Implementing the Vision of the Modern Mainframe

Transforming Core Mainframe Code with Generative AI

By Jason Bloomberg

Table of Contents



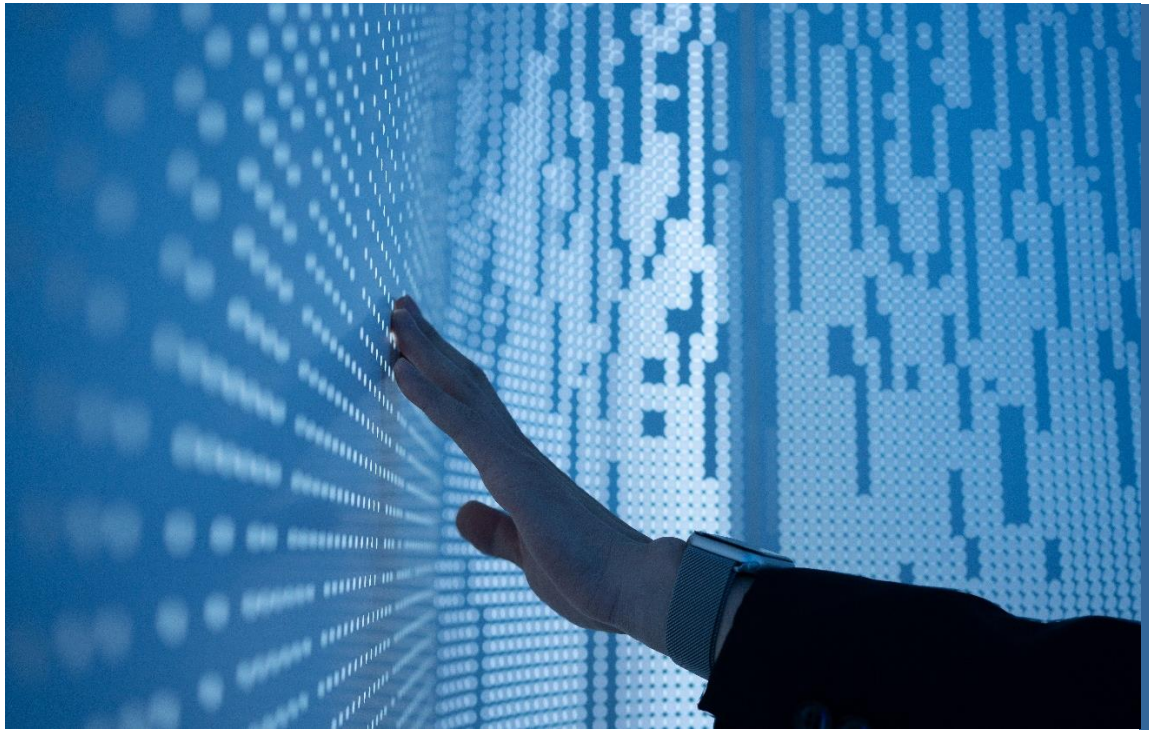

Mainframe applications run the global economy – even though many mission-critical apps are decades old.

Mainframe-based organizations face a growing challenge: how to modernize critical core mainframe applications to meet evolving business needs?

Historically, fear of breaking older applications has impeded progress with modernization. Today, however, modern development tooling simplifies the process of modernization, empowering developers to make the changes the business requires.

Understanding the structure and flow of such applications is insufficient. The missing capability: the ability to explain what critical core mainframe code *actually does*.

With the assistance of generative AI, however, code explainability – and with it, successful modernization – is finally within reach.



Overcoming the Fear of Core Mainframe Code Modernization

No other platform in the history of computing compares with the mainframe. Sixty years on, this venerable technology continues to provide mission-critical capabilities to the largest enterprises on the planet.

This combination of longevity and mission criticality, however, comes with its own problems – problems unique to the mainframe environment.

Many of the most important mainframe applications – think bank core transaction processing and airline reservation systems, for example – have been in operation for decades.

As time went on, generation after generation of mainframe programmers would tweak these applications, adding new code as necessary to meet changing business requirements.

The older applications mainframe developers maintain consist of layers of different code, from numerous programmers, following evolving practices over the years.

Meanwhile, mainframe coding best practices have continued to evolve along with the rest of the IT industry. From procedural to object-oriented and now to cloud native, mainframe programmers do their best to keep up with the times.

