture? Which are they?
- How was the work coordinated in your domain/workstream?
- How did you prioritize work during the program? (e.g., in your team or domain)

- Did you encounter dependencies while performing your work? Which are they?
- How did you program manage dependencies? (samples?)
- Which is the role of governance in managing dependencies?
- Did you encounter challenges with managing dependencies? (examples?)
- What type of knowledge was important to share between teams in this project?
- What kinds of practices were used to share knowledge?
- How did you measure your progress? (samples?)
- Which are the key metrics you report? How did you report them?
- How did you follow the status of the overall programme?
- Did a team know what other teams were working on?
- Were you able to distribute workload among teams?

We believed that it is crucial to understand the existing enterprise architecture of the banks. We have conducted another set of interviews with software and infrastructure engineers from the banks' information technology departments. In these interviews, we explore the technical architecture of the underlying information technology systems, main processes of their IT operating model such as project portfolio planning, software design and development, configuration and release management, etc and how they organized to execute these processes. Answers to these questions helps us to draw the picture of their as-is state and helps us to evaluate the reasons for their decisions and choices.

4. THREE CASES OF LEGACY MIGRATION

In this section, first, we compare the demographics, motivations, and legacy architectures of each Bank. Then, we discuss the target architecture, migration strategies, migration planning approaches, and prioritization criteria identified and followed by each bank and the reasons that lead to the decision.

4.1. Description of Cases

We refer banks in our case study as BankA, BankB, and BankC. Table 4.1 summarizes the number of ATMs, customers, and total assets of these three banks. Regarding the total assets, each bank is classified in the same category as they rank in the top four private banks in Turkey. In terms of the total customer volume, BankA and BankC are leading, where BankB only has half of their volume. However, the number of digital customers of all three banks, combining active customers that use the web and mobile channels, are comparable. BankA leads in terms of the number of ATMs and branches as they value face-to-face interaction with customers as their tradition.

Table 4.1. Comparison of cases in terms of the number of ATMS, branches, customers and total assets

	Bank A	**Bank B**	**Bank C**
# of ATMs	6600	4300	5200
# of Branches	1200	850	900
# of Customers	19M	9.5M	18.5M
# of Digital Customers	8.9M	7.2M	9.3M
Total Assets	430B₺	370B₺	385B₺

Legacy Technology Stack. All banks had monolithic architectures prior to their transformation. Table 4.2 summarizes the context of the banks before the transformation of their legacy technology architectures.

Table 4.2. Legacy technology stack of banks

Legacy Tech. Stack	**Bank A**	**Bank B**	**Bank C**
Presentation Layer	Java (Web&Mobile)	Java (Web&Mobile)	Java (Web&Mobile)
Integration Layer	API, BPM, ESB	API	Mainframe, ESB
Backend Layer	Custom Application with Java/J2EE and .NET Mainframe (IMS) with COBOL	Custom Application with Java/J2EE	Mainframe (CICS) with Coolgen and COBOL
Data Layer	Mainframe DB2, Oracle	Oracle	Mainframe DB2

Before the migration, BankC was highly dependent on its monolithic mainframe core banking solution. Thanks to the highly modular and parametric mainframe architecture, BankC was still able to deliver business needs conveniently. With the rise of online banking, BankC has adopted service-oriented architecture (SOA) and developed Java-based web and mobile applications. These custom-developed applications are integrated with the mainframe using an Enterprise Service Bus (ESB) [11]. However, online banking increased the number of transactions on the mainframe and amplified the need for shifting towards open systems due to high operational costs.

BankB had already built its technology architecture on open systems before its migration. Still, BankB's 3-tier open system legacy was architected as a large monolithic core that is modular but not parametric. Core banking modules had very few cross-product service components. The backend of the monolithic core was developed in Java/J2EE, served through four channel applications. Channel applications are also developed in Java, and integration among layers was through API Gateway, and data is stored in Oracle databases.

The legacy technology stack of BankA had the burden of long-term co-existence of both the mainframe architecture and the multi-tier architecture with open systems. Most of the backend logic was on the mainframe; the multi-tier open system custom backend applications were developed in Java/J2EE with web and mobile applications developed in Java. The data were stored in mainframe (DB2 database) and open systems (Oracle databases). As a result, the bank spent a significant effort in data synchronization. To utilize SOA, BankA built an Enterprise Service Bus, an API Gateway, and Business Process Management (BPM) [11] layers to address integration requirements. As its legacy was running in product silos, there were a vast number of redundant functionalities. There were very few shared cross-product services such as product engine, pricing engine, fees and commission module used by product silos. New products and services could be developed mostly by new design and development rather than configuration. Besides, the monolithic core was neither modular nor parametric.

Regarding the infrastructure layer, all banks ran their legacy systems on-premises. Due to the strict regulations that restrain banks from moving to the cloud, this typical trait was mainly an obligation.

Although the architectures of the banks were classified as monolithic, the size of their application and service inventory stocks varied. Whether the bank's reliance was on the mainframe or open systems, the technical components like the number of databases, the number of APIs, and the number of applications were different than each other. Table 4.3 lists the banks' application and interface inventory.

Table 4.3. Application and interface inventory of banks

	BankA	BankB	BankC
# of Channel Applications	15	14	12
# of External APIs	17	50	70
# of Internal APIs	1648	125	350
# of Screens	1645	1250	8800
# of Domains(Service Groups)	19	29	200
# of Backend Services	4114	3000	12500
# of Distinct Core Databases	305	3	9

4.2. Key Motivations

In this section, we discuss R1: Which are the *key motivations* for banks to start their transforming from legacy systems towards next-generation architectures? Three banks have different motivations that lead them to initiate a transformation program.

BankA's pain points were limited multi-channel capabilities, low scalability, and low data quality for regulatory requirements. BankA would like to improve its multi-channel capabilities by introducing an omni-channel capabilities to its channel architecture. Several points to point integration of product and architecture components: redundancy in management of identical products in different silo applications made it difficult for BankA to make new products and features available for its customers through information systems. BankA would like to change its core applications to provide flexible product offering and bundling capabilities through the parametrization of key applications. BankA would like to move to an application and integration architecture that allows higher scalability and performance at a lower cost.

