Refactoring the Approach to Space Situational Awareness (SSA) Legacy Application Modernization

Arne Gerhardt
Deloitte Consulting LLP
Scott Radeztsky
Deloitte Consulting LLP

ABSTRACT

With so many technological advances and participants in space technology, maintaining SSA is paramount to the National Defense Strategy. Currently, some organizations are running programs well beyond their expected end of life, either because previous modernization efforts have failed or the criticality these systems provide cannot afford downtime to be refreshed. Our work in this paper aims to investigate best practices to modernize legacy systems and codebases in a way that minimizes cost and time to modernize while preserving essential legacy SSA calculations and mission capabilities. We present best practices to leverage automated tool suites such as innoWake™ in secure environments, showing how teams can refactor legacy code into modern languages, deploy on modern infrastructure, and test to prove that the modern code and converted data behave exactly the same as the legacy system. We also present use cases with the innoWake™ suite to mine legacy source code for logic beneficial for SSA developers and operators, for instance, to uncover obscured relationships between code and data visually or to identify the software or performance bottlenecks that hinder the application's ability to scale substantially (e.g., increase the number of objects tracked, or the number of agency applications involved / informed). For the special case of systems leveraging FORTRAN and Assembler (common across Space systems and SSA in particular), we highlight how to identify key outdated constructs and deprecated language features that impede maintainability and modernization. This paper shows it is possible to take a new path to transforming complex legacy systems that do not require a multi-year effort to completely overhaul. The authors hope the results and use cases presented compel our readers to take a serious look at this proven, selective-modernization approach, which allows agencies to continue to use what's already working while producing an in-parallel modernization and exact copy of the system with modern language, security, and infrastructure benefits.

1. INTRODUCTION

One of the goals of Deloitte's research and engineering teams is to create a fully automated capability that converts SSA system legacy code (e.g., FORTRAN, Natural, Assembler, COBOL) and corresponding legacy data and screens into object-oriented applications (e.g., on Java and C#/.NET) that can run on modern infrastructure, use modern relational databases, and smoothly and securely integrate with cloud-based distributed platforms. These types of solutions produce a modernized system that functions identically to the legacy system but at a fraction of the cost and timeline of rebuilding or rewriting legacy systems from scratch. Such an approach also minimizes the impacts on no-fail missions, operators, interface partners, and Information Technology (IT) operations teams, since they produce no impact on or require no downtime for the running system. Producing modern Java or .NET (C#) code also results in a shorter learning curve when onboarding new engineers to a testing, development, or management team because it is similar in flow/structure as well as naming conventions of the legacy system.

Given that so many mission-critical systems deployed are well beyond their recommended end-of-life date, finding a way to migrate code accurately and efficiently from ancient programming languages running on unsupported platforms to newer, better-supported languages that run on modern, scalable platforms is vital for ensuring the long-term success of SSA missions. We show how modernizing opens a path to solve typical legacy system shortcomings and enables needed mission capabilities such as expandability (add or track more objects), maintainability (simpler to replicate data or deploy updates), and scalability (both widening the pool of developers qualified to develop and maintain the system's capabilities, while also alleviating many of the hardware and software limitations associated with deploying on very aged infrastructure). Using world-class, proprietary technology unique to Deloitte, this paper presents results for an application modernization framework that can migrate a wide variety of antiquated systems into modern solutions within a minimal time frame and with zero impact on legacy system uptime.

2. APPROACH

This unique style of legacy transformation leverages our structured, methodical approach to gauge and attack gaps in the legacy technology baseline in a way that avoids the risks inherent in multi-year modernization projects (e.g., delayed caused by unknown requirements, scope creep, lost or hidden relationships within or between code and data, inconsistent integration, and testing). The first step of this approach includes converting databases, code, and screens to modern languages/technologies. Once converted, the resident capabilities of popular and well-known modern technologies can be used, greatly simplifying the follow-on steps for additional debugging, use of DevSecOps, or improving mission capabilities like system expansion, advanced analytics, and scalability to accept more inputs/data sources. Additionally, such an approach enables parallel system functionality across legacy and modern systems throughout the entire project—clients may keep adding features or perform operations on the legacy system up to the cut-over date—as well as supporting modular deployment and testing as part of the cut-over and ATO (Authority to Operate) acceptance. Extensive testing practices validates data inputs/outputs of modernized systems match the legacy behaviors, and our methodology inherently prevents data loss or code-data relationship loss during the modernization process. Other benefits include removing limits to a system's performance (for instance, the ancient TCP/IP thin ethernet), maintaining the granular security across users, code, and data, and all the go-forward benefits: a modernized system can take advantage of modern infrastructure and tools to meet and exceed the speed and capacity needed to handle current data loads that are critical to integrated SSA capabilities.

3. METHODOLOGIES USED

During our research and engineering to develop this process, Deloitte employed an approach that leverages innoWake™ and TruNorth™, a suite of Deloitte proprietary tools with capabilities that assist in legacy application modernization within secure environments. In this approach, we use the AI capabilities of TruNorth™ to consolidate and create deeper insights across the set of code, data, mission, and infrastructure assets, and use innoWake™ to provide features such as legacy discovery and automatically refactoring code from older systems while maintaining functionality and compatibility. The process also migrates and modernizes the system's legacy data structures into today's 3rd-normal / SQL-based formats, completely in sync with the newly refactored code's needs. While innoWake™ uses a unique, automated, risk-reduced, and proven methodology for converting mainframe applications to languages familiar to contemporary software engineering teams, such as Java or .NET, it also has capabilities that simplify and automate other time-consuming, complex, or high-risk processes such as mining source code for relationships across a legacy system's codebase and automated database migration with an unmatchable level of efficiency and accuracy.

To expedite the modernization process and relieve workload stresses on system engineering teams, innoWake™ includes an automated migration process that reads source code and data and then transforms it automatically and in a repeatable way using a predefined schema. The key to this process is that it is consistent and produces the same result each time it runs. Any identified errors are addressed centrally in the transformation engine, allowing the fix-and-refactor process to continue until all code is addressed. This feature also allows for the parallel operation of both the legacy and refactored systems during implementation without a code freeze, as modifications to the legacy system during the engagement can also be automatically migrated to the new environment through the same process. In addition, handling code refactoring in this way also brings the benefit of reducing the risk associated with manual migration methods, which are prone to introduce human error in the code migration process. Instead, the "stored intelligence and engineering" process eliminates common sources of data loss such as the misconversion between character sets across new and legacy systems entirely.

To effectively consider this type of a system modernization effort successful the result must be one-to-one with the original system, or in other words: the modernized system must be functionally equivalent to the original and not break compatibility within the system or integration with other systems. For a modernized application to be a successful one-to-one recreation of the original system, the structure (e.g., source code and data and application screen modules) of the legacy application platform is converted to a new platform with the fewest possible changes to achieve an accurate conversion of the legacy system functionality. Such an approach also minimizes the impact on system operators as they transition into managing the new solution, reduces the risk of missing existing functionality, and accelerates testing and validation as the resulting system is identical in function to the familiar legacy system. When done correctly, a system modernization endeavor using this approach should be 'invisible' to all other systems that rely

on data from the modernized application, minimizing the impact on other operations. This process developed by Deloitte is unique and uses patented technology.

