Don't kick the COBOL can down a crumbling road

Kicking the legacy modernization can down the road costs significantly more every year. This white paper looks at the challenges of maintaining legacy applications, the modernization options available and our refactoring approach.

Recently, the US Government Accountability Office (GAO) released an eye-opening report which found that roughly 80% of the government's $90 billion IT budget is spent on maintaining aging technology , and the increasing cost is shortchanging modernization. The report illustrates how kicking the legacy modernization can down the road costs significantly more every year which has resulted in a significant decline in development, modernization, and enhancement activities between 2010 and 2019. While many of the findings in the report are astonishing, such as the revelation that the Defense Department's Strategic Automated Command and Control System, which is used to send and receive emergency action messages to U.S. nuclear forces still uses 8-inch floppy disks for storage, the vast majority of the legacy upkeep falls into the COBOL realm. Among available options, we assert that automated refactoring is the fastest, most cost-effective, and safest way to alleviate reliance on legacy infrastructure, databases, and the application code that supports them.

Vintage is in, but restoration has its limits

The public version of the GAO report identified ten federal agencies' systems most in need of modernization based on attributes such as age, criticality, and risk. GAO then analyzed agencies' modernization plans for said systems against key IT modernization best practices. While the names of the systems were redacted in the report, the details paint a clear picture: COBOL. The most common thread among them is COBOL, which is cited as a risk due to the dwindling number of people available with the skills needed to support it.

The U.S. Air Force's "System 1" provides configuration control and management to support wartime readiness and operational support of aircraft, among other things. Unfortunately, the costs associated with maintaining the system have been steadily increasing due to poor documentation, aging infrastructure, and the shrinking availability of manpower to maintain the legacy code. As a result, Air Force officials expect annual costs to rise from $21.8 million in 2018 to approximately $35 million in 2020.

The Department of Education's "System 2" processes and stores student information and aids in the processing of federal student loan applications. First implemented in 1973, the

WHITE PAPER

system runs approximately one million lines of COBOL on an IBM mainframe. Officials stated that the agency would like to modernize the system to eliminate their reliance on COBOL, simplify user interactions, improve integration with other applications, respond to changing business requirements more quickly, and decrease development and operational costs.

The Department of the Treasury’s Internal Revenue Service’s (IRS) "System 6" houses taxpayer data. Many IRS processes depend on output, directly or indirectly, from this data source. This system is written in Assembler and COBOL, complicating maintainability, and suffering from the risk of staff attrition. Ultimately, it has become increasingly more expensive to operate, and due to talent scarcity, the agency is forced to pay a premium to hire staff or contractors with the knowledge to maintain these systems.

From nightmare to reality

The dawn of 2020 brought with it an unexpected compelling event of enormous proportions-an ensuing global pandemic. Nearly every person on the planet has been affected in some way, from shifting to work from home models overnight, to losing their jobs entirely. Millions have fallen ill, and hundreds of thousands continue to lose the battle against COVID-19. The impact of the pandemic is immeasurable, and many federal, state, and local agencies found themselves in high demand, where the risks their legacy systems posed quickly became a reality.

One of New Jersey's mainframes, which supports its unemployment platform, buckled under the pressure of a 1,600% rise in applications, from 9,500 the week of March 14 to 156,000 the week after. These legacy systems can struggle to adapt to new user cases or unexpected surges in demand because of their undocumented monolithic nature, long complex waterfall cycles, limited extensibility and integration capability, and most notably the diminishing workforce capable of writing the mid-twentieth century code that powers them. In a press conference, Governor Phil Murphy lamented, *“"We have systems that are 40-plus years old. There'll be lots of postmortems and one of them on our list will be how the heck did we get here, when we literally needed [COBOL] programmers."*

The big iron behemoth

Realizing the state of agility required to weather major compelling events is a difficult prospect, especially for agencies tethered down by cumbersome systems and processes with a deep heritage such as mainframes. The impact and breadth of these legacy systems is not to be underestimated. They play an enormous role in governments and business around the globe today, particularly among those with long histories. Many find it astounding that 80% of the world’s corporate data resides in or originates from mainframes running technology that is more than seventy years old. To some, it is difficult to understand why agencies continue to use such old technology for their critical applications, especially in an era of accelerating change.