BankB's motivations were improving quality, reducing waste efforts, increasing collaboration and ownership, increasing scalability, and reducing technical debt. BankB

was introducing hundreds of system or application changes into their information systems yearly. Complexities of making a change were also increasing exponentially due to the loosely coupled structure of the applications and configuration and release management processes. To introduce these changes at the required speed, the bank is duplicating its efforts for designing components, lack of test automation, large monolithic application packaging, and manual processing of release management processes creates lots of wasted effort that is not delivering any value to the bank. BankB was looking for efficient service development and delivery process where they minimize the effort of redundant testing, release, deployment management, and overall quality of the codes being developed. Besides rapid scalability is required in order to serve high number of digital customers that have higher fluctuating behaviour. Traditional legacy banking architectures either in mainframe or open systems, unless it is hosted on cloud is required to have hardware architecture that support these peak usage behaviour with required redundancy and this comes with significant cost. BankB would like to make their applications cloud-enabled so that it benefits from cloud in terms of cost.

BankC's primary concerns were high IT costs and high attrition rates of qualified employees. BankC was having challenges attracting and retaining talent due to the mainframe technology they have been using for many years. Young engineers and software developers were looking for work where they can practice new technologies. After practicing Cobol and understanding the basics of software architectures in the banking industry, these talents were looking for jobs in other banks or different industries to practice new architectures or software programming languages. BankC realized that to sustain their leadership position in banking technologies, they have to upgrade their technology stack to attract the best people and retain them longer. Increased number of visits to their bank accounts or execution of payment or credit card transactions has increased the total number of transactions significantly. The cost of mainframes is typically measured by the amount of money paid per MIPS. As a result, BankC started to have higher costs than its peers for running its core banking applications. Therefore they have a strong business case in migrating their core banking from mainframe-based architecture to open systems. In a very dynamic business environment and highly

volatile markets, BankC would like to have an architecture where they could introduce changes more frequently and without sacrificing their systems' quality and robustness.

To compete in the digital era, BankA was the first bank to initiate a transformation program. They have started their program in 2014 and completed by 2019 as a transformed organization that evolved into a customer–centric model. To reflect on technological advances and cost pressures, BankB and BankC aimed to build an IT infrastructure for the next decades and to compete with non-banks; therefore, they targeted to adapt to a digital ecosystem-centric model. BankB's transformation program is planned to be completed by 2023. BankC's plan is in a broader time range, with a target to complete the program by 2025.

Target technology architecture. Taking motivations as the driving force and target functional model as the lever, banks have adopted different target technology architectures based on their needs. BankA continues to utilize SOA by using ESB and BPM solutions. They adopted omnichannel architecture with business process-driven logic. BankA's infrastructure remains on-premises.

BankB and BankC adopted similar architectural principles with different solutions. They both follow microservices architecture [12], where BankB has finer granularity. BankC has coarser granularity. Their experience on the open platforms was the primary reason for the granularity of their application components. BankB has already running on open platforms and has the necessary tools and infrastructure to monitor and manage high number of independent microservices components. BankC used to rely on the mainframe's resilient and robust management systems. They prefer to build coarse-grained components, manage them and decided to split them into smaller services at later stages of their transformation. Both banks adopted light-weight frontend architectures, API Gateway as integration layer, CI/CD pipelines [13] for delivery automation, and container platforms to orchestrate their cloud-ready applications. BankB has built its private cloud infrastructure [14], but BankC's infrastructure re-

mains on-premises.

4.3. Migration Strategies

In this section, we explore the different strategies for migration to answer R2: What is the *strategy* for application migration?

Every migration journey needs to define a clear migration strategy for every application based on a holistic application analysis. The selected strategy fundamentally affects the expected migration effort, the potential benefits, and possible long term cost savings of the new operations model.

After building the target technology architecture, all three banks have identified migration strategies to move their applications and services from legacy to newly built platforms. Re-architecting is the standard strategy for all of them; still, they approach differently while defining the strategy.

BankA already has SOA-based applications; therefore, their primary strategy was to retain the applications with small modifications and remediations. Re-architecting was the second strategy to follow, mainly used for mainframe offloading. As BankB's legacy architecture was on open systems, their primary strategy was to rehost the applications. They lift and shift the applications from legacy to a newly built cloud platform with minor enhancements. The main driver here is to migrate applications to new architecture rapidly and phase-out legacy systems for cost reduction. After rehosting, they start to re-architect the applications based on new architectural principles and develop microservices. BankC was heavily dependent on the mainframe, and application migration means re-architecting for them. Therefore, they primarily re-design and re-implement their applications from scratch, involving business stakeholders in the transformation program. Applications with low criticality and minimum transaction are classified to retain. For commodity applications that provide non-differentiating functionality, replacing them with a commercial off-the-shelf product is

another strategic decision for BankC.

All three banks have selected different migration planning approaches based on their working cultures and organizational structure. BankA aspired to plan the transformation in a structured way with a focused group leading the program; therefore, they followed the centralized approach. The time required to complete the evaluation process was relatively less for BankA compared to other banks, as they executed a series of workshops. Overall program plan and prioritization were prepared much earlier, decided by the architecture team in the initial planning phase. However, BankA struggled during the execution phase due to incorrectly captured domain-specific needs and misidentification of common components. As the prioritization was solely based on expert judgment and previous architecture team experience, the transformation program fell behind in terms of business value realization.

Principles of microservices architecture and agile delivery led BankB to follow a domain–driven migration planning; thus, domain-specific requirements have been captured by involving domains in an early stage. The know-how is cascaded to the project teams smoothly. As architectural compliance responsibility is on the domain teams, the inconsistency of designed solutions against target technology architecture principles was a significant drawback that BankB has faced. Another pain point was the double effort spent developing common services as every domain developed such services under its responsibility with limited reusability.

