International Journal of Computational and Experimental Science and ENgineering (IJCESEN)

Vol. 11-No.4 (2025) pp. 8585-8593
http://www.ijcesen.com

ISSN: 2149-9144

Research Article

Digital Transformation in Healthcare Technology: Modernizing Legacy Expense Management Systems

Laxmi Pratyusha Konda*

Independent Researcher, USA.
* **Corresponding Author Email:** reachpratyu.k@gmail.com- **ORCID:** 0000-0002-5997-7850

Article Info:

DOI: 10.22399/ijcesen.4269
Received : 25 September 2025
Revised : 25 October 2025
Accepted : 11 Novemeber 2025

Keywords (must be 3-5)

Cloud-native architecture,
microservices,
healthcare expense management,
digital transformation,
legacy system modernization

Abstract:

The healthcare technology market is under pressure to redesign its old systems that are not very scalable, have integration issues, and are not very efficient in manual processing. This article is a case study of the change of a legacy healthcare expense management system to a cloud-native microservices platform in a large financial services company. The project substituted a single-facade legacy screen scraper system with the medical expense management application based on the use of recent technologies such as Java/J2EE, Spring Boot, REST/FHIR APIs, and Microsoft Azure. The transformation delivered impressive results in accuracy of automated file processing, a decrease in manual intervention, an improvement in performance, and user engagement. The exploration looks into implementation methodology, technical architecture choices, challenges, measurable results, and lessons learned that can be used in similar digital transformation programs in healthcare technology platforms. The case reveals that well-considered modernization of the legacy can provide significant gains even with the already-present challenges, and it can be used to build premises of continuous innovation with better agility, scalability, and developer productivity. Some of its critical success factors came out, such as architectural choices that took into account cloud-native microservices, strict engineering practices, agile-based practices, and continuous people investment in terms of skills, change management, and team morale. With the future of healthcare technology being value-based care and interoperability standards, consumer-centered experiences, the platforms founded on a modern architectural basis will become a more and more distinguishing feature between the market leaders and organizations that are still bound by the shackles of legacy.

1. Introduction

The healthcare technology field is facing a more sophisticated problem: to unify the limitations of the old system with the growing need for real-time processing, integrations, and better user experiences. An example of such tension between operational need and technological obsolescence is healthcare expenses management platforms that process millions of transactions each year on behalf of participants in the Health Savings Account (HSA) program, employers, and insurance companies. Research examining the relationship between software delivery performance and organizational outcomes demonstrates that elite performing organizations deploy code 46 times more frequently than low performers, with lead times for changes that are 2,555 times faster, while simultaneously achieving failure rates that are seven times lower and recovery times that are 2,604 times faster [1]. Many financial services institutions continue to operate expense management systems built on outdated architectural paradigms that fundamentally limit scalability, integration capabilities, and innovation potential, often allocating 60-80% of their technology budgets merely to maintain existing systems rather than driving innovation.

The transition toward cloud-native microservices architectures represents more than a mere technological upgrade; it constitutes a fundamental reimagining of how healthcare expense management systems can deliver value to stakeholders. The microservices architectural pattern decomposes applications into loosely coupled services that implement specific business capabilities, enabling organizations to structure engineering teams around these services and deploy

them independently [2]. This architectural approach addresses critical challenges inherent in monolithic architectures, including the difficulty of understanding and modifying large codebases, the obstacle-laden path to adopting new technologies due to framework lock-in, and the fundamental inability to scale individual application components independently based on their specific resource requirements [2]. However, this transformation pathway presents formidable obstacles that organizations must carefully navigate, balancing innovation imperatives against operational continuity requirements while maintaining data integrity and regulatory compliance—particularly critical in healthcare environments governed by interoperability standards such as Fast Healthcare Interoperability Resources (FHIR).

This academic analysis is a case study of a massive digital transformation project in one of the biggest financial services companies. The project entailed breaking down an old healthcare cost tracking system (designated legacy screen scraper system) and installing a new generation cloud-native medical expense management application. The change made use of modern technologies such as Java/J2EE, Spring Boot, RESTful and FHIR-compliant rest APIs, and Microsoft Azure infrastructure to drive significant operational advances: 98% on automated file processing, 60% lowered manual intervention needs, 30% improved performance, and 25% improved user engagement metrics. These outcomes align with empirical findings indicating that high-performing technology organizations exhibit 2.5 times lower change failure rates and recover from incidents 24 times faster than their lower-performing counterparts [1]. The analysis proceeds by examining the legacy system's limitations, the architectural vision guiding the modernization effort, implementation methodologies, quantifiable outcomes, challenges encountered, and lessons derived from the transformation experience, offering guidance for organizations across regulated industries confronting similar legacy modernization imperatives.

