White Paper

# How to Pay Off Your Technical Debt with Legacy Modernization

## Modernize Your Legacy Code for a Cloud-Native World

**Jason Bloomberg**
President, Intellyx

**August 2021**

In today's mad rush to modern technologies and approaches like cloud-native computing, it's tempting to leave older, legacy applications behind.

Such a move, however, is often a mistake. Legacy applications typically still provide value, as the business logic they execute is often mission-critical for the enterprises they support.

Fortunately, there are approaches for modernizing legacy code that gives organizations the best of both worlds: modern cloud-native architectures that preserve the value of legacy business logic while also enabling organizations to lower their long-standing technical debt.

## The Legacy Code Modernization Priority

What do you think of when you hear the word *legacy*?

In common parlance, your legacy is something important you leave behind for your descendants to remember you by – or perhaps, a particularly favored descendant themselves.

In IT circles, in contrast, *legacy* has a strictly negative connotation. The word conjures up images of ancient, creaking computers and their tape drives spinning as they slowly execute decades-old programs in dead languages no one remembers.

In truth, even in the world of computing, the first meaning of legacy may be more accurate. If you're still running that legacy application, you're running it for a reason – because it still provides value. Even better, it's already paid for.

Maybe the application isn't an issue. Legacy might also refer to the data, or perhaps the platform or its architecture.

Nevertheless, we use the word legacy to suggest that a particular application or system acts as a ball and chain on our organization, limiting our ability to innovate and otherwise deal with a changing business and technology environment.

We use the word legacy to suggest that a particular application or system acts as a ball and chain on our organization, limiting our ability to innovate and otherwise deal with a changing business and technology environment.

True, the reason we care about legacy is that it is still providing value, but such value collides with the pressure to innovate.

This dichotomy presents itself in many industries, for example, in insurance. An insurance carrier's business depends on mission-critical data processing tasks like

underwriting, rating, and claims processing. Typically, such processes run in legacy applications on legacy systems – providing unquestionable value while also limiting the carrier's agility and, thus, ability to innovate.

Keeping such legacy as-is will make an enterprise uncompetitive while ripping it out is cost-prohibitive and often worsens the situation. There's got to be a better way.

## Why 'Lift and Shift' is Rarely the Answer

For decades, IT executives were faced with two unpalatable options for legacy application modernization: *rip and replace* or *leave and layer*.

When a legacy application still provides value, ripping and replacing is a high-risk option. The replacement may not adequately meet the organization's needs, and the transition from old to new may be overly disruptive to the enterprise.

However, the leave and layer approach isn't much better. True, adding APIs or connectors to a legacy application can extend its lifetime, as can leveraging robotic process automation (RPA) to script its user interface.

The problem: leave and layer adds to technical debt, kicking the modernization can down the road. Any shortcomings of the legacy app remain an obstacle to the enterprise.

Cloud computing has added a third option: *lift and shift*. In theory, this approach makes sense: simply migrate the legacy app off of its antiquated platform into the cloud, and voila! It's modernized.

Lift and shift, however, rarely delivers on this promise. Just like leave and layer, it also adds to technical debt, and it doesn't help with legacy applications written in languages that won't run in the cloud.

Despite its seeming straightforwardness, lift and shift into the cloud is more of a misnomer than a real solution. After all, it's not the application itself we want to preserve, it's the *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 all the legacy business logic that still provides value to the enterprise.

## Software Architecture: The Biggest Stumbling Block

Lift and shift simply won't solve your modernization problems – if only for the architectural challenges.

If the legacy application is monolithic, then it will remain monolithic in the cloud. Even if your starting point is object-oriented (OO), the modularization that microservices require doesn't generally line up with the OO approach to modularization. As they say, the devil is certainly in the details.

There is one other important consideration. While in some cases, the modernization requirement is to leave certain legacy business logic as-is, but more frequently, the organization requires the ability to update that business logic moving forward.

In some cases, the modernization requirement is to leave certain legacy business logic as-is, but more frequently, the organization requires the ability to update that business logic moving forward.

In other words, refactoring – updating the structure and organization of the code without changing its functionality – may or may not be the only priority. Updating that functionality may also be a requirement.

In either case, refactoring may be a necessary (albeit not always sufficient) requirement for any legacy modernization project, especially if such refactoring involves architectural modernization.

Updating the code is important. Updating the architecture – moving from monolithic or legacy object-oriented to cloud-native, say – is even more strategic for one reason: architecture transformation can lower your technical debt.

## Balancing Automation and Human Expertise

The obvious solution to the problem of legacy apps written in cloud-unfriendly programming languages is to translate those programs line-by-line into a modern language that will run in the cloud.

Indeed, there have been tools for conducting such translations on the market for years. Such line-by-line translation on its own, however, rarely addresses the business need. It barely touches the technical debt, and it infrequently leads to maintainable code moving forward.

For many organizations, the goal of legacy code migration is to end up with maintainable modern applications, which means a combination of microservices as well as user interfaces that meet the needs of today's increasingly mobile workforce and customer base.

To achieve these goals, code migration must do more than translate legacy code. It must transition legacy software architecture to microservices architecture – a task that today's tools cannot fully automate.

The missing piece of this puzzle is *human expertise*. Software, after all, is a human product; we've yet to invent AI smart enough to create quality code, especially in the context of modern software architectures.

It's also important to remember that increased agility is one of the primary benefits of legacy modernization. Organizations want to go from less flexible to more flexible software as they update the code itself.

Building flexibility into software once again requires the human factor. Therefore, we must incorporate software engineering best practices across the lifecycle of the software in question, bringing best practices like iterative development to the modernization initiative.

