

White Paper

# A Graceful Path to Legacy Modernization

# Contents

## 03 Executive Summary

## 04 Legacy Modernization Terminology

- 05 Infrastructure Modernization
- 06 Application Modernization
- 06 Infrastructure vs. Application: The Chicken or the Egg?
- 07 The Gordian Knot
- 09 Graceful Modernization – Taking the First Step

## 10 LzLabs Language Runtime™ (LzLR™)

- 10 LzWorkbench™

## 11 Bringing it All Together

## 11 LzLabs - With you All the Way

# Executive Summary



**Mark Cresswell**

Chief Executive Officer,  
LzLabs

As an organization builds up a portfolio of applications to support its business, there is a need for modernization to be understood. As technological possibilities grow and business models change, the demands for modernizing a company's IT infrastructure increase. But when it comes to modernizing mainframe environments, while "the corporate mind is willing", often "the company flesh is weak". Investments are sunk, applications work (albeit in standstill fashion), cost of modernization is too high and the risk of migration often considered too great.

This document discusses why the perception of modernization risk is so high with mainframe applications, and introduces the features of LzLabs Software Defined Mainframe, which have been designed specifically to overcome these challenges.



**Dale Vecchio**

Chief Marketing Officer,  
LzLabs

## Legacy Modernization Terminology

Before diving into a discussion on the complexity of legacy modernization, it is worth establishing some important terminology and definitions about legacy modernization as an objective.

Applications, developed to support business requirements, are never static; they change along with the business.



Modern development environments, infrastructure and DevOps processes have benefited from tremendous innovation in recent years, enabling IT to more rapidly evolve applications to support these changing business requirements.

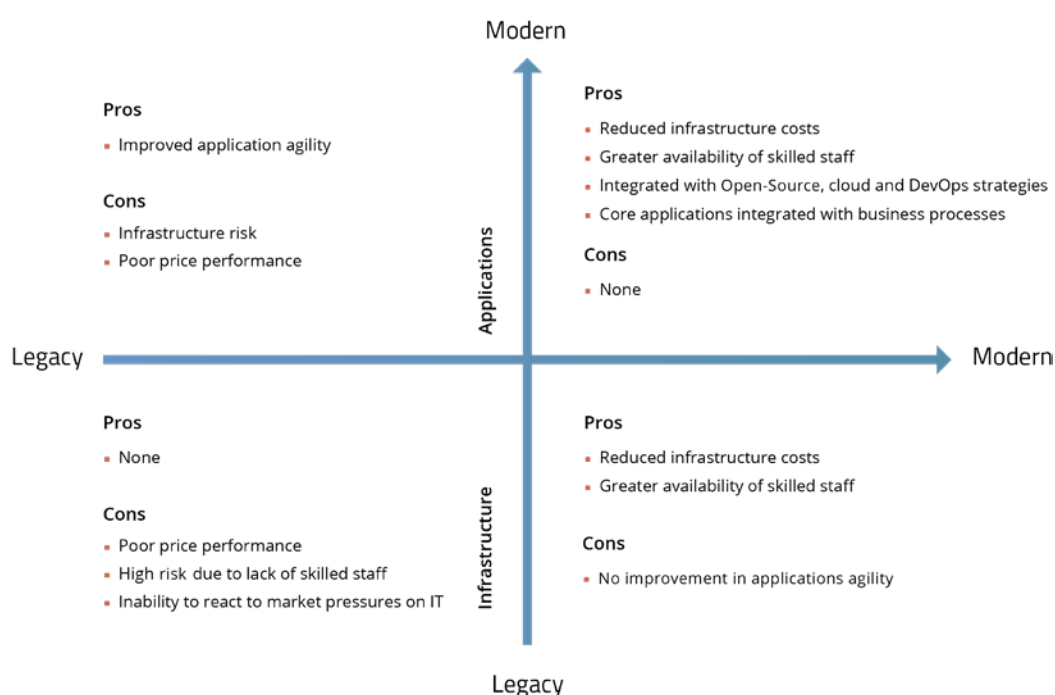
These innovations are almost entirely absent on legacy mainframe systems.

Consequently, there is mounting pressure to modernize applications that currently run on mainframe systems, so-called legacy applications.