2. Legacy System Constraints and Modernization Imperatives

2.1 Architectural Limitations of the Screen Scraper Platform

The incumbent legacy screen scraper application represented a traditional monolithic system architecture that, while functional, exhibited fundamental limitations characteristic of legacy platforms. The system enabled HSA participants to track medical claims, process payments, and upload receipts through a web-based interface, serving over 36,000 enrolled users annually. Despite achieving a 20% improvement in adoption rates through incremental enhancements, the platform's architectural foundations constrained future development possibilities. The monolithic architecture pattern, while offering the benefits of simplified development, deployment, and testing in early application stages, creates substantial barriers as applications scale beyond initial use cases [3]. The screen scraper system exemplified how monolithic applications package all functionality into a single deployable unit, whether as a single executable, web application archive (WAR), or enterprise archive (EAR) file, requiring complete redeployment even when modifications affect only isolated components [3]. This architectural approach resulted in the entire application stack being deployed as an indivisible unit, preventing the organization from independently scaling high-demand components or implementing targeted updates without comprehensive system-wide coordination.

The application's reliance on scheduled batch processing and screen scraping techniques for retrieving claims and accumulator data from external carrier portals epitomized the brittleness of legacy integration approaches. Although the implementation of a Scala-based parser improved data accuracy by 25% through enhanced validation logic, the fundamental methodology remained labor-intensive and vulnerable to disruption. Any modifications to carrier portal interfaces could precipitate integration failures, necessitating immediate remediation efforts. Furthermore, integration patterns utilizing combinations of REST APIs and web scraping created ongoing maintenance overhead inconsistent with sustainable operational models. The monolithic architecture's tight coupling between components meant that changes in one module could inadvertently affect seemingly unrelated functionality, requiring extensive regression testing across the entire application before any deployment [3]. From a technical architecture perspective, the monolithic design precluded independent scaling of discrete system components, thereby constraining resource optimization based on varying workload patterns. Although the introduction of CI/CD pipelines lowered the deployment time by 40 percent, coordinated release between modules that were closely integrated was still required, which restricted deployment agility. These architectural limitations basically hampered the ability of the organization to react quickly to the changing market demands and competitive forces.

2.2 Converging Pressures Necessitating Transformation

Several driving forces came together to make a strong business argument for extensive platform change instead of further gradual improvement. The expectations of the users had changed dramatically, and the participants started to require mobile-first experiences, real-time updates on the status of the claims, and smooth cross-device functionality. The batch nature of the processing paradigm exemplified by the legacy system was deeply incompatible with the user demands, as demonstrated by the modern digital experience of immediate response and constant availability. Operational inefficiencies had a physical cost to the organizational resources because manual intervention requirements to process files took up a large amount of analyst capacity, which could be re-allocated to more valuable tasks like exception investigation, process improvement initiatives, and better customer service. Even with the improvement made in the error rates by the enhancement of the validation logic, they still had to be reviewed and corrected by humans, creating bottlenecks in the operations.

Regulatory and industry standardization trends introduced additional pressures for modernization. The healthcare industry's progressive adoption of FHIR as the standard framework for health data exchange created expectations for interoperability that legacy API patterns could not readily satisfy. FHIR represents a next-generation standards framework designed to enable electronic exchange of healthcare information through a common set of resources, an extensible data model, and a RESTful API architecture supporting structured data exchange across disparate healthcare systems [4]. Research examining FHIR implementation demonstrates that the standard facilitates interoperability through resource-based representations of clinical and administrative concepts, with specifications defining over 140 distinct resource types encompassing patient demographics, clinical observations, diagnostic reports, medications, procedures, and billing information [4]. The FHIR specification supports multiple data formats, including JSON and XML, enabling flexible integration patterns while maintaining semantic consistency across implementations [4]. Completing FHIR compliance by changing existing screen scraper architectures seemed less and less feasible than a ground-up redesign, because the underlying impedance mismatch between old integration patterns and the resource-oriented FHIR model would have been all too much work, and would have touched practically every integration point. The competitive aspect of the HSA market also accelerated the modernization imperatives, with several providers seeking employer deals and individual participants' enrollments, making the market situation where the capability of quickly implementing differentiating features became strategically important.