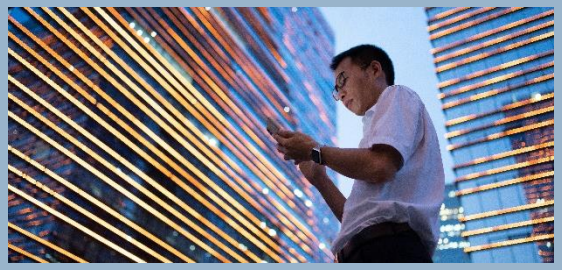
As a result, the older applications they maintain consist of layers of different code, from numerous programmers, following evolving practices over the years.

Struggling with the Fear of Change

It's no wonder that we call such applications 'hairballs' or 'spaghetti code.' Maintaining such applications has been a convoluted and difficult task, for fear that any change might break something important.

For IT executives struggling with mainframe hairballs, such fear is their natural reaction. Fear that changing something will break an old, yet mission-critical application.

It's not simply the possibility that such applications are inherently fragile. The problem is that no one knows just how fragile they are.



The good news: it's possible to untangle the spaghetti. Modern mainframe development tools not only simplify the maintenance *of core mainframe applications; they facilitate the* modernization *of them.*

Such fear impedes any effort to modernize mainframe applications. As a result, leaving them alone altogether becomes the default choice. Tweaking them a bit around the edges may be necessary, but only in the direst circumstances.

The good news: it's possible to untangle the spaghetti. Modern mainframe development tools not only simplify the *maintenance* of core mainframe applications; they facilitate the *modernization* of them.

Such tools help mainframe developers:

- Visualize mainframe code, shining the light on how the software functions
- Understand the interactions, dependencies, and relationships within and between programs
- Break monolithic applications (aka 'hairballs') into smaller callable components, facilitating the move to modular architectures that provide greater flexibility.

In spite of these new capabilities, there is still one piece missing from the puzzle. Such tools have never explained what the code actually does. It's always been left to the developer to piece together core business functionality, line by line.

In spite of these new capabilities, however, there is still one piece missing from the puzzle. Such tools have never explained *what the code actually does*. It's always been left to the developer to piece together core business functionality, line by line.

That is, until now. By applying generative artificial intelligence (genAI), modern tools like BMC AMI DevX Code Insights with BMC AMI Assistant can fill this gap, providing the code explainability that has always been the missing piece, not only of core code maintenance, but also modernization in particular.

Transforming Mainframe Code to Conform to Modern Standards and Architectures

As the last section would suggest, code maintenance and code modernization are two quite different things.

Maintenance requires keeping code in place. Developers fix issues as they come up and add new functionality as the business needs evolve.

Maintenance can also mean some level of refactoring (rewriting code to improve it without adding functionality) – but such refactoring is limited by the adage 'if it ain't broke, don't fix it.'

Modernization considers the target architecture of the entire IT landscape and then places the transformation efforts for any legacy mainframe code within that target architecture.

With maintenance, fear rules the day. Do as little as possible to core mainframe applications to meet the business need – and whatever you do, don't break them.

Modernization, on the other hand, considers the target architecture of the entire IT landscape and then places the transformation efforts for any mainframe code within that target architecture.

Viewing Mainframe Applications through a Modern Lens

Most modern IT architectures today are cloud native – leveraging cloud best practices to deliver dynamic software at scale, in hybrid IT environments that combine the best of the cloud and on premises.

To participate in this vision for modern software, mainframe developers must transform core code to be well-documented, modular, and microservices-based – with functionality that is transparent to the broader application development team, now and into the future.

Microservices represent a modern approach to writing efficient, self-contained software. Cloud native architectures depend upon microservices, because they are essential to the scalability benefit of the architecture.

When modernizing mainframe applications by transforming them into microservices, organizations place the mainframe squarely into their cloud native environment, taking advantage of the scalability and maintainability benefits that microservices provide in the distributed computing world.

As a result, development teams must address the core code maintenance problem as well. It's far more straightforward to maintain microservices on the mainframe than their 'spaghetti code' predecessors.

Modernizing mainframe code, therefore, provides benefits beyond the core advantages of cloud native architectures. It enables organizations to get out of the business of maintaining older core code – an activity that perpetuates risk and delivers little business value.

Such modernization also provides human benefits as well, as new generations of developers find modernized applications to be easier to work with than core code and the work more in alignment with their career goals.

Modernization even helps resolve the tribal knowledge conundrum – where the organization depends too heavily on a few developers who know how the older code works.

Modernized applications democratize the development process, reducing the dependence on tribal knowledge and opening mainframe development to a wider range of individuals.

