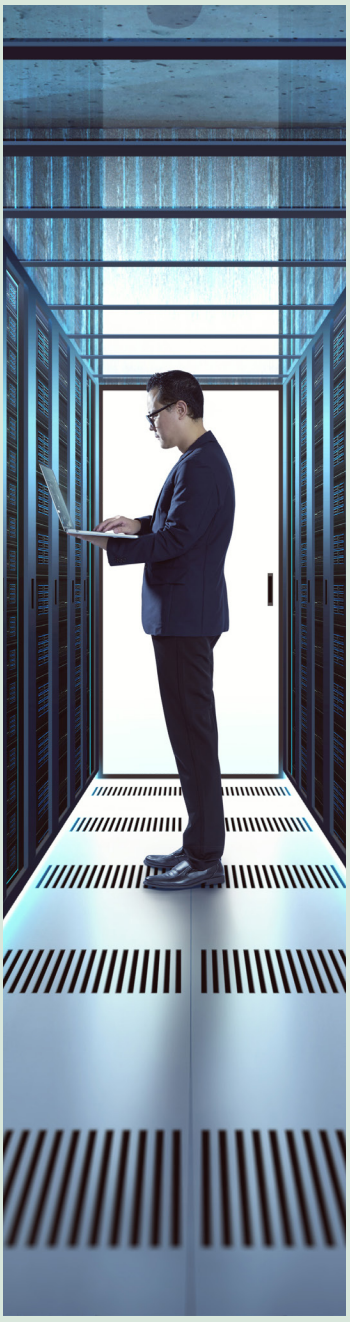
WHITE PAPER



RECIPES FOR SUCCESSFUL LEGACY TRANSFORMATION TO CLOUD

Infosys®
Navigate your next

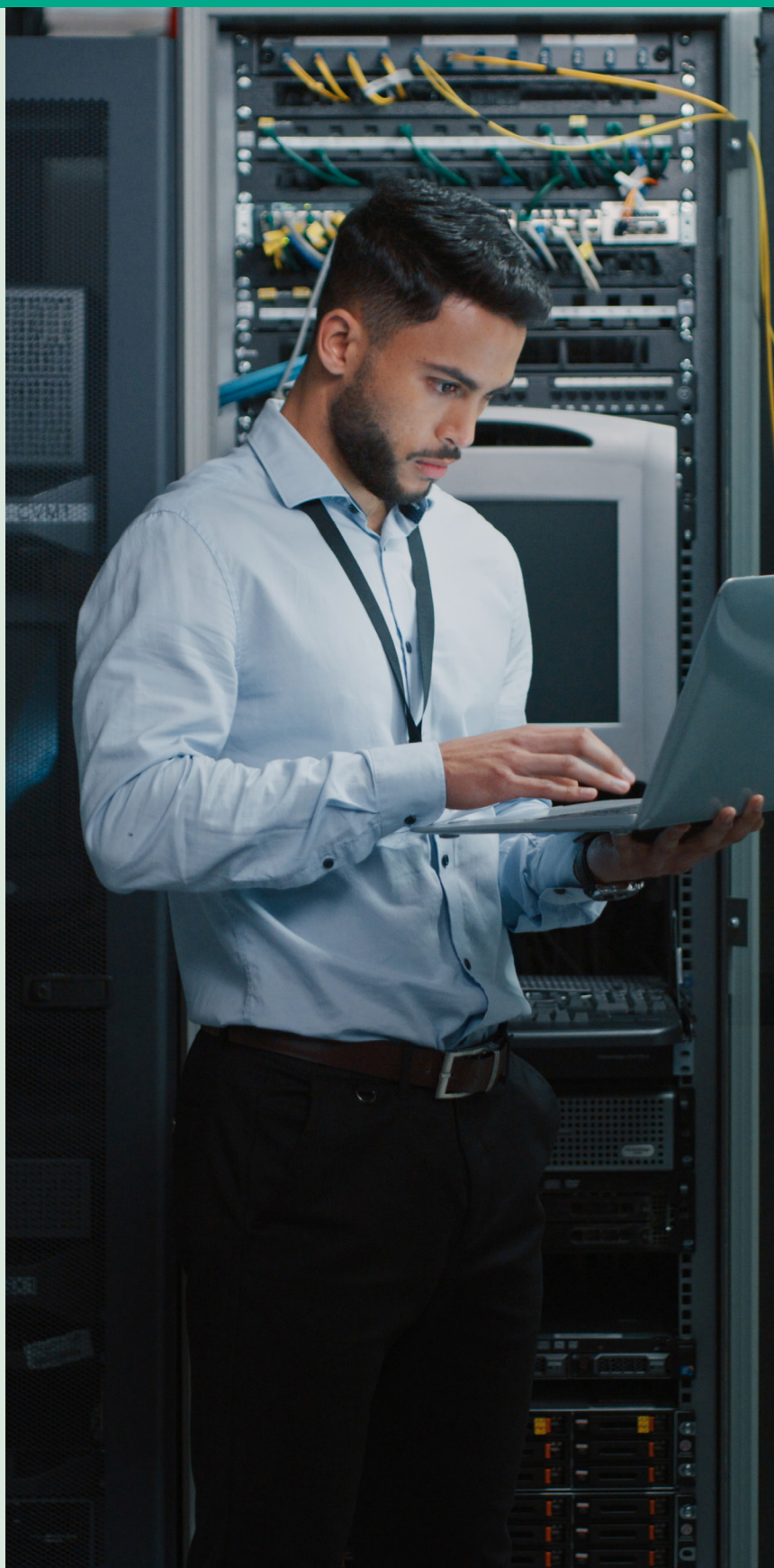
Moving to the cloud is a mandate for enterprises big and small to stay competitive and relevant and reap both business and technical benefits of agility, scalability as digital transformation is enabled, and growth. This journey to the cloud will go through different routes and pitstops depending on the size and composition of the current IT estate and the stability that's expected due to business and regulatory needs during a major technical revamp and migration before finally reaching an end state that's stable enough to consider decommissioning of the legacy landscape. In this case study, we share our experiences and learnings from a large modernization and transformation of the core PLM portfolio of one of the largest retailers worldwide.

Context and business scenario

The landscape to be transformed was a custom PLM stack built in the late 1990s and early 2000s. It encompassed the key business and engineering functions of a new product launch, packaging and R&D required for their product to be created end-to-end from conceptualization to market launch. The kind of apps and technologies were typical of that period for core engineering apps, i.e., desktop/thick client VB backed by 2 Oracle databases, each having over 5 TB of data. There were also a few apps on .Net, SharePoint and Crystal Reports, though the majority was in VB. Overall, there were around 40 apps of varied sizes, from some having a couple of screens with a handful of users and others with over 500 screens and 5K varied users ranging from product managers and packaging developers to chemists and R&D scientists. So, the customer needed clear business and technical objectives to achieve this transformation.

Business and technical objectives

- Resolve the technical risks associated with EOL and unsupported technologies and platforms
- Gain user productivity efficiency by using modern technologies
- Reduce new product launch time
- Reduce TCO
- Use scalable architecture to support future business expansion

