Migration Decoded

To keep pace with the unrelenting, swift pace of new technology, IT organizations need an integrated software migration framework that spans everything from effort estimation/impact assessment, baselining and acceleration tools, through methodology selection, merging and finishing criteria, as well as validation.

Executive Summary

In today's fast-paced business place where technology advancements and disruptive innovation rule the day, the shelf life of technology stacks and related application software is being compressed rapidly. This forces IT organizations to continuously invest in migration and new technology platforms to stay current with the dynamic needs of the modern digital enterprise. The volume of these investments is indicative of how far behind the technology stack is or how fast newer advances in the technology have evolved.

Recent data points make this challenge abundantly clear:

- According to industry research, a double digit percentage of overall IT budgets is allocated to software migration, modernization and rationalization, annually.
- The success rate of migration projects is low compared with other types of IT, due to numerous factors. In fact, more than half of migration projects either fail or exceed budgets, according to industry research. The bottom line: Migrations are often complex and exceedingly risky!

To unleash the market potential and reduce the risk involved in migrations, we have created a framework called 10 Migration Success Factors (MSF 10). We have implemented different types of migrations in the last five years with a large strategic client and have extracted 10 key areas that we believe must be mastered to succeed in a migration project. These factors are based on various real-time experiences, such as version upgrades, total code rewrites, compliance to reference architectures (a prescription of common components, and software products and their versions), data migration, new data center migration – or any combination of the above.

While there are many other specific factors that may influence the success of a migration project, the generic 10 listed in this white paper are critical and applicable in any given migration context. This list will vastly improve the odds of project success for any project/program manager involved with migration planning.

1

Estimation and Impact Assessment

What does it take to migrate? Estimating the unique values, for starters. Each migration is unique and complex, across a variety of tasks. Therefore, scientific estimation methods such as function point (FP), use case points (UCP), Cocomo model, etc. are not effective in estimating the migration effort. We suggest adopting a story points/work breakdown structure (WBS) or a complexity points-based sizing approach. Often, this requires a quick proof of concept (POC) to understand the tasks involved in the migration before the final estimations can be determined. The POC not only helps the estimation process, but also helps in understanding and solving the challenges involved.

- A sample complexity factors list from one of our large Java-based migrations includes number of EAR/WAR files.
- Total Web services/APIs provided.
- Number of POM files impacted.

A standard technology-based questionnaire should be prepared to help in the assessment of the current state of the platform and application.

A standard technology-based questionnaire should be prepared to help in the assessment of the current state of the platform and application.

An impact analyzer (custom-built or provided by the product owner) is typically used to identify the impacted components and code, which will form the input for the estimate.

2

Baseline Current State

Code is the only truth. Due to rapid technology changes, the shelf life of many applications is rapidly compressing. Because of this, no artifact (design, requirements, test cases, etc.) exists to accurately reveal the current functionality, nonfunctional requirements benchmark, etc. Therefore, it's essential to baseline the current state of the application from the code base of the platform. This can be done as follows:

- **Create functional baseline:** See what's working and what's not, based on a comprehensive functional test suite. This will also record any defects in the current system.
- **Create nonfunctional baseline:** Baseline performance, scalability of the application/platform for critical user scenarios.
- **Code quality baseline:** Baseline the code quality against parameters in the specific technology (e.g., PMD, Checkstyle and FindBugs for Java).
- **Create security vulnerabilities** and accessibility baseline as applicable.
- **Baseline the current build,** deployment infrastructure, data archival and retention policies, and disaster recovery requirements.
- **Perform gap analysis for the as-is** (what is it supposed to be vs. actual in the point of departure (POD)). For example: exceptions that exist in as-is for compliance to reference architecture.

3

Migration Approach

The approach taken should form your path forward.