BankC has followed a different path by combining two approaches. With a hybrid approach in place, BankC benefits from the advantages of domain–driven execution and centralized planning and control function. Firstly, they have defined company-wide common components to reduce replicated functionalities and avoid double effort. Moreover, a central governance body ensures the compliance of solutions to functional and technical design principles. The central body also provided the ability to define priorities and plan in line with transformation goals. Still, BankC could not meet every domain-specific need and spent additional effort to clarify details at the domain

level due to limited responsiveness to product level requirements. Establishing a team and organizing workshop execution was a time-consuming activity that BankC had overcome with the central governing body.

4.4. Prioritization

In all our selected cases, banks' overall attitude was to define the prioritization criteria from a technical aspect, with a motivation to overcome the technical obstacles they are facing with the legacy architecture. Besides, the business and delivery objectives of banks also reflected while defining these prioritization criteria. Therefore, the principles of new technology architecture and target banking architecture were two key inputs. The prioritization assessment was commonly conducted as a survey by distributing a set of questions to the organization to understand every domain's needs. The outcome was a list of prioritized applications and a transformation program plan based on the questionnaire results.

BankA conducted the prioritization activity by a central authority, mainly a team composed of enterprise and solution architects, at the beginning of the transformation program for each application. The enterprise acknowledged this prioritization assessment as the guideline to initiate the execution stage. However, in BankB and BankC, the prioritization assessment was executed by each domain per application.

The central architecture team of BankB provided a checklist, a content to conduct a survey, to support domains. Domains provided inputs to these surveys and highlight how they prioritize their applications within the domain. They also provided a high-level project plan and their effort estimations with associated resource requirements. The governance team compiles these inputs and comes up with a consolidated roadmap for the transformation program. BankB did not conduct any governance on the solution designs for each domain. Domains that have their plans completed and access to the required resources are allowed to kick-off their plans while providing updates to the governance entity for the progress and updates on their plans.

BankC followed a more formal approach to identifying which projects to start. A template to gather data about the application and projects lists are created. The data about applications gather and verified through these templates. Each application is assigned to a scope calculated through a formula that takes the business value, technical feasibility, inter-dependencies, application life-cycle, and application priorities. The governance team and the solution architecture team assess these applications and their scores and come up with a roadmap where applications are migrated in waves. BankC's architecture board and solution architecture team ensures consistency among the designs and seeks common services among applications. The main difference between BankB and BankC was on the central architecture governing body, which oversees technical dependencies and schedules the execution plan, did not exist in the former, but it was present in the latter.

During the transformation program's execution phase, BankA performed the solution development activity during the initial planning for each domain/ application. BankB, instead, performed this activity in significantly later stages, mostly during the technical design of the relevant project for the domain/ application. The main difference of BankC was centrally governing the domain solution architecture concerning architectural standards and common services, where the domains are the only responsible for the blueprinting in the domain–driven migration approach.

4.5. Measuring the Success

In this section, we discuss the mechanisms used to measure the legacy transformation program's success to answer R4: How to measure the transformation program's success?

The accomplishment of a legacy migration transformation program requires a dedicated tracking approach that clarifies understandings and interpretations. Based on the observations of our cases, we have experienced that robust program governance is a key success factor for the consistent alignment of projects with overall transforma-

tion objectives and guidelines. The program governance focuses on business outcomes delivered, ensures alignment with business priorities, and manages the program by providing standard processes to drive consistent reporting of status. It orchestrates project teams regarding risks and dependencies, provides a consolidated program level schedule and budget, and establishes a result tracking mechanism driven by value delivered throughout the program.

It is a common approach for banks' transformation office to set quantitative and qualitative metrics to show the progress achieved. These metrics measure the transformation program in terms of technology, processes, and people. Based on the program goals and the function that drives the transformation, banks tend to set technical-driven or business-oriented metrics. Those that set technical-driven metrics struggle to get long term commitment from top management as the business value delivered is vague. Contrary, banks that set business-oriented metrics keep track of the program efficiently yet struggle in technical aspects such as quality and resilience, which are revealed in later stages of the program and may cause significant fractures. Therefore, it is essential to find the right mix in between.

The metric setup approach is tightly related to the migration approach selected at the beginning. Metrics are defined, owned, and tracked by the central governance team in centralized migration. Each domain defines, owns, and tracks its metrics separately in domain-driven migration. In a hybrid migration, the central governance team defines the metrics and assigns them to domains, domains own, track, and report.

As the metrics are defined and assigned, the program tracking process is run parallel to the migration execution. The information collection process is executed periodically (e.g., quarterly), metrics are calculated based on the data gathered, and reports are published regularly.

BankA's waterfall approach for shaping the transformation program has considered mostly traditional program management KPIs as seen table 4.4. The program was

composed of several initiatives grouped under two main domains. KPIs under transition to cloud based infrastructure monitor implementation of the new infrastructure and migration of all artifacts to the new data center. Migration of the applications from legacy architecture to the new architecture and decommissioning of the legacy applications was used as a key measure to monitor the progress and success of transition to the next generation architecture. BankA executed the whole program as an IT transformation program. During the execution of the program, bank's business objectives are measured and monitored separately and they wasn't any link established between them.

Table 4.4: Key Performance Indicators of BankA

Main Objective Area	Metrics
Transition to Cloud Based Infrastructure	Completion Ratio of Shell & Core Items
	Completion Ratio of Security Requirements
	Completion Ratio of Infrastructure Implementation
	Tracking of Outages Due to Migration
	Completion Ratio of IT Inventory Migration
Transformation to Next Generation Application Architecture	Progress of Planned FE & Middleware Development
	Progress of Multichannel Management Development
	Progress of Implementing New Components
	Progress of .NET Software Development Architecture
	Completion Ratio of Infrastructure Foundations
	Re-Platforming Velocity of Applications
	Progress of Legacy Application Decommissioning
	Application Performance Improvements
	Applications Availability Improvements

BankB has defined a detailed list of KPIs that would help them to monitor their progress over the program. They have defined a different set of metrics for program workstreams. Summary of these metrics reported to bank's business units to help them

follow. BankB defined KPIs as seen in table 4.5 grouped under five work streams. These KPIs were specific to the type of activities grouped under these workstreams. Migration to cloud based infrastructure, implementation of CD/CI delivery, application of new methods and tools, transition to modular and scalable architecture, transition on budget are the metrics groups owned by program workstreams. Similar to BankA, BankB did not connect the program objectives with the bank's business objectives.