3. Architectural Vision and Implementation Methodology

3.1 Cloud-Native Microservices Architecture

The cloud-native microservices architecture is the paradigm chosen by the medical expense management application initiative, which was a conscious break with the monolithic design patterns. This style of architecture allowed discrete system capabilities to develop, deploy, and scale independently, and directly overcome the flexibility limitations that curtailed the evolution of the legacy platform. The microservices paradigm enabled the agility of the organization, as autonomous teams were able to possess full verticals of functionality, lessening the overhead of coordination and increasing the speed of delivery. Microservices architecture is a type of cloud-native application design, which organizes applications as a set of loosely coupled services, each with particular business capabilities and communicating using lightweight protocols like HTTP/REST API [5]. This architectural pattern enables organizations to build applications composed of small, independent services that run as separate processes, allowing development teams to work autonomously on different components without extensive coordination requirements [5].

The technical stack selection balanced proven enterprise stability with contemporary development productivity. Java/J2EE and Spring Framework were the platforms with strong business logic implementation foundations, featuring a mature ecosystem and a rich collection of libraries. The use of convention-over-configuration concepts and built-in support of containers meant that Spring Boot made microservices development easier with less boilerplate code and faster development. REST APIs served as the primary integration mechanism for internal service communication, while FHIR-compliant APIs addressed external carrier and vendor interactions, aligning with healthcare industry interoperability standards. Microsoft Azure provides comprehensive cloud infrastructure capabilities, including platform-as-a-service offerings, managed services, and global distribution capabilities supporting high availability

requirements. The microservices approach delivers several critical advantages, including the ability for development teams to select optimal technology stacks for specific services rather than being constrained by organization-wide technology standardization, enabling polyglot programming where different services can utilize different languages, frameworks, and databases based on their specific requirements [5].

Several architectural principles guided design decisions throughout the implementation. Service boundaries aligned with business capabilities rather than technical layering, enabling teams to deliver complete features without extensive cross-team coordination. Stateless service design facilitated horizontal scaling and simplified failure recovery by eliminating session affinity requirements. Asynchronous communication patterns using message queues decouple service dependencies, improving overall system resilience and enabling independent service evolution. Database-per-service patterns ensured data encapsulation, though pragmatic compromises allowing shared databases during transition phases balanced theoretical purity against delivery practicality. The microservices architecture's fundamental characteristic of service independence means that individual services can be developed, deployed, updated, and scaled independently without requiring coordination across the entire application ecosystem, dramatically accelerating innovation cycles and reducing deployment risks [5].

3.2 Core System Components and Integration Architecture

The medical expense management application platform comprised specialized microservices addressing distinct business capabilities within the healthcare expense management domain. The Carrier Consent File Manager generated weekly consent files representing authorization agreements between clients and insurance carriers, processing over 10,000 user records per execution cycle. The service was used to withdraw the data of the participants from PostgreSQL and Oracle databases, process the data based on the specifications of the required format of the carrier, and provide files through secure file transfer protocols. The overall validation regulations, data quality verification, and exception management systems were successful at 98 percent, which is a significant increase over the manual processing methods that are subject to human error. The Carrier Exception File Manager was used to process reject files periodically that were sent back by the insurance carriers, and claims or transactions were identified that could not be processed because of a data problem, eligibility issue, or policy issue. The service parsed carrier-specific file formats, correlated rejections to original transactions, and updated system status accordingly. Automation of this previously manual process reduced human intervention requirements by 60%, eliminating tedious analyst tasks involving manual review and system updates for rejected items.

3.3 Infrastructure, DevOps, and Operational Excellence

Microservices were packaged with their dependencies in Docker containers and ensured uniformity across the development, testing, and production environments, as well as making deployment processes easier. Kubernetes managed the deployment, scaling, and lifecycle management of containers, allowing a 30% performance boost using smart resource allocation policies and automatic scaling response to load changes. Kubernetes emerged as the standard orchestration platform for containerized applications, providing automated deployment, scaling, and management capabilities that transform how organizations operate distributed systems [6]. The platform enables velocity through its declarative configuration approach, where developers specify desired application states, and Kubernetes automatically handles the operational complexity of achieving and maintaining those states across clusters of machines [6]. This orchestration feature is specifically useful to microservices architectures in which dozens or hundreds of services need to be orchestrated, with Kubernetes automating service discovery, load balancing, and health checking that would otherwise involve a significant amount of hand operational overhead [6].

