International Journal of Computer Techniques–IJCT Volume 13 Issue 1, January - 2026

Open Access and Peer Review Journal ISSN 2394-2231 https://ijctjournal.org/

RESEARCH ARTICLE OPEN ACCESS

An Empirical Analysis of Monolith-to-Microservices Migration in Enterprise Banking Systems

Sameena Begam Savukath Ali
Southern New Hamsphire University, New Hampshire
sameenabegam.savukathali@gmail.com

--*************************----------------------------------

Abstract:

Historically, the banking sector has predominantly used large monolithic technology to perform their crucial business operational tasks (like managing accounts, processing payments, lending activities, evaluating potential risks, and producing reports required by regulatory agencies). When banks initially built their technological systems, regulations and banking operations were relatively stable; consequently, banks designed their monolithic banking systems specifically with an emphasis on stability, high levels of transaction integrity, and centralized decision-making. For years, this technology enabled banks to successfully function at the enterprise level and gave banks nearly total control over data integrity, security, and compliance with regulations.
Unfortunately, as the financial services sector has changed, so too has the need for banks' use of monolithic technology. Digital banking, faster payment methods, mobile-friendly customer experiences, new open banking and regulatory compliance rules, and continuous updates to regulations are exerting unyielding stress on banks' existing systems. Most banks' technologies were created to handle tight coordination between all releases, extended testing periods, and implementation of all components of an overall technology at once; therefore, introducing new features and addressing customer requests and/or compliance issues can take many months or even years to implement successfully. Since it is usually impractical for banks to scale their individual functional areas independently of each other, they tend to manage their resources and operate their systems in a less than efficient manner. Therefore, more banks are now implementing technology using a Microservices architecture model, allowing for greater agility, resiliency, scalability, and cloud compliance than monolithic architectures allowed [1][2].

This research paper analyzes both the technical and organizational lessons learned from migrating acquired inherited monolithic core banking systems to a microservice-based architecture. On the technical side, a comprehensive perspective on the many core challenges faced by organizations in their journey toward microservice implementations is explored. Such core challenges include defining appropriate service boundaries within a decomposed application ecosystem, decoupling a tightly integrated legacy database, managing distributed transactions, managing operational complexity associated with distributed systems, and (through Restructured continuous integration and continuous delivery (CI/CD) pipelines) structuring the environment to support independent service deployments [10]. In banking environments, these technical challenges can be especially challenging, as the issues of data integrity, data availability, and data consistency are critical.

Additionally, the study discusses how several organizational factors greatly impact the chances for migration success. These factors include the need to restructure teams around business-aligned services, retrain skillsets toward cloud-native and DevOps methodologies [10], and evolve governance models to achieve the proper balance of service autonomy and regulatory compliance. Through a case study using two separate financial institutions' successes and analyzing the need to remodel existing business structures through architectural patterns (the "Strangler Fig pattern"), through domain-driven designs [8], through decentralized data management and DevOps enablement [10], this research demonstrates that using an incremental migration strategy will result in significant increases in deployment frequency, improved scalability of systems, and greater operational resilience.

Overall, the conclusion of the research paper is that to modernize core banking platforms successfully through microservice migrations requires businesses to view microservices migrations as a holistic transformation of their architecture, operational processes, and organization, rather than just a Technical refactoring effort.

Keywords — Microservices Architecture, Monolithic Systems, Banking Systems Modernization, Domain-Driven Design, DevOps, CI/CD Pipelines, Distributed Systems, Data Consistency, Cloud-Native Architecture, Regulatory Compliance

---------------------------------------************************---------------------------------

I. INTRODUCTION

In the past, banking companies have used one big system to manage everything from account management, through payment processes and risk assessments, as well as all lending procedures and regulations. A major feature of the monolith was the ability to centralize business logic, data management and security controls into a single unit of deployment. This resulted in a strong level of consistency between transactions, predictable performance for banks and a central set of rules for governing bank transactions. Monolithic designs met the stricter operational and regulatory demands placed on financial institutions at that time and provided banks with a stable platform to deliver trusted, scalable services to their customers and maintain greater control over access management, data integrity and audit capability for many years.

Today, however, with customers demanding digital banking and mobile technology, the limitations of the monolithic design are becoming increasingly clear. Many banks were not prepared to support the demands for quick feature delivery, elastic resources and availability required by the proliferation of mobile and real-time payment methods and 24/7 online banking services. Changes to even small areas of function required system-wide coordination, extensive regression testing and scheduled down time. In addition, the rapid increase in regulatory changes meant that banks were being pressured to adapt their legacy systems quickly without sacrificing stability.

The emergence of digital banking services, the establishment of APIs based on Open Banking regulations, and the growth of cloud-based computing are all factors that have propelled the rapid adoption of Microservices as a means of Modernizing financial institutions' infrastructures. The Open Banking regulations establish requirements for the secure and scalable provisioning of APIs to third-party providers, while the cloud's dynamic scalability and immediate resource provisioning make it easier for banks to modernize their infrastructures through Microservices [4]. Consequently, many financial institutions are breaking apart their legacy Monolithic systems and replacing them with Microservices that can be deployed independently. This approach to infrastructure Modernization improves those institutions' Agility, Availability, and Operational Resilience [1][2]. Additionally, by aligning Microservices with individual and distinct Business Capabilities, Microservices enable the Horizontal Scaling of an Organization's operations and the separation of Faults within a System, thus creating Space and Opportunity for FASTER cycles of Innovation while allowing financial institutions to modify or replace individual Service Components to meet their Business Needs without affecting their entire System [3].