The reason is quite simple. The legacy systems are stable and robust. They perform satisfactorily and continue to meet the functional requirements around which they were originally built. However, these systems have passed through many hands over many years, often without proper documentation of features or functional relationships. As the technology, infrastructure, and architecture of the agencies and businesses around them changes, the burden of retaining them will continue to grow.

Exorbitant operating costs

Mainframes are expensive. Exactly how expensive depends on several factors including size, complexity, usage, and the latest licensing contracts with major vendors. Based on insights from our customers, a mainframe's annual per MIPS cost falls somewhere between $2,000 and $5,000 USD. A comparable workload can run in a FedRAMP compliant environment such as the AWS GovCloud (US) for just 8-12% of that, which is a material difference.

The "2020 Mainframe Modernization Business Barometer Report" found that on average, organizations could save $31 million if they modernized the most urgent aspect of their legacy systems, and while individual results can certainly vary, modernization is a proven cost reduction strategy.

Don’t kick the COBOL can down a crumbling road

Diminishing skills & resources

Developers who understand procedural languages such as COBOL are becoming increasingly difficult to find. Simply put, people who know how to work with COBOL and mainframe technology are steadily retiring. To make matters worse, most universities no longer offer mainframe instruction since no one would dream of using procedural languages like COBOL for greenfield development projects anymore. According to a 2018 Forrester Consulting study, enterprises have lost an average 23 per cent of specialized mainframe staff in the last five years, a whopping 63 per cent of these vacancies remain unfilled. As the shortage of experienced programmers grows, the risk of relying on a shrinking talent pool and the cost of these resources will continue to rise.

Sluggish response to competitive pressure

Lack of frameworks, productive and advanced IDEs, debugging tools, and test automation add significant time to development cycles on the mainframe. Organizations relying on maintaining and extending legacy systems using waterfall development methods have a very long time-to-market for new business needs and respond slowly to challenges from customers, constituents, or competitors. In recent years, the successes of digital transformation, underpinned by modern agile practices such as continuous integration and DevOps, have highlighted the troubles associated with sluggish development cycles in legacy environments. According to the 2020 Mainframe Modernization Business Barometer report, 85 per cent of respondents preferred agile development practices over traditional waterfall methods, with 33 per cent stating that modernizing would allow them to be more reactive to market changes.

To make matters worse, the cumbersome nature of these old systems has significant downstream effects. Even when new applications are developed using modern technologies, integrating these with core business functionality running on mainframes is a highly constrained, time intensive, and risky task.

What are your options?

Maintain status quo

The first choice for any scenario is always to do nothing, to ‘let it play out’. The inherent complexity and perceived risk of upending critical mainframe applications can be unnerving for many organizational leaders. Unfortunately, as time passes, dealing with legacy systems will become increasingly difficult – less available expertise, more application development backlog, and more money spent on licensing and rare, expensive talent.

Third party off-the-shelf solutions

This approach focuses on replacing mainframe application functionality with packages and components available from third party vendors. A positive of this approach is a reduced amount of source code maintenance since most vendors shoulder responsibility for fixing production bugs and implementing new functional enhancements.

However, commercial off-the-shelf (COTS) packages offer standard domain business processes that often differ from the homegrown mainframe application they are meant to replace. Reuse of existing business logic is not possible; therefore, some level of business process re-engineering or customization of the third-party solution will be required. Both processes can be time consuming and expensive. In our experience, the older and more critical a legacy system is, the higher the likelihood of significant customization requirements.

Rehosting

Rehosting, sometimes referred to as replatforming, is ideal for organizations that wish to retain legacy code as-is while moving away from difficult to integrate non-relational databases and expensive legacy infrastructure. With rehosting, the application code is shifted into an emulation environment (a proprietary piece of software) where it can run on modern, distributed systems without change. The underlying legacy database is commonly refactored to a relational model during the system migration, opening it up for integration with modern business intelligence tools and systems. The rehosted code interacts with the new database through the emulation software and can reside on modern infrastructure on premises or in the AWS Cloud.

Refactoring

Refactoring is a broad term that is used to describe an array of solutions, all focused on changing the original codebase to meet the organization’s needs. These solutions can be categorized across an automation gradient, with transcoding at one end and manual rewriting at the other.