CI/CD pipelines that were executed through Jenkins automated the code-to-production deployment pipeline. Unit testing with JUnit, integration testing with Mockito, and API contract testing with Postman collections and automated security scanning were all pipeline stages. The inclusion of infrastructure-as-code solutions with Azure Resource Manager templates allowed deployment of environments with reproducibility and the ability to recover in case of a disaster, and approach infrastructure configuration with the same rigor as application code. Additional considerations were put on monitoring and observability during the implementation, where it is noted that distributed systems need to be well instrumented in order to be successfully operational. The distributed tracing feature allowed visualizing the flow of requests between more than two services and

allowed identifying bottlenecks in performance and troubleshooting. Kubernetes provides built-in abstractions for scaling applications horizontally by adding more container instances or vertically by allocating additional resources, with automatic load balancing distributing traffic across healthy instances while continuously monitoring application health and automatically restarting failed containers to maintain service availability [6].

4. Measurable Outcomes and Qualitative Transformations

4.1 Quantitative Performance Metrics

The new expense management application implementation delivered substantial measurable improvements across multiple performance dimensions, validating the transformation investment. Processing accuracy for the Carrier Consent File Manager reached 98%, representing a significant advancement over manual processes inherently susceptible to human error. The validation logic, automated data quality checks, and comprehensive testing protocols contributed to this reliability level. The remaining 2% error rate primarily reflected edge cases in source data quality rather than processing defects, suggesting that further improvements are required in source data governance rather than processing logic refinement. Cloud computing fundamentally transformed enterprise IT by delivering computing services, including servers, storage, databases, networking, software, analytics, and intelligence over the internet, enabling faster innovation, flexible resources, and economies of scale [7]. Organizations migrating to cloud infrastructure typically pay only for the cloud services they actually utilize, reducing operating costs, improving infrastructure efficiency, and enabling dynamic scaling as business needs evolve without substantial capital expenditure requirements [7].

The benefits of operational efficiency were also seen most dramatically, namely, in automated exception file processing, where the number of manual interventions decreased by 60%. This efficiency would be translated to hundreds of analyst hours saved per month, which could now be directed toward exception investigations, process improvement projects, and activities providing more customer support that are more in line with the capabilities of the analyst and the value added to an organization. The elimination of repetitive manual tasks additionally improved analyst job satisfaction and reduced error rates associated with attention fatigue in monotonous work. Cloud platforms deliver several critical advantages, including speed and agility, where vast computing resources become available within minutes rather than weeks or months, providing organizations with tremendous flexibility and reducing pressure on capacity planning [7]. The global scale inherent in cloud computing enables services to be delivered from geographically distributed datacenters optimized for performance, with cloud providers achieving economies of scale that translate into lower variable costs for customers compared to on-premises infrastructure investments [7].

System performance improvements of 30% resulted from the confluence of multiple optimization efforts: efficient microservice design patterns, database indexing and query optimization, strategic caching implementation, and Kubernetes-based intelligent resource allocation. Sub-second response times for claim processing enabled real-time user interactions, replacing previous batch-oriented delays that frustrated users and limited system utility. The platform processed 42,000+ user records weekly, contributing to a 25% increase in user engagement metrics for HSA participants. Improved user experiences, accelerated transaction processing, and mobile accessibility drove higher adoption rates and active usage patterns, with user satisfaction ratings reflecting positive reception of enhanced healthcare expense management capabilities. The cloud-native architecture was robust, as shown by the system reliability measures. The uptime of OAuth 2.0 authentication, FHIR-compatible APIs, and high-availability infrastructure of Azure reached 99.9, which is significantly higher as compared to the availability of legacy systems. Fault isolation was also enhanced by the microservices architecture, which avoided the cascading failures that would otherwise cause the compromise of entire systems as a frequent problem with monolithic architectures, where the failure of a single component spreads extensively.

4.2 Qualitative Organizational Transformations

In addition to measurable indicators, the change initiative also brought qualitative changes that have essentially redefined organizational potential and strategic orientation. The microservices architecture led to a drastic reduction in time-to-market when rolling out new features because it allowed them to be independently deployed without central coordination throughout the platform. This agility made the organization responsive to the market demands and the competitive pressures and turned software delivery into a limiting element to a strategic enabler. Microservices architecture

emphasizes organizing application functionality around business capabilities rather than technical layers, with each service owned by a small team responsible for the complete service lifecycle from development through production operations [8]. This organizational structure enables decentralized governance where teams make localized technology decisions appropriate for their specific service requirements rather than conforming to enterprise-wide standardization mandates that may not suit all use cases [8].