Iterative software development is a central Agile principle, but it actually predates the Agile movement by several years. We now widely recognize iterative approaches as the best way to lower the risks due to unclear or changing requirements in the face of a dynamic infrastructure.

It's important to realize, therefore, that any legacy modernization project would benefit from an iterative approach over the lifecycle of the initiative.

Translating legacy code to a modern language is but one piece of the puzzle. This lifecycle also includes the integration of transformed legacy code with new code and architecturally-driven changes, as well as the resolution of deployment, testing, and updating challenges.


Translating legacy code to a modern language is but one piece of this puzzle. This lifecycle also includes the integration of transformed legacy code with new code and architecturally-driven changes, as well as the resolution of deployment, testing, and updating challenges.

Iterative approaches also help mitigate the challenge of scope creep, where new requirements appear during the course of the initiative.

Given the big bang modernization efforts of past decades often took years to plan and implement, scope creep typically presented a risky impediment to successfully completing the initiative. In fact, many modernization efforts failed completely because of scope creep.

Our best advice: avoid waterfall thinking, especially when it comes to modernization. You never want to plan on completing a modernization effort before moving onto other tasks. Instead, take an iterative approach that includes modernization alongside new application development in each iteration.

## Deploying Modernization Efforts in Practice

Perhaps the greatest challenge with legacy application modernization is the fact that the original application is still running and will continue to run until your team is ready to cut over to the modernized application.

In some cases, it's possible to phase in the new application – but not always. It may also be advantageous to run the new and old applications in parallel for a while, A/B testing the new software while the old software meets most of the day-to-day needs.

How practical such parallel operation might depend upon several factors, including the status of the underlying data infrastructure. If your modernization initiative includes transitioning from a legacy database to a modern one, whether to synchronize that cutover with the application cutover is a question you must resolve ahead of time.

If your organization has the luxury of a weekend of downtime to effect the transition, so much the better – but in today's 24 x 7 digital world, you may not have that luxury. So be sure to plan accordingly.

## The Importance of Testing

Testing modernized applications also present some unique challenges. In some cases, the goal of the modernization effort is functional equivalence: the new application is supposed to behave precisely the same as the old. In such cases, testing is straightforward, as the new application must pass the same tests that the original one did.

However, in most cases, there is an additional requirement to improve the functionality of the application – if only its user interface. It's unlikely that you'd want to modernize a mainframe 'green screen' application, only to have the new interface work like the old one.

Testing in such situations requires more thoughtful planning. It's essential to ensure the functionality that you want to maintain works properly while also allowing new functionality (either business logic, user interface, or some combination) to pass the necessary tests.

## Scoping and Management across the Lifecycle

Finally, the proper scoping and management of architecture transformations is absolutely essential to the success of any modernization initiative.

Today's modernization efforts typically focus on cloud-native architectures consisting of containers and microservices running in Kubernetes-supported environments. But even if cloud-native is still squarely in your future, you will likely be transitioning to a Web-based n-tier architecture at the least – an architecture that also represents a transformation of your original legacy architecture.

The architectural plan for your software goes hand in hand with the infrastructure that will support your software runtime. For example, if you plan to run Kubernetes in a hybrid private cloud/public cloud environment, your software architecture should take that plan into account.

The architectural plan for your software goes hand in hand with the infrastructure that will support your software runtime. For example, if you plan to run Kubernetes in a hybrid private cloud/public cloud environment, your software architecture should take that plan into account.

In any case, you must be sure to consider the entire modernization lifecycle. Synchrony Systems offers its Modernization Lifecycle Platform (MLP) that supports this lifecycle from analysis and planning to transformation and remediation to build and deployment to testing and production release. The MLP applies iterative, automation-driven modernization processes to generate production-ready, modernized applications.

## The Intellyx Take

While careful planning and execution can reduce the risks of modernizing legacy business logic, transforming older application architectures to cloud-native, we might still step back and ask whether legacy code has a place in cloud-native computing at all.

The answer is: yes, absolutely, if doing so meets the business need.

Cloud-native computing, after all, is more than microservices and containers and Kubernetes. It is, in fact, a paradigm shift in enterprise IT overall, consisting of fundamental transformations to all aspects of hybrid IT – including on-premises assets generally and legacy applications in particular.

Cloud-native computing is, in fact, a paradigm shift in enterprise IT overall, consisting of fundamental transformations to all aspects of hybrid IT – including on-premises assets generally and legacy applications in particular.

And yet, while bringing legacy business logic forward to modern computing architectures may very well be the best decision in certain circumstances, it is nevertheless not always the ideal choice.

In particular, make sure you are avoiding the sunk cost fallacy: don't continue to rely upon an existing asset simply because it cost you money in the past if a new asset will better meet the business need moving forward.

Sometimes retiring that legacy asset really is the best choice – and the most direct approach for eliminating technical debt. In many situations, however, there are better approaches for retiring such debt while maintaining the value of legacy assets at the same time.

Making the right decision can be difficult. Be sure you consider all the factors, in particular, the modernization approaches from companies like Synchrony Systems.

## 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 Synchrony Systems, Inc.

We help enterprises transform legacy in-house applications to modern technologies while preserving business-critical functionality. Synchrony's Modernization Lifecycle Platform (MLP) is a scalable, cloud-based platform for managing and executing end-to-end migrations and modernizations of legacy IT applications to modern software architectures and platforms. It enables automated code conversion, transformation, remediation, and upgrades of millions of lines of code in minutes, ensuring consistent, reliable, and repeatable results. MLP was named a 2019 SIIA CODiE Award Finalist for Best Emerging Technology and 2018 SIIA CODiE Awards finalist for Best DevOps Tool.