At one end of the refactoring gradient lies automatic migration solutions that use transcoding, or line-by-line conversion of the source language (COBOL in our example) to the target language (Java for example). This type of conversion is often described as “compile-time conversion” and tends to be inexpensive and highly automated. In a transcoding model, a piece of licensed software is used to recompile online and batch applications from COBOL into Java so they can be deployed to an industry-standard Java application server. The Java these solutions produce is colloquially referred to as “JOBOL” because by their nature, compiler-driven conversions produce a procedural Java, not truly object-oriented code. These solutions cannot accurately adhere to common object-oriented concepts and paradigms such as encapsulation, abstraction, modularization, and loose coupling.

Procedural, line-by-line JOBOL is heavily constrained, offering very few options for performance tuning, optimization, and extension because it is virtually un-maintainable “spaghetti code”. As a result, many compile-time transcoding providers suggest that developers maintain and extend the original source COBOL, compile into Java using their transcoder, and deploy into modern operating environments, rather than attempt to maintain the JOBOL natively. In some cases, transcoding solution providers offer proprietary software tools to aid developers stuck working with the resulting procedural Java, but these tools are limited in scope and output more JOBOL, reducing the hope of code maintainability further, while locking customers into additional software and licensing. While compile-time transcoding solutions are inexpensive and highly automated, the JOBOL they produce is the least performant and maintainable on the refactoring automation gradient.

Conversely, manual rewrites inhabit the opposite end of the gradient as the least automated, most expensive, and highest-risk solution for modernization through refactoring. Significant time and effort is required to recreate the legacy systems’ wealth of functionality with newly written application code that is stable and error-free- the scope and complexity of this effort is often vastly underestimated. Legacy systems such as mainframes house business-critical applications which have been the subject of extensive change and constant evolution over decades, passing through countless development cycles. Most of these environments are not well documented, and their complexity and interconnectedness is universally underestimated. The blind spots this phenomenon creates is often the harbinger of extreme scope creep, leading to disastrous results. In one example, the California Department of Motor Vehicles embarked on an effort to modernize their core systems by rewriting them in 1991. After six years of effort with nothing to show for it, the project, clocking in at nearly $16 million USD over-budget, was cancelled.

Rewrite projects can take years to complete, and by the time the new applications are ready to deploy, the systems they were designed to replace have changed so significantly, their newly developed replacement is obsolete before it hits production. Even in optimal circumstances, rewrites require extensive code freezes to effectively deploy which are simply unattainable.

On balance, at the center of the refactoring gradient, is tailored, rules-based automated refactoring, the sweet spot for modernizing procedural COBOL to object-oriented languages such as Java.

We take a unique approach to achieving the highest quality, lowest-risk, tailored, rules-based automated refactoring solution possible. COBOL-to-Universal (CTU), a proprietary automated toolset, combined with proven methodologies and refactoring processes, reduce the risk and pain of migrating away from COBOL and its surrounding infrastructure.

Don’t kick the COBOL can down a crumbling road

Our Approach

Our Automated COBOL Refactoring solution, harnessing a powerful combination of proprietary COBOL-to-Universal (CTU) software and an iterative transformation methodology, delivers a modern Java application based on fully maintainable open systems. The CTU refactoring software has processed billions of lines of code for many of the world’s largest enterprises and governments, and is actively and continuously evolving with each modernization project to maximize code quality, reduce risk, and ease the migration away from legacy.

This quality-focused, rules-based automated refactoring approach, reduces cost, allows for deeper integration, and unshackles organizations from outdated languages and expensive infrastructure, while unlocking infinite customization potential to meet any business requirements. Once the application is refactored, developers can change and extend application functionality directly and easily using object-oriented concepts and paradigms.

Legacy modernization is a complex undertaking that involves far more than dropping code into conversion tools and pressing compile. It is important to understand the source environment, determine potential challenges and how to overcome them prior to conversion, iteratively test the results, and tune the tooling to adjust for any necessary changes in the refactoring process. It is the combination of a proven modernization methodology enacted by domain experts harnessing powerful tools that sets us apart. As such, each automated refactoring initiative we deliver follows a proven, multi-phase methodology.