Developer experience improvements manifested in multiple dimensions. Well-defined service boundaries and modern technology stacks increased the productivity and professional satisfaction of the developers. The engineers enjoyed the experience of contemporary frameworks, cloud-native design patterns, and automated workflows to minimize toil and focus on value-adding activities. Fewer technical debts and less ambiguous codebases reduced the onboarding of new members of the team, enhancing the efficiency of organizational learning and effectiveness in knowledge transfer. The microservices architectural style has been adopted to facilitate polyglot programming and polyglot persistence, whereby various services use different programming languages, frameworks, and data storage technologies that are suited to a particular functional need [8]. By having this diversity in technology, teams can utilize the best tools to apply to each area of a problem and still have loose coupling via the standard communication protocols [8]. The stakeholders gained confidence during the process of multi-year transformation as they were updated by the transparent reporting, frequent shows of working software, and quantifiable progress towards the set goals. Executive sponsorship has continued to be strong, which is essential in terms of organizational commitment and resource allocation in order to support the transformation initiative in the long term. The improvement in competitive positioning was based on the increased capabilities of the platform and market leadership, as the updated platform offered the necessary grounds for competitive differentiation and market positioning.

5. Challenges, Mitigation Strategies, and Lessons Learned

5.1 Technical Challenges and Resolution Approaches

Legacy system migration while maintaining operational continuity presented substantial technical risks requiring careful management. The mitigation strategy employed phased rollouts, operating both legacy and modern systems in parallel during transition periods to ensure continuity. Feature parity analysis ensured the new platform matched legacy functionality before cutover, preventing capability regression. Data migration pipelines incorporated comprehensive validation mechanisms and rollback capabilities to minimize data integrity risks, recognizing that data corruption or loss could prove catastrophic in healthcare financial contexts. The DevOps handbook emphasizes that organizations successfully navigating digital transformations adopt deployment strategies enabling small batch sizes and frequent releases, with high-performing organizations deploying changes multiple times per day compared to low performers deploying monthly or quarterly [9]. These deployment practices reduce risk by limiting the scope of each change, enabling faster feedback loops, and facilitating rapid rollback when issues emerge, fundamentally transforming how organizations balance innovation velocity against operational stability [9].

Integration complexity is multiplied through coordination with numerous external carriers, each presenting unique API specifications and data format requirements. The development team addressed this challenge through abstraction layers and adapter patterns that insulated core services from integration variability, enabling consistent internal interfaces despite heterogeneous external systems. Comprehensive testing using mock services validated integrations before carrier production environments became available, preventing schedule dependencies on external system availability from constraining development velocity. Research demonstrates that comprehensive automated testing strategies prove essential for maintaining quality in rapidly evolving systems, with high-performing technology organizations investing 15-20% of development time in test automation infrastructure that enables confident refactoring and continuous deployment [9]. The integration of automated testing throughout development pipelines, rather than relegating testing to discrete phases, enables rapid detection of defects when remediation costs remain minimal [9].

Performance optimization required multiple iterative cycles to achieve sub-second response time objectives. Database query analysis identified inefficient queries requiring rewriting or appropriate indexing strategies. Caching strategies reduced redundant database calls for frequently accessed data. Application profiling tools pinpointed bottlenecks for targeted optimization efforts, enabling resource-efficient performance improvements. Load testing under realistic workload conditions validated performance

characteristics before production deployment, preventing performance surprises under actual usage patterns. Organizations implementing DevOps practices establish comprehensive telemetry and monitoring capabilities enabling proactive problem detection, with leading organizations instrumenting applications to expose performance metrics, business metrics, and operational health indicators that facilitate data-driven decision-making and rapid incident response [9].