Although Microservices offer significant Benefits, migrating Core Banking Systems onto Microservices still presents significant Challenges and Risks to the financial institution. Core Banking Systems represent the financial institution's core business and production Systems and have the potential for disrupting Services and/or the flow of funds if the Migration is not properly planned, executed, and governed. Core Banking Systems are almost always deeply integrated within the regulatory controls of the financial institution and have become increasingly more complex as a result of decades of incremental Development. Therefore, it is critical for successful Migration Initiatives to provide uninterrupted Service continuity, ensure Data Integrity and Transactional Correctness, as well as provide for secure and strictly auditable Migration processes. Introducing Architectural Flexibility without Introducing Instability requires careful planning, incremental execution and strong governance.

This research paper investigates a number of the most important experiences that banks have obtained from moving their mainframe systems into microservices from both a technical and an organizational perspective. The paper discusses the technical challenges banks face when breaking up their services into smaller microservices, decoupling the information used by those services, and managing data across geographically distributed sites as well as managing operational complexity [6][7]. The paper also addresses the organizational requirements in developing a microservice-based architecture including restructuring of teams, developing new skills and competencies required for microservices development, developing new governance models, and complying with regulations in a heavily regulated environment like banking. The paper is based on current research, and contemporary research as well as evidence from real-life banking institutions illustrates the best practices and patterns for banks to enable them to safely modernize while maintaining the level of reliability and compliance that is required to operate in a highly regulated banking environment.

II. PROBLEM STATEMENT

Legacy Banks, with their emphasis on centralized databases, were mostly developed using tightly coupled monolithic applications. The Tight Coupling of business logic, presentation, and data access results in a single point of failure where changes in any of these functional areas can affect other functions within the application. As banks have grown, the number of features, regulatory compliance, integration capabilities, and so on, has increased as well. As a result, the monolithic codebase continues to increase in size and complexity until it becomes virtually impossible to maintain, scale, or evolve the application in a controlled way.

Due to this Tight Coupling between functions, a change made to one function will usually need extensive analysis, a full regression test of the entire application, coordinated releases between multiple teams, and so on. In addition, the time line for deployment cycles is extended and less frequent; therefore, the opportunity to identify and fix defects is decreased, making it challenging to react rapidly to changing business or regulatory requirements. Additionally, scaling to accommodate variations in demand is also inefficient because the system is provisioned as a whole as opposed to provisioning only for the specific area(s) experiencing high levels of traffic. All of these issues combined to create increased Operational Risk and Technical Debt for Large Banks.

As a result of the difficulties mentioned above, many financial services organisations are beginning to follow a microservices strategy to modernise their systems [1]. The concept of microservices is that breaking down a large system into smaller, individual services, each linked to an individual capability (business capability) provides for far better management than that found in a large system which has a common database. This also allows for a shorter time period to be used to send out new systems and therefore makes it easier for an organisation to scale up its operations by rapidly building cloud-native environments [4] as well as the possibility that the larger systems will be able to utilize the underlying cloud infrastructure.

The greatest challenge when converting to this type of architecture is being able to achieve it without affecting the organization's business operations, the ability to continue to meet regulatory standards and keep your data secure. Identification of service boundaries and ownership of responsibilities is vitally important when breaking down tightly coupled business logic [8]. The introduction of separate databases complicates this further, as financial systems require very strict consistency and transactional requirements. The complexity is further increased when communication between distributed services must occur, managing the impact of failure, latency and data synchronization.

Simultaneously, organizations also need to retain critical non-functional requirements during the migration process; therefore, maintaining governance over data and applying security controls to ensure compliance with audits is paramount. Additionally, the operational reliability of services must be preserved through increased distribution of systems, and teams will require a transition to new development patterns and deployment methodologies. For this reason, organizations will need to reach higher DevOps maturity, including implementing automated tests and continuous integration/continuous delivery (CI/CD) [10] along with extensive monitoring to ensure proper utilization of resources. These factors all indicate that migrating from monolithic banking systems to microservices is a complex change requiring an entirely new approach to architecture, operation and the organizational structure of all impacted personnel.

III. CHALLENGES

Technical Challenges

Service Decomposition

The most complicated and significant challenge in transitioning to microservices from a monolithic banking solution is defining suitable service boundaries [6]. In legacy systems, business logic is usually interdependent among various functions so isolating responsibility is extremely hard. Ineffectively defined service boundaries can lead to:

1) Increased inter-service communication
2) Greater data redundancy
3) Increased latency

Ultimately this negates the benefits of microservices. This issue is especially challenging within a micro-frontend composition since client applications generate a high volume of request when they are retrieving real-time updated data from their associated backend services and are also maintaining a set of ordered interactions of those services.

To separate services effectively, an organization needs to fully understand their business processes and domain connections. Domain-Driven Design (DDD) [8] provides an organized structure by which organizations can achieve this by defining service boundaries based on business capabilities instead of by way of technical layer definitions. By creating bounded contexts, and aligning services around specific business domain functions, organizations decrease the amount of coupling and increase the amount of cohesiveness between services. DDD encourages ongoing collaboration between domain experts and engineers, so that technical implementations adequately represent banking concepts in the real world, e.g., accounts, transactions, payments and lending. Additionally, using a single shared ubiquitous language throughout the organization promotes better understanding, reduces confusion and, supports the long-term maintenance of the overall system [8].

