Intellyx White Paper

Legacy Migration vs. Modernization: The Challenge of Coexistence

Jason Bloomberg

December 9, 2015

Ever since the first vacuum tube burned out in the first commercial digital computer, calls for IT modernization have targeted older, legacy systems. Over the years, one plan after another for retiring increasingly aged mainframes, midrange systems, databases, and enterprise applications have crossed many a CIOs desk.

Yet while finally chucking that old system and replacing it with new and shiny tech may sound appealing, in practice such full-fledged migrations are rarely practical or cost-effective. Instead, many enterprises are realizing that older systems still provide value. Thus legacy modernization strategies are better off focusing on *coexistence* rather than all-out migration.

However, every organization's legacy context is different, as are its business goals. As a result, there are a plethora of migration, modernization, and coexistence scenarios, each with its own pros and cons. IT leaders must now understand the breadth of options open to them in order to make the right decisions about how to deal with legacy systems and applications.

Multiple Migration and Modernization Scenarios

Fortunately, legacy migration and modernization strategies have come a long way since the days of vacuum tubes. The 2000s brought Service-Oriented Architecture (SOA) to the table, helping organizations abstract legacy assets as Web Services. Today, the evolving notion of SOA has given rise to modern Application Programming Interfaces (APIs).

Understanding the modern challenge of coexistence, therefore, depends upon properly learning the lessons of SOA and APIs. While SOA was a mixed success, it unquestionably moved the modernization ball forward. Not all migration and modernization scenarios take advantage of SOA lessons learned, however. To illustrate these challenges, this paper breaks down such challenges into six basic scenarios.

The illustrations below are simplified representations of each scenario, as the real world context is inevitably more complicated. Nevertheless, most migration and modernization scenarios tend to follow one or more of the following patterns.

The 'lipstick on the pig' scenario

The first two scenarios are dangerous oversimplifications – common failure scenarios that don't adequately address the complexities of modernizing a legacy environment.

First on the list: the 'lipstick on the pig' scenario. In this scenario the modernization team simply exposes a legacy system via an API – with no consideration of new requirements the existing system cannot deliver. This scenario is shown in figure 1 below (blue represents legacy assets, while yellow represents new capabilities).

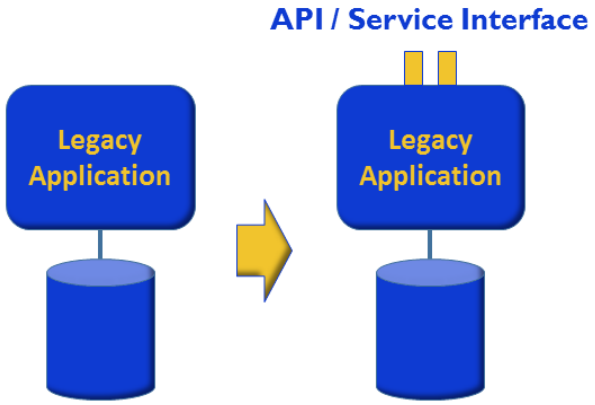

Figure 1: Lipstick on the Pig Scenario

This scenario is of limited value, as it offers little additional flexibility and no new functionality. However, it was one of the most common failure scenarios for SOA initiatives, as many organizations incorrectly assumed that simply exposing a legacy asset with Web Services was all that SOA was about.

The overly simplistic "heart lung machine" scenario

The second scenario is also an overly simplistic anti-pattern that nevertheless became a common pitfall for SOA initiatives. With this 'heart lung machine' scenario, the organization first exposes legacy functionality with one or more APIs. Next, it builds or buys a new system that has an API that precisely matches the API of the legacy system – a goal that has superficial appeal but is difficult to achieve in practice.

MANY ORGANIZATIONS INCORRECTLY ASSUMED THAT SIMPLY EXPOSING A LEGACY ASSET WITH WEB SERVICES WAS ALL THAT SOA WAS ABOUT.

Next the team syncs the data between the two systems and finally cuts over requests from the old system to the new one seamlessly, thus relieving the legacy system of all its duties. At that point it's simple to fully retire legacy system. See figure 2 for an illustration of the 'heart lung machine' scenario.