Table 4.5: Key Performance Indicators of BankB

Main Objective Area	Metrics
Transition to Cloud Based Infrastructure Architecture	Completion Ratio of Cloud Services Readiness
	Application's Cloud Transactions Ratio
	Ratio of Cloud Transactions over all Transactions
	# of Applications on Cloud Architecture
	Completion Ratio of User Authorization Definitions
	Container Security Standards Compliance KPIs
CI/CD	Product Backlog Ratio
	Ration of Successful Deployments
	# of Defects over # of Application's Test Cases
	Test Efficiency Ratio
	Average Defect Fixing Duration
	Test Coverage Ratio
	Automated Tests Pass Ratio
	Data Quality Test Pass Ratio
	Automated Analysis Coverage Ratio
	Fixing Ratio of Analysis Findings
Bringing New Technologies and New Competencies	Employee Satisfaction Survey Score
	Technology Dissatisfaction Score in Exit Interviews
	Turnover Rate Reduction
	Trainings Completion Ratio
	Hiring Ratio from Top Universities

Continued on next page

Table 4.5 – continued from previous page

	Applications Ratio from Top Universities
Transformation to Modular and Scalable Architecture	Cost per Code Line per Application
	Technical Debt Reduction Ratio
	Average Service Response Time
	FE Service Response Time
	Service Response Ratio
Transformation On Budget	Project Budget Compliance Ratio
	Infrastructure Budget Compliance Ratio
	Transformation Budget over Total IT TCO
	Cost Reduction per Milion Transactions

BankC applied a value-driven program management framework where they calculate the expected value of each program actions, priotize accordingly and execute if necessary. They have defined KPIs as seen in table 4.6, grouped under three strategic objectives. Re-Imagine next-generation architecture, shift to CD/CI, and people transformation objectives were the strategic objectives area where these KPIs were grouped. BankC chose to measure value tangibly and drives to prioritize applications dynamically as they progress over the program. All metrics were connected to the strategic objectives connected to the bank's business objectives.

Table 4.6: Key Performance Indicators of BankC

Main Objective Area	Metrics
Re-imagine Next Generation Architecture	# of New APIs in the Catalog
	# of New Web Components in the Web Catalog
	Ratio of Transactions in the New Architecture
	Ratio of Applications in Production
	Common Business Services Transformation Ratio
	Ratio of MIPS Reduction per Application

Continued on next page

Table 4.6 – continued from previous page

Shift to CD/CI	Product Backlog Variance
	Fix Duration of Broken Builds
	Reduction in Time to Recover Production Failures
	Average Resolution Time of Production Defects
	Operational Incident Ratio
	Automated Unit Test Coverage Ratio
	Technical Debt Reduction
	Budget Compliance Ratio
	Infrastructure Saving from Transformation
	Cost Reduction per MIPS
People Transformation	Turnover Rate Reduction
	Employee Engagement Survey Scores
	Hiring from Top Universities
	Ratio of Trained Inhouse Resources

Despite different approaches to measuring progress and success of the program, banks have several common KPIs although they use different names for KPIs. Table 4.7 outlines the common KPIs among the cases.

Table 4.7: Common Key Performance Indicators

Common Metrics	BankA	BankB	BankC
Cloud Services Readiness	X	X	
Security Standards Compliance	X	X	
Migration to Cloud Architecture Ratio	X	X	
Reusable Components Ratio	X	X	X
Ratio of Application in Production	X	X	X
Product Backlog Variance	X	X	X
Ratio of Transactions on New Architecture	X	X	X

Continued on next page

Table 4.7 – continued from previous page

Automated Test Coverage Ratio		X	X
Defect Fixing Duration		X	X
Availability/ Avr. Service Response Time	X	X	
Technical Debt Reduction		X	X
Cost Reduction per Code Line		X	X
Cost Reduction per Million Transaction		X	X
Total Budget Variance		X	X
Infrastructure Budget Variance		X	X
Transformation Budget/IT TCO		X	X
Employee Engagement Trends		X	X
Attrition Growth Rate		X	X
Hiring from Top Universities		X	X
Training Completion Ratio		X	X

4.6. Discussion and Key Takeaways

Industrial trends. Our interviews with the chief technology officers of the banks reveal that their decisions are affected by the software architecture trends. For the cases of BankB and BankC, the chief technology officers agree that SOA pattern provides a straightforward solution to their problems, yet they opt for the microservice architecture for it is the trending architecture pattern during their transformation period.

Pattern selection. The most urging issues of the current operation influence the design pattern chosen for the target architecture. For example, the redundancy requirement of BankA over the application architecture force the bank to simplify the application architecture, eliminate redundant interfaces and components, and increase reusability in the cross-product components. As a result, the bank preferred to ap-

ply the SOA pattern over a well-defined functional banking architecture. BankB had problems with high amount of redundant efforts for development, issues during the release of new the versions and difficulties in configuring the management processes. To solve these problems, the bank adopted domain–driven design patterns, the microservice architecture, and the agile development method. BankC suffered from increasing processing costs due to rapidly increasing online transactions from their high number of digital customers, which forced them to move out from the mainframe they were peacefully living in the last 20 years.

Functional application models. Historically, the functional application models for banks evolve from product–centric to segment–centric and manufacturing-centric to customer–centric and, recently, to ecosystem-centric functional models. The existing legacy architecture and business requirements of the banks define their target functional application model. BankA has been operating under a manufacturing-centric functional model and strived to apply a more customer–centric banking model. BankB and BankC claimed that they have already been using customer–centric models and aimed for migrating to ecosystem-centric functional models to serve their customers through their ecosystem partners. Therefore, these two banks implemented strong API layers in their integration architectures with lightweight application architectures at the front-end layer.