Data Decoupling and Consistency

In the past, banks used large centralized systems that were built around relational databases to enforce a strong system for managing transactions and keeping data safe. This made it easy for the way a bank managed its consistency but it created a close relationship between parts of the system, thus limiting the ability to scale. In contrast, a Microservices architecture has a decentralized data model where each microservice owns their own data and therefore introduces new issues with consistency across many services [12], especially with regards to a financial services environment where the requirements for precision and reliability in the operation of financial transactions are very high.

To deal with these requirements of consistency and resilience in their systems for managing financial transactions, banks typically implement architectural patterns such as Saga Orchestration and Event Sourcing [12]. The Saga architectural pattern allows banks to organize their distributed transactions by breaking them into a series of local transactions where each local transaction is managed by a microservice. The use of compensating transactions also provides a way to recover from failure scenarios. An Event Sourcing approach allows services to model all state changes in the system as an unchangeable stream of events where a service can recreate its state from the historical state changes and thus maintain its consistency over time. Both of these architectural patterns provide for the eventual consistency model and the increased resilience of the bank's systems, however, these new patterns introduce increased complexity with respect to monitoring, governance and error handling. In the banking industry, a mismanaged transaction can have serious implications and therefore require a high level of governance and control over the way these architectural patterns are designed and implemented within the banking industry.

Operational Complexity

The complexity of operating in a microservices environment arises from the fact that most application components are designed to communicate with each other over a network instead of all being bundled together within the same piece of code (as in traditional applications) [7]. Network latency, partial failure of services, and cascading dependencies can cause service outages, which is not the case with traditional systems. Therefore, operating in a microservices environment requires sophisticated tools and techniques to provide the level of reliability expected by customers.

To effectively manage the complexities of operating in a microservices architecture, organizations must implement strong observability [10]. Centralized logging, distributed tracing, and comprehensive metric collection are vital for providing visibility into how services interact with each other and how the overall system behaves. If organizations do not implement these practices, it takes longer to identify the cause of a service failure or a performance bottleneck, causing operational time to recover from an issue to increase. The operational reliability of microservice banking platforms is based on proactive monitoring and automated recovery mechanisms, and establishing clear incident response processes that can successfully respond to failures without affecting mission-critical financial operations.

CI/CD and Deployment

Building a microservices-based architecture means rethinking how we build and deploy software [10]. In a microservices-based approach, a single, monolithic release process (which usually means lots of coordination to deploy infrequently) is replaced with a CI/CD pipeline for each microservice that supports building, testing, and deploying independently. Each microservice needs its own automated testing strategy (that works for both the service and the pipeline), deployment process (to ensure it can roll back to a previous version), and an automated rollback strategy should the new version fail to deploy.

In addition to being a shift in how we think about software delivery, implementing this new approach will require significant investments in tooling, redesigning our processes, and making culture changes. Each team will need to implement automated testing at several different levels (i.e., unit tests, integration tests, system tests), continuous integration practices, and Infrastructure-as-Code (IaC) [10] to manage their deployment environment consistently. In the banking world, CI/CD pipelines must include security checks as part of the CI/CD process, as well as compliance checks before a deployment can take place; therefore, they must also include approval workflows so that we can meet the regulatory requirements of our industry. The success of making this shift will not only be technical in nature but will also require a culture change toward an emphasis on continuous delivery, shared responsibility for keeping the production environment stable, and accountability for what happens after a release goes live.

Organizational Challenges

Cultural and Structural Change

Microservices have revolutionised many different sectors and they've radically changed certain industries. The Banking industry in particular has already been through a structural and cultural upheaval due to the rapid shift of technological resources. Banks traditionally have always been organised around service delivery models that are centralised, where teams operate based upon their function. The organised communication structure of an organisation with three levels of hierarchical governance can present serious challenges when it comes to the implementation of microservices within the centralised model in that microservices require decentralised, product-oriented teams who are responsible for their services from inception to deployment and post-production support.

According to Conway's Law [13], the structure of an organisation directly affects the architecture of its systems as the way people communicate within an organisation ultimately determines how a system is designed and built, so building successful microservices on top of a centralised organisational model ultimately results in highly-coupled microservices, whether they be overlapping ownership or ambiguity with regards to responsibilities of a service. As such, achieving successful migrations will require an intentional redesign of the organisation at the same time as having established cross-functional teams who are assigned to a specific business domain. Each member of the cross-functional team must have the authority to make decisions independently of one another while also being accountable for the quality, reliability, and compliance of the service provided to the customers. Without this shift of culture, it will be nearly impossible to reap all of the technical advantages associated with microservices.

Skill Transformation

Adopting microservices requires that engineers and operators develop skillsets very different from those required previously [14]. Developers along with other Platform Engineers will need to possess the ability to utilize Cloud Platforms, Containerization Technologies [4], Orchestration Frameworks and DevOps Tooling to design and build distributed systems. This type of skillset goes beyond simply developing applications and includes an understanding of Infrastructure Automation; Observability; and Reliability Engineering.