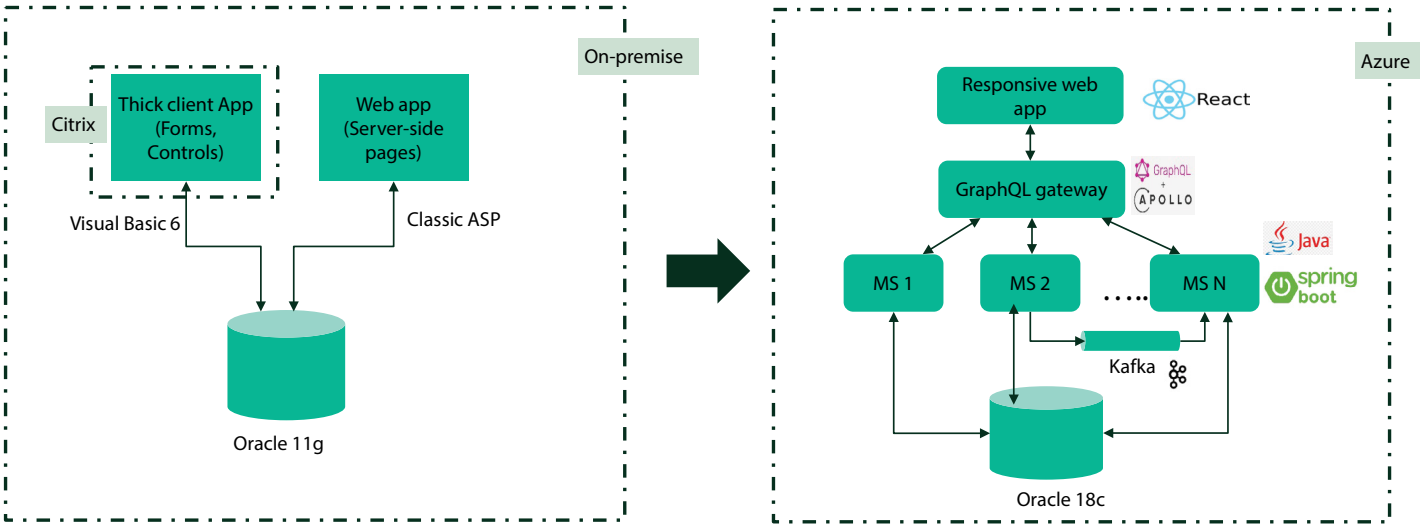


Solution approach

An extensive assessment of the existing landscape from the business process, technology, architecture, data, users, and operations point of view was done to arrive at the solution recommendations. It was found that almost 90% of the apps needed a full re-architecture and rewrite to a microservices based cloud-native tech stack. However, the remaining 10% could be migrated using a combination of lift and shift and re-platforming. This included the database, which had most of the business logic embedded within views, stored procedures and other DB objects. This decision was taken because EOL technologies had to be eliminated while keeping core logic and feature parity intact with no risk to the business processes.

The transformation was planned in two phases. The first phase focused on as-is migration with a modern tech stack and architecture as a foundation. The second phase involved transforming business processes and eliminating any technology debt.

Figure 1 shows the simplified view of the legacy and target architectures.



- 2-tier architecture
- Thick client apps installed locally or accessed via Citrix
- Web apps usable only on desktop with limited interactivity
- Oracle DB user-based credentials and roles
- Limited tools for troubleshooting, support and operations
- Complex manual build/deployment process

- N-tier horizontally scalable architecture on cloud
- Responsive web app with intuitive and consistent UI/UX across platform, hosted on Azure cloud
- SSO with AD credentials and well-defined roles framework
- Distributed logging and tracing with automated alerts
- Fully DevSecOps build and deployment pipelines

Figure 1 Legacy and target architectures

The following sections discuss key challenges encountered, architecture and design considerations, and solutions.

Microservices as core and business capabilities

As part of the initial assessment based on available product documentation, discussions with subject matter experts (SME) and product owners and high-level code and app flow analysis, it emerged that most of the smaller apps had a unique role in the overall business process and hence represented a single business capability. We identified the functionalities within these as candidates for separate microservices. However, the larger ones were true monoliths that served several functions for multiple user groups. The data used was also spread across with upstream/downstream dependencies on the smaller apps. The identification of services corresponding to these was based on **domain driven design** with one service built around a core domain object like Product, Formula and Bill Of Material (BOM). Apart from business capabilities, the core services required for user management, master data handling, and messaging were also developed. Figure 2 shows how apps could leverage these core and business services for their realization. Some of these services didn't have an exact match in the legacy world but were needed for the new microservices based design paradigm.