5.2 Organizational Challenges and Change Management

The legacy technology skills would not be transferred to the cloud-native microservices systems, and considerable organizational investment in capability development was needed. To develop the required competencies, the organization adopted extensive training systems, career qualifications, and practical training workshops. The combination of having some of the most learned cloud engineers with members of the team moving out of legacy technology helped speed up the transfer of knowledge through the practical working environment. Accepting that learning curves would temporarily reduce development velocity proved necessary, requiring organizational patience and realistic expectation management with stakeholders. Continuous delivery implementation requires organizations to establish deployment pipelines automating the build, test, and release processes, enabling reliable software releases through comprehensive automation of build, test, and deployment activities [10]. The continuous delivery approach emphasizes maintaining software in a deployable state throughout development, with every code change triggering automated build and test processes validating that changes have not introduced defects or regressions [10].

Change management for users accustomed to screen scraper workflows required thoughtful support approaches. User research-informed design decisions ensured the new platform maintained familiar interaction patterns while introducing improvements, thereby reducing adoption friction. The phased rollouts were done in terms of early adopters programs, where feedback was obtained before the real implementation to perform an iterative improvement to the project on the basis of the real experience of the user. The transition was facilitated by comprehensive documentation, training resources, as well as helpdesk support, which takes into consideration that technical superiority is not the sole guarantee of future adoption. The coordination between cross-teams in the distributed sites spread across various time zones demanded intentional communication habits. Establishing core collaboration hours accommodated time zone differences while maintaining synchronous communication opportunities. Asynchronous communication norms and comprehensive documentation practices mitigated coordination challenges inherent in distributed work.

5.3 Critical Success Factors and Transferable Insights

Several critical lessons emerged from the transformation experience with applicability to similar modernization initiatives. Investment in observability capabilities from project inception proved invaluable, as implementing comprehensive monitoring, logging, and tracing retroactively in distributed systems substantially increases troubleshooting complexity. The continuous delivery methodology emphasizes that deployment pipelines should incorporate comprehensive automated testing, including unit tests, integration tests, acceptance tests, and performance tests, with quality gates preventing defective code from progressing toward production environments [10]. This testing pyramid approach ensures that the vast majority of defects are detected through fast-executing unit tests, with progressively smaller numbers of defects requiring more expensive integration and acceptance testing for detection [10]. Pragmatism in service boundary design balanced theoretical microservices ideals against practical delivery realities. API contracts as first-class artifacts improved frontend-backend coordination and integration testing effectiveness. The distributed teams were given the necessary rhythm by Agile ceremonies, keeping everyone on track and allowing continuous enhancement by means of frequent retrospectives. The management of technical debt demands strict discipline, and capacity allocation ensures that shortcuts are not accumulated, which will result in a progressive decrease in productivity.

Table 1: *Software Delivery Performance Characteristics [1][2]*

Performance Indicator	Elite Organizations	Low Performers	Architectural Enabler
Deployment Frequency	Multiple times daily	Monthly or quarterly	Microservices independence

Lead Time for Changes	Minutes to hours	Weeks to months	Loosely coupled services
Change Failure Rate	Minimal occurrences	Significantly higher	Fault isolation patterns
Recovery Time	Rapid restoration	Extended downtime	Service autonomy

Table 2: *Architectural Pattern Comparison [3][4]*

Architectural Aspect	Monolithic Pattern	FHIR-Compliant Microservices
Deployment Unit	Single indivisible package	Independent service components
Technology Flexibility	Fixed framework choice	Polyglot technology selection
Scaling Granularity	Entire application replication	Component-specific scaling
Integration Standard	Proprietary coupling	Resource-based interoperability
Data Format Support	Single format constraint	Multiple formats with semantic consistency

Table 3: *Cloud-Native Infrastructure Capabilities [5][6]*

Infrastructure Element	Traditional Approach	Cloud-Native Implementation
Service Communication	Direct coupling	Lightweight protocol APIs
Application Deployment	Manual configuration	Declarative orchestration
Resource Allocation	Static provisioning	Dynamic scaling policies
Technology Governance	Centralized standardization	Decentralized team decisions
Container Management	Manual oversight	Automated lifecycle control

Table 4: *Operational Excellence Dimensions [7][8]*

Capability Domain	Cloud Computing Benefit	Microservices Advantage
Resource Acquisition	Available within minutes	Service-specific optimization
Cost Structure	Pay for actual utilization	Independent scaling efficiency
Geographic Distribution	Global datacenter reach	Fault tolerance through isolation
Team Organization	Infrastructure flexibility	Business capability ownership
Technology Selection	Platform service variety	Programming language diversity