Multiple migration strategies. In all cases multiple migration strategies are followed based on the previous transformation experiences, the existing legacy architectures, the target architectures, and the budget constraints of the banks. BankA, which run on an existing hybrid architecture (mainframe and open system architecture), did not consider a new core banking implementation from scratch due to their past experiences on failed migration to a package core banking software and transformation costs. BankA preferred retaining existing applications by re-engineering them (retain strategy), developing cross-product components that eliminate redundant applications (re-architect strategy), and gradually migrating to a business process based

operating model. BankB, which has previously migrated fully from the mainframe environment, preferred to re-platform existing monolithic applications to a different architecture (re-host strategy). BankB also designed and implemented their primary application domains in microservices architecture pattern (re-architect strategy) relying on the domain–driven design and cloud-ready technology stack. On the other hand, BankC which has not conducted a major core banking transformation program in the last decades chose to retain some of the applications, to replace some others and to re-architect their core banking system as a whole through a more extended transformation program.

Governance organizations and design patterns. The culture and availability of strong governance structures such as architecture boards, solution architecture teams, and target architecture patterns play an essential role in how migration planning is conducted. BankA, which has a strong architecture governance organization and a top-down management style, conducted their migration planning exercises relying on central teams. domain–driven design pattern and microservices architecture as target design pattern let the teams conduct planning exercises through domains that maintain an application under a functional domain.

Prioritization. Regardless of the planning approach selected, all banks develop a prioritized list of projects in which they plan to migrate legacy application components to the target architecture. BankA conducted a bottom-up analysis for each project. It identified high-level application inventories to be updated and estimated the effort required. BankB chose to apply a survey and asked the domain architect to evaluate and score the applications they will re-architect under their domain and develop a comprehensive list and application migration prioritization. BankC developed a scoring model based on weights of prioritization criteria and the nature of the applications or domains to be migrated. All these prioritization lists helped them to develop a roadmap that shows them two or three years horizon with more detailed plans for the coming quarters or years. It is essential to prioritize so that program governance

over scope, schedule, and budget is managed. On the other than considering the very complex nature of banking application architecture and duration of these programs and continuously changing environment, it was never possible to stick to these plans. The solution seems to develop a more agile program management approach where these prioritize frequently change while still demonstrating that the program delivers the highest possible value. We believe that value-driven program management [15] [16] practice would provide the most optimized prioritization.

Robust project management. All three studies confirmed that a robust migration project management is required to ensure the continuity, progress, and quality of the transformation. A robust project management ensures the alignment of business and IT functions. Project management is also crucial for quality assurance throughout the transformation, by providing standard processes, actions to drive consistent reporting of status, risks, issues, and resources. A robust project management enables planning and monitoring the project by centralizing the project level plan, budget, dependencies, and backlog.

Measuring business value. It is a common approach for the transformation office of banks to set quantitative and qualitative metrics to show the progress achieved. These metrics measure the transformation program in terms of technology, processes, and people. Based on the program goals and the function that drives the transformation, banks tend to set technical-driven or business-oriented metrics. Those that set technical-driven metrics struggle to get long term commitment from top management as the business value delivered is vague. Contrary, banks that set business-oriented metrics keep the track of the program efficiently yet struggle in technical aspects such as quality and resilience, which are revealed later stages of the program and may cause major fractures in the program. Therefore, it is important to find the right mix in between. The metric setup approach is tightly related to the migration approach selected at the beginning. Metrics are defined, owned, and tracked by the central governance team in centralized migration. Each domain defines, owns, and tracks their own met-

rics separately in domain-driven migration. In a hybrid migration, central governance team defines the metrics and assigns to domains, domains own, track and report. As the metrics are defined and assigned, the program tracking process is run in parallel to the migration execution.

Continuous alignment. As the banks progress through the transformation program and migrate the applications to the new platform, migration evaluation and prioritization criteria are subject to change. Continuous alignment of these criteria with program objectives is essential to succeed in the application migration.

5. RELATED WORK

In this chapter we overview related work focusing on case study research in software engineering and migration to cloud, microservices, SOA and large scale program management practices.

Case studies in software engineering. Runeson and Höst provide guidelines for conducting case study research in software engineering [10] [17]. Verner et al. present a framework with finer granularity for industrial case studies [18]. In this work we follow the steps stated by [10] and consult the framework presented in [18] when needed. Santos presents strength and weaknesses from planning and performing member checking to justify the findings of case studies performed in a software engineering [19]. We tried to apply narrative accuracy and interpretive validity member checking techniques in one-to-one interviews and elaborating our findings collected through surveys in the workshops.

Service Oriented Architecture. Errad and Anand investigate key approaches to transform legacy applications into services to participate in an enterprise-wide SOA. They present a decision framework to guide architects in selecting the optimal combination of legacy modernization options [20]. Gouigoux and Tamzalit share experiences of a single migration from monolith to service–oriented architecture [21]. Main benefits from this transformation are listed as reusability, changeability, and efficiency. A. Megargel et al. [22] investigate bank's technology infrastructure, IT governance processes and maturity of SOA practices to find out their impacts on time-to-market of new products and services. He conducted interviews and case study surveys with CTOs from eight banks operating in Asia and conclude SOA maturity plays an important role in digital transformation. He recommended establishing centre of excellence in SOA and implementation of a well–architected ESB increases maturity of SOA. Among our cases, BankA's has the similar motivation and selected SOA as their design pattern

to increase their time-to-market. BankA implemented ESB and has established SOA center of excellence.