Replacing Fear with Courage

For any organization to take the leap into modernizing their mainframe applications, *fear* must give way to *courage*.

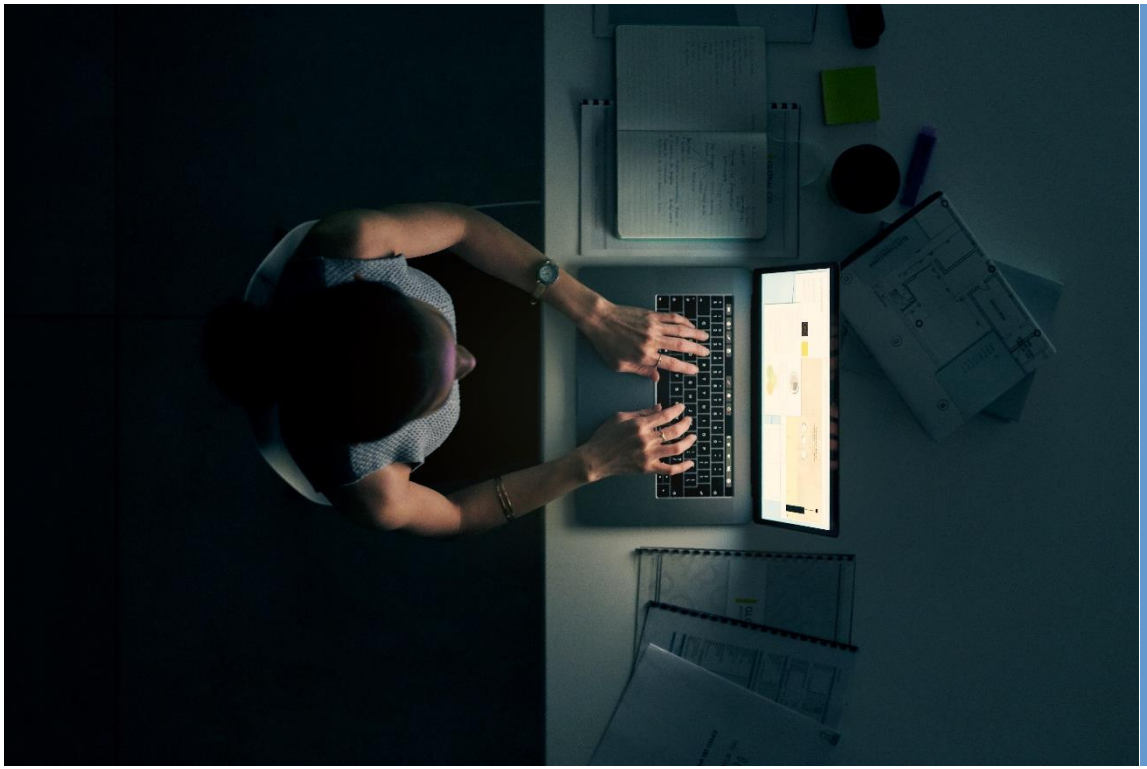
True, it's always possible that the transformative change that modernization represents might lead to issues in the short term. But the way to deal with fragility isn't to leave the applications alone, it is to *make them less fragile*. Such is the promise of modernization.

Code explainability is the essential enabler that empowers developers to reduce the perceived fragility of core mainframe applications.

Fundamentally, such applications are not really that fragile. After all, they've been running for decades in mission-critical environments.

This fragility is more a matter of perception than reality. It is only because people don't know what the applications are doing do they conclude that they may be fragile.

Once code explainability comes into play, however, then developers can identify what issues older code suffers from and how best to resolve those issues – reducing the risk that they will inadvertently break something.



The Secrets behind Mainframe Modernization

Most mainframe apps are written in COBOL, so it's no surprise that there are many COBOL modernization tools on the market.

Typically, such tools convert COBOL to Java (or perhaps another modern language). Simple, right? Push a button and you've just solved all your modernization challenges.

In many cases, COBOL modernization tools take a line-by-line approach, leaving all the spaghetti in place, only now it's Java spaghetti instead of COBOL. The result is no more maintainable than before and doesn't take advantage of any of the benefits of modern architectural approaches.

Not so fast. In many cases, these tools take a line-by-line approach, leaving all the spaghetti in place, only now it's Java spaghetti instead of COBOL. The result is no more maintainable than before and doesn't take advantage of any of the benefits of modern architectural approaches.

In many cases, in fact, the resulting code is unreadable, further limiting the ability for developers to modernize it. Testing such code is also difficult.