The aim of this modernization is to enable Legacy Applications to take advantage of the innovations available in contemporary environments and therefore respond more rapidly to the needs of the business. These innovations fall into two broad categories:



- **Innovations in infrastructure.** These include the vast body of Open-Source solutions that significantly reduce the development burden and enable free access to cutting edge software; commodity hardware; cloud infrastructures and other cost-effective models for scaling capacity; analytics and machine learning services; highly optimized development environments; testing and build processes; and containerization, to name a few.
- **Programming languages.** Gone are the days of having to develop in archaic procedural languages such as COBOL or PL/1 on green screen terminals. Today's object-oriented development languages, such as JAVA, supplemented with modern development environments, make programmers far more productive.



Legacy modernization thus falls into two similar categories

1. **Modernizing IT infrastructure** – placing applications into an environment where they can scale easily and cost-effectively, exploit Open-Source solutions, and benefit from DevOps toolchains and so on.
2. **Modernizing applications** – rewriting, or transcoding into modern languages, only those legacy applications or programs that are most in need of modernization.

Let's look now in more detail at the implications of these two modernization vectors.

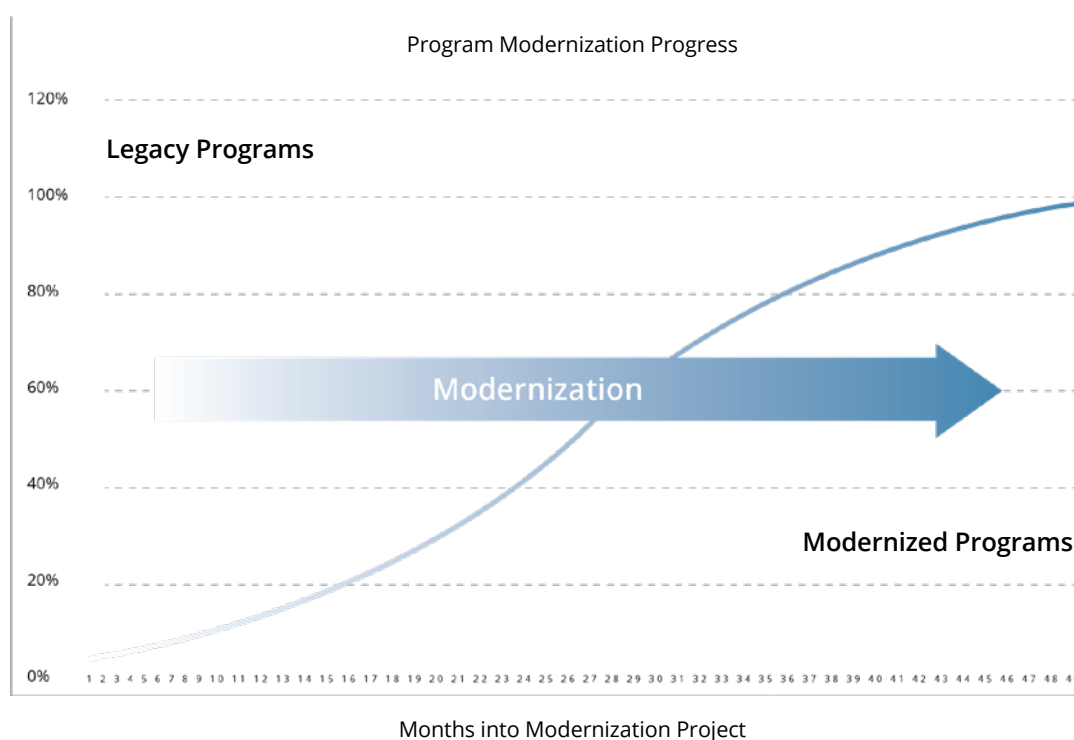
## Infrastructure Modernization

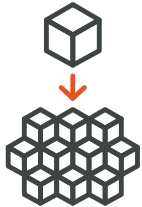


The foundation of infrastructure modernization is the movement of legacy applications, written and compiled to run on IBM Z computers, to the Linux operating environment running on x86 architecture computers; either on-premise or in the cloud. Using conventional methods to rehost these legacy applications into x86/Linux requires all the programs that make up the legacy application be recompiled and, in many cases, changed.