Microservices and cloud migration. Jamshidi et al. [23] details a pattern–based approach for multi-cloud architecture migration. JF Zhao and JT Zhou et al. [24] discuss similarities and difference of migration strategies. In our case studies, the main migration problem is the one from the monolithic legacy system to modular architecture. Migration to cloud is a part of the problem, but not the main focus for BankA and BankC. Re–hosting is one of the migration strategy for BankB. BankB looks for ways to migrate existing application to cloud architecture with limited changes. Balalaie et al. [25] report on the migration to a cloud-native microservices architecture of a single company. Our work confirms the need for distributed data governance, reusability, and scalability as the drivers for the migration. Bucchiarone et al. presents the changes in the architecture from a monolith microservice architecture in the banking domain [26]. The focus of discussion in [26] is on the changes in the architecture. Dragoni et al. analyse the same case from a mission critical perspective [27]. Fan and Ma comment that two drawback of the migration to microservices are complex structure and increased resource use in their experience report on the migration of a web and mobile application to microservices [28]. Di Francesco et al. survey 17 participants from the industry for their experiences on migration to microservices [29] focusing on the challenges in execution of the migration. He considered migration to services into three steps: reverse engineering, architecture transformation and forward engineering. He first identify the migration activities in each of these phases and analyze the challenges faced by the participants in these subprocesses. Another survey is conducted by Taibi et al. [30] where the participants answer questions on the migration process, cost, and data overhead. In a recent study A Megargel et al. compares the monolithic and microservices styles of application architecture in banking industry and proposes a methodology for transitioning from monolithic to cloud-based microservices [31]. We portray the existing architectures and focus on the similarities and differences in their motivation. Our study analyzes migration strategy, and the detailed planning approach

for the migration from legacy to the target architecture. Our our domain is limited to the banking industry.

Service identification during planning. Banks commented that one of their problem is finding adequate granularity and cohesiveness of their services when thinking of transforming, evolving and scaling existing applications. Li and Zhang et al. [6] give a dataflow-driven approach to identifying microservices from monolithic applications. Baressi and Garriga et al. [32] investigate a method for identifying microservices through interface analysis. He note that finding the adequate granularity and cohesiveness of microservices, both when starting a new project and when thinking of transforming, evolving and scaling existing applications, is a problem. In this thesis, he introduces a solution based on the semantic similarity of foreseen/available functionality described through OpenAPI specifications. C Schröer et al. [33] conduct a literature review on microservices identification approaches and classify the content based on the software development process. Mostly monolithic applications were suitable for decoupling and conversion to microservices. Applications with higher data consumption are not recommended. In case core banking applications, cross–products services such as commission and fee engine, product factory engine, information provider services were considered as best candidates for microservices. In our cases the banks did not consider applying any automatic service identification techniques due to estimated complexities and lack of documentation. C. Bandara et al. [34] also claim that with correct tool support, it would be possible to get insight into the possible services which can be found in the existing monolithic system at code level without worrying about the existence of architecture diagrams, system artefacts and people who know the legacy system well. He propose a toolkit to analyze the monolithic systems and propose the best ways to decompose the functionality into set of microservices. Highly complex architecture of core banking application may not allow to implement such methods standalone.

Governance and Measurement. Programs at size of our cases requires a strong and proven governance mechanisms. SU Haq et al. [35] investigate effectiveness of

several program governance mechanisms and proposing a moderation model on how to measure it. Kerzner et al. [36] shares best practices in the project management metrics, KPIs, and dashboards in his book "A guide to measuring and monitoring project performance". He grouped metrics into four categories; business-based, success-based, project-based, and project management process metrics. He highlighted that the right person who should be doing the asking could select the right metrics. Sanchez and Robert et al. [37] propose to help to measure the achievement of a portfolio's strategic objectives taking into account the realization of key benefits. In our cases banks realized the benefit of do it, but follow different approaches when selecting specific KPIs. Shahin and Mahbod [38] present a method on how to prioritize KPIs where there are several different KPIs to monitor. Our cases introduce more than fifty KPIs without identifying the priorities among them. In order to ensure proper governance some sort of prioritization would be helpful where there are multiple KPIs alarming actions in different direction. C Karrenbauer et al. [39] study in this area and present an evaluation approach with an optimization model for improving project portfolio management, support value creation, and goal achievement.

Dependency Management. Stray and Moe et al. [40] states managing dependencies between teams and within teams is critical when running large-scale agile projects. Babinet and Ramanathan et al. [41] discusss the challanges of dependency management in large agile environments. In transformation programs such as our cases large-scale software development work is carried out simultaneously in parallel by many developers and development teams. Dependency management is stated as one of the critical success factors for legacy system migration programs. I. Pashchenko et al. [42] investigate decision-making strategies for selecting, managing, and updating software dependencies. N Sekitoleko et al. [43] outline difficulties in technical dependency management and their communication during the program. V. Stray et al. [40] examine planning methods in agile large-scale project and how these methods address different types of dependencies. High number of dependencies in our cases requires development of automating identification and monitoring of these dependencies.

Enterprise Architecture and Migration. N. Silva et al. [44] recommend a set of migration rules to automate migration of enterprise architecture. We observe an attemp in BankA migration strategy to migrate some of the processes through automation. Sun and Chen [45] propose domain based integrated enterprise technical architecture for a universal bank, helping to build common patterns to manage bank's technical IT applications and infrastructure. This approach was very similar to the migration prioritization approaches followed by BankB and BankC. BankB applies domain–driven design and apply similar patterns in domains. BankC uses enterprise functional model as an input to migration prioritization model. Agievich and Skripkin propose using a change matrix to manage enterprise architectures migration [46]. Similar tools are used in all our cases to manage the migration. Furda et al. investigate multitenancy, statefulness, and data consistency for the migration of enterprise architectures [47].

6. CONCLUSION

In this thesis we present a multi-case study of legacy system migration from the banking domain. Three cases have a comparable number of digital customers but differ in terms of total number of total customers, and application and interface inventories. All three transformations took place between 2014 and 2020. Throughout the migration projects, we interviewed the members of projects with different ranks and roles in the projects. We conduct an in-depth analysis of the motivations for the migration, migration strategies, and prioritization for the migration. We discuss the key takeaways and lessons learned for such high-scale migration projects.

We observed similarities in motivations of the banks for the migration even though they had different legacy banking architectures such as reducing costs and improving quality although each bank had its specific triggers for the migration. The most urging issues heavily influenced the target architectures of the bank. The decision makers of the migration projects also admitted that architectural trends were another strong influence for that decision.