It is also important to recognize that a migration project to transition a legacy system using this methodology results in an experience quite different from what might be encountered in a traditional system development lifecycle. Time emphasis is shifted from a sequential requirements and design processes to one with a very clear and agile and model-driven conversion and comparison process. This translates into project teams that build unique skills and joint understanding of the system as they employ techniques and tools unique to a refactoring effort. These differences are not a potential source of friction for legacy developers and operators but rather an opportunity that builds side-by-side knowledge transfer as legacy and modern teams build shared system knowledge as they work on refactoring and testing like-for-like behaviors. Project leadership should look to build these complementary teams during onboarding processes and despite the departure from a traditional from-scratch development lifecycle, help teams embrace these additional benefits on top of the other project structures that will be familiar to project members, such as the Agile methodology. Though much shorter (often 2-5x shorter) than traditional rewrite projects, the end result is a team deeply versed in how the system runs and operates.

3.1 FIVE APPROACHES TO APPLICATION MODERNIZATION

Expecting each system modernization effort to look identical would be an unrealistic expectation. As a result, Deloitte has identified five distinct approaches an organization can take when modernizing legacy systems to yield the best results. These approaches involve carrying out different aspects of system investigation and modernization, and a modernization effort may even combine approaches depending on the project's needs. As seen below, these five approaches include Discovery, Mining, Transformation, Legacy DevOps, and Modernization: a breadth of solutions well suited for any business case. Below are descriptions of each approach, along with what actions each contain and some use cases showcasing situations for which they prove most useful.

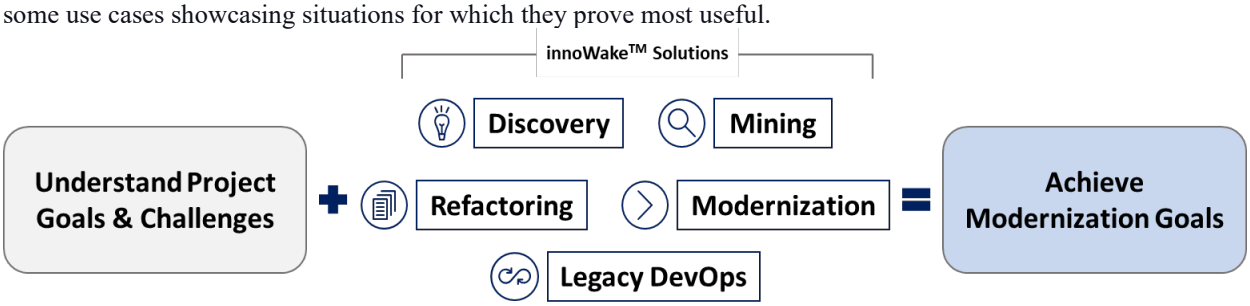

Fig. 1. Diagram depicting the five modernization approaches to help reach modernization goals

Choosing the Discovery approach to modernization is a good first step for all approaches, but especially a good choice when mission leadership must quickly develop a high-level view of an organization's legacy or legacy-modern technology landscape – a vital first step in determining how to best achieve modernization goals, but especially important for legacy codebases where much of that knowledge is "institutional," or embedded in the code itself, or even worse in the memory of a few essential operators (or worse still, a few folks who have already retired). This tool-based analysis (using supported products certified and securely used in many state and federal agency data centers and tightly restricted facilities) can be automated using tools such as innoWake™ Discovery, which produces a detailed set of tabular, visual, and searchable insights about the existing codebase and data on which a mission relies. Additionally, Discovery programs provide legacy system developers invaluable code diagnostics information (e.g., cyclomatic complexity) and identify system integration points with third-party tools and external services that are often hidden or forgotten by being hard-wired or referenced in the code.

For situations requiring a deeper investigative look at a legacy system's underlying architecture than Discovery offers, a Mining approach can augment Discovery to produce the desired results. Mining a legacy codebase involves parsing and indexing each line of code to identify the more profound, non-static relationships within a line of code or between lines of code; for instance, how they connect with the files, function calls, data, and other insightful information that documents how the legacy system behaves, and what architectural patterns or business rules are currently in place. Despite the more intensive approach, the Mining process can be automated using technology such as innoWake™, significantly reducing the time and labor involved to capture these flows across the entire locus identified for mining. Code Mining also has the added benefit of helping to identify redundant or unused code, which can be discarded and

thus reduces development and testing time to modernize any new application. Mining provides deep technical and functional information vital to identifying the call chains, data flows, and business functions of legacy code and as input to determine a strategic modernization roadmap.

As mentioned, both the Discovery and Mining output can be pulled into TruNorth™ where AI is used to pre-processes the data ingested to jumpstart portfolio creation and Deloitte's AI-fueled algorithms continuously identify optimization opportunities, including intelligent suggestions for component groupings that could be addressed together (modernization guided by mission-led "wave planning"). The suite also contains an AI-powered chatbot that helps teams interact with their portfolio using normal business or mission language, and to create trackable modernization scenarios at the scale of interest (e.g., for individual applications or multiple data centers).

When migrating legacy code into modern languages, the Transformation approach is vital to translate code seamlessly and efficiently from legacy languages such as COBOL, Natural, PL/1, FORTRAN and Assembler into modern languages such as Java and .NET (C#), running on modern platforms and servers. Using tools such as innoWake™ allows code transformation to happen automatically, and existing code, data, and UI assets port into a new system that is more easily maintained than the legacy implementation. Transforming and refactoring code with this approach has the added benefit of moving legacy systems away from languages that have dwindling industry knowledge or support, but in a way that preserves important SSA legacy calculations and mission capabilities while simultaneously addressing key outdated constructs and deprecated language features that impede maintainability and modernization.

A fourth approach can be taken utilizing Legacy DevOps for infrastructure, where full code transformation is unnecessary or impossible. This surgical approach involves helping to maintain legacy infrastructure more efficiently by leveraging modern tools such as contemporary Individual Development Environments (IDEs) and designing automated workflows that enable an organization to minimize friction and improve the efficiency of the maintenance process within or across languages and tools. This approach uses a continuous integration strategy which aids organizations in monitoring progress and continuously adding new optimizations to stay agile and consistently review and revise DevOps procedures as necessary. Additionally, taking a Legacy DevOps approach can help make legacy systems more familiar and accessible to developers more accustomed to modern tools and languages, as it helps to put in place a workflow that is easy to understand and familiar, potentially reducing onboarding times for new members.