The LzLabs Software Defined Mainframe® (SDM) eliminates the requirement to change and recompile programs being rehosted. This makes SDM the only plausible option for infrastructure modernization when source-code is not available or easily identifiable for recompiling. LzLabs explores these points in detail in this whitepaper.

Once in a Linux environment, legacy applications can benefit from a vast array of tooling to optimize their ongoing exploitation. They can take advantage of cost-effective scaling options; modern monitoring, high-availability tools and many other facilities available on Linux on x86. Furthermore, for those legacy applications that continue to be maintained, modern DevOps processes and development environments are now available. Rehosing legacy applications onto Linux on x86 makes IT able to more rapidly respond to changing business requirements.





## Application Modernization

One optimal outcome for most companies is to have their entire application portfolio based on modern languages or packages. Where legacy applications are concerned attaining this means converting the individual programs from COBOL or PL/1 (and in some cases even more arcane languages) into modern languages such as JAVA. In so doing, an organization can tap into much larger pools of development talent and tooling. Furthermore, the door to an entire world of Open-Source innovation is opened, which alters entirely the trajectory of application development for these hitherto legacy applications.

Yet modernizing applications in this way is a non-trivial undertaking using conventional methods - a subject we explore in more detail in the section The Gordian Knot below.

---

### **"A Journey of a Thousand Miles Starts with but a Single Step"**

---

This famous quote, attributed to Lao Tzu, has been used in support of many business problems. Legacy application modernization is certainly one for which the maxim applies. In this section, we explore the logical and most practical steps required for an organization to travel from legacy applications to a thoroughly modern infrastructure and application portfolio.

## Infrastructure vs. Application: The Chicken or the Egg?

The first question to face when looking to modernize an application is whether to start with the infrastructure or the application itself.

The organization could attempt to modernize the applications in place. At first blush this method seems the least disruptive and most agile, but very quickly the effort would run into many of the challenges, previously highlighted, that have caused the mainframe to be abandoned as a contemporary platform for application modernization. Using Object COBOL is the only way to enable interoperability with those programs being modernized, and recompiled.

With the whole process being conducted on a legacy mainframe, costs and resource retention become a practical issue also. Organizations that run mainframes carefully control access to mainframe resources due to the punitive costs of exceeding licensed capacity; indeed, this resource contention point is another common reason for companies moving ongoing development off the mainframe.

When taken collectively, it is easy to understand how these factors have conspired to thwart most modernization initiatives that choose to modernize applications first.

By modernizing the infrastructure first, the issues identified above are circumvented.

As mentioned in the previous section, LzLabs has made the process of modernizing infrastructure, a step on the road to application modernization, more straightforward. A full description of all the capabilities of SDM, in respect of infrastructure modernization, is beyond the scope of this paper, sufficed to say by eliminating the need to change and recompile programs, or modify data formats, the majority of barriers to rehosting a legacy application onto Linux on x86 are eliminated.