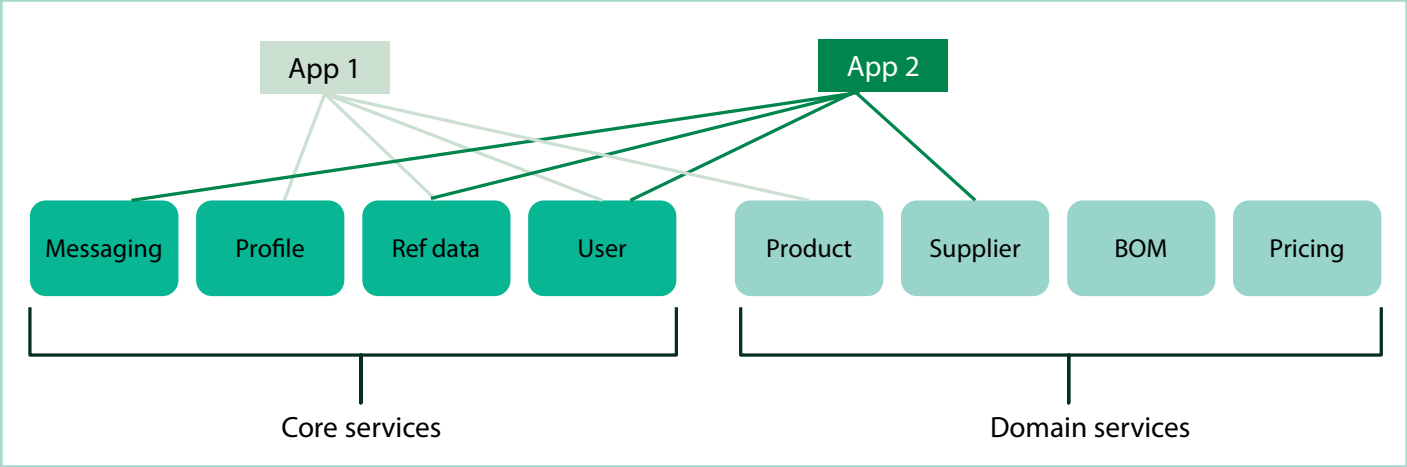


Figure 2 Microservices landscape

In this first phase, once the services were identified, the migration approach comprised the following key steps:

- Extract business logic from the legacy code and database, and move it to the new stack service layer, leaving the DB as a pure data store only.
- Retain the logic within the DB (stored procedures) for cases that involved complex calculations and nested data retrievals involving 1000s of DB queries to process a single API call to keep performance intact. This transformation was planned in phase 2 of the initiative to remove all technical debt.
- Perform analysis of complex screens that required data from multiple sources and the service API design such that they could be rendered from a single API or within two to three calls in the worst case.
- Conduct tradeoff analysis and proof of concepts on UI/gateway aggregation of multiple API responses vs. data aggregation at a single service backend. We chose the latter based on performance SLA considerations.
- Keep the cross-service communication asynchronous using Kafka and avoid chaining of calls.
- Perform load tests at 10x the expected user load for critical user flows to ensure performance is within acceptable limits.

Co-existence and data migration

A key requirement for this transformation was to have real-time bidirectional data synchronization. It was needed since the apps and users would be migrated to the new platform incrementally, and the data flow would happen from both platforms during the transition phase. A few use cases had been performed (from the same user role or different use role) to complete some of the business processes and workflows, ensuring data sync in real-time in both systems. This was referred to as co-existence and had a major impact on the logical and physical data models for the microservices. The key design and implementation approaches for data modeling and synchronization included:

- The data partitioning was done based on schema per service, with each service owning the writes to its schema, while reads could be done across schemas using cross-schema views. The big legacy schemas (Schema1, Schema2) with up to 500 tables were split into smaller schemas (S1, S2 etc. in in Figure 3) based on the domain model, having 20 to 50 tables each.
- Data synchronization was done using Oracle Golden Gate (OGG), and configurations (source<-> target table mappings) were carefully analyzed and grouped to keep dependent tables together and have maximum parallelism for high performance.
- Keeping the table structures similar between both DBs helped to simplify the OGG configurations and keep the data corrections minimal in cases where OGG couldn't replicate due to network issues or conflicts.
- The data synchronization was continuously monitored using custom scripts that compared data to the column level values, and mismatches were reported and corrected.