All banks applied multiple migration strategies based on their transformation experiences, existing legacy architectures, target architectures driven by technology trends, and cost pressures. Culture and availability of strong governance structures such as architecture boards, solution architecture teams, and target architecture patterns play an essential role in how migration planning is conducted. We observed that as the banks progress through the transformation program and migrate the applications to the new platform, migration evaluation and application prioritization criteria are subject to change.

Finally, continuous alignment of these criteria with program objectives is essential to succeed in the application migration. As applications are migrated, banks evolve to new technology architectures and retire their legacy systems. On top of the techno-

logical evolution, our cases aimed to adopt new methodologies like DevOps and agile to maximize the benefits from their new architectures. These changes also drive our cases to shift towards modern IT operating models.

Reflecting on our experiences in this case study, we suggest the following lines for future research: First, we consider it essential to replicate the study in organizations in different industries. Banks have substantial monolithic architectures, and the modernization of these systems cannot be completed in a single transformation program. On the other hand, several organizations with a relatively small monolithic legacy system require modernization programs with similar motivations. Approaches or techniques used for identifying the migration strategies and prioritization of application may be different

Second, we find it interesting to investigate governance methods used during the execution of the transformation programs. Organizations maturity, culture and experience is similar programs seem to have impact on the way governance is handled and specific processes are executed. Program tracking, dependency management and how the overall value is measured should have a huge impact on the successful outcome of these programs. This could include, for example, challenges and success factors for large-scale agile transformation programs (Dikert and Paasivaara at al. [6]) as the lenses to guide the collection and analysis of case study data in future research.

Third, we acknowledge that the way the monolithic is decomposed and microservices are identified is critical design decision for organization on how to architect the new system. We therefore think, it is interesting to investigate automatic service identification techniques and frameworks (Baresi and Garriga at al. [32]), (Chen and Li at al. [48]) over modernization of legacy systems in banking.

REFERENCES

1. Pérez-Castillo, R., B. Mas and M. Pizka, "Understanding Legacy Architecture Patterns", *2015 International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE)*, pp. 282–288, IEEE, 2015.

2. Zimmermann, A., R. Schmidt, K. Sandkuhl, D. Jugel, J. Bogner and M. Möhring, "Evolution of Enterprise Architecture for Digital Transformation", *2018 IEEE 22nd International Enterprise Distributed Object Computing Workshop (EDOCW)*, pp. 87–96, IEEE, 2018.

3. Hu, S.-J., "Method and System for Integrating Core Banking Business Processes", Google Patents, Jan. 24 2006, US Patent 6,990,466.

4. Hariharan, N. and K. Reeshma, "Challenges of Core Banking Systems", *Mediterranean Journal of Social Sciences*, Vol. 6, No. 5, p. 24, 2015.

5. Erl, T., *Service-Oriented Architecture: Concepts, Technology, and Design*, Pearson Education India, 1900.

6. Dikert, K., M. Paasivaara and C. Lassenius, "Challenges and Success Factors for Large-Scale Agile Transformations: A Systematic Literature Review", *Journal of Systems and Software*, Vol. 119, pp. 87–108, 2016.

7. Robson, C., *Real World Research: A Resource for Social Scientists and Practitioner-Researchers*, Vol. 2, Blackwell Oxford, 2002.

8. Stake, R. E., *The Art of Case Study Research*, Sage Publications, 1995.

9. Yin, R., "Designing Case Studies", *Qualitative Research Methods*, pp. 359–386, 2003.

10. Runeson, P. and M. Höst, "Guidelines for Conducting and Reporting Case Study Research in Software Engineering", *Empirical Software Engineering*, Vol. 14, No. 2, p. 131, 2009.

11. Liu, Y., E. Hu and X. Chen, "Architecture of Information System Combining SOA and BPM", *2008 International Conference on Information Management, Innovation Management and Industrial Engineering*, Vol. 1, pp. 42–45, IEEE, 2008.

12. Thönes, J., "Microservices", *IEEE Software*, Vol. 32, No. 1, pp. 116–116, 2015.

13. Steffens, A., H. Lichter and J. S. Döring, "Designing a Next-Generation Continuous Software Delivery System: Concepts and Architecture", *2018 IEEE/ACM 4th International Workshop on Rapid Continuous Software Engineering (RCoSE)*, pp. 1–7, IEEE, 2018.

14. Chilipirea, C., G. Laurentiu, M. Popescu, S. Radoveneanu, V. Cernov and C. Dobre, "A Comparison of Private Cloud Systems", *2016 30th International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, pp. 139–143, IEEE, 2016.

15. Kerzner, H., F. P. Saladis *et al.*, *Value-Driven Project Management*, Vol. 1, John Wiley & Sons, 2011.

16. Aitken, I., *Value-Driven IT Management*, Routledge, 2012.

17. Host, M. and P. Runeson, "Checklists for Software Engineering Case Study Research", *First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*, pp. 479–481, IEEE, 2007.

18. Verner, J. M., J. Sampson, V. Tosic, N. A. Bakar and B. A. Kitchenham, "Guidelines for Industrially-Based Multiple Case Studies in Software Engineering", *2009 Third International Conference on Research Challenges in Information Science*, pp. 313–324, IEEE, 2009.

19. Santos, R. E., C. V. Magalhães and F. Q. da Silva, "Member Checking in Software Engineering Research: Lessons Learned from an Industrial Case Study", *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pp. 187–192, IEEE, 2017.

20. Erradi, A., S. Anand and N. Kulkarni, "Evaluation of Strategies for Integrating Legacy Applications as Services in a Service Oriented Architecture", *2006 IEEE International Conference on Services Computing (SCC'06)*, pp. 257–260, IEEE, 2006.

21. Gouigoux, J.-P. and D. Tamzalit, "From Monolith to Microservices: Lessons Learned on an Industrial Migration to a Web Oriented Architecture", *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*, pp. 62–65, IEEE, 2017.

22. Megargel, A., V. Shankararaman and T. P. FAN, "SOA Maturity Influence on Digital Banking Transformation", *IDRBT Journal of Banking Technology*, Vol. 2, No. 2, p. 1, 2018.

