PROWESS SOFT

PROWESS SOFTWARE SERVICES

WHITEPAPER

Integration Tool Migration

Considerations and Execution of Integration Tool Migration

Abstract

The paper discusses the various parts of an integration solution and how a migration strategy may be built for successful migration from one integration tool to another.

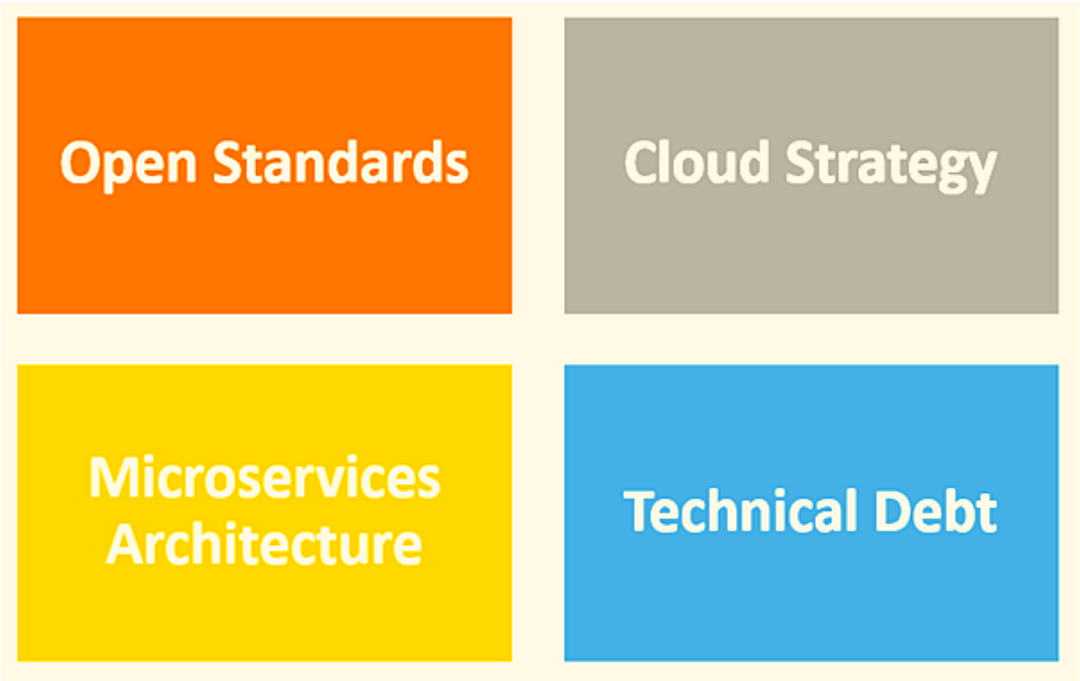
CONTENTS

Introduction

Enterprises use multiple technology tools to implement their business functions. With the technology tools at their disposal, enterprise architecture teams implement frameworks and other reusable artifacts, accelerators, and tools that fill in the functionality whitespaces between the tools and the business use cases addressed by enterprises. Over time, as enterprises keep adapting to evolving business needs, the underlying technology stack may be found wanting. Whether it is a move from a monolith to SOA, or from SOA to Microservices; adopting a cloud strategy, or a specific technology use case pattern such as streaming, if the existing underlying tool does not support it, it may make sense for enterprises to start migrating their solutions into newer tools that support their roadmap. With newer technology paradigms for solutioning, it may be quicker to reach desired target architecture state through more recent tools than persist and make do with existing ones.

Triggers of Migration

Below is a set of triggers that are often encountered. What are yours?



Open Standards

As technology evolves, it brings newer standards in implementing integration solutions. Existing systems in an enterprise may be composed of multiple types of connections. The integration touchpoints may be non-standard connections, the endpoints may be socket-based connections or a mix of numerous point-to-point connections that are not reusable. It is also possible that the current implementation standards may be out of date.