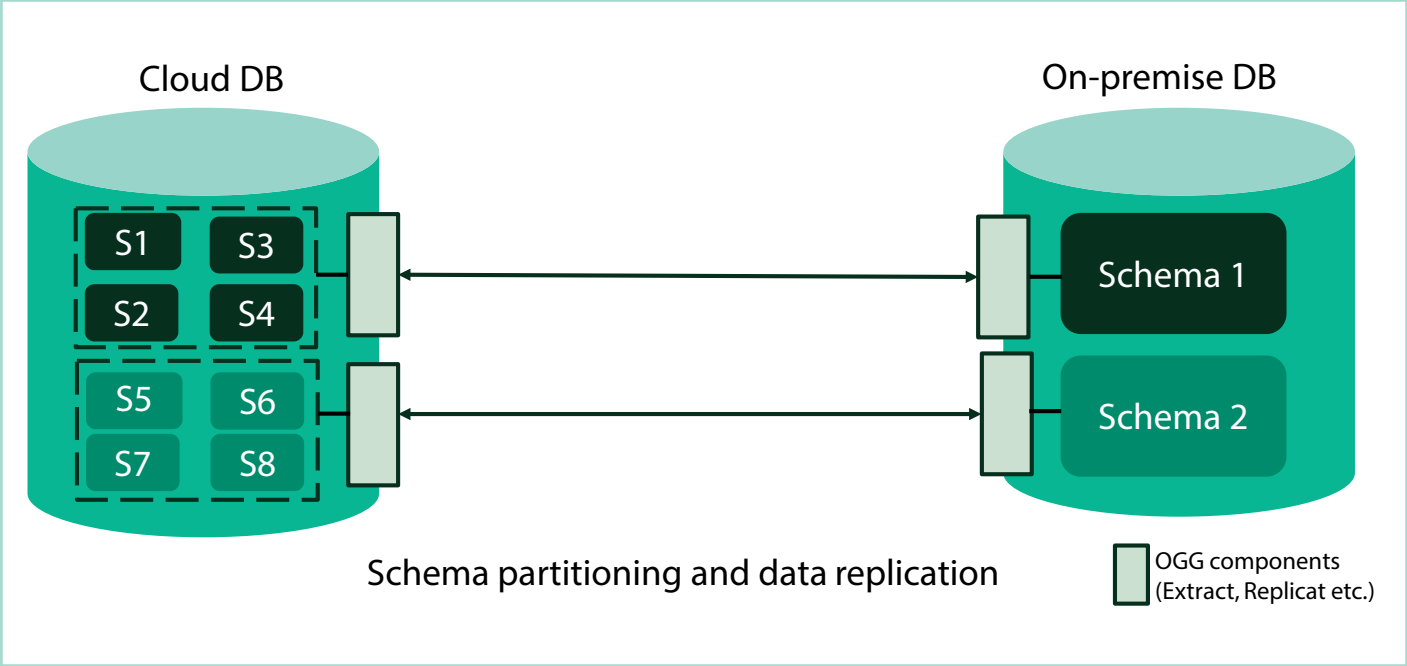


Figure 3 Schema partitioning and data replication

The data migration was done in phases along with the respective application releases, and OGG sync was set up for the migrated tables in line with the migration. After every migration, a comprehensive data validation report was generated to ensure the data consistency was maintained, and any delta generated during the release window was manually synced.