The final approach to upgrading legacy systems is the Full Modernization approach, which involves completely rebuilding a legacy system using modern technology to maximize system capability, usability, reliability, and performance. Taking this approach helps organizations to leverage new technologies, such as Cloud Native architecture or enhanced cybersecurity or analytics tools that legacy systems would have no way of implementing. The length and intensity of this step can vary from project to project depending on the scope and requirements of the project owner. Still, they may be vital to ensuring the long-term success of an aging mission-critical system. It should be noted that these approaches are not definitive and may vary in scope and detail depending on the needs of an organization's modernization initiative. By identifying distinct approach methods for system modernization efforts, Deloitte has successfully enabled our consulting teams to focus on exactly what a client's organization needs to meet its goals.

3.2 LEVERAGING CODE MINING TO GENERATE CODEBASE INSIGHT

After Discovery, gathering insight into the structure, architecture, and relationships within a legacy codebase is a crucial next step to planning the most effective course of action for a modernization effort. Though an ideal way of compiling this information would be to consult the original authors or operators of the legacy system, the reality is such individuals are only sometimes available or those that have not yet retired may have only a partial understanding of the minute details of a system. Amplifying these issues is the fact that many legacy program features and operations have updated over years or decades, and documentation of these changes to mission-critical systems often wanders from what is implemented; this possibility poses a significant threat not only to the success of modernization efforts but also to vital aspects of national defense and security. In the past, there were few alternatives to manually reviewing codebases to establish relationships line by line, which was extraordinarily costly and time intensive. This fact alone often made necessary modernizations cost prohibitive. To overcome this hurdle, Deloitte has dedicated considerable engineering effort to build automated code mining capabilities into the innoWake™ suite of modernization tools,

allowing code miners to eliminate manual effort and errors, which produces results and reduces the turnaround time of a modernization endeavor considerably.

The Mining process identifies and documents the relationships between different parts of a system's source code and data. At the highest level, these relationships help developers establish how components within the system interact with each other, which helps identify the best attack vectors for beginning a modernization. Our methodology is capable of mining enormous, expansive codebases (across multiple languages) to generate graph-based visualizations of call chains and control flows that drive insights and enable custom drill-downs into system dependencies and relationships that exist between components. Interactive web applications in the innoWake™ suite provide insight into the legacy application using interactive dashboards, visualizations, collaborative rules, extraction workflow, and source code viewers to aid in modernization efforts.

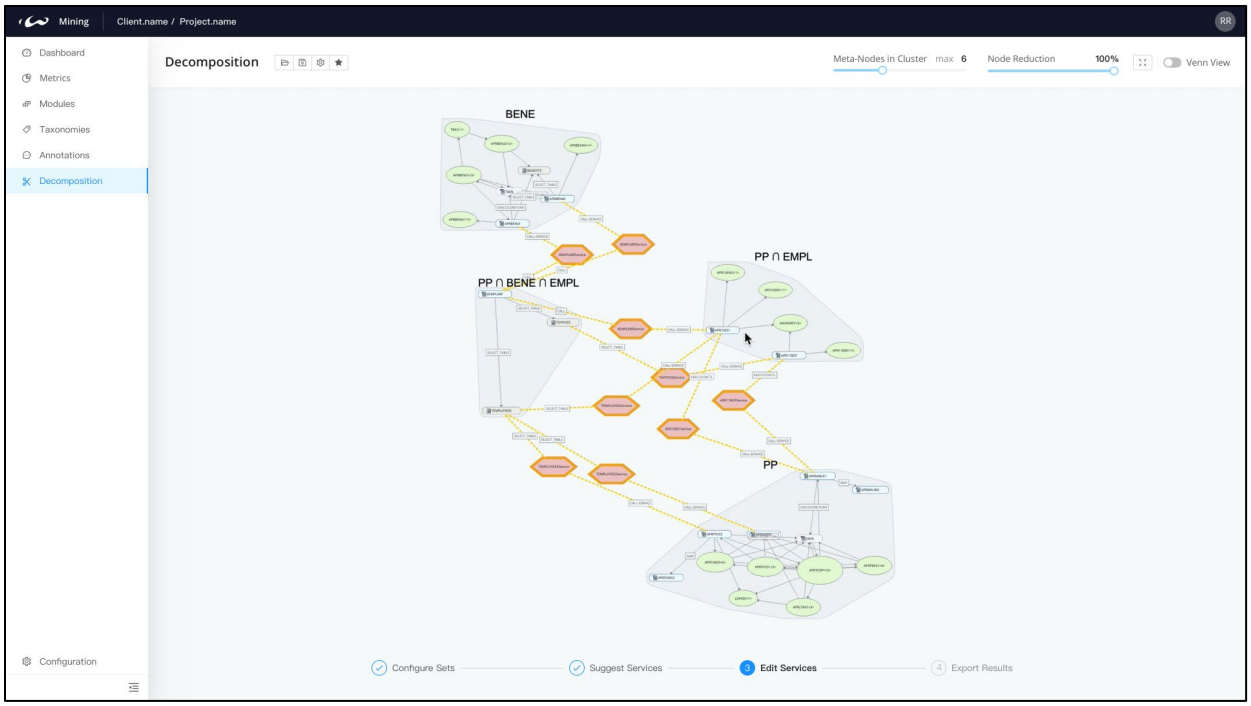

Fig. 2. A screenshot of one view in the innoWake™ interface showing isolated process groups and the graph of their interactions, determined by mining a code repository

As seen in the figure above, visual representations of the relationships existing within a codebase help developers and project management teams understand how technical workflows connect to various system processes. Our process mapping technique uses a taxonomy-based approach to identify processes and maps them to a unique identifier that identifies and tracks these connections across the entire codebase. innoWake™ is uniquely capable of providing quantitative and visual interaction with the legacy codebase to index these relationships and serves as an invaluable aid for retroactively creating documentation of systems with incomplete or non-existent documentation. Though graphically mapping relationships helps communicate a system's software architecture, its importance goes beyond that. Producing these kinds of visual aids enables developers to quickly and effectively drill down or conduct what are traditionally time- and labor-intensive tasks, such as identifying dead or redundant code, evaluating batch job sequencing to seek opportunities for parallelization, and tracking the flow of hard-coded values or deprecated language features to improve system maintainability. Furthermore, establishing these relationships early allows for the analysis of system data access and interfaces to identify areas where duplicate data access queries exist or expose areas that could benefit from becoming a microservice or modern, API-based utility in the architecture.

As a mirror to the code insights, the mining process can also trace data flow patterns across a system or application. Automated mining tools enable project staff to automatically conduct code detection and classification, business rule identification, call chain analysis, data lineage, and pattern detection, which can uncover patterns and similar logic to identify clusters of parallel code. Tool-supported cluster analysis helps quickly identify code patterns shared inside

the clusters and find places to improve redundant code or algorithms. Field tracing and data lineage track variables across the codebase and support analysis of fields, files, or tables, while rule mining provides a collaborative workflow for identifying, validating, and extracting rules via the innoWake™ IDE. Data model analysis helps to visualize, digest, and understand data models and identifies sources, record types, and the use of data within the system. The innoWake™ data dictionary capability can perform automated rules candidate identification and the translation of code variables to simple English.