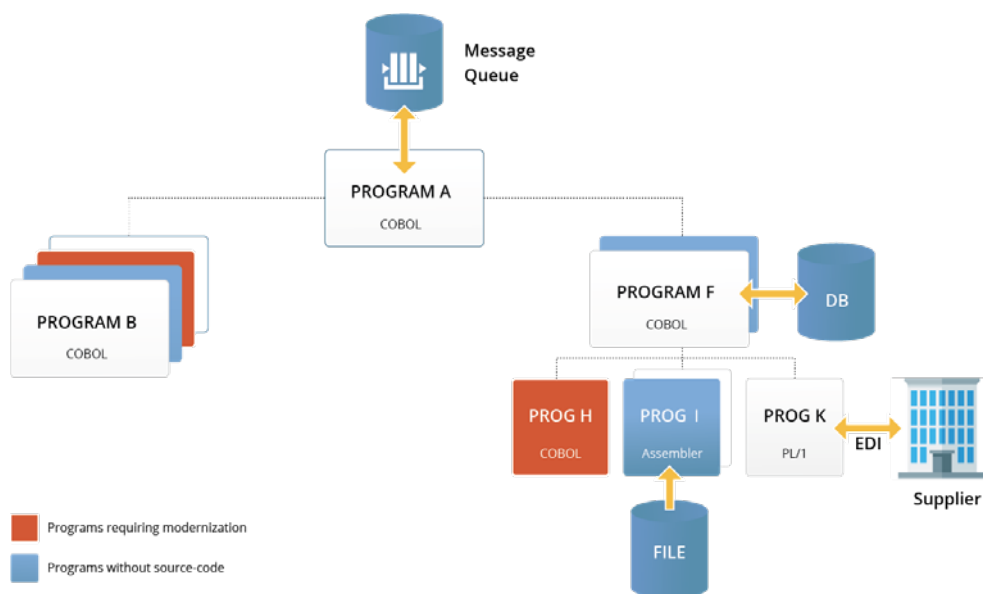


Figure 1 - Simplified Mainframe Application

## The Gordian Knot

The moment an organization begins to review what's required to modernize even a single program within a legacy application, the intricate web of interdependencies between programs and files means the scale of the task grows...rapidly. To illustrate this point, consider this typical but hypothetical and highly simplified legacy application:

The above example shows a hypothetical example of a business that has a legacy application made up of 11 programs. The business has determined that there would be value in modernizing three of these programs, but that it is not necessary to change the functionality of the other eight.

In this example, program A is triggered by a message arriving. Program A retrieves the message and starts processing; a typical entry point for a mainframe application.

The processing of the message determines whether program B is launched or program F. When program F loads, it performs database I/O, which leads it to run some combination of programs H, I and K, and other programs upon which those three depend. Program H is written in COBOL and the organization has a repository of its requisite source code. Program I is written in Assembler, with no available source code, but it reads a very important control file. The program was written in 1980 and the staff with knowledge of its functionality are long retired. Program K writes out an Electronic Data Interchange (EDI) file subsequently used for communication with a supplier.

The organization running the application would like to modernize this application to support a more modern interface to the supplier and have the application react to A/B testing information from its customer-facing website. The business has determined that such changes would enable it to obtain better pricing from its suppliers and streamline its inventory management.

As you can see from the diagram above, the dependencies between the various programs and files, which constitute the application, are very tight. In some cases, those dependencies are embodied in programs for which the source-code no longer exists.

**An important consequence of the legacy application being active on Linux x86 infrastructure is the flexibility with which full-scale modernization can be pursued.**

Any modernization strategy that requires access to the original source-code is going to run in to issues that can only be solved with rewrites of those missing programs, not to mention the recompilation of the programs for which source-code remains. Again, the challenges of this approach are well documented and are discussed in more detail in the aforementioned whitepaper.

In our example, let's assume that the infrastructure for our hypothetical application has already been modernized using the SDM, enabling this application to run on x86 hardware without any changes to the source-code or any recompilation. The legacy application could run reliably, albeit without any modernization-related changes yet, on a modern infrastructure supporting a production workload with the same scale and serviceability as during its previous life as a mainframe application.

An important consequence of the legacy application being active on Linux x86 infrastructure is the flexibility with which full-scale modernization can now be pursued.

Purists would have you believe that re-writing all these programs in a modern language from scratch is the only way to create thoroughly modern applications.

But, even for this relatively simple application, such an endeavor could run into many months, or even years. And it may simply be impossible, as the original legacy application is likely to undergo "Business as Usual" changes during the modernization process. As a result, modernization will be chasing a moving target. This assessment is increasingly well-understood by seasoned IT professionals, consequently the inevitable strategy for application modernization has become procrastination.

The truth is that not all the constituent parts of a legacy application need to change for it to be modernized. Analysis of the hundreds of components that make up a typical legacy application will yield an understanding of which need a more intrusive level of modernization and which don't. As previously mentioned, in our fictitious example above, only 3 out of 11 programs really need to be touched to achieve the business objective.

What determines whether a project, to modernize a legacy application, will succeed or fail is the extent to which those components of the legacy application, which DO NOT need modernization, are forced to change as well.