- **Choose the right team model.**
- **A centralized team:** This is a separate team that will be employed for migration. The team will complete the migration to meet the acceptance criteria and hand it over to the "owning" team. This will be well-suited for simple and stable applications.
- **A federated team:** This is a team formed with a combination of a target technology expert team and an owning team (subject matter experts), which will complete the migration. This is suitable when:
  - The target technology is fairly new.
  - The application is complex.
  - The velocity of the application changes are high.
- **Choose a factory model,** as applicable, to achieve economy of scale through industrialization. This is suitable when:
  - The number of applications to be migrated is high.
  - There is a possibility of repetitiveness in migration.
  - Separate teams/application production lines (APLs) can be set up for each logical step of the migration process.

Establishing entry-task-validation-exit (ETVX) will be essential for successful APL execution. It's expected to achieve at least 15% to 20% of cost savings in low volume cases, and above 20% where application volumes are high. In our research, we observed these efficiencies through three dimensions:

- Increased human efficiencies.

- Reduced effort via automation.
- Improved collaboration and communication with the supporting teams.

Choose between a big bang vs. phased approach. Consider the following factors before making a decision:

- Risk to the business.
- Complexity and size of the platform.
- Complexity involved when releasing software in phases.
- Cost involved in maintaining two parallel platforms.

4

Accelerators

We recommend the use of accelerators to automate the migration process, not just for speed and efficiency but for quality and consistency. Depending on the context, numerous accelerators are available for product migrations, but be aware: you may need to build custom accelerators for certain application migrations.

Irrespective of the context, consider the following accelerators.

- **Tools for impact analysis:** It's very important to understand "what" is being impacted by the migration project. Manual analysis of the code-base/components is generally tedious and error-prone, and it might increase risk since it can lead to incorrect estimates and misunderstanding of the migration implications. Third-party tools exist to aid in this process; in fact, small utility tools are available for custom applications.
- **Create cookbooks:** Capture step-by-step tasks and instructions in a shareable document to ensure consistent quality and help improve productivity. This will help improve productivity. Also, conduct a POC to test these steps as well as to help document learnings and best practices.
- **Automation:** Build tool-set/utilities for repetitive tasks, particularly if the migration is focused on a group of applications. Before building, assess the trade-off between the effort savings and cost of building a tool-set.

  Examples include:

  - Impact analyzer.
  - Utility to set up developer environment.
  - Utilities for automating the migration of POD source code to point of arrival (POA)-compliant source code.
  - Data migration utilities.
  - Data validation scripts post migration of the data into POD.
  - "Sample application" to validate the environment; for example, a "sample war" (Web archive) in the Java world with different integrations of the platform can be used to test the newly created environment.
  - Log analyzers.
  - Test data-creating utilities for load and performance testing and functional testing.

5

Merge Strategy

Business teams constantly introduce change. It's therefore important to have a strategy and roadmap for ongoing changes while the migration is taking place to the POA. Key considerations in this regard include:

- **Create a merge strategy:** Define merge points at which the ongoing changes are merged/synchronized with POA codebase.
- **Plan a freeze period:** Define a freeze period for ongoing changes before the production release goes live. It should be as short as possible to avoid business impact.
- **Isolate POA expected behaviors:** The POA's expected behavior, when combined with other changes, will complicate migration activities. It becomes hard to separate issues originated by the migration vs. ongoing changes. There is a risk of attributing existing production bugs to migration tasks. A functional baseline that creates a "baseline current state" will help to resolve this.

It's important to have POA validation/test strategy in place to make sure before and after functionality is the same (except for the ongoing changes, if any) and nonfunctional parameters are equal or better.

6

POA Validation/Test Strategy

Did I break something? It's important to have POA validation/test strategy in place to make sure before and after functionality is the same (except for the ongoing changes, if any) and nonfunctional parameters are equal or better.