Combined, these capabilities provide an efficient and streamlined process for mapping and planning the translation of legacy systems into modern software solutions and serve as an incredibly effective tool for building a clear baseline that mission-critical systems can use to stay properly up-to-date and maintained going forward. We have found having a clear understanding of the functional aspects of a legacy system before modernization work begins is crucial to planning effective migrations and minimizing the potential risks of upgrading legacy systems.

3.3 AUTOMATED REFACTORING OF LEGACY LANGUAGES

Legacy systems that have been built and operated across decades rarely have only one legacy language or legacy platform but have grown to leverage multiple versions of languages, utilities, and 3rd party products (e.g., FORTRAN for SSA computational libraries and Assembler on mainframes for speed, COBOL or Natural for Agency operations or business rules, and a mix of VSAM, DB2 or Adabas for the data). For this reason, we find our application modernization approach generally needs to leverage several methods for automated code conversion, all of which we have successfully used with the innoWake™ product suite. A key goal of any modernization effort to migrate legacy codebases to a new, modern language is to refactor existing code to preserve functionality and maintain consistent operations between the legacy system and the resulting modernized system. A better understanding of the current state of conversion tools and desired end states for the target system will rapidly determine which path makes the most sense for modernizing the system. In some cases, it makes sense to use an open-source or third-party code converter to translate code from legacy to a modern language. One pain point identified by Deloitte in many modernization projects is the need for viable Commercial off-the-shelf (COTS) refactoring tools capable of effectively parsing and migrating FORTRAN and Assembler into modern languages. To remedy this issue, Deloitte has developed its own proprietary FORTRAN and Assembler parsers compatible with the innoWake™ suite of modernization tools to identify common inconsistent and troublesome structures and deprecated features for those languages. We find it incredibly valuable to quickly and automatically scan a FORTRAN codebase to identify the hot spots or troublesome routines that will require special attention, such as those that declare Common blocks, Global variables, Implicit typing, or find code that leverages things like Equivalent and GoTo statements, arithmetic IF statements, double precision math, or CASE statements. Finding and plotting out these routines identify hot spots that affect many other routines and flows (upper left, below) versus a one-off that can be fixed without impacting other routines or flows (lower right, below).

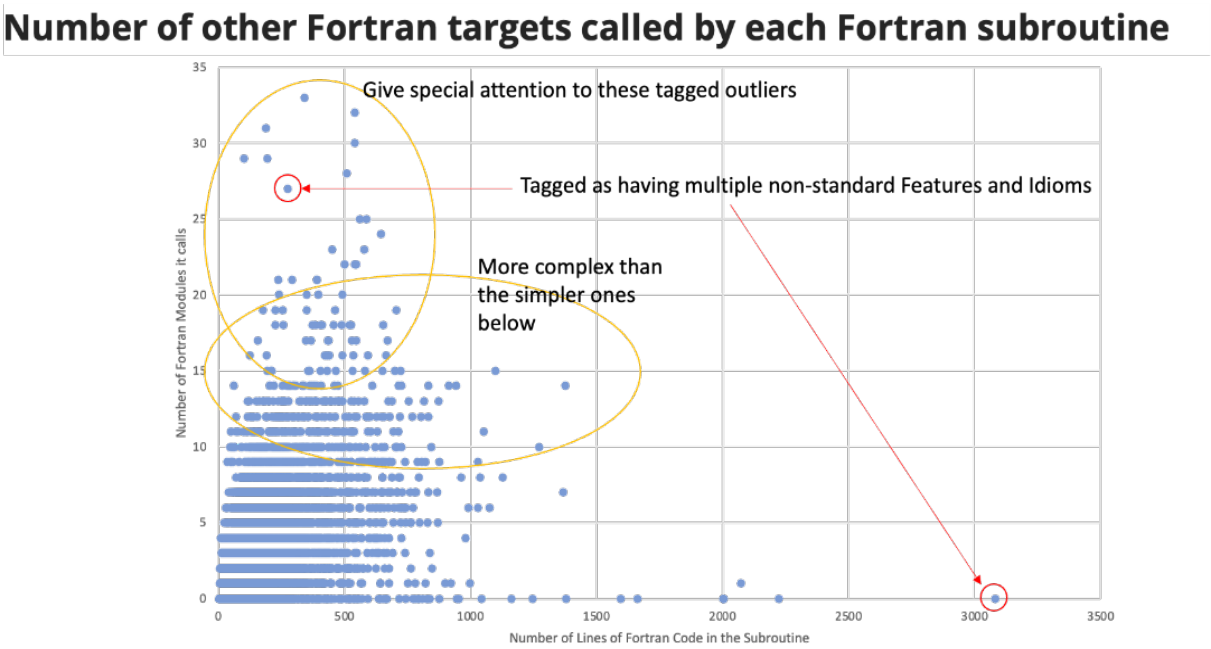

Fig. 3. A scatter plot example using innoWake™ mining output: for each FORTRAN subroutine, plot its lines of code against how many other FORTRAN subroutines it calls. Also highlight if the subroutine was flagged to contain non-standard / non-modern features (i.e., GOTOs, common blocks, etc.)

We also find many SSA systems have a mix of FORTRAN applications that also interact with Assembler routines, for instance, to provide computational speed for complex orbital math not found in FORTRAN libraries, provide a user interface or operational control point, or to communicate with the underlying mainframe system components and external system integrations. We find automated tooling enables a quick understanding of the relationships between these modules, which in turn informs our understanding of the level of complexity or effort to modernize this hybrid set of components: whether it be digging in to fix a widely used fundamental service like the mainframe EZASMI TCP/IP stack, or a simple one-off that provides the menu of operational jobs from which to select for database cleanup. We present a few additional FORTRAN-Assembler example visuals here, highlighting how our tooling lets us quickly identify outliers or prioritize waves of additional analysis or repair on those outliers.