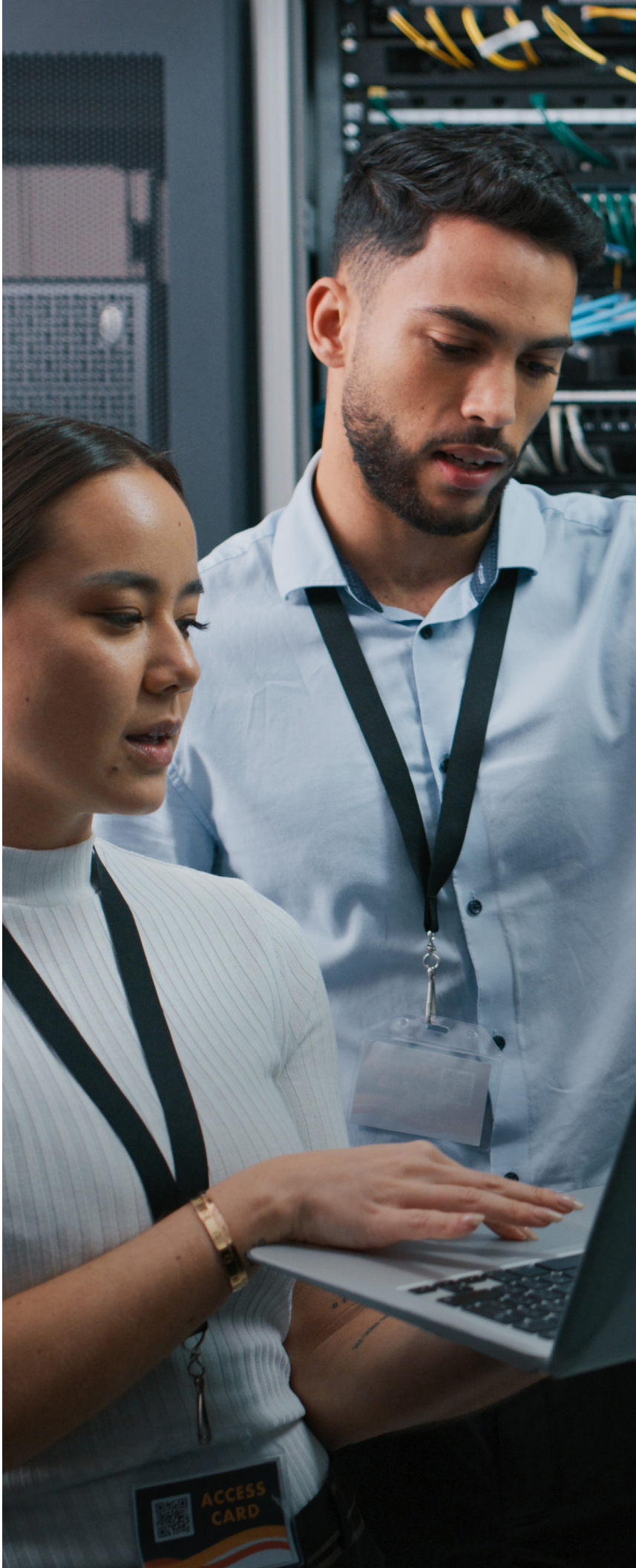
User experience revamp

A unified user experience across the applications was a key guiding principle for the transformation. The existing apps were mostly thick client based and installed on the user's machine or accessed via Citrix. Each app had its own login; in some cases, the same user had multiple credentials per app based on the role they represented. As part of the UX revamp, a single landing page was created, which listed all the apps the user had access to, their key action items, stats and preferences. To enable this, a user management data model and service was created, which mapped all the multiple identities in the legacy system to a single user ID and migrated the required attributes from all the disparate user data sources into the new data model. This included their roles and permissions, which were handled using custom Oracle roles in the existing system. In addition, the user ID was linked with SSO, resulting in a seamless login and access experience across apps.