As teams migrate to microservices, they will have to learn how to use these new tools, operate in a new environment with different failure modes, work with increased levels of complexity with distributed systems, and adapt their methodologies. In addition, teams in regulated banking environments must deal with the potential for huge financial and compliance issues as a result of errors caused by having an incomplete knowledge of the skills required to support microservices. To help teams develop their abilities and build confidence while continuing to maintain operational stability, teams will benefit from having structured training programs that include mentoring, as well as gradual strategies to introduce microservices into their organizations.

Governance and Compliance

Governance and compliance can be among the greatest challenges to an organization's ability to implement microservices architecture in banking organisations. Banks have extensive regulatory frameworks to adhere to regarding the auditability, access, protection of data and operational resilience of their systems. With traditional monolithic systems, regulations and other controls are typically controlled and enforced through one central area. Therefore, it is relatively easy for banks to govern and monitor their compliance using this type of system.

The complexity of the microservices architecture is that instead of one centralised governing body directing the activities of all services involved in the solution, there are numerous independent services being developed and managed by several teams [5]. This increases the potential for divergent designs, which then leads to inconsistently enforced policies. Regulatory compliance and auditability become difficult when there is no clarity in the designated direction or when there are no restrictions on the manner in which a team can develop service(s) to meet the compliance requirement. Governance needs to be adapted to provide a method for managing the services and associated technologies developed through a microservices architecture, while allowing the team to leverage their own creativity. Instead of managing every design aspect of a microservices-based service, banks should outline and provide standardised policies, reference architecture, and automated processes that are uniformly applied to all services.

By doing this, banks can enable teams to innovate and independently while also adhering to regulatory compliance and other security mandates. The use of automated compliance checks, standardised CI/CD pipelines, and shared platform as a service capabilities help provide a strong balance between autonomy and control. In summary, a successful governance process for microservices based design and development is dependent upon: 1) aligning the organisation's structures, processes, and rewards to meet regulatory obligations, and 2) following architectural best practices.

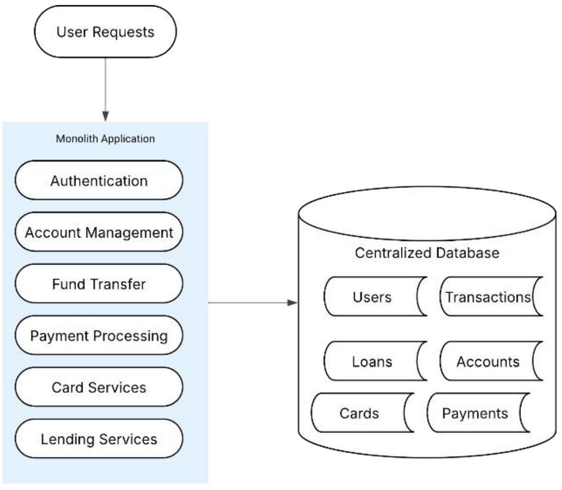
IV. SOLUTION APPROACH

Migrating from a traditional monolithic banking system to a microservices architecture can be done by using a well-considered and phased approach rather than attempting to create a new application that replaces all features of the existing one [1]. Modernization of banking systems requires consideration of factors, such as maintaining regulatory compliance, operating within a regulated environment and ensuring operational stability. Migrating to a modernized application in an incremental manner allows financial institutions to maintain the highest level of customer trust while limiting risk associated with migration.

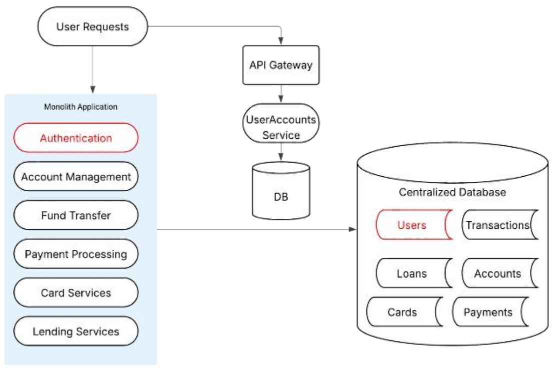
Incremental Decomposition (Strangler Fig Pattern)

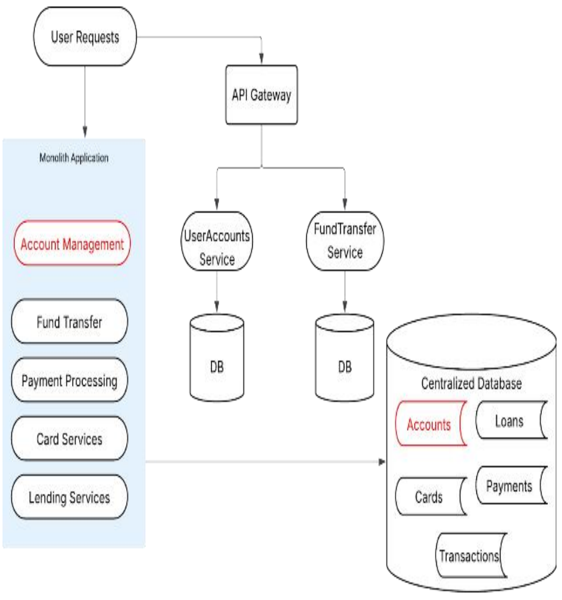
The Strangler Fig Pattern [1] allows banks to make incremental modernizing changes to their current systems without disrupting what they already do. Through this technique, a bank can implement new microservices on top of the old monolith while directing traffic for specific functions to the microservices instead of the monolith.