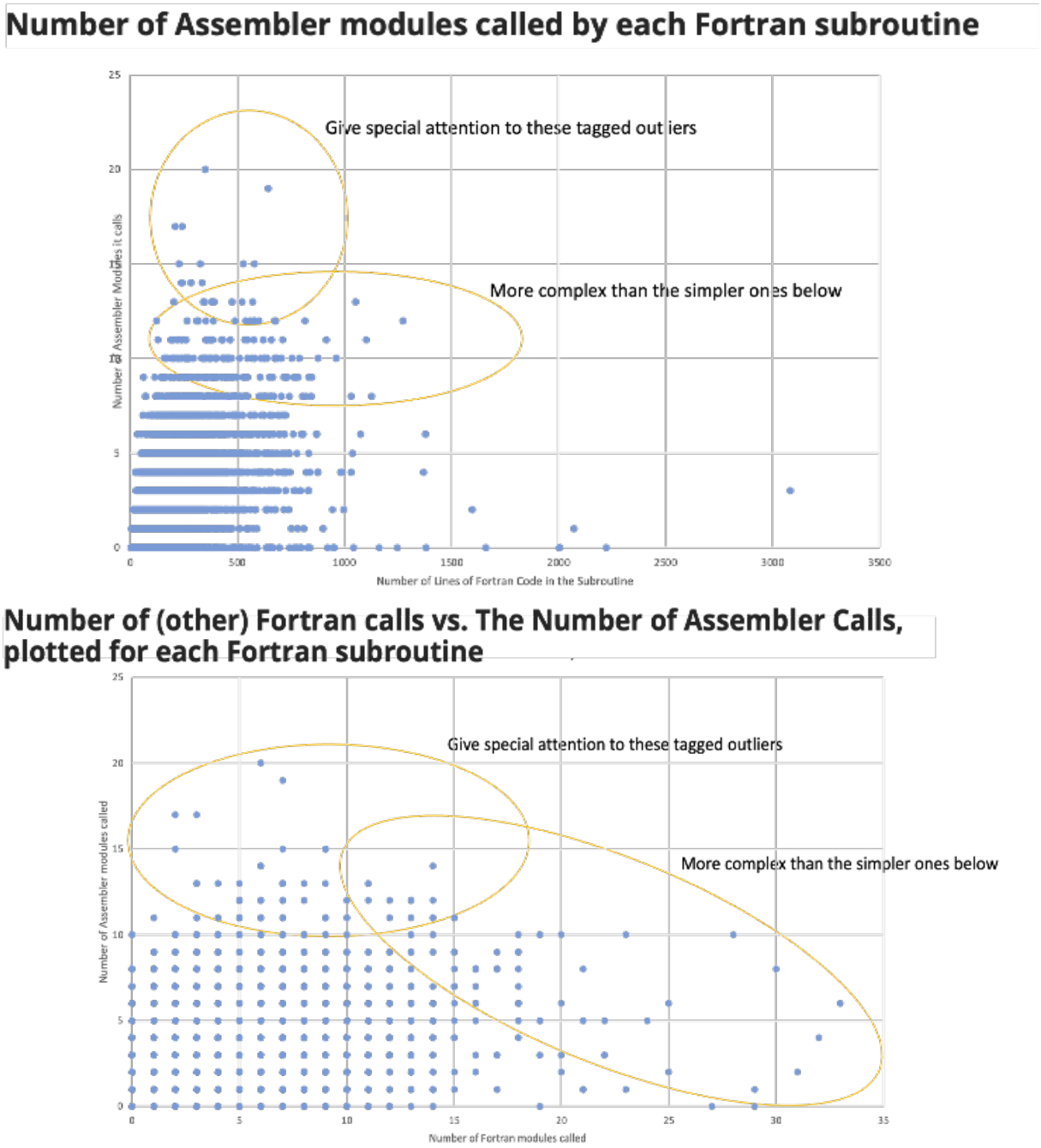

Fig. 4. Two other scatter plot examples using innoWake™ mining output to examine patterns and complexity.
Top: Is it stand-alone, or does it leverage many other calls? (for each FORTRAN subroutine, plot its lines of code against how many other Assembler modules it calls/uses)
Bottom: what is the composition of the calls it leverages? (for each FORTRAN subroutine, plot how many other FORTRAN subroutines it calls against how many Assembler modules it calls/uses)

We find these types of analyses invaluable to allow teams to identify areas for investigation and to prioritize work. Is a module more stand-alone, or is it very busy as seen by how it calls or uses many other routines and modules? As seen above as we overlay other things flagged in discovery and mining, we can quickly identify candidates for simple treatment (transformation via refactoring) versus those that require deeper analysis or surgical attention to determine a path forward (i.e., can we do a simple cleanup before transforming, or is there something structural that warrants a different path for that module or part of the system). This innoWake™ output is also useful input for enterprise

standards and enterprise architecture decisions: e.g., is there a global pattern we wish to impose to clean up all the CASE statements or double precision variables we've flagged across all subroutines, or will we allow each module owner to take an approach they think best? Should we consolidate some of these routines before transforming, or just keep them as the first set of microservice candidates after transforming? Combining mining to aid our transformation process or combining the (typically in legacy systems) multiple languages in that process, helps us (and our clients) make fact-based decisions. Our completely automated tooling allows us to know we haven't missed any instances across the codebase where FORTRAN or Assembler or COBOL or other languages have these issues.

To make the refactoring process as straightforward as possible, automated code transformation of millions of lines (of COBOL, for instance) is very quick within an Eclipse IDE window via our plug-in, with automatic transformation never taking more than 60 minutes to complete when refactoring the entire codebase. The automatic nature of this migration considerably cuts back on development time, as developers have little need to focus on transforming code logic and syntax from one language to another, as programmatic parsing will accurately and predictably migrate programs across languages. Projects that utilize automatic code transformation tools see vast reductions in labor costs, as a process that may take an experienced team weeks – or even months – to complete can then be completed automatically in less than an hour, letting developers focus their skillset on other aspects of the modernization process.

Such a tool also enables large-scale modernization projects without the need for completely rewriting a legacy codebase. Specifically pertaining to the SSA mission, automated refactoring significantly reduces the risk of reworking a legacy system from the ground up, as there is no chance of human error or the need to modify existing code logic or unique orbital calculations. Especially for systems that are involved in well-understood scientific calculations, such as the SPADOC system modernization currently underway for the U.S. Space Force, the algorithms contained within the existing code are time-tested and guaranteed to be robust, accurate, and secure, meaning a rewrite of the codebase would be an inefficient use of time and money and induce undue risk since the current system is already capable of the necessary calculations required for the mission. Instead, refactoring the existing code to a modern language ensures both the legacy system and modernized system behave identically, reducing the need for significant retraining of end-users while simultaneously enabling the integration of contemporary hardware and software solutions. From Deloitte's experience in numerous application and system modernization projects, we have seen that usually only 10-20 percent of a legacy system's codebase needs any serious work or optimization, meaning that leveraging this technology can help organizations significantly speed up project turnaround time while cutting back on the cost associated with the effort considerably.