Assess and design

As the primary transaction engines for core business functions, the applications will have inevitably been augmented, tweaked, and extended multiple times by a multitude of developers. The longer the system has been around, the more technical debt accumulated, and the less the organization knows about its inner workings. The trouble is, making changes or decommissioning systems without a full understanding of the impact is incredibly risky, leading mainframes to remain in place far longer than they should.

To make matters more complicated, the way in which companies and governments operate has fundamentally changed. For example, people do not purchase airline tickets the same way they did in 1987. However, they are probably using the same underlying systems to reserve seats from their smart phones that their travel agents and airline reps used.

Ideally, stakeholders should have total visibility into the legacy environment to inform platform decision making, raise awareness around potential transformation challenges, and gain a clear understanding of integrations and ancillary dependencies. With a comprehensive picture of the contents and interrelationships between application components, CIOs, enterprise architects, project managers, and developers will realize a significant reduction in the scope, cost, and risk of migrating away from the legacy environment. We deliver this clarity and omniscience by beginning every modernization effort with a legacy systems assessment, which informs target environment design.

In the assess and design phase, organizations discover artifacts they did not know they had, relationships they did not realize existed, and assets that are no longer in use. It consists of two key activities. First, the automated application assessment, which clearly defines the current state of applications and databases in the legacy environment. To prepare for this phase, Advanced Modernization Platform as a Service (ModPaaS) for AWS is installed in a customer-managed environment through the AWS marketplace. ModPaaS is used to access and house source code and other shared materials for the assessment and subsequent transformation process. eav is the toolset used to process the automated application assessment, in which all application components are cataloged, cross-referenced, and missing components are identified, collected, and added to the inventory in an iterative fashion. At the conclusion of the assess and design phase, the customer may retain access to eav and ModPaaS, where all original source code can be archived for future access and traversal by customer stakeholders.

Next, our team highlights topics that could be troublesome during maintenance or augmentation, which must be addressed prior to the transformation effort. Once these areas of concentration are identified, customer teams and our experts work together to implement appropriate solutions. A few examples of common areas of concentration in COBOL assessments include resolution for non-fully qualified names, scoping of variables, and planning for the handling of pre-processor statements.

The second part of the assess and design process is an operational and infrastructure assessment. This cross-references application assessment findings to ensure a complete understanding of the entire legacy landscape, seeding hardware and other operational requirements for the appropriate target distributed systems architecture. We work with cross-functional teams to understand necessary service levels, performance requirements, and ancillary software products in use, informing target environment design decisions such as the size and composition of the infrastructure hardware, systems software, monitoring apparatus, data storage, etc.

In many cases, organizations have mature, standards-driven and widely used DevOps tool chain pipelines in place. In some cases, the continuous integration (CI) and continuous development (CD) philosophies that drive DevOps are new to the agency or the development team responsible for the applications being modernized. During the operational and infrastructure assessment, we will work with client stakeholders to understand whether the preference is to deliver into a DevOps pipeline or not, and if so, whether assistance is needed in developing it.

If a pipeline exists, we can tailor the delivery process to accommodate its constraints. If not, we will work with the customer to design and create the target platform DevOps tool chain pipeline including the pre-delivery test cases as well as standard reporting, basic approval workflow, and the configuration of typical role types. This process follows five steps:

> **Stage 1: CI/CD framework** - Select and deploy a CI/CD framework based on the tools of the customer's choice (e.g. Je.g. "AWS CodePipeline or alternative third party solutions such as Jenkins)".

> **Stage 2: Source Control Management (SCM**) - Integrate the preferred Source Code Management (SCM), e.g. AWS CodeCommit or alternative third party solutions such as GitHub with the AWS CodePipeline, solution with CI/CD tooling based upon the selected pipeline architecture

> **Stage 3: Build Automation** - Configure and deploy the selected build automation tools (e.g. AWS CodeBuild or alternatives third party solutions such as Maven, Gradle, Cake, etc.) across the deployment pipeline based on clean, compile, test, and deployment location customer preferences, and on the agreed-upon pipeline architecture.

> **Stage 4: Application Hosting Environment** - Configure the application target environment, servers/containers, and appropriate management systems and processes (e.g. Tomcat, Jetty, Docker, Kubernetes).