- **Functional comparison:** Compare the functional results of the migrated applications against the functional baseline. You can adopt frameworks such as Selenium or use custom tools for this comparison.
  - A typical UI for an application can be compared using image comparison – even at the pixel level – to make sure nothing is changed due to migration. Selenium, for example, can be used to capture the screenshots and compare before/after snapshots. For any application, the test results at the time of the baseline will be used to certify the target state.
- **Validate nonfunctional requirements (performance, scalability, etc.):** The baseline will be compared for critical use cases to ensure better or equal performance, scalability, etc. post migration.
- **Create a strategy for data migration and compliance to reference-architecture as applicable.**
- **Create a test-data strategy:** It is often extremely challenging to capture the right test data for validation. The cleansed data from the POD system can be loaded to overcome this challenge.

7 Infrastructure Strategy

Build infrastructure to POA. Many migrations require procuring all new infrastructure (Web, app, database servers, new OS, etc.) and it's a cumbersome process in many large corporations. Things to consider:

IT organizations need a mechanism in place to validate the environment before the actual application is deployed in the target environment.

- **Procurement process:** For physical, virtual, cloud-based infrastructures, almost every organization will have a defined process with timeline service level agreements (SLAs) for every step. From there, they need a complete understanding of the migration process since it impacts the migration schedule.
- **Add lead times into planning:** Based on SLAs, the environment readiness will become a hard dependency in the plan. For example, the system and integration testing (SIT) environment should be ready at least two weeks before SIT starts.
- **Isolate the environmental issues:** IT organizations need a mechanism in place to validate the environment before the actual application is deployed in the target environment. This will help to isolate the issues and possible resolutions before the application is validated. Often, IT organizations use a test app with all critical integration scenarios for this purpose.

8 Done Criteria

When is the migration considered complete? Though it's a common practice to have done/done or acceptance criteria for any project, it's more important for migration projects. Typical acceptance criteria will include:

- **Backward compatibility:** How far back are the versions to be supported? For example, older browser versions to be supported for UI applications, or old versions of Web services.
- **Functional validation compared to baseline.**
- **Nonfunctional validation compared to baseline.** This means no degradation of code quality, compliance with current SLAs for performance, vulnerabilities, etc.

9 Coexistence Strategy

Don't shut down the POD before completely standing on POA.

- **Have a fallback strategy:** Depending on the production stabilization period and approach (big-bang vs. phased), the strategy should include considerations for data synchronization, batch processes, disaster recovery environments, switch for traffic diversion, etc.
- **Consider a switch/toggle approach:** Design a possible parameterized switch between old and new systems to toggle between the environments with minimum impact to the business. Having a comprehensive coexistence strategy becomes a minimum requirement to accomplish this.

10 Change Management

Last but not least, successful migration projects depend on change management mastery. To do this:

- **Create a communication plan to dependent teams for all integrations:** Include synchronous, asynchronous, batch. Ensure that there is no impact to the existing protocols.
- **Train support and application teams:** It's important to train the support personnel and development teams to take care of the POA platform.
- **Manage the change to disaster recovery plan, backup compliance and data retention requirements.**

Looking Ahead

The problem of migrating to modern technologies is only going to grow because of accelerated technology advancements. To deal with this, MSF 10 provides a methodical approach and comprehensive array of tactics. For MSF 10 to reach its full potential, IT organizations must:

- Thoroughly understand their current software/ application environment.
- Comprehend the business criticality and the impact of the software to be migrated.
- Apply detailed architectural governance.
- Build an organizational culture open to adopting new technologies as they emerge.

Well-defined planning, appropriate estimation and sound execution methodology will go a long way towards mitigating the risk of complex migrations. Our MSF-10 framework is a comprehensive set of tried and true strategies based on experiences gleaned from real-time implementations. This will help with proposing, planning and executing any type of migration.

About the Author

Suresh Chinnam is a Project Director within Cognizant's Banking and Financial Services business unit. In this role, he is responsible for delivery of digital platforms for a large cards and payments client. Suresh received his master of technology degree in computer science and engineering from IIT Madras. He can be reached at Suresh.Chinnam@cognizant.com.