Typically, the large-impact functions of a bank that are still fairly isolated from one another, like user authentication, payment processing, and account inquiry, will be moved first using this Pattern because they typically have less complexity and few dependencies on other parts of a banking system. Once a function is moved to an API or routing layer and begins receiving traffic, the monolith is gradually reduced in scope. This will enable banks to migrate their most important capabilities without the risk of shutting down or encountering pervasive bugs.

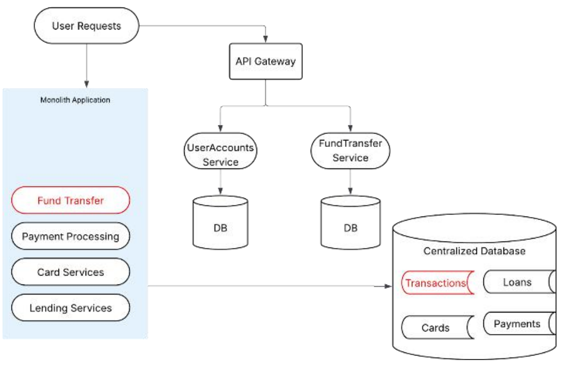


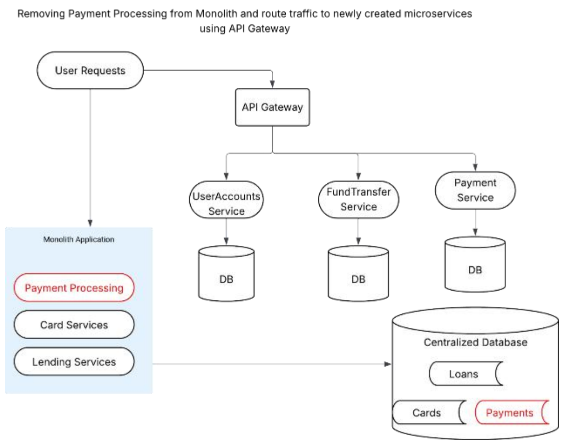
Removing User Authentication operations from Monolith and route traffic to newly created microservices using API Gateway

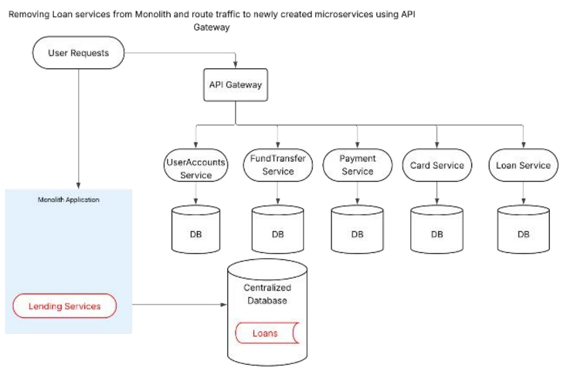


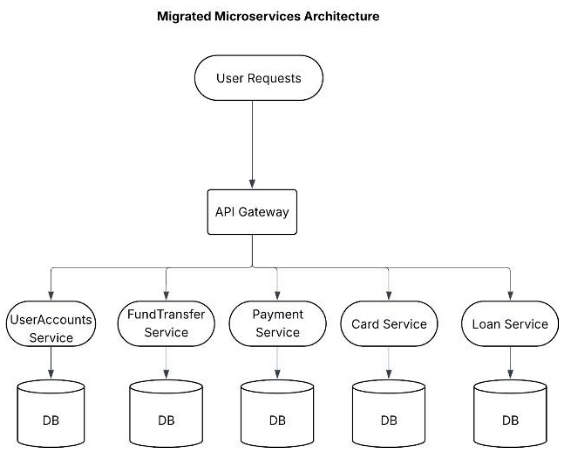


Removing Fund Transfer from Monolith and route traffic to newly created microservices using API Gateway









Having a gradual approach to migrating functions allows the business area and technical team time to further test their architectural decisions, improve operations, and build confidence as they transition more complex functions.

Domain-Driven Design (DDD)

Domain-Driven Design [8] supports the alignment of digital banking system architecture to core banking domains like accounts, transactions, payments, lending, and customer profiles by providing an architecture model that understands the needs of the banking business.

Instead of decomposing digital banking systems based on technical layers alone, DDD emphasizes capturing the business model as a holistic representation of the banking capabilities and captures the business process ownership by defining the actual boundaries of ownership of business capabilities as they exist in the real world.

The key concepts of DDD are essential to successful migration to microservices. Boundaries between Domains, or "bounded contexts," clarify the domain boundary and the consistent definition of models and key terms [8]. Context mapping clarifies how Services interact with each other and how they transfer data, providing clarity around their interfaces and reducing ambiguity, risk of unintended coupling, and the potential for misunderstandings.

The use of a "ubiquitous language" [8] creates a common understanding of the business concepts and provides opportunities for effective communication among Domain Experts, Developers, and other Stakeholders, which reduces the chance for confusion and misunderstandings.

The application of DDD also provides organisations the ability to design Services that are cohesive, loosely-coupled to one another and well-aligned with their organizations area of responsibility. The DDD approach also allows for scalable design of Services, allows for easier Change Management and also improves future Maintainability of the overall System.