Sometimes, just expanding the audience from internal stakeholders to external ones may trigger a need for change. The current solution may have been implemented based on internal stakeholder needs but is not necessarily designed for external use. If a newer business use case requires exposing the services to external stakeholders, the existing tools may not be ready. An API Gateway may need to be added. Even if the API Gateway is added, the services may need to be standardized now that external stakeholders are starting to use it. If the existing tool does not support standardization out-of-box, that may be a reason for adopting new tools.

Cloud Native Solutions

The advent and availability of cloud infrastructure as a commercially viable alternative has transformed application architecture and design. Many systems designed and implemented in the past decade did not necessarily have the cloud on their minds. The technology product companies that did not foresee the cloud as a major player haven't had their cloud-plan ready. Subsequently, the companies that had implemented older products had to migrate to newer tools that provided the ability to build and deploy applications on the cloud.

Some enterprises may have tried to re-host the existing on-prem applications onto the cloud but probably learned that re-host does not necessarily make them cloud-native.

Microservices Architecture

The earlier design of services was focused on providing end-to-end functionality, which did not lend the flexibility that Microservices do today. As part of microservices architecture, smaller units of functions were designed whose lifecycle was managed independently. The tools that were needed to build and deploy microservices are vastly different than the traditional tools that were hitherto used.

By nature of being smaller and more manageable discrete functions, it became a natural way to deploy as cloud-native applications that were easy and quick to scale-up or scale-down. With these considerations, it became a strategic decision for many enterprises to move away from existing product stack into products that supported microservices out-of-box.

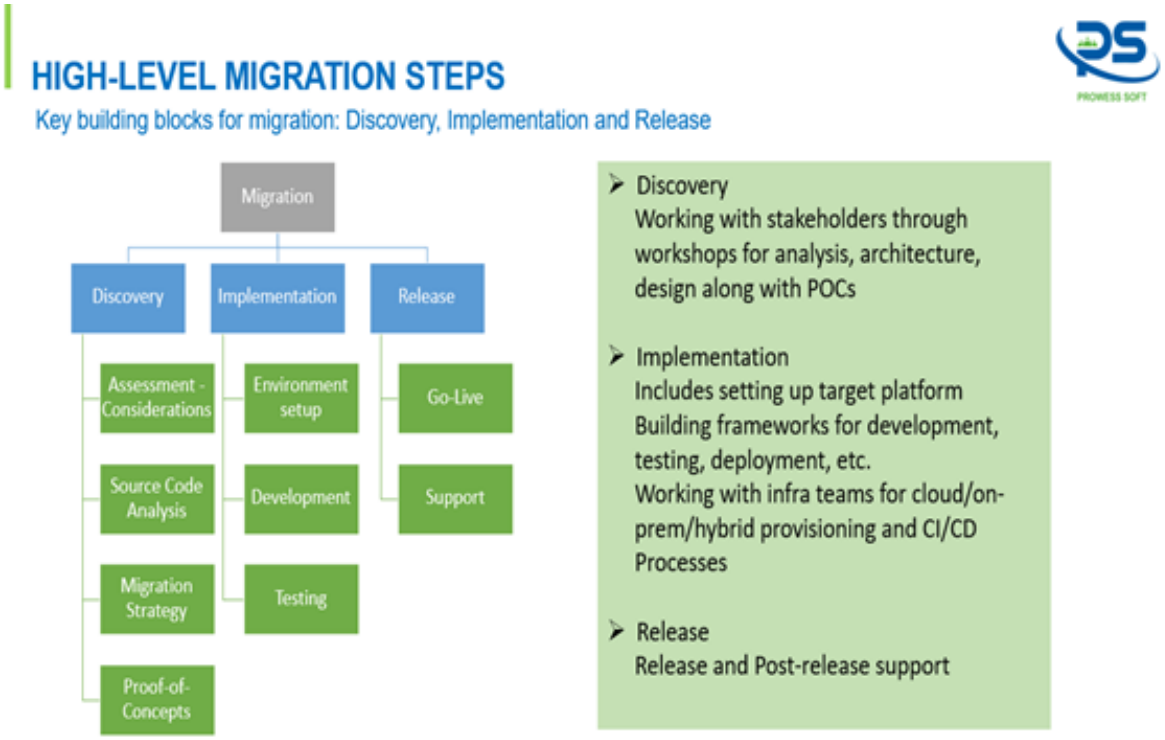
Enterprises that had existing applications as large monoliths or as services that packed end-to-end functionality, started looking for tools that helped with breaking application logic into manageable microservices.