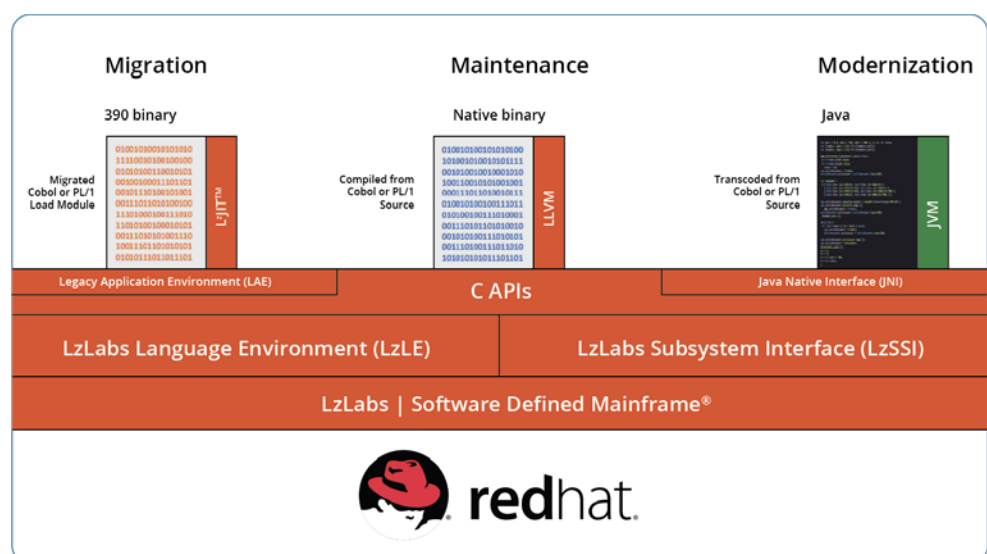


Figure: 2 - SDM Application Modernization Architecture



## Graceful Modernization – Taking the First Step

A better approach is to modernize the application, once it is running on Linux on x86, in a step-by-step manner. However, without a technology like SDM, this is impossible.

The tight coupling between the constituent programs of the legacy application means that even if we just want to modernize Program F, all the programs with which Program F interacts would also need to be changed. This is due to the mechanisms used by tightly-coupled legacy programs to call each other. Old COBOL and Assembler programs use low-level operating system facilities to interoperate with each other. These facilities have almost no capability to invoke modern programs such as Java in a natural manner. As previously stated, for companies choosing to modernize an application on a mainframe the only option is to rewrite those programs, not requiring modernization, in Object COBOL.

If the legacy application is running within SDM on Linux, this problem is entirely solved.

An organization can gracefully modernize just the programs that need it most, without having to touch any of the other programs with which the modernized program interacts.

SDM does this via a common runtime model and subsystem interface for all its supported languages.

---

**An organization can gracefully modernize just the programs that need it most, without having to touch any of the other programs with which the modernized program interacts.**

---

## LzLabs Language Runtime™ (LzLR™)

LzLR exists primarily to ensure the runtime requirements of the rehosted applications are satisfied in the same way on Linux as they are on a legacy mainframe.

However, this common runtime model has enabled LzLabs to enhance runtime features to enable seamless interoperability between legacy programs and the programs of modernized applications. Stated differently, SDM enables an existing COBOL program, which has been rehosted onto a Linux operating system, without any changes or recompilation, to “CALL” a JAVA program naturally; and vice versa. Nothing needs to be done to either the COBOL program or the JAVA program to make this happen. Simple configuration of the SDM environment is all that is required.



In the hypothetical example above, the company wishing to modernize the legacy application could simply convert Program F to JAVA, and use standard JAVA methods to invoke Programs H, I and K. Furthermore, the entry Program A, would be able to continue to invoke the new JAVA program using its existing CALL interface without any changes or recompilation.

Some readers may have, quite rightly, balked at the phrase “simply convert Program F to JAVA”. When done manually, the process is far from “simple”; rewriting significant amounts of code is time consuming and risky. At LzLabs we have developed tooling that can automate many aspects of this conversion.

## LzWorkbench™

LzWorkbench provides an Eclipse-based development framework, that is designed for programmers to enhance existing COBOL and PL/1 applications. Completely new programs can even be developed and interoperate with existing mainframe binaries seamlessly.