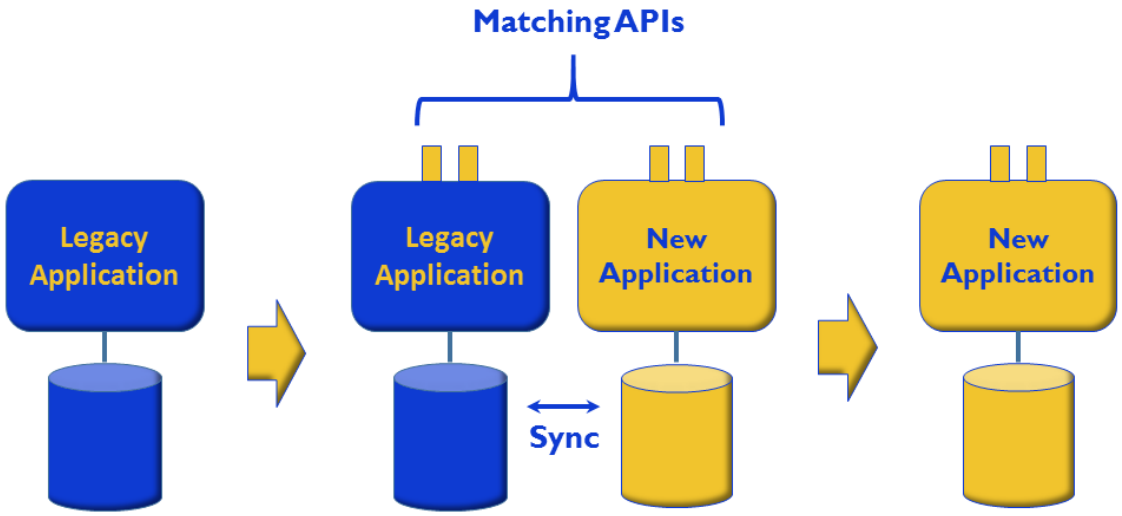

Figure 2: The 'heart lung machine' scenario.

This scenario sounds good but is difficult or impossible to implement in practice. Building matching APIs is nearly impossible. Syncing the data may also be impractical, given the old and new data stores may have incompatible architectures.

Furthermore, all access to the legacy system must take place through the API for there to be any hope of a seamless switchover – a requirement that is also difficult to achieve in practice.

THE 'HEART-LUNG MACHINE' SCENARIO SOUNDS GOOD BUT IS DIFFICULT OR IMPOSSIBLE TO IMPLEMENT IN PRACTICE.

The basic augmentation scenario

The 'basic augmentation scenario' is the first of the six scenarios that organizations have been able to use reliably to solve real-world modernization challenges – although it also comes with its own set of problems.

With this scenario, the organization exposes legacy systems (including legacy applications and databases via existing stored procedures) via one or more APIs.

Then it adds a new system or enterprise application, for example, customer relationship management (CRM), enterprise resource planning (ERP), or business process management (BPM) apps, each of which has its own API. The APIs for old and new thus coexist. The basic augmentation scenario appears in figure 3 below.

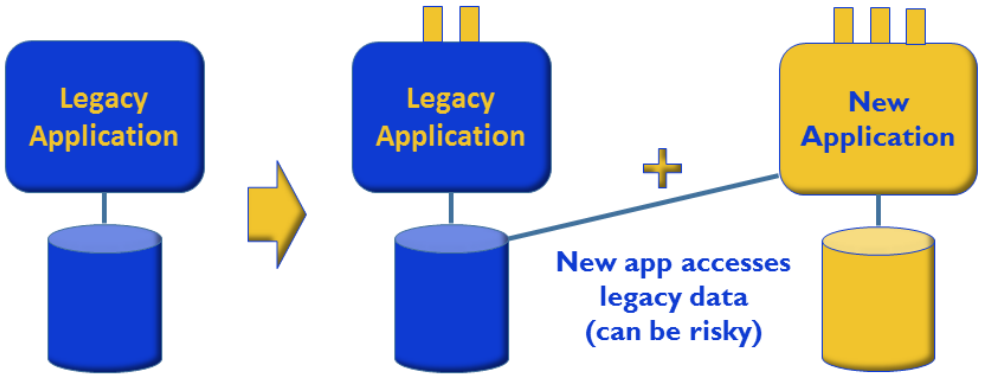

Figure 3: Basic Augmentation Scenario

Once the new application is operational, the team then coordinates the older and newer applications. Such coordination can include publishing all APIs via the same API directory or marketplace as well as composing legacy APIs with APIs from newer applications in order to build new, composite applications.