4. Conclusions

The effective migration of the previous screen scraper system to the cloud-based expense management platform proves that properly planned modernization of legacy projects can bring significant value even with the challenges and risks that they are associated with. The transformation registered notable quantitative results in processing accuracy, reduction of manual intervention, performance improvement, and increased user engagement, and foundations on continued innovation through better organizational agility, system scalability, and developer productivity. A number of critical success factors were revealed to be vital to transformation effectiveness. The technical foundations of the realized benefits of the realisation came as a result of architectural designs welcoming cloud-native microservices, containerization, and API-first design. Strict engineering principles, such as extensive testing, automation of continuous integration, deployment, and observability of operations, provided quality and reliability that were in line with the needs of the healthcare financial services. Cross-functional teams in Agile approaches brought about incremental value in dealing with the inherent complexity in large-scale system transformation. Above all, long-term investments in individuals in terms of skills training, change management, and maintenance of team morale transformed technical skills into materialized organizational performance. The latest medical expense management application domain is a unique area of concern due to regulatory mandates, complicated integration platforms, and data sensitivity needs. Nevertheless, the patterns and approaches to implementation, as well as the lessons that were shown, are not industry-specific. Companies in the controlled industries that are facing constraints of legacy systems can implement this change framework to suit their specific situation. This is a long process

that has to be supported by the organizational patience, the long-term investment, and the strong devotion, yet the results of the work are the efficiency of operations, the satisfaction of the users, the competitiveness, and the possibility of innovations, which can be evaluated as the worthy results of the considerable efforts. With the ongoing development of healthcare technology, where value-based care patterns, improved interoperability levels, and more consumer-friendly experiences are the order of the day, the platform that was developed based on modern architecture principles will increasingly distinguish leaders in the market from organizations limited by legacy systems. The difference between organizations with modern and legacy platforms will probably continue to expand as the compound effects of higher agility and innovation capacity will increase with time. This example can inspire and offer practical insights to organizations that are taking on similar organizational changes, as it presents a roadmap on how such organizations can navigate the intricate process of legacy lock-in to cloud-native opportunities that can liberate, as opposed to restrict, strategic goals.

Author Statements:

- **Ethical approval:** The conducted research is not related to either human or animal use.
- **Conflict of interest:** The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper
- **Acknowledgement:** The authors declare that they have nobody or no-company to acknowledge.
- **Author contributions:** The authors declare that they have equal right on this paper.
- **Funding information:** The authors declare that there is no funding to be acknowledged.
- **Data availability statement:** The data that support the findings of this study are available on request from the corresponding author. The data are not publicly available due to privacy or ethical restrictions.

References

[1] Nicole Forsgren, "Accelerate: The Science of Lean Software and DevOps Building and Scaling High-Performing Technology Organizations", 2018. [Online]. Available: https://dl.acm.org/doi/10.5555/3235404

[2] Microservices.io "Pattern: Microservices Architecture," [Online]. Available: https://microservices.io/patterns/microservices.html

[3] <packt>, "The monolithic architecture pattern,". [Online]. Available: https://subscription.packtpub.com/book/web_development/9781789133608/1/ch01lvl1sec02/the-monolithic-architecture-pattern

[4] Carina Nina Vorisek, et al., "Fast Healthcare Interoperability Resources (FHIR) for Interoperability in Health Research: Systematic Review," PubMed Central, 2022. [Online]. Available: https://pmc.ncbi.nlm.nih.gov/articles/PMC9346559/

[5] IBM, "What are microservices?" IBM. [Online]. Available: https://www.ibm.com/think/topics/microservices

[6] Brendan Burns, et al., "Kubernetes: Up and Running, 3rd Edition", 2022. [Online]. Available: https://www.oreilly.com/library/view/kubernetes-up-and/9781098110192/ch01.html

[7] Chiradeep BasuMallick, "What is cloud computing? Definition, benefits, types, and trends," Spiceworks, 2021. [Online]. Available: https://www.spiceworks.com/tech/cloud/articles/what-is-cloud-computing/

[8] Martin Fowler, "Microservices: A definition of this new architectural term," eapad.dk. [Online]. Available: https://eapad.dk/resource/microservices-a-definition-of-this-new-architectural-term/

[9] Gene Kim, "The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations", PubMed Central, 2016. [Online]. Available: https://dl.acm.org/doi/10.5555/3044729

[10] Jez Humble, PictureDavid Farley, "Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation," PubMed Central, 2010. [Online]. Available: https://dl.acm.org/doi/book/10.5555/1869904