Decentralized Data Patterns

Microservice Architecture necessitates that organisations examine the challenge of moving away from a centralised model of data management and instead accept a decentralised ownership of data [12]. Each microservice takes ownership of its own data set, defines the right storage technologies for both functional and non-functional requirements, and manages data consistency for the application. Most importantly, Microservices can leverage polyglot persistence when a microservice chooses between relational databases, document database, nNoSQL database, or event stores based on the needs of the service for consistency, performance, and scalability.

In addition, Event-Driven Architecture (EDA) [12] embodies a loosely coupled approach by enabling asynchronous communication between services. If EDA is combined with Event Sourcing and Saga Orchestration, a highly distributed and multi-service architecture can achieve data consistency and independent evolution of microservices. In highly regulated environments such as Banks, the design of the Microservice architecture and decentralised data structures must follow strong governance to meet regulatory compliance, Accountability, and Auditability.

If governance is followed, decentralised data architecture provides the ability to scale and provide resiliency while also facilitating Regulatory Compliance, Traceability, and Accountability.

DevOps Enablement

To take full advantage of microservices, the banking industry must adopt DevOps [10]. Automation through CI/CD pipelines allows teams to deploy services as they are ready without affecting each other's development timelines, increasing the frequency of development cycles and securing reliable software delivery. Infrastructure as Code [10] provides a method to create environments consistently and repeatedly, thus minimizing configuration drift and reducing operational risk.

The standardization of deployment and scaling through container orchestration platforms [4] helps streamline these processes, while real-time system monitoring and observability [10] provide proactive performance issue detection and failure identification. The combination of these technologies leads to improved system responsiveness, resilience, and recovery.

The implementation of DevOps in heavily regulated banking environments must also include the integration of security scanning, compliance checks, and approvals into the DevOps pipeline workflow to maintain governance. By incorporating controls into automated workflows, banks will have the ability to achieve the benefits of increased delivery velocity without sacrificing regulatory assurance, ultimately allowing for the scalable modernization of the banking industry.

V. RESULTS

Evidence from research and industry is abundant and clear: when organisations transition from monolithic banking systems to Microservices, they benefit from numerous operational dimensions that allow for better performance than before [9]. Examples of these are deployment frequency; instead of performing quarterly or other infrequent deployments, organisations can now conduct weekly or daily deployments because they are able to deploy each service independently. As a result, teams can implement incremental improvements more quickly, respond to dynamic regulatory requirements faster than before, and mitigate risks associated with deploying multiple components at once.

In terms of scalability, it is also quite noticeable how much better microservice-based architectures provide increased scalability than traditional monolithic banks [9]. When scaling up a monolithic banking application, typically additional resources are consumed by the entire application at one time as opposed to only the individual microservice being scaled. As an example, many of the services offered by banks include payment and account inquiry functionalities that have fluctuating transactional volumes and traffic patterns; therefore, a microservices architecture allows for individual services to scale based on the demand being placed on them. In addition, the ability to independently scale microservices provides significant infrastructure cost savings since the infrastructure costs associated with a peak volume of transactions will be significantly lower than what would be incurred if the application was to scale as a complete monolith.

The resilience of migrating to the microservices architecture was another valuable outcome [2]. The separation of functionality into distinct components results in less chance that a failure of one will have a cascading effect across the entire system, which is referred to as fault isolation. Along with automated recovery and monitoring systems, this results in reduced risk of system-wide downtime and increased overall service uptime. This increased resiliency is extremely important for banking platforms that require constant operation and adhere to the strict regulatory requirements of uptime.

Maintainability will improve as the monolithic structures are broken up into smaller, modular services with specific functions associated with those services [1]. Modularization will reduce the risk of long-term technical debt because it will allow for easier understanding, testing and evolution of code within each service. Additionally, teams can concentrate their efforts on improving individual services without impacting other parts of the system that may not need improvement, thus enabling continued evolution of better and more sustainable software development over the long term.

When combined with the above, these benefits show that when properly governed and managed, a microservices migration can provide an organization with significant operational advantage, while at the same time supporting the compliance and reliability needs of the modern-day banking world.

6. Case Study: Core Banking Platform Modernization

A significant financial institution is currently modernizing its Core Banking Application that was built using a monolithic Java Application. The Core Banking Application is essential to run all of the core functions of the financial institution, such as Account Lifecycle Management, Customer Onboarding, Compliance, Risk, and Payment Processing.

As time went on, the Core Banking Application has become increasingly complicated due to the number of tightly coupled modules, usage of a single Shared Relational Database, and quarterly release schedule. As a result, the financial institution was unable to quickly respond to regulatory changes and/or changes from the business.

The main objectives for the modernization program were to provide improved delivery agility, increased resilience, reduced operational risk, and enabling the required Cloud-Native Scalability [4] of the Core Banking Application. Since the Core Banking Application is critical to the bank, an incremental migration strategy was adopted that focused on service continuity, regulatory compliance, and mitigating operational risk.

Designing Domains and Service Boundaries

A Modern Banking System is implemented through Domain Driven Design (DDD) analysis [8]. Well-defined bounded contexts are constructed to identify the core banking functions and the business domains they operate under. The banking functions are defined by the following Customer Onboarding, Payment, Account Management and Regulatory Reporting. Once defined, the banking functions and the associated business domains established the boundaries for each of the microservices. Each microservice has defined boundaries; therefore, all items related to a business domain are included within a microservices and very few dependencies between one microservice and other microservices exist.