Challenges with the basic augmentation scenario arise, however, at the data layer. Data architects must make a critical decision: either maintain entirely separate data stores to support the legacy APIs vs. the new applications, or empower the new applications to access legacy data directly, as opposed to accessing such information via the API layer.

Neither alternative is without its own risks. Maintaining separate, coexisting data stores can lead to disparate versions of the truth, thus complicating any master data management initiative in the works.

However, allowing new applications to update data in legacy databases directly as opposed to going through APIs is also asking for trouble, since bypassing legacy business logic can lead to consistency or governance issues, or in the worst case, can actually corrupt data.

ALLOWING NEW APPLICATIONS TO UPDATE DATA IN LEGACY DATABASES DIRECTLY AS OPPOSED TO GOING THROUGH APIS IS ASKING FOR TROUBLE.

The basic modernization scenario

The lowest risk of all scenarios is the basic modernization scenario, but it's not right for every organization. In this scenario, the team exposes legacy applications and stored procedures as APIs. Then over time, it adds new functionality to the legacy applications and updates the APIs accordingly.

Putting new code on old systems is not always feasible, of course – but in some situations, this alternative provides the most cost-effective, low-risk approach to legacy modernization. The basic modernization scenario appears in figure 4 below.

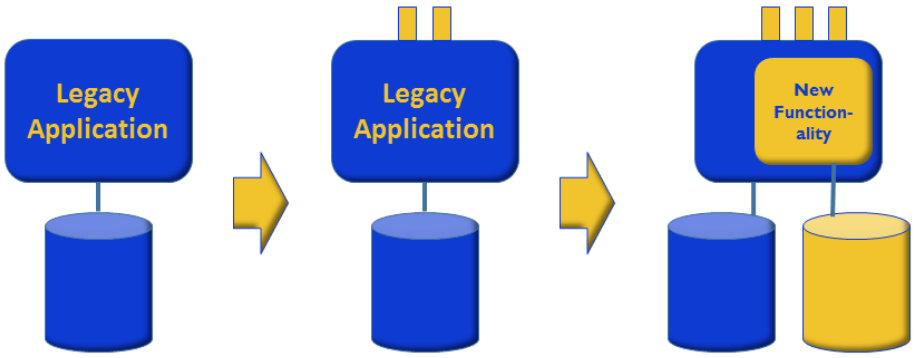

Figure 4: The Basic Modernization Scenario

There are a number of challenges with this scenario, however. It requires coding on legacy systems, which often requires the skills of seasoned professionals who can be expensive and difficult to find. It also requires the abilities of seasoned architects who understand the role the legacy system plays in the modern environment.

The basic reduction scenario

The idea behind the basic reduction scenario is to transition functionality off of legacy applications to newer applications, while allowing for the fact that the legacy systems will still likely continue to provide some value even after this migration is complete, and thus must coexist with the newer applications.

This scenario begins much like the basic augmentation scenario, where the team adds new applications alongside existing, legacy applications, and then coordinates their respective APIs. Once the new applications are up and running, however, developers migrate key functionality from older to newer applications. The basic reduction scenario appears in figure 4 below.

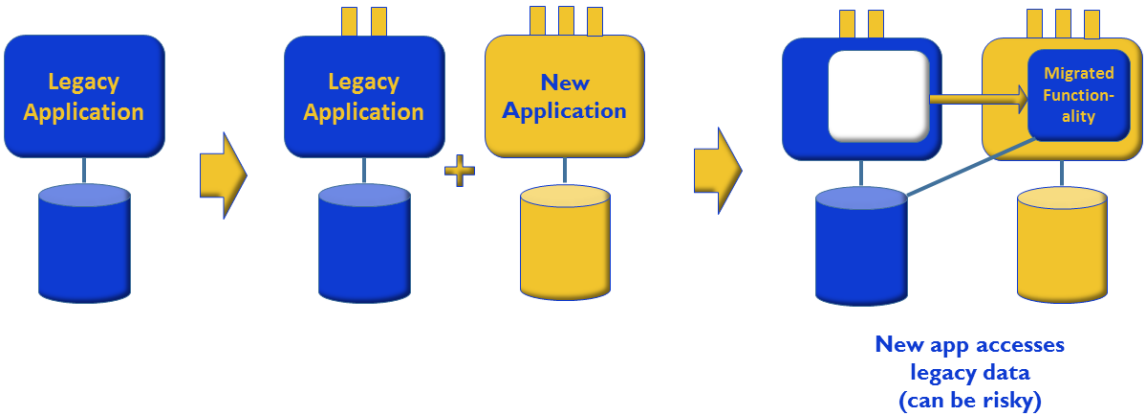

Figure 5: The Basic Reduction Scenario

In some cases the new functionality is identical to the legacy code it replaces, but more often than not, migrating application functionality is a perfect opportunity to update the associated business logic to meet new or changing requirements.