Reduce Technical Debts

Enterprise applications accumulate technical debt as they deliver functionality through their application lifecycle processes. This technical debt may be planned to be addressed at an appropriate time, usually based on the business priorities. There is another way of accumulating technical debt; although not planned, it may come through significant corporate events such as mergers or acquisitions. Such events set in motion a lot of enterprise architecture activities. There would be discovery sessions that include assessing the integration topology of both entities, analysis of current technology landscapes, analysis of value proposition in making the needed changes, etc.

More often than not, there are relatively significant changes planned on the enterprise roadmap to ensure the synergies due to the merger are realized. The changes include migrating solutions into one of the tools in use and decommissioning the other.

Migration

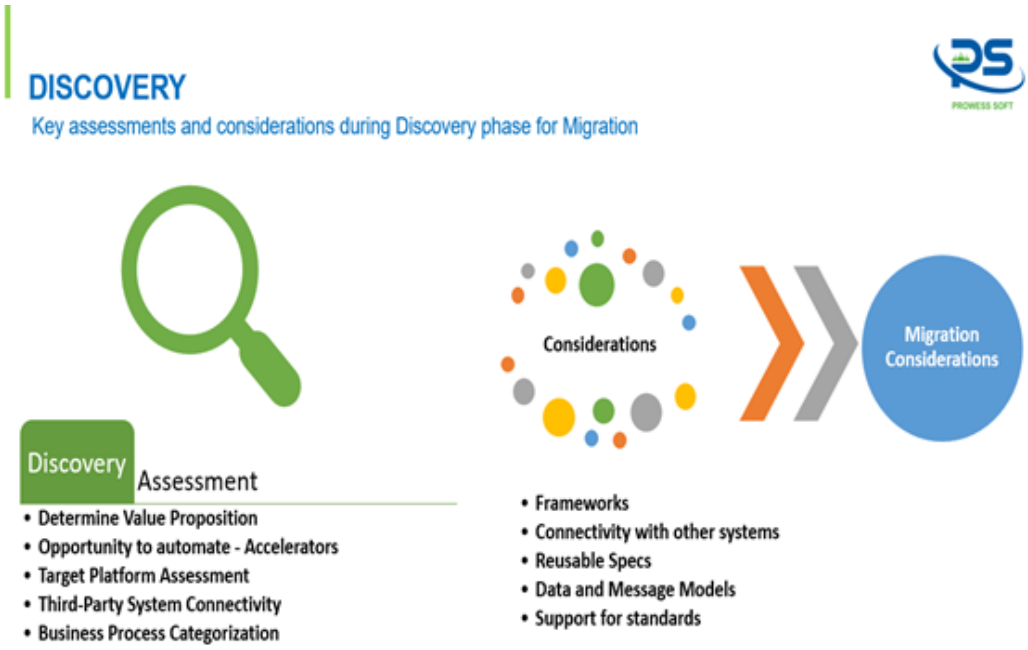


At a high-level, migration needs the following steps

- Discovery
- Implementation
- Release

Discovery

The discovery phase may be conducted through workshops between enterprise systems stakeholders to arrive at the Migration Strategy. The discovery may include analysis of current state architecture, existing code, performing certain proofs-of-concept (POCs) to ascertain conformance to design patterns, POCs for connectivity with other enterprise applications and external systems, etc. The discovery phase brings about a specific set of outcomes that guide the rest of the phases. This phase may define a sequence of activities to build through the newer technology platform, starting with building the core functional blocks, common artifacts, and the order of migration of other applications based on this information.



One of the first tasks as part of the workshops is to be clear on the benefits of migration. There could be multiple triggers to embark on the migration exercise, and identifying the value proposition and documenting them through consensus would be the first step. In bringing about the consensus, there could be a lot of inputs one could get as part of the workshops, sometimes from teams that do not see the value of migration, and these discussions can help document where the "value" of migration is. These workshops also help with prioritizing the applications for migration. Some application migrations may bring more benefits to the enterprise as a whole than others.

