White Paper

How to Maximize Application Modernization ROI

Jason Bloomberg
President, Intellyx

April 2022

Given the prevalence of legacy technologies in today's enterprises combined with the incessant pressure to innovate, CIOs are now struggling more than ever to implement cost-effective modernization strategies.

Fortunately, there are more modernization techniques available today than the risky 'rip and replace' or 'leave and layer' options from previous decades.

In many situations, the best approach to modernizing legacy applications is to leverage automated code migration tools like those from Mobilize.Net.

Mobilize.Net can translate legacy applications into modern code, ready for ongoing refactoring and new development leveraging modern tools, architectures, and environments.

Breaking Free of the Modernization Dichotomy

Over the last several decades, CIOs have struggled with a central modernization dilemma – whether to leave legacy systems as-is, exposing them via APIs or emulation technologies – what we call *leave and layer*, or rip them out entirely, known as *rip and replace*.

There are substantial risks of cost overruns and project failures inherent in retiring legacy assets. Such project failures can result from projects running over budget and behind schedule, introducing new critical defects in business logic, or simply not meeting customer requirements.

Avoiding such risks has generally led to the increasing presence of older technology. However, over the last twenty years or so, one innovation after another from cloud computing to AI have chiseled away at this growing accumulation of legacy.

There are substantial risks of cost overruns and project failures inherent in retiring legacy assets. Such project failures can result from projects running over budget and behind schedule, introducing new critical defects in business logic, or simply not meeting customer requirements.

This challenge has always boiled down to an economic argument: how much is an out-of-date system costing the organization, vs. the all-in cost of modernizing that legacy – including all the indirect costs of making the transition from old to new, including downtime, retraining, and resistance to change.

Today, new technologies and modernization approaches have shifted this economic argument by reducing the risks inherent in modernization.

Your starting point should include understanding the various 'new world' modernization options open to you, and how to make the appropriate decisions about how to move forward given your particular legacy challenges.

The right strategy will likely be a hybrid approach, consisting of combination of any or all of several modernization approaches, including leaving certain legacy assets in place as well as either modernizing assets by updating them in place or modularizing them and modernizing certain areas of their functionality.

Rewriting legacy assets entirely – perhaps the riskiest and most expensive alternative – has actually become a last resort. In order to rewrite complex legacy applications, the development team must fully understand the functioning of the legacy application, in spite of limited or nonexistent documentation, convoluted code patterns, and the lack of full functional regression test suites.

Instead, leveraging tooling like Mobilize.Net's to replace existing applications' source code with modernized versions of the same applications reduces the risks and costs inherent in modernization while providing a code base for future innovation.

Expanding Modernization Options

Cloud computing has added a third option to the traditional modernization dichotomy of rip and replace or leave and layer: *lift and shift*.

In theory, this approach makes sense: simply migrate the application's compiled executable and its associated runtime off its antiquated platform into a virtual machine in the cloud.

However, lift and shift isn't in reality a modernization option at all. It is more of a cloud enablement solution that leaves applications in their unmodernized state.

After all, it's not the application executable itself we want to preserve, it's the *application functionality and associated business logic* that the application represents.

The ideal end-state would be to have an entirely new application – one that followed modern cloud-native best practices – that kept whatever legacy functionality and business logic still provided value to the enterprise.

To overcome this challenge, we can't simply deal with the compiled executable. We must work directly with the source code by translating legacy programs into a modern language that will run on modern systems and architectures – in particular, in the cloud.

The ideal end-state would be to have an entirely new application that kept whatever legacy functionality and business logic still provided value to the enterprise. We can't simply deal with the compiled executable. We must work directly with the source code by translating legacy programs into a modern language that will run on modern systems and architectures – in particular, in the cloud.

Calculating the ROI of Modernization

How, then, should organizations make the decision about how to modernize a particular application? There are three factors to consider:

- *The business priorities driving the modernization requirement* – do customers require new capabilities that existing approaches to updating legacy applications can't deliver cost-effectively? Is there a regulatory compliance driver? Does the legacy application have security vulnerabilities? Is modernization essential to remain competitive? Be sure to consider all relevant business priorities before making the decision how and what to modernize.

- *The mission criticality of the targeted applications and systems* – how important to the organization's day-to-day business are the targeted applications and systems relative to other assets in the enterprise? Mission criticality affects the urgency of a modernization initiative as well as its risk profile, as the business risk of failure of such assets is particularly high, largely because operating systems, libraries, and other components fall out of support.
- *The all-in cost of the modernization* – Sometimes it makes sense to keep a legacy application or system if the cost of modernization is too high. Don't assume that all modernization is a priority until you factor in the costs of remediation.

It's important to consider all of these factors in order to calculate the return on investment (ROI) for any modernization initiative before making any decisions about a modernization strategy.

To calculate the ROI for a modernization initiative, calculate the costs and benefits of executing such a strategy and compare that calculation to the costs and benefits of not executing the strategy and instead maintaining the status quo, across a reasonable time window that would allow for completing the initiative:

- *Maintaining the status quo*: sum the all-in costs of ongoing maintenance with the opportunity costs inherent in not complying with whatever business requirements the existing legacy is preventing the organization from achieving.
- *With modernization*: calculate the difference between the value to the business of the short-term benefits of modernization and the long-term strategic benefits of greater innovation that will result from the modernization initiative and the sum of the all-in costs of the modernization initiative
- *Calculate ROI*: The ROI initiative will be the modernization value minus the status quo value, divided by the status quo value. For a more accurate calculation, take into account the time value of the money required for the initiative (say, by incorporating the interest costs based on the assumption that the organization financed the modernization effort).

For example, assume a 5-year window for the modernization initiative. During that time, the all-in cost of maintaining the status quo is, say, $1 million. The cost of the

modernization is $500,000, and the benefit to the business during that time is $2.8 million.