However, changes to such logic can complicate interactions with legacy data stores, just as such changes cause issues with the basic augmentation scenario. The main difference between these two scenarios, however, is that with the basic reduction scenario, the migrated functionality must access either the legacy data store or the new data store associated with the new applications.

As a result, many organizations must also migrate the data corresponding to the migrated functionality – as well as the associated data structure and metadata, including indices, triggers, and stored procedures. Such migration can be deceptively complex and error-prone.

MIGRATING THE DATA, AS WELL AS THE ASSOCIATED DATA STRUCTURE AND METADATA, INCLUDING INDICES, TRIGGERS, AND STORED PROCEDURES, CAN BE DECEPTIVELY COMPLEX AND ERROR-PRONE.

The agile modernization scenario

The final scenario we'll consider is the agile modernization scenario. As with other scenarios, the team begins by exposing legacy applications and stored procedures via APIs and then adding new applications like CRM, BPM, or ERP applications that also have their own APIs. Then like the basic augmentation scenario, the organization also adds new functionality to the legacy environment.

Over time, therefore, the team continues to modernize the legacy applications, while at the same time maintaining and updating new applications, all the while coordinating their APIs. In other words, this scenario provides the most flexible approach to coexistence of all the scenarios in this paper. The agile modernization scenario appears in figure 6 below.

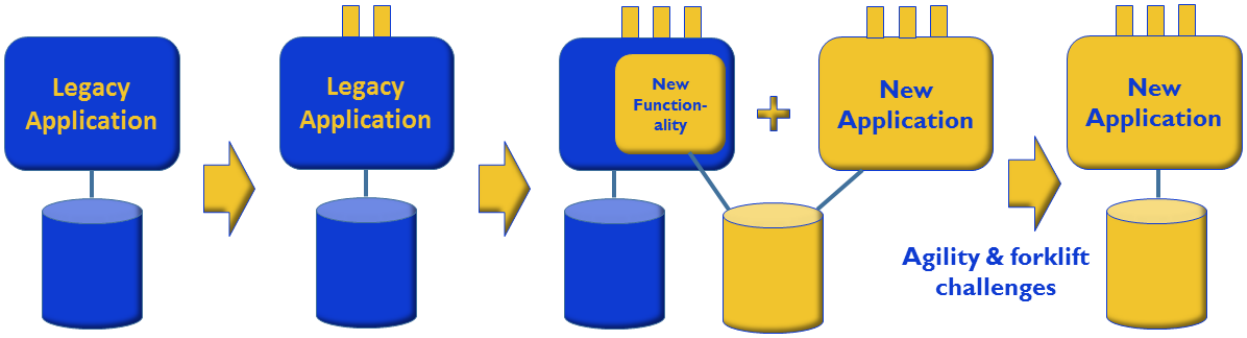

Figure 6: The Agile Modernization Scenario

In essence, modernization continues to take place in the legacy environment, while simultaneously occurring among newer applications – giving the organization the most flexibility of any of the scenarios.

However, because this scenario maintains both old and new data stores that must coexist, and furthermore, new functionality on the legacy systems may update the newer data stores, the organization may run into the same master data management issues that the basic augmentation scenario faces.

Keeping both old and new data stores may not be the long term plan, however. The agile modernization scenario is also an important intermediate step on the road to fully retiring legacy systems, and in practice, is the most realistic alternative to the impractical 'heart lung machine' scenario.

Essentially, over time the organization relies increasingly heavily on the new application functionality. At a particular point in time, then, the team cuts over from the legacy applications to the new applications. However, just as with the heart lung machine scenario, such cutovers are easier said than done.