> **Stage 5: Code Testing Coverage** - Assist in deployment and configuration of code testing frameworks (e.g. JUnit, EasyMock, Selenium, etc.) and code quality tools (e.g. SonarQube, Cobertura, Emma, etc.) that appropriately interface with the chosen CI/CD environment to maximize deployment automation.

We will also work with customer specialists to deploy and configure middleware automation tools (IaC) such as "Ansible, Chef, Puppet, AWS CloudFormation, and others,", and others, while consulting with subject matter experts from the legacy and target environments along the way.

At the conclusion of the assess and design phase, a complete catalog of the legacy environment is delivered, including areas that require special consideration by the teams, which are isolated and summarized so that they can be addressed when the transformation processes begins. Details regarding use of third-party utilities and products in both batch and online applications are organized and presented alongside findings and decisions pertaining to target operational and infrastructure setup. All of this information is harnessed to coordinate a refined project plan, which kicks off the transformation process.

Transform

The transform phase begins after the assess and design process is complete and a strategy for moving ahead is established. During this phase, we perform pre-delivery functional testing and delivers assets into the target DevOps pipeline based on pre-defined work packets. While customer teams build, test, and deploy these work packets in the target environment, we refactor the next work packet, iterating until the entire in-scope application estate has been transformed.

Transformation focuses on two components, the data and the applications. Automated data migration is performed alongside the automated application refactoring activities. In addition, the target environment and relevant operational infrastructure is built out, alongside any necessary operational data migration activities. Throughout the transformation, precious business logic from the legacy system is preserved, enabling deeper integration, cloud migration, and customization to meet business requirements.

Automated data migration

Data migration is an important component of every legacy modernization project. Each environment is handled differently, from pre-relational to relational migrations, to targeting a variety of database types, and everything in between. Although the tasks associated with this phase vary widely across automated COBOL refactoring projects, steps include the de-construction of the legacy database definitions into metadata artifacts and entry of additional rules based upon customer workbook entries which include column and table naming date type formats, overrides for redefines and group level clauses, and preferences for each statement and clause in the resulting DDL. The generation of brand new DDL defines a complete relational database providing the same data access as the legacy database.

Automated application refactoring

Our automated refactoring philosophy is centered on producing functionally equivalent, human-maintainable, quality-driven code positioned on AWS cloud ready target frameworks and platforms. COBOL-to-Universal (CTU), our proprietary toolset, powers the automated refactoring process.

Translating a procedural language such as COBOL, to an object-oriented language such as Java presents a myriad of challenges. Thus, CTU was designed around these absolutes:

> The refactored application must behave exactly the same as the original application and produce the same results

> The refactored application must be human-maintainable and follow object-oriented concepts and paradigms

> The refactored application must perform as well or better than the original application in the target modern environment

> The resulting applications should be cloud-ready and delivered through an automated code pipeline using a standard DevOps toolchain and best practices

CTU deconstructs the legacy codebase to isolate source code and classify down to the field level. It then refactors the assets using customized rules based on customer requirements and standards. This refactoring process includes flow normalization, code restructuring, data layer extraction, data remodeling, and packaging for reconstruction. Upon reconstruction, new object-oriented code structures are generated and deployed into the target environment and enabled by CTU’s native Java framework. To find out more about the refactoring process, take a look at our Automated Application Refactoring paper.

Operational and infrastructure implementation

During the transform phase, all operational and infrastructure components required by the modernized applications are provisioned, configured, implemented, and integrated. This includes the underlying hardware platform, and all required third-party software. QA/Test environments are built out first, and applications are tested to ensure complete functionality before deploying to the final target environment. If necessary, specific operational data migration-related tasks are completed during this phase, including the migration of required batch job schedules, security profiles, and any archived data.

Test, deploy, and support

The test and deploy phase begins with the receipt of work packets through the automated DevOps toolchain as they are delivered from iterative modernization activities in the transform phase. If issues arise, the work packet is passed back to us where we review the code, adjust our tooling, reprocess the work packet, and deliver back into the DevOps pipeline to resume testing and deployment. This process repeats until the entire modernized estate has been deployed.