An in-depth analysis would be needed to identify corresponding activities/tasks among the older and newer platforms and verify whether it is an accelerator or may be developed to convert the configuration of the corresponding tasks from one platform to another. For example, the table below identifies some activities used in each of the following platforms that achieve the goal of mapping.

	TIBCO BW	MuleSoft	WSO2	Web Methods
Activity	Mapper	DataWeave script	XSLT Mapping	Map Service
Data formats	XSD	RAML	XSD	IS Schema

Based on the table above, an accelerator may be developed to convert existing activities from one platform to another. Identifying such activities as part of the discovery phase and assessing the feasibility, and building these tools greatly helps during the implementation phase.

Implementation

The migration strategy produced after the discovery phase contains components that may be worked upon in sequence. The phase typically starts with extending the POCs that were created during discovery into standard reusable artifacts that can be part of enterprise application frameworks. It may consist of building observability frameworks, configuration for DevOps tools or CI/CD tools, frameworks for connectivity to internal and external systems, etc. These artifacts become the building blocks for applications that are yet to be migrated.

Product mapping is an important consideration, both at the discovery and implementation phase. In the discovery phase, the activity is focused on mapping a tool from an older platform to a corresponding tool in the newer one. During implementation, it is about building frameworks around the selected solution. For example, if building an in-memory cache is part of the solution, the older platform may have chosen a proprietary tool. In comparison, the newer platform may utilize an open-source solution, for example, Redis, as an in-memory cache. In the implementation phase, one could build the reusable framework around operations that may be invoked with the Redis store in the target platform. Some platforms provide their products for various functionality that may be considered; for example, in the TIBCO platform, ActiveSpaces is an option if your target platform of choice is TIBCO.

An important but often underplayed aspect of Implementation is the need to train teams on the newer frameworks. Oftentimes, in large corporates, the lack of skills in technical platforms may be compensated through contractors. However, the existing technical teams are the ones with a wealth of functional knowledge. The cost and effectiveness of training existing personnel on the newer technologies often beat the cost of training on the large amounts of functional knowledge that existing teams accumulate over the years. However, it may be assessed subjectively based on the scope and context of the migration.

Building shared artifacts allows for proceeding with the migration of individual applications. Based on the dependencies that were identified in the discovery phase, and the priorities, appropriate applications may be selected for migration. The usage of accelerators would be a key differentiator in this phase. Accelerators help

- Ensure common tasks are automated, reducing manual effort
- Ensure consistency across all processes
- Over time, as the accelerators mature, they become a powerful time-saving mechanism for large scale migrations

One of the critical activities of the Implementation phase is testing. Many tools in the market can help with regression and performance testing. An enterprise with well-defined processes would have testing tools and standards in place, whether for a migration project or otherwise. The appropriate tools may be leveraged to work toward regression and performance testing. Keeping the older and newer systems “alive” is recommended to ensure periodic tests are performed and any inconsistencies may be identified and fixed.

This phase also involves setting up the runtime for the target environment. This may include one-time setup and configuration of reusable CI/CD pipelines.

Release

As part of the implementation, the release scripts would be in place. Ideally, there would be parallel environments running for some time that can help with a backup for unforeseen runtime issues and as validation through periodic checks for ensuring consistent results across both environments.

Any tasks related to provisioning, setting up appropriate users and roles, support training, documentation of Standard Operation Procedures (SOPs), and handing-off to support teams, would be part of this phase. A good amount of time may be spent with support teams for live support in the initial days until Support teams can pick up and handle maintenance chores.

Building common artifacts, setting up the new runtime environment, performing the application migration tasks, testing, setting it up for deployment for the first application, and supporting through the initial release provides valuable insights into existing application and the effort involved in the migration process. By identifying the opportunities for improvements in the first migration cycle and applying the learnings to code migration, automation, and CI/CD pipelines, the process matures and becomes repeatable as more applications are targeted for migration.