Because the team continues to add new functionality to the legacy environment, at some point it must terminate such updates and transition all new code to the new environment – a challenge that may not be feasible if the organization depends upon regular, continued updates in the legacy environment.

Transitioning the data can also present a problem, as applications continue to update the data in the legacy data environment. This situation can lead to the forklift problem: at some point in time, the team must move any remaining data all at once to the new system, typically within a maintenance window.

Because of these problems inherent in full migrations off of legacy systems, even with the agile modernization scenario, the coexistence state typically becomes the most satisfactory final state. The bottom line: coexistence almost always trumps full migration off of legacy systems.

THE BOTTOM LINE: COEXISTENCE ALMOST ALWAYS TRUMPS FULL MIGRATION OFF OF LEGACY SYSTEMS.

Analysis: How to Decide which Scenario is the Right One

There is no one scenario that is right for every organization, and furthermore, each of the scenarios has its advantages as well as its risks. Deciding the best approach for any particular company, therefore, can be a difficult task in and of itself.

The most important starting point for making this decision is to understand the business pain points that are driving the modernization/migration initiative in the first place. Only devote resources to efforts that will address such pain points – in other words, 'if it ain't broke, don't fix it.'

Many IT executives fall into this trap when considering the fate of legacy systems. They conclude that because systems are older or leverage languages or other technology that may no longer be popular that retiring such systems is a priority. In fact, maintaining legacy systems that coexist with newer systems on an ongoing basis is usually – but not always – the lower risk and more cost-effective alternative.

That being said, decision makers should take the age and maintainability of any legacy system or application into account when putting together their modernization strategy. Clearly, if hardware is failing and irreplaceable or if the vendor of critical software is terminating its support for such applications, then such factors should impact the decision to migrate.

The challenge of maintaining necessary skills is also an issue – but not a black or white one. Certainly some legacy coding or operations skills are becoming nearly impossible to hire for, and no one wants to learn them either, due to the fact that such skills are unlikely to be transferrable. But many legacy skills –

COBOL and CICS skills, for example – promise years of continued demand, as well as a healthy community of practitioners.

It's also important to consider the challenges as well as the opportunities of data migration. Migrating off of legacy databases can certainly present numerous challenges, but also affords an organization the rare opportunity to clean up obsolete or problematic data. Much as downsizing from a large house to a condo is a rare excuse to get rid of the detritus of decades, so too is the opportunity that data migration presents.

Finally, putting together the right API strategy and corresponding architecture is a critical success factor for any migration or modernization initiative. APIs do not stand alone, but rather form the glue that cements old and new together, as well as bringing together the full range of newer software capabilities in the organization and beyond.

OpenLegacy's solution platform can support and facilitate such an API strategy. It provides the tools and features necessary to enable developers to automatically generate APIs from legacy on premise systems, edit and enhance those APIs, and fully deploy to the targeted environments. Furthermore, OpenLegacy's management console offers security and role management, performance and usage monitoring, and automated API testing and verification.

OPENLEGACY'S SOLUTION PLATFORM PROVIDES THE TOOLS AND FEATURES NECESSARY TO ENABLE DEVELOPERS TO AUTOMATICALLY GENERATE APIS FROM LEGACY ON PREMISE SYSTEMS, EDIT AND ENHANCE THOSE APIS, AND FULLY DEPLOY TO THE TARGETED ENVIRONMENTS.

Keep in mind, however, the lessons of the first two scenarios: the 'lipstick on the pig' and 'heart lung machine' scenarios. Both of these scenarios heavily depended on APIs, but were nevertheless overly simplistic and thus typically unfeasible in practice. One of the most important challenges, therefore, is to avoid going down the wrong path.

Choosing the right path depends in large part on using the right tools. The API quick enablement benefits that OpenLegacy can provide can make the difference between an expensive, time-consuming failed legacy migration effort and a successful, cost-effective migration/modernization initiative that leverages coexistence for ongoing business value.

When used properly, APIs provide a comprehensive abstraction layer that simplifies interactions with legacy applications, modern enterprise applications, stored procedures, and cloud-based SaaS applications.

But make no mistake – such simplification requires robust architecture, careful migration and modernization strategies, and the right tools in order to increase the chances of success. Be sure to leverage a tool like OpenLegacy to reduce risks and increase the chances of success for any legacy migration or modernization initiative.

OpenLegacy is an Intellyx client. Intellyx retains full editorial control over the content of this white paper.