Testing is a critical phase of any modernization project and typically accounts for over 50 per cent of the entire modernization effort. Although key business logic is retained through automated refactoring, the underlying application is adapted and refactored to operate within the new target operating environment. Thus, a detailed validation of the migration is extremely important, and the best means of achieving that validation is with thorough testing. Validation testing typically includes stages such as functional (regression), non-functional (performance), integration, high availability, disaster recovery, security, and user acceptance testing. A major misconception is that testing starts only once refactored applications become available to validate. Testing begins prior to even starting down the modernization path, by ensuring that appropriate test plans and test-related artifacts (test cases, test data, test results, etc.) are available to conduct the necessary testing and validation of the migrated applications. To assist with this, we offer a test strategy workshop (typically on-site) during the assess and design phase. Test-related areas of concentration are discussed, and solutions are addressed. A key deliverable of the test strategy workshop is a test strategy report providing an overview of the testing approach with detailed responsibilities, milestones, and ownership of each of the test-related service activities. Testing, deployment, and support activities can vary widely. As such, the processes described in the following paragraphs are intended to be a baseline.

Testing

Our testing service activities include pre-delivery testing against a mutually defined subset of the refactored COBOL application using a test plan with documented test scenarios provided by the customer. A customer-led baseline for testing service activity defines and executes tests on the existing system to capture and record the expected results. We then run the tests against the same data on the refactored system, identify, investigate, and fix discrepancies in the expected behavior of the modernized application by modifying the automated tooling rules to ensure the refactoring process is fully automated. Pre-delivery testing consists of select test cases from the available functional tests, which represent parts of the application. We also provide application discrepancy correction throughout the different customer-led testing stages, where any problems found related to the application and data migration are fixed.

Testing requires the greatest amount of customer involvement of any phase of the project. Thus, the impact of customer resources during testing play a major role in the overall success of the project, both in completeness of the testing, and the duration of the project itself. In addition to offering the test strategy workshop, we can also provide support in the form of a test lead, test manager, and additional test resources to augment a customer test team.

The automated refactoring process does not require code freezes, which is a major risk reducer for customers, however, it is necessary to bring the new codebase up-to-date. Therefore once pre-delivery testing is complete and any discrepancies in application behavior are resolved, we perform a code refresh to ensure that any changes that took place in the legacy application environment during the conversion process are accounted for, refactored into the target language and environment, and tested.

Deployment and support

We work closely with customer teams to ensure a smooth and error-free transition into production. Part of this transition includes the cooperative construction and testing of a go-live production cutover plan to reduce the potential risks associated with application deployments. The go-live plan typically includes multiple dry run cutovers and fall back procedures to validate and fine tune the cutover process. Information is captured related to the time, to implement the production cutover (and the time to delay specific processing while the final transition is taking place), ensuring that all data is migrated and ready for production in the time allotted. The plan also assesses overall go-live readiness and validates the documented process. We provide on-demand assistance during the warranty period following the deployment, as well as post-migration support of the modernized application.

Conclusion

Companies and governments are struggling with aging applications and the rusty infrastructure they call home. CIOs understand the inevitability of migration to modern languages and platforms, but these systems and the resources supporting them are complex and highly intertwined with core business operations and processes. As a result, IT leadership often chooses to “kick the can” of modernization down the road, assuming that the risks of doing anything are greater than the risks of fiddling with these integral systems at all.

The reality however, is that the risk of doing nothing far outweighs the risk associated with optimizing or modernizing the existing environment through informed decision-making. These systems are ticking time-bombs, poised to explode at any given moment with unforeseen impact due to lack of understanding, visibility, and a skilled resource pool to clean up the mess.

Through automated refactoring, organizations are able to retain the business rules and look and feel of a legacy system while extricating themselves from procedural code very few people understand, and infrastructure that costs more every year to keep in operation. However, not all modernizations are created equal. When choosing a modernization provider, it is important that the tools and services they have at their disposal meet the needs of the organization and supporting resources in the future.

With more than 35 years of experience and over 500 successful migration projects, we are uniquely qualified to deliver turnkey COBOL modernizations using our proven process and software, ensuring a smooth transition to the target state.

More information

w modernsystems.oneadvanced.com **e** hello@oneadvanced.com

UK +44 0333 230 1884

Ditton Park, Riding Court Road Datchet,
Slough, Berkshire, SL3 9LL

US +1 855-905-4040

3200 Windy Hill Road, Suite 230 West,
Atlanta, GA 30339