Moving from thick client windows based apps to web apps were the other major shift in terms of UX and UI implementation. A few best practices followed were:

- The legacy apps had features using keyboard shortcuts, drag and drop across windows and tight integration with other apps like Outlook and Excel. These were redesigned considering the browser experience with appropriate menus, controls and forms.
- There was no limit on the amount of data a table could hold in the legacy world, which was replaced with paginated grids and lazy loading/infinite scroll experiences. In addition, some screens and flows were entirely revamped in consultation with the end users to simplify the business process.
- Better data integration enabled improved user productivity by eliminating the need to copy/paste data between apps, replacing it with a simple lookup or auto population based on context.
- A common component library and style guide definition ensured a consistent look and feel across the apps while enhancing developer productivity. However, there were certain cases where licensed UI libraries had to be used to meet the business-specific UI and data loading requirements.

Extensive documentation and training videos were created to help transition users to the new system. The user feedback was excellent, and the transformation eliminated their UX pain points with the old system.



Development and release strategy

A strong DevOps culture and release process was built in from the start, with refinements done along the way. The project team was over 300 strong at its peak, spread across multiple locations and time zones with UI, backend, database and QA skills. They were augmented with a cross-functional team of architects, business analysts, DevOps engineers and program leads. There was a high level of automation in the build, deployment, and regression testing suites. Code quality gates were set up and regularly monitored for compliance. The branching/merge strategy had rules to ensure all merges to the deployable branches were done through approved pull requests only. A stable build was deployed to the QA environment for functional validation at logical points in time. This involved manual testing to compare the apps between the legacy and rewrite versions to ensure they match functionality and do not have any response time degradation in the new apps.

Each production release was meticulously tracked with a cutover plan that listed each team's activities and duration. Then, the same was validated with a dry-run in a production-like environment.

Key learnings and best practices

During this major transformation journey, there were many learnings from people, process and technology aspects. Appropriate course corrections were done to ensure it was on track and culminated successfully. A few key ones are summarized below

- The documentation gives a start for legacy apps, but a thorough analysis of the app, its flows (beyond the happy path) and integrations are necessary to understand its true complexity. This may not always be evident from a black box view, and source code analysis is the only way to identify potential issues and dependencies related to its migration.
- Discuss with the SMEs and product owners the features that are really needed in the new version, since the legacy app could have obsolete features that are still reachable. In addition, a prioritized product backlog needs to be created before commencing the migration.
- Demo UI and UX change early to the users for early feedback, even though it's built based on signed-off specs and wireframes.
- Identify each app's top performance critical scenarios and establish response time baselines using data volumes similar to that used in Production. Then, validate that the same or better metrics are achieved in the new system using manual and load testing with the expected concurrent user load.
- Set up a robust logging, monitoring and alerting framework and ensure the services and endpoints send the required logs and traces to this system and review them for usefulness. Use an Application Performance Monitoring (APM) tool to generate the performance profile and identify /resolve issues during the development/QA phase.
- Perform data validation and comparison frequently to ensure replication works as expected. Lay out a plan to identify and rectify data issues.
- Keep track of the dependencies of the new platform on the existing one and design such that the former can function with minimal/no changes once the legacy system is ready to be decommissioned after full cutover. This will help in a smooth decommissioning phase rather than requiring a full-blown system analysis/assessment again.

Conclusion

Cloud transformation programs that require a complete re-architecture must consider multiple system design, process aspects, and weigh their pros and cons before taking a path forward. It becomes even more challenging when there's a need for a legacy system to continue with partial use cases along with the new system. The strategy to be applied and the robustness of the overall transformation approach and release process is critical for success. Also, a phased approach by segregating technology transformation along with long-term architecture and business process transformation makes better sense in such large transformations. This helps in de-risking EOL unsupported technologies much faster. Special attention should be paid to data migration, user migration, training and support related areas apart from technical transformation. Organization level change management also plays a crucial role in such transformations.

About the Authors

Joseph Alex
Principal Technology Architect, Infosys

Krishna Markande
Associate Vice President, Senior Principal Technology Architect, Infosys

For more information, contact askus@infosys.com

Infosys®
Navigate your next