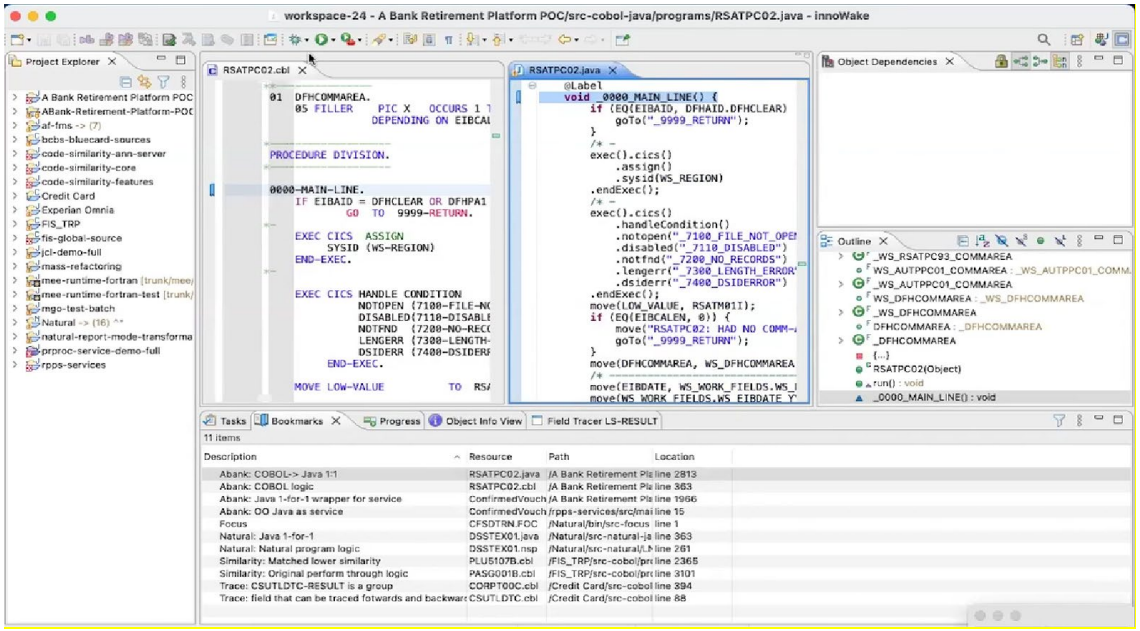

Fig. 5. A screenshot of an Eclipse environment showcasing legacy code (left) and modernized code (right) side by side in an Eclipse IDE environment

Additionally, the automated refactoring process gives developers lots of flexibility in updating and maintaining the legacy system during modernization. Since legacy code can be translated near-instantly, especially for smaller segments of system code, developers accustomed to the legacy system can continue to update the legacy code during the modernization process as the modernization team can refactor those changes and integrate them into the modernized system easily. The frequency of synchronizing legacy code with the in-progress modernized code can be determined by what makes the most sense for the project (i.e., daily, weekly, or biweekly). Still, most crucially, the legacy system can be maintained in vital ways, such as applying security updates and patches while the modernization is still on going. The benefit of this approach is that functionality can be delivered in a more modular and agile manner. For example, given the modernized back end works identically to the legacy system, FORTRAN code can remain intact and merely interface with the new relational database. Critically, organizations do not have to wait for the entire system to be modernized before running the new versions of their code and to start realizing the benefits of modernization.

The refactoring process also takes care to retain much of the original source code's formatting, interpreting the modernized code straightforwardly for engineers familiar with the syntax and structure of the legacy code. This process helps developers familiar with the older system transition smoothly into maintaining the new system, as there may be a learning curve associated with this process, especially for developers more accustomed to older programming languages. Finally, refactoring code into modern languages allows re-engineering code to be simpler since a mainframe-centric design in the code is no longer necessary, enabling rapid development leveraging new technologies, such as replacing legacy green screens with HTML5 webpages that run in a browser.

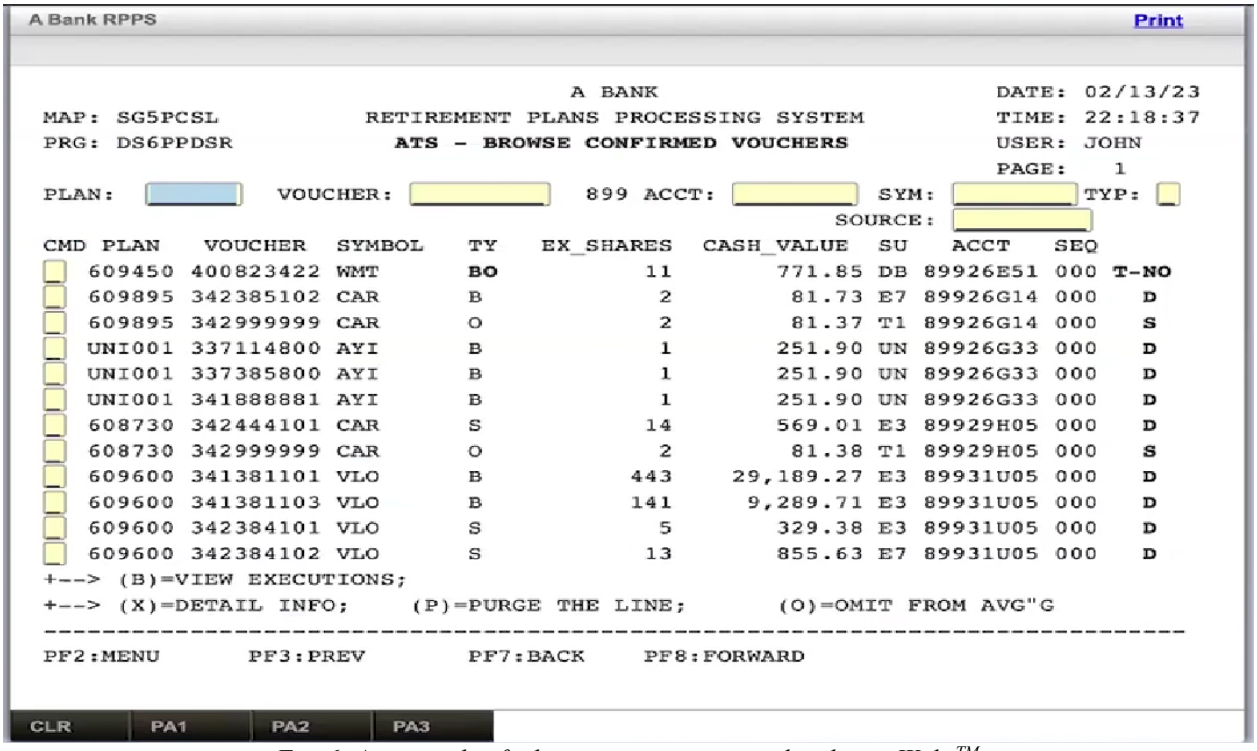

Fig. 6. An example of a legacy screen converted with innoWake™ to run in a modern web browser