Service boundaries were established through a culture of shared ownership. By establishing a shared ownership of a service by all banks' employees, a common language [8] between business stakeholders and the engineering teams was developed. The development of a shared ownership culture enhanced communication between teams, ensuring there is no confusion regarding how to implement a requirement, and ensuring there are no discrepancies between the terms used by a bank and the terms used by the regulators.

Cross-Functional Services Ownership

To implement the new architecture of the bank, the employees were realigned around services, instead of being organized into functional silos [13]. Each micro service was owned by a cross-functional team that was responsible for developing, testing, deploying, supporting and maintaining it. The establishment of the ownership model led to increased accountability and reduced the number of hand-offs between teams, which enabled faster decision-making and improved accountability for service operations.

With the establishment of a cross-functional service ownership model, the teams were also able to work within defined governance guidelines to manage their own release schedule while maintaining compliance with the governing body. Establishing a culture of shared ownership significantly improved service delivery and allowed the bank to realize the maximum benefits associated with the use of microservices.

Event-Driven Approach to Data Consistency

One of the biggest technical hurdles was to break apart the shared monolithic database while continuing to maintain transactional consistency for financial workflows. The bank implemented event-driven communication patterns and Saga orchestration [12] to manage distributed transactions across services. Each service owned their data and published events for other services to be notified of state changes in their own data (for example, payment transactions or account updates). As a result, all workflows could continue to be consistent without the need for distributed database transactions.

While this method enhanced the resiliency and scalability of the overall system, it needed to be governed, monitored and have reliable failure management in place. To provide assurance of accuracy and traceability (which are critical in financial systems), the bank implemented compensating transactions, idempotent event processing and audit trails.

CI/CD Pipeline Implementation & Observability Foundations

In order for independent service deployment to be possible, the bank invested heavily in DevOps enablement [10]. The bank established an automated CI/CD pipeline for each microservice including unit test, integration test, security scan and compliance checks. It used Infrastructure as Code [10] to standardize the environments and eliminate configuration drift across development, test and production stages.

In conjunction with this effort, the bank established a complete observability framework [10] with centralized logging, distributed tracing and real-time metrics. This type of capability enables teams to monitor how services behave, identify and resolve issues quickly and decrease mean time to recovery.

Results and Lessons

The modernization process provided the bank with quantifiable metrics in terms of higher delivery speed, a more stable enterprise system, and an increase in operational visibility. The frequency of deployments increased greatly with the shift from a coordinated release every quarter to independent deployments by the individual teams. The better fault isolation capability minimized the negative impact of an issue occurring in a service, and the increase in observability enabled quicker incident resolution. All of these increased capabilities were vital to the bank maintaining compliance and being able to provide an audit trail while going through the modernization efforts, as a result of strong governance and traceability mechanisms.

This particular study exemplifies that a successful modernization of a Core Banking system cannot happen without not only changing the architecture but enhancing the entire service delivery production ecosystem. The combination of Domain-Driven Design [8], Service Ownership, Event-Driven Data Patterns [12], and DevOps Enablement [10], in conjunction with Organizational Transformation, was fundamental in achieving sustained successful outcomes in the modernization effort. This evolution supports the conclusion that Microservices Migration is a Holistic Transformation initiative which requires the complete interplay of Technology, Operations, and Organizational Structure.

7. Outcomes

The modernisation of the core banking platform's technology stack had a significant positive impact on business operations through the transition from a monolithic architecture to a microservices based architecture. The following represent the three main operational metrics of this transition, including:

Release Frequency

In the previous monolithic architecture, changes were released every quarter because all components within a single unit had to be tested and deployed together. Consequently, changes of any size required multiple teams to coordinate their work heavily before being able to deliver the change. This resulted in an increase in the time that it took to deliver a change. After transitioning to the microservices architecture, the frequency of releases increased to weekly, or about 6 times the amount of releases of the previous release model [10]. This increase was due to the ownership of independent services, and the fact that there were independent deployment pipelines for each service, enabling each team to deliver smaller, incremental changes to the services independent of full platform releases. As a result, the increase in the frequency of releases enabled faster delivery of new features and functionality, quicker regulatory updates, and reduced risk with large bundled releases.

Deployment Lead Time

The average deployment lead time during the previous monolithic architecture was approximately 4-6 weeks, including time for development, integration testing, approval workflows, and coordinated planning for a single release. The average deployment lead time with the microservices architecture was 1-3 days, an approximately 85% reduction [10]. The decrease in deployment lead time is attributed to automation being incorporated into Continuous Integration and Continuous Delivery (CI/CD) processes, service-level testing, and Infrastructure as Code (IaC). Shortened deployment lead times allowed the organisation to rapidly respond and deploy changes to meet customer and regulatory needs while providing the same level of reliability associated with deployments.

The failure of a change

The monolithic architecture type exhibited 18 per cent to 22 per cent failure rate related to change. This was caused mainly as a result of the dependencies created by making changes across multiple tightly-bound modules. What was observed was that defect occurrence in one problematic area could lead to a defect occurring in other modules with no apparent relation, thus increasing the probability of negative outcomes in production. Once the system was migrated to a micro-service-based architecture, the failure rate of change dramatically decreased to 7 per cent to 9 per cent, an overall percentage reduction of approximately 60 per cent [2]. Fault-isolation improvements, smaller change-scope maintenance, and testing based on service provided greatly decreased the probability of experiencing a deployment failure.