LzWorkbench also has a transcoding feature that automatically converts COBOL into JAVA. This generated code will work immediately as it preserves all calls to subsystems in the correct syntax. It is also designed to be easily read and understood by competent JAVA programmers. The result is an automatically generated, fully-functional starting point for the modernization exercise.

## Bringing it All Together

---

For the organization running our fictitious legacy application, the journey from an old, impenetrable system, which is slow to respond to business requirements, expensive and awkward to run and maintain, to a thoroughly modern system, which enjoys all the benefits of open systems innovation, access to talent, agility and cost, can now be accurately and credibly charted.

The journey starts with the re hosting of the entire legacy application to run within SDM on Linux/x86. Made possible by the core-capability of SDM to enable legacy applications to run without source-code changes, recompilation or data format changes on Linux/x86 systems.

This step of the journey benefits from a very low risk profile as neither the legacy application, nor its data are forced to change in any way. The same binary executables are operating on the same data formats.

Once in this Linux/x86 infrastructure, LzWorkbench, in conjunction with other modern development tools and processes, can be used to provide a platform for ongoing maintenance of programs within the legacy application. Crucially, LzWorkbench can be used to fast-start the next step of the journey - application modernization.

With the benefits of LzWorkbench the organization can now gracefully modernize each individual program, at its own pace with minimal risk. An individual program can be transcoded to JAVA, without changes to a single additional component, and the entire legacy application will continue to work identically. Small, incremental modifications can subsequently be made to the JAVA program in support of further modernization requirements. The entire process fits perfectly with today's agile development methodologies, tools and processes.

The result will be a thoroughly modern application, operating in a thoroughly modern infrastructure, achieved with minimum risk or disruption.

## LzLabs – With you All the Way

---

The journey of modernization is not easy. Continuing to depend on a declining skill set and diminishing vendor community is no longer an acceptable alternative. By breaking up the modernization journey into manageable pieces, moving forward over time to the desired end state is the most realistic approach. Migrating existing application systems to packaged software solutions, or rewriting them entirely from scratch may seem enticing, but don't be fooled. The marketplace is a graveyard of failed projects that tried to take this approach. LzLabs is dedicated to providing realistic solutions to this journey. Utilizing the Software Defined Mainframe and LzWorkBench to reduce the risk of your modernization journey is a balanced approach for any mainframe modernization project. Don't procrastinate and don't take risks in the hopes of technological purity. The first step of your modernization journey should begin with LzLabs.

---

**By breaking up the modernization journey into manageable pieces, moving forward over time to the desired end state is the most realistic approach.**

---



---

## About LzLabs

LzLabs is a software company that develops innovative solutions for enterprise computing customers, including its **LzLabs Software Defined Mainframe® (SDM)**. The company was founded in 2011 and is headquartered in Zürich, Switzerland. The SDM liberates and enables customer legacy applications to run unchanged on both Linux hardware and Cloud infrastructures. Thousands of mainframe transactions are processed per second, while maintaining enterprise requirements for reliability, availability, serviceability, and security. Our software solution provides unrivaled compatibility and exceptional performance, dramatically reducing IT costs. LzLabs' offices in Switzerland and the UK are home to highly-experienced mainframe experts and modern IT thought leaders from across the globe.

---

## Contact Us

 **LzLabs GmbH**  
 **@LzLabsGmbH**  
 **info@lzlabs.com**

LzLabs GmbH  
Richtiarkade 16  
CH-8304 Wallisellen,  
**Switzerland**

+41 44 515 9880

25 Templer Avenue  
Farnborough  
Hampshire, GU14 6FE  
**United Kingdom**

+44 (0)1483 319185

## lzlabs.com/products

LzLabs®, the LzLabs® logo, LzLabs Software Defined Mainframe®, LzSDM®, LzOnline™, LzBatch™, LzRelational™ and LzHierarchical™ are trademarks or registered trademarks of LzLabs GmbH.  
z/OS®, RACF®, CICS®, IMS™ and DB2® are registered trademarks of International Business Machines Corporation. Linux is a trade mark or (in some countries) registered trademark of Linus Torvalds. All other product or company names mentioned in this publication are trademarks, service marks, registered trademarks, or registered service marks of their respective owners. Other third party marks are the trademarks or registered trademarks of their owners.