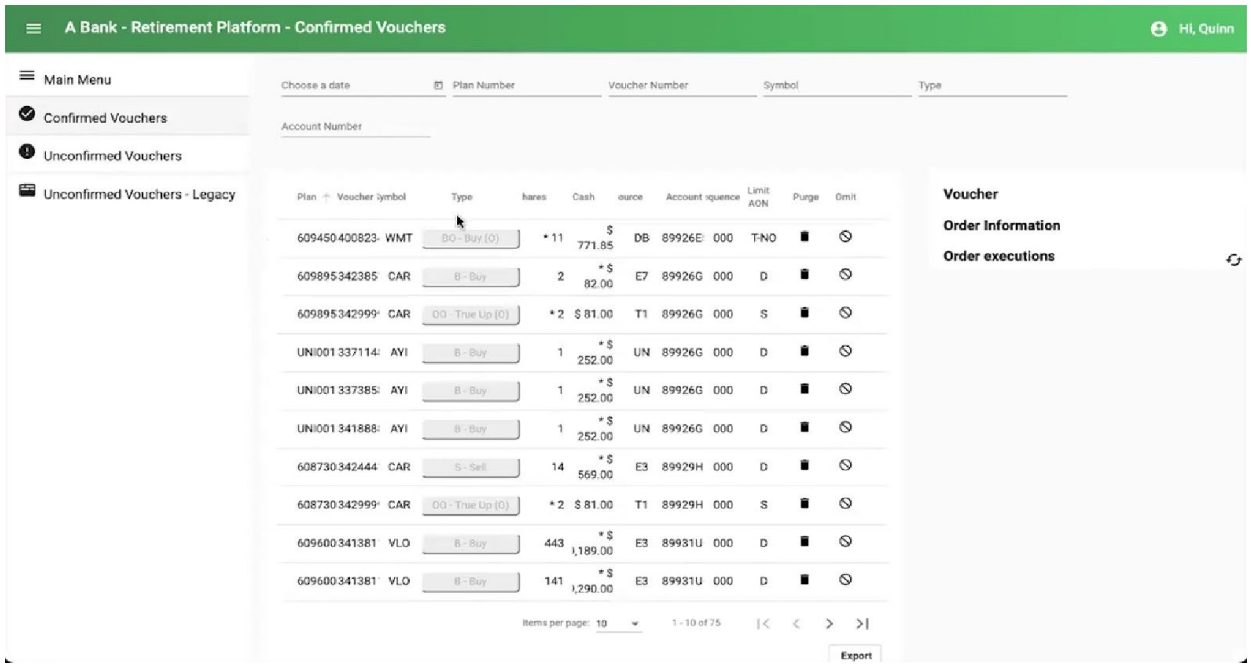

Fig. 7. The same screen from Figure 4 redesigned to utilize an HTML5 interface

The practice of using automatic code refactoring as a method of migrating legacy codebases has successfully been completed by Deloitte on numerous occasions for mission-critical systems involving both government and commercial projects.

3.3.1 OUR APPROACH TO REFACTORING

Unique Feature of our Platform and Approach	**Benefits of Refactoring Approach for Legacy SSA**
100% automated conversion of legacy languages to Java or .Net	Lowest possible risk as no possibility for human error
Patented code conversion	Supported products create verifiable, maintainable converted code that does not increase code size
Eclipse plug-ins for legacy and Java developers	Easy transition of existing maintenance staff to new environment
No code freeze	Ongoing feature development standard maintenance can continue throughout transition to new technology, no need for parallel / double maintenance in the old and new systems
No change in displays transitioning to web browser	End users require practically no training, high UAT acceptance with consistent security and 508 compliance
100% automated data migration	Consistent and verifiable: no data loss, no risk of data corruption
Incremental data migration	Incremental roll out into production
DB2 and legacy data bridge	Allows operating legacy and modernized system in parallel without data synchronization, reducing risk and removing necessity for big bang deployment
Test case recording in 3270 terminals	Ability to record production use of existing system with built-in 3270 emulator, recording all interaction. Conversion of data stream to Junit test cases allows automated regression testing giving high confidence in system correctness, reduces risk
Legacy database conversion to COTS RDBMS	Retains all legacy features like MU/PE, Superdescriptors, sort order, record types, converts to Unicode, ability to access data with COTS tools and languages

Flexible FORTRAN conversion	Allowing FORTRAN conversion to modern languages with innoWake™ tools or 3rd party conversion tools gives great flexibility in choice of target architecture and future, further modernization
Further tool supported modernization	Further modernization, including rules extraction, data abstraction, micro service creation, or cloud-native mapping based on a high degree of automation allows retaining investment in current application while creating a platform for the future

Fig. 8. A breakdown of the benefits of the Transformation approach

3.3.2 OVERCOMING OTHER COMMON ROADBLOCKS

FORTRAN conversion is an important client pain point that Deloitte has encountered while engaging in system modernization efforts. Traditionally, modernizing legacy Natural code and ADABAS databases were another challenge due to the technologies' difference from modern solutions. To surmount these issues, Deloitte's Research and Development team has integrated a proprietary code parser and data migration tool into innoWake™ to specifically facilitate modernizing these challenging implementations.

We have determined that to make this translation most efficient and accurate; three major steps must be completed. The first involves running a diagnostics tool to determine any missing code, which third-party tools or utilities are used by the Natural/ADABAS, FORTRAN, or COBOL, the usage of any Assembler code, whether the application runs online or batch, any interfaces used, code complexity, and code coverage (which is usually around 95%). This helps establish a high-level overview of how the system behaves and serves as an expansion of the code-mining process. Second is achieving 100% code coverage by extending the parser to use new constructs found in the source code frequently or by changing project code for singular instances of syntax violations. Doing so ensures that 100% automated code conversion remains possible. The third is to finally run the automated refactoring tool to convert legacy code to Java and regularly migrate any ongoing changes in the legacy system to the modernized Java code to ensure consistent behavior across the two versions.

By following this process, application modernization teams can produce quality code that is completely sustainable and accessible to a much broader range of engineers, leading to a higher-quality system long term. Just like the FORTRAN parser process, this conversion process mirrors logic structures from the original source code, thereby improving accessibility to developers and maintainers of the legacy system. Additionally, modern tools such as the Eclipse IDE, Subversion, Jira, Jenkins, and Git can all be applied to any legacy code that remains in the system, making maintenance and version control a much simpler process for system managers.

3.3.3 ELIMINATING ERROR IN THE REFACTORING PROCESS

One of the most significant advantages of automated conversion over manual conversion of legacy systems is that it eliminates any risk of data loss or corruption by handling data reliably and consistently and accounting for syntactical differences between the source and target language. One of the most common sources of error that get introduced during manual refactoring of code is the conversion from the EDCDIC character set, which is commonly utilized in mainframe systems, to the ASCII/Unicode character set used in most modern systems due to the different sort order between the two and different binary representation. This might seem benign at the surface, but it can cause issues that cannot be detected at compile-time and require an expert developer to identify and troubleshoot.

Run-time errors are traditionally far more challenging to troubleshoot since these kinds of errors can only be detected when executing a particular segment of code which yields unintended results and maybe later apparent. These contrast with compile-time errors, which a code compiler can identify as the source code compiles into an executable program. The issue with the different sort orders between the character sets is that while the code in both the legacy code and refactored code is equivalent, due to the way characters are organized and identified at the lowest level within the EDCDIC and ASCII/Unicode character sets can yield completely different results.