Rollback duration

The time required to complete a rollback operation within a monolithic system ranged from approximately two to four hours; this was due to the entire application needing to be redeployed and the state of shared database(s) needing to be restored. On the other hand, the use of Microservices allowed rollback operations to be completed in less than 15 minutes, resulting in a percentage improvement of approximately 90 per cent [10]; because services ran independently via a service-level deployment model that used the concept of immutable infrastructure, this allowed for rapid reversal of the service without

affecting other services. Because of this rapid turnaround, the operational risks associated with rollbacks have been significantly reduced, along with greatly enhancing overall incident response efficiency.

Overall, the metrics demonstrate quantifiable improvements in speed of product delivery, reliability, and operational resiliency achieved by changing from a monolithic to a microservice-based architecture. From this data we know that by the investment in quality governance and strong DevOps practices [10], substantial enhancements can be made to the baseline functionality of a core banking application while maintaining and preserving its integrity.

Metric	**Monolith**	**Microservices**	**Improvement**
Release frequency	Quarterly	Weekly	~6×
Deployment lead time	4–6 weeks	1–3 days	~85%
Change failure rate	18–22%	7–9%	~60%
Rollback time	2–4 hours	<15 min	~90%

VI. CONCLUSIONS

The transition of monolithic banking systems into microservice based solutions is a significant change and should not simply be seen as an architectural transformation [1]. By providing numerous benefits such as scalability, agility and cloud readiness, Microservices will only deliver on these promised benefits if the process is treated comprehensively (i.e. – encompassing Operations, Governance and Organisational Culture) and as a complete change to the entire business model.

Accurately Modernisation through Microservices requires a disciplined approach to Service Decomposition (developing a clearly defined Service Portfolio) and an ability to decouple Data from Functions [6]. Additionally, distribution of

Transactions must be managed effectively. Domain Driven Design (DDD) [8], Saga Orchestration [12], Event Driven Architecture [12], and Incremental Migration (using the Strangler Fig Pattern) [1] are some of the key methodologies used today to mitigate Risk, while ensuring Transactional Integrity, as well as, overall System Stability. Without these methodologies, Microservices may lead to increased Fragmentation, decreased Performance, and increased Operational Instability [7]; especially in highly regulated industries like Banking.

Additionally, organisational aspects related to migration must be considered. Thus, based on these findings, microservice Based Architecture changes may necessitate changes in team structures, competencies, and governance processes [13]. Using cross functional, service aligned teams with complete ownership from start to finish helps organisations make quick decisions and be accountable for their work. DevOps Practices [10] establish the automation and observability needed to use distributed systems effectively. Governance needs to shift away from a centralised/manual approach to a policy-based, automated guardrail framework that supports regulatory requirements but does not limit innovation [5].

The case study evidence supporting this paper provides measurable increases in frequency of deployment, lead time, resiliency and maintainability when organisations are able to align both technical and organisational transformations [9]. By supporting incrementalised modernisation efforts, Banks can achieve continuous innovation, while also maintaining the ability to Audit, and fulfill Regulatory Requirements. Most importantly, implementations used in this study demonstrate that an implementation can be completed without sacrificing stability for speed. The Governing Body provides the Guidelines to implement a future focused model.

Overall, Financial Institutions that treat the migration to microservices as a complete transformation, utilising Architecture Discipline, DevOps Enabling Technologies [10], Organisational Redesign [13], and Compliance Centring Governance Mechanisms [5], will have the best potential for continuing to Modernise their Core Platforms in a Sustainable Manner. As regulatory requirements continue to evolve and customers will increasingly demand digital banking solutions, Banks must adopt a holistic approach to Modernisation if they are to remain in business.

REFERENCES

[1] [1] Newman, S. *Building Microservices*. O'Reilly, 2015.

[2] [2] Lewis, J., Fowler, M. "Microservices." martinfowler.com, 2014.

[3] [3] Dragoni, N., et al. "Microservices: Yesterday, Today, and Tomorrow." Springer, 2017.

[4] [4] Pahl, C. "Containerization and the PaaS Cloud." *IEEE Cloud Computing*, 2015.

[5] [5] Taibi, D., Lenarduzzi, V., Pahl, C. *IEEE Cloud Computing*, 2017.

[6] [6] Fritzsch, J., et al. *IEEE Software Engineering Conference*, 2019.

[7] [7] Bogner, J., et al. *Journal of Systems and Software*, 2019.

[8] [8] Evans, E. *Domain-Driven Design*. Addison-Wesley, 2004.

[9] [9] Villamizar, M., et al. *IEEE Cloud Computing*, 2016.

[10] [10] Bass, L., Weber, I., Zhu, L. *DevOps*. Addison-Wesley, 2015.

[11] [11] Richards, M. *Microservices vs SOA*. O'Reilly, 2016.

[12] [12] Gysel, M., et al. *European Conference on SOCC*, 2016.

[13] [13] Chen, L. *IEEE Software*, 2015.

[14] [14] Thönes, J. *IEEE Software*, 2015.