The ROI would equal [($2.8 million - $500,000) - $1 million]/$1 million, which equals 130%, not factoring in the time value of money.

The Modern Tradeoffs of Legacy Technology

There are many sides to legacy. Legacy might refer to applications, operating systems, or the underlying hardware. Within applications, legacy might indicate the programming language, the software architecture (procedural vs. object-oriented, for example), or the business logic itself. Legacy may also refer to the human skills necessary to keep legacy assets up and running.

Legacy might refer to applications, operating systems, or the underlying hardware. Within applications, legacy might indicate the programming language, the software architecture (procedural vs. object-oriented, for example), or the business logic itself. Legacy may also refer to the human skills necessary to keep legacy assets up and running.

For many executives facing the modernization challenge, focusing on reducing technical debt can help clarify the decision-making process. Technical debt refers to expedient decisions in the past that grow into problems for today's leadership or decisions that made sense at the time but have grown into liabilities – problems that only get worse over time (hence the 'debt' metaphor).

Paying down technical debt essentially means cleaning up the messes of a predecessor. If the mess is bad enough, cleaning it up may be a necessary investment – even though

given the option, any CIO would rather spend their limited budget on new functionality that drives innovation.

This tradeoff is at the core of any modernization decision. Spend too little on it, and legacy will continue to impede the organization's ability to innovate and respond to the needs of customers. Spending too much, however, squeezes the budget for innovation, which also puts the organization at a competitive disadvantage.

How Mobilize.Net Improves Modernization Business Outcomes

The ideal approach to modernization is to understand the specific needs of the business moving forward and where to best invest modernization resources to achieve those business goals.

In some cases, retiring an entire system or application and replacing it with a rewritten or commercial off the shelf (COTS) application is the best move, but neither approach typically offers the best ROI.

The decision of what and how to modernize depends in large part upon the mission criticality of the assets in question.

For non-critical assets, simply leaving them be might be the best choice – or at the least, pushing off the modernization decision until later. In other cases, replacing a legacy app with a modern COTS app may be the most cost-effective option.

For mission-critical applications – or applications that at minimum are unique and provide a competitive advantage – there are four main options:

- *Leave and layer* – Leave existing application in place but access it via a new API layer or emulation technology
- *Repair* – When existing applications mostly meet new requirements, but they require some updates or bug fixes
- *Rewrite* – When the existing application fails to provide required functionality

- *Migrate* – Automatically migrate application functionality and business logic in order to reduce costs, risk, and time compared to a rewrite.

Mobilize.Net offers a set of tools and related services that provide for automated migration of legacy application code that costs substantially less than rewriting applications while reducing risk and time to market.

This migration preserves the functionality and business logic of the original application while opening it up to further development and allowing it to run in a modern environment.

Once the modernization team has leveraged Mobilize.Net to migrate the application, including it in a modern software lifecycle is straightforward. Now that the application code is in a modern language, developers can use modern tools and best practices like CI/CD and DevOps to incorporate once-legacy functionality and business logic into modern applications.

Now that the application code is in a modern language, developers can use modern tools and best practices like CI/CD and DevOps to incorporate once-legacy functionality and business logic into modern applications.

The benefits of the Mobilize.Net automated migration approach to modernization include:

- Mobilize.Net's advanced modernization tools can move legacy workloads from old platforms to new ones
- Automated migration doesn't require modernization teams to recode complex business rules and processes

- Automated migration translates the code directly from the source language to the destination language, using deep semantic analysis to avoid introducing functional defects
- Automated migration puts a functionally equivalent, modernized application into production, replacing the original application
- With automated migration, the modernized application is up and running in a shorter timeframe than manual rewrites – and the development team can promptly work on refactoring and improvements
- It's possible to retire the legacy application much more quickly than for a manual rewrite project, thus reducing the costs and risks of maintaining it while the rewrite is taking place.

Modernization with Mobilize.Net is no magic wand, as it is but one step in the full modernization process rather than the final goal. The tools can, however, dramatically improve the ROI of modernization and help organizations get a handle on their burgeoning technical debt.

The Intellyx Take

Modern enterprises now have the means to modernize legacy applications within a new context that recognizes that there is far more going on with today's modernization than in the old days of the rip-and-replace dilemma.

True, in some cases the business requires entirely new, custom applications, but it may just as likely need to update existing applications, connect new, modular capabilities to older apps, or pull together SaaS and on-premises assets in various ways to meet changing customer needs.

Automated migration of legacy source code is an essential part of this modernization mix and can actually facilitate modernization scenarios that are broader and more strategic than modernizing individual applications.

Once the development team has modern, fully functional source code, it can support whatever innovation the business and its customers require.

About the Author: Jason Bloomberg

Jason Bloomberg is a leading IT industry analyst, author, keynote speaker, and globally recognized expert on multiple disruptive trends in enterprise technology and digital transformation.

He is founder and president of Digital Transformation analyst firm Intellyx. He is ranked #5 on Thinkers360's Top 50 Global Thought Leaders and Influencers on Cloud Computing for 2020, among the top low-code analysts on the Influencer50 Low-Code50 Study for 2019, #5 on Onalytica's list of top Digital Transformation influencers for 2018, and #15 on Jax's list of top DevOps influencers for 2017.

Mr. Bloomberg is the author or coauthor of five books, including *Low-Code for Dummies*, published in October 2019.

About Mobilize.Net

Mobilize.Net builds the world's highest fidelity source code translation technology. Millions of developers have used Mobilize.Net technology to successfully modernize billions of lines of code. Mobilize.Net solutions enable customers to reduce risk, cost, and time while moving applications to the platforms businesses demand today. Find out more at https://www.mobilize.net.