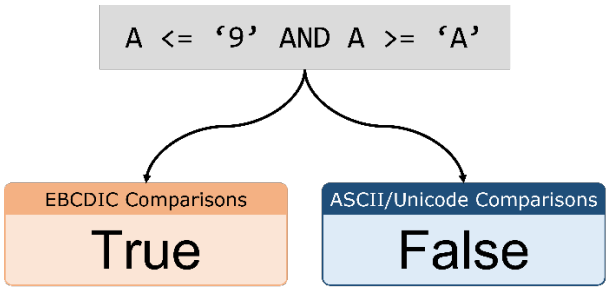

Fig. 9. One example of comparisons that produce different results in EBCDIC and ASCII/Unicode character sets is this test for a variable being alphanumeric

Ensuring that a refactoring solution can identify and remedy these discrepancies is vital for ensuring that a modernized system performs as expected. In cases such as the one described above, innoWake™'s configuration can maintain compatibility through a variety of means, and with respect to EBCDIC to ASCII conversion, a commonly made conversion when migrating mainframe systems, there are numerous built-in safeguards to reduce the need for manual correction of refactored code. When storing EBCDIC code, innoWake™ converts the data to Unicode without changing any characters contained within the data, as all EBCDIC characters are available in Unicode. Whenever a computation using EBCDIC characters is found in the source code, innoWake™ refactoring solutions internally perform all the needed calculations on the EBCDIC data in its own format, despite the data being stored in Unicode, which ensures accurate results. Additionally, any redefined data structures remain identical. If binary data is involved, the Java runtime will execute its computations or data in EBCDIC format internally, creating the same results as the computations on the mainframe did. Other legacy database features, such as MU/PE in ADABAS or multiple record types in VSAM are retained and converted to RDBMS features as necessary. Additionally, views are introduced to represent the data in a relational fashion for use of relational database tools, such as BI, ETF, and so on.

4. FUTURE DEVELOPMENTS

While our process has successfully modernized many legacy systems, we are constantly pushing to find new technologies and processes to integrate into our methodology to minimize required development time and associated costs. One area where we see growth for application modernization efforts leveraging innoWake™ would be through increasing the breadth of languages the platform is natively capable of refactoring. Support for languages has primarily focused on areas and languages that the industry needs the most support for, meaning systems relying on more niche or obscure programming languages might still need to fully take advantage of refactoring and code-mining software. Other areas include languages that are difficult to translate into modern languages due to the unique ways such languages handle information as highlighted in our discussion of FORTRAN, a notoriously difficult language to translate into modern object-oriented languages that only recently saw native innoWake™ support.

Deloitte also sees potential in enhancing the current AI and Machine Learning technologies (AI/ML) we already have in our products, for instance to enhance the efficiency and capabilities of the innoWake™ and TruNorth™ suite of tools. Leveraging these technologies can generate deeper insights and more efficient and refined code during the refactoring process, minimizing the amount of human intervention during the process. Additionally, these tools can enhance the code mining process, helping to generate documentation for legacy and machine-refactored code as if a developer wrote it by taking advantage of Natural Language Processing (NLP) technology.

We are committed to continually refining our processes to help support clients most effectively during their application modernization endeavors. Maturing technologies such as AI/ML and NLP have the potential to enhance the capabilities of software refactoring technologies by shortening necessary turnaround time and reducing the costs associated with engaging a system modernization project.

5. RELEVANCE TO THE SSA MISSION

Across the SSA/SDA portfolio, many missions' critical systems are running on a legacy technology stack. These systems may have existing performance constraints, be unable to easily leverage advancements in technology (i.e., cloud, AI/ML, DevSecOps), rely on dwindling institutional knowledge and select vendors, and require higher costs to

maintain. Many systems in the SSA/SDA portfolio – whether because of the criticality of their function or because prior modernization efforts have failed – are operating well beyond their advised end of life. Our solution and approach allow us to modernize legacy systems while minimizing risk and ensuring mission processing continues without degradation.

Ultimately, Deloitte has successfully leveraged innoWake™'s refactoring capabilities in many mission-centric critical conversions from mainframe technology to modern, distributed or cloud-based architectures.

Currently, Deloitte is employing this methodology in modernizing a legacy system for the U.S. Space Force, on a Space Defense Operations Center (SPADOC) system tracking objects within Earth's orbit, demonstrating the applicability of this process to a broader range of legacy systems still in use that are critical to the SSA/SDA missions. Naturally, because of how mission-critical such a system is, minimal impact on day-to-day operations during and after the modernization process was paramount and a key consideration of the U.S. Air Force (USAF) when selecting a vendor for the project. The modernization process involved migrating an existing ADABAS database into a modern relational database as well as refactoring more than a million lines of code across several languages, including COBOL, FORTRAN, ADAMINT, Natural, and Assembler so that the system could be moved from a mainframe architecture to one that can take advantage of modern hardware. Additionally, the U.S. Air Force required that the solution be capable of being flexible to facilitate the integration of future technologies and could complete the modernization process within a period of 16 months.

Preceding Deloitte, the USAF had numerous other attempts to modernize the SPADOC system, none of which were successful, largely due to the complex process involved with refactoring FORTRAN and Natural code. The USAF needed a contractor to deliver a strategy that automatically refactored such languages, leaving a limited selection of viable vendors. Ultimately, Deloitte's transformation process leveraged automated code mining and refactoring through innoWake™, which was the only method capable of satisfying all the requirements set forward by the USAF to guarantee system performance and data integrity upon completion. Throughout Deloitte's involvement in transforming the SPADOC system for the U.S. Space Force, we have learned a lot about the tools we've created for system modernization efforts and the processes we currently have in place. The innoWake™ suite of tools provides the right foundation for doing code transformation, especially for doing it in an automated and repeatable way. However, we also recognize that these refactoring tools are not magic code converters or replacements for developers who understand the architecture of the legacy system. Having access to the right subject matter experts who can analyze a system and understand its underlying architecture is critical to avoid unexpected setbacks and delays during the process. Establishing and maintaining relationships with developers and users of the legacy system is key to determining where the modernization team's efforts are best spent to expedite the production of a projects minimum viable product and can help to identify regions of depreciated or dead code that developers need not carry over to the modernized system, speeding up development time. The connections we establish with those who can collect a clear understanding of the legacy system are a vital part of our approach, and it plays a large role in what makes our approach to modernization using innoWake™ so effective. A key area where we have identified a need for improvement in our processes is communicating the current testing infrastructure of the legacy system before work begins on producing a modernized system. Understanding what metrics, the current system is measured with can play a significant role in determining how developers could best approach the modernization effort and enable us to provide an apples-to-apples comparison of performance from the legacy system to the modernized one.

6. CONCLUSION

This paper shows it is possible to take a new path to transforming complex legacy systems that do not require a multi-year effort to completely overhaul. The authors hope the results and use cases presented compel our readers to take a serious look at this proven, secure, selective-modernization approach, which allows agencies to continue to use what's already working while producing an in-parallel modernization and exact copy of the system with modern language, data, security, and infrastructure benefits.