Driving Mainframe Modernization with Code Explainability

Instead, it's important to clean up the original applications first. Resolve the interdependencies and obscure logic of the original COBOL before attempting to transform it into a modern language.

To accomplish this admittedly difficult task, it's essential for developers to understand the code they are modernizing. This *explainability* of the code is perhaps the greatest advantage that genAI provides to the mainframe developer.

While today's mainframe modernization tools can help developers understand the structure, organization, and logical flow of core applications, only with the help of genAI can they get a straightforward, natural language explanation of the business functionality of the code.

Explainability is the key to addressing the fear of modernization that has impeded such efforts for so long. Once developers understand the code that genAI can provide, they can proceed with the necessary modernization efforts.

Leveraging Explainability for Better Documentation

The explainability benefit of genAI can also address challenges with the documentation of older code. In many cases, documentation is insufficient or missing – so developers must also create new or improved documentation as they go along.

Creating new documentation is a perfect use of genAI, as it's an important benefit of explainability. GenAI can take existing code and generate passable explanations of how the code is supposed to behave, saving developers time and improving the readability of the code well into the future.

New and improved documentation is essential for aiding developers' ongoing maintenance and modernization tasks. It is also important for operators as well, as these professionals must understand the code that they are running in production.

Modern Quality Assurance on the Mainframe

It's also essential for modernized mainframe applications to comply with today's quality assurance practices. Testers use modern tools for applications in the distributed environment, and they want to use the same or similar tools for the mainframe.

Modern hybrid applications combine the capabilities of multiple platforms, where the mainframe is only part of the story. For example, a mobile banking app might connect to the cloud, which in turn integrates with the mainframe for core transaction support.

Testing such applications is an end-to-end affair. The mainframe must be an integral part of such quality assurance efforts.

The explainability benefit that genAI brings to the mainframe helps with testing as well, as automating tests requires that testing tools have an inherent understanding of the required functionality of the code they are testing.

Without explainability, such automation takes place in the dark – leveraging hand-written requirements and arbitrary test cases in the hopes that the result is quality code.

Explainability moves the testing ball forward, giving testers running automated tests greater visibility into the inner workings of the code under test – including mainframe code.

The Intellyx Take

GenAI is only one part of the mainframe modernization story. In addition to explaining business functionality to developers, it can also help generate readable documentation for both older and modernized code.

In some cases, it can even generate code, although this capability is new to market and may not be adequate for mission-critical applications yet.

The broader story for mainframe modernization is the role the mainframe plays within today's hybrid cloud native architectures.

The mainframe is but one platform among many – with advantages unique to the platform.

No other platform can compare with the mainframe for its high availability, resilience, and transaction processing capabilities. It's no wonder that the mainframe has survived sixty years as the cornerstone of so many enterprises' transaction and data processing efforts.

No platform can survive over several decades, however, without staying current. Business needs evolve as do architecture and software development best practices.

The good news: mainframe-based organizations can have the best of both worlds – the long-term benefits of the mainframe combined with the modern advantages of hybrid, cloud native computing.

This promise of a modern mainframe is not simply for today. It is a promise for the future as well, as technologies continue to evolve. Just as the mainframe played well with Linux and Java, it now plays well with cloud-based systems that leverage GPUs for optimizing AI workloads.

And tomorrow? The mainframe is likely to play well with quantum computers and other future technologies we have yet to imagine. As a result, mainframe modernization will continue to be important now and well into the future.

About the Author

Jason Bloomberg is the founder and managing director of enterprise IT industry analysis firm Intellyx. He 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 #14 on the *Top 50 Global Thought Leaders on Cloud Computing 2024* and #10 on the *Top 50 Global Thought Leaders on Mobility 2024*, both by Thinkers 360. He is a leading social amplifier in Onalytica's *Who's Who in Cloud?* for 2022 and a *Top 50 Agile Leaders of 2022* by Team leadersHum.

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

About Intellyx

Intellyx is the first and only industry analysis, advisory, and training firm focused on customer-driven, technology-empowered digital transformation for the enterprise. Covering every angle of enterprise IT from mainframes to cloud, process automation to artificial intelligence, our broad focus across technologies allows business executives and IT professionals to connect the dots on disruptive trends. Read and learn more at *https://intellyx.com*.

About BMC

BMC empowers 86% of the Forbes Global 50 to accelerate business value faster than humanly possible. Our industry-leading portfolio unlocks human and machine potential to drive business growth, innovation, and sustainable success. BMC does this in a simple and optimized way by connecting people, systems, and data that power the world's largest organizations so they can seize a competitive advantage.