23. Jamshidi, P., C. Pahl and N. C. Mendonça, "Pattern-Based Multi-Cloud Architecture Migration", *Software: Practice and Experience*, Vol. 47, No. 9, pp. 1159–1184, 2017.

24. Zhao, J.-F. and J.-T. Zhou, "Strategies and Methods for Cloud Migration", *International Journal of Automation and Computing*, Vol. 11, No. 2, pp. 143–152, 2014.

25. Balalaie, A., A. Heydarnoori and P. Jamshidi, "Microservices Architecture Enables Devops: Migration to a Cloud-Native Architecture", *IEEE Software*, Vol. 33, No. 3, pp. 42–52, 2016.

26. Bucchiarone, A., N. Dragoni, S. Dustdar, S. T. Larsen and M. Mazzara, "From

Monolithic to Microservices: An Experience Report from the Banking Domain", *IEEE Software*, Vol. 35, No. 3, pp. 50–55, 2018.

27. Dragoni, N., S. Dustdar, S. T. Larsen and M. Mazzara, "Microservices: Migration of a Mission Critical System", *arXiv preprint arXiv:1704.04173*, 2017.

28. Fan, C.-Y. and S.-P. Ma, "Migrating Monolithic Mobile Application to Microservice Architecture: An Experiment Report", *2017 IEEE International Conference on AI & Mobile Services (AIMS)*, pp. 109–112, IEEE, 2017.

29. Di Francesco, P., P. Lago and I. Malavolta, "Migrating towards Microservice Architectures: An Industrial Survey", *2018 IEEE International Conference on Software Architecture (ICSA)*, pp. 29–2909, IEEE, 2018.

30. Taibi, D., V. Lenarduzzi and C. Pahl, "Processes, Motivations, and Issues for Migrating to Microservices Architectures: An Empirical Investigation", *IEEE Cloud Computing*, Vol. 4, No. 5, pp. 22–32, 2017.

31. Megargel, A., V. Shankararaman and D. K. Walker, "Migrating from Monoliths to Cloud-Based Microservices: A Banking Industry Example", *Software Engineering in the Era of Cloud Computing*, pp. 85–108, Springer, 2020.

32. Baresi, L., M. Garriga and A. De Renzis, "Microservices Identification through Interface Analysis", *European Conference on Service-Oriented and Cloud Computing*, pp. 19–33, Springer, 2017.

33. Schröer, C., F. Kruse and J. M. Gómez, "A Qualitative Literature Review on Microservices Identification Approaches", *Symposium and Summer School on Service-Oriented Computing*, pp. 151–168, Springer, 2020.

34. Bandara, C. and I. Perera, "Transforming Monolithic Systems to Microservices-An Analysis Toolkit for Legacy Code Evaluation", *2020 20th International Conference on Advances in ICT for Emerging Regions (ICTer)*, pp. 95–100, IEEE, 2020.

35. Haq, S. U., D. Gu, C. Liang and I. Abdullah, "Project Governance Mechanisms and the Performance of Software Development Projects: Moderating Role of Requirements Risk", *International Journal of Project Management*, Vol. 37, No. 4, pp. 533–548, 2019.

36. Kerzner, H., *Project Management Metrics, KPIs, and Dashboards: A Guide to Measuring and Monitoring Project Performance*, John Wiley & Sons, 2017.

37. Sanchez, H. and B. Robert, "Measuring Portfolio Strategic Performance using Key Performance Indicators", *Project Management Journal*, Vol. 41, No. 5, pp. 64–73, 2010.

38. Shahin, A. and M. A. Mahbod, "Prioritization of Key Performance Indicators: An Integration of Analytical Hierarchy Process and Goal Setting", *International Journal of Productivity and Performance Management*, 2007.

39. Karrenbauer, C. and M. H. Breitner, "Value-Driven IT Project Portfolio Management: Tool-Based Scoring, Selection, and Scheduling", *International Research Workshop on IT Project Management 2020*, 2020.

40. Stray, V., N. B. Moe and A. Aasheim, "Dependency Management in Large-Scale Agile: A Case Study of DevOps Teams", *Proceeding of the 52nd Hawaii International Conference on System Sciences (HICSS 2019)*, University of Hawai'i, 2019.

41. Babinet, E. and R. Ramanathan, "Dependency Management in a Large Agile Environment", *Agile 2008 Conference*, pp. 401–406, IEEE, 2008.

42. Pashchenko, I., D.-L. Vu and F. Massacci, "A Qualitative Study of Dependency Management and Its Security Implications", *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1513–1531, 2020.

43. Sekitoleko, N., F. Evbota, E. Knauss, A. Sandberg, M. Chaudron and H. H. Olsson, "Technical Dependency Challenges in Large-Scale Agile Software Development",

International Conference on Agile Software Development, pp. 46–61, Springer, 2014.

44. Silva, N., F. Ferreira, P. Sousa and M. M. da Silva, "Automating the Migration of Enterprise Architecture Models", *International Journal of Information System Modeling and Design (IJISMD)*, Vol. 7, No. 2, pp. 72–90, 2016.

45. Sun, J. and Y. Chen, "Building a Common Enterprise Technical Architecture for An Universal Bank", *2010 International Conference on Management and Service Science*, pp. 1–4, IEEE, 2010.

46. Agievich, V. and K. Skripkin, "Enterprise Architecture Migration Planning Using the Matrix of Change", *ITQM*, pp. 231–235, Elsevier, 2014.

47. Furda, A., C. Fidge, O. Zimmermann, W. Kelly and A. Barros, "Migrating Enterprise Legacy Source Code to Microservices: On Multitenancy, Statefulness, and Data Consistency", *IEEE Software*, Vol. 35, No. 3, pp. 63–72, 2017.

48. Li, S., H. Zhang, Z. Jia, Z. Li, C. Zhang, J. Li, Q. Gao, J. Ge and Z. Shan, "A Dataflow-Driven Approach to Identifying Microservices from Monolithic Applications", *Journal of Systems and Software*, Vol. 157, p. 110380, 2019.