Addressing the Elephant in the Room - Modernizing Legacy Software and Architectures

Erik Chaum
Naval Undersea Warfare Center
Newport, RI 02871 USA
USA

erik.chaum@navy.mil

Philip Newcomb
The Software Revolution, Inc.
Kirkland, WA 98034 USA
USA

pnewcomb@tsri.com

ABSTRACT

Modernizing legacy mission critical systems and architectures is very expensive! To try to meet these challenges, system architects and engineers are increasingly using model-based and rule-driven system engineering methods. This paper discusses Architecture Driven Modernization (ADM) standards and methods, a relatively new type of system modeling, specifically defined to maximize computer-aided assessment, transformation and refactoring enabling modernization of valuable legacy software. Real-world ADM use cases show that dramatic modernization cost and time reductions are routinely possible. This paper reviews the state of ADM technology and practice and goes on to suggests that ADM standards, tools and methods could lead to further potentially disruptive business process changes that have significant benefits for military system developers.

1.0 CONTEXT

Each year military organizations spend considerably more resources on maintaining and modernizing legacy software systems and architectures than they do developing new capability! The US Government Accounting Office recently reported that "Of the $79 billion federal agencies budgeted for IT in 2011, $54 billion (about 69 percent) was reported to have been spent on the operations and maintenance of existing legacy IT systems"[1]. Architecture frameworks (AF) enable model-based system engineering (MBSE) design and evaluation methods and are increasingly being used for new system definition and design work. Similarly, they are being used to document "as is" and "to be" mission architectures as a basis for enterprise planning and decision making. Unfortunately, traditional MBSE and AF are of limited utility when there are no detailed system and architecture models or documentation. The enthusiasm for MBSE methods is understandable, but we must broaden our focus to find architecture definition and evaluation methods suited to modernizing aging legacy military systems.

Many factors contribute to the pressure to modernize, including: the need to add or deprecate capabilities; hardware and software obsolescence; corrective actions needed to address system deficiencies and vulnerabilities; changes to achieve enterprise-level interoperability and efficiencies; and evolution to new system designs and technology patterns (e.g., "the cloud"). More generally, there is a steady pressure to modernize, despite business and technical risks, as a way of reducing total ownership costs (TOC) that span hardware, software, manpower and maintenance.

1.1 Technical Debt

Military organizations typically have a huge investment in millions of lines of legacy application and architecture middleware code. The source codes in these software portfolios may; 1) date back decades, 2) use a

mix of modern and archaic software languages, protocols, and architectures, 3) have been written by now retired subject matter experts (SME), 4) have been previously partially refactored to deal with new requirements or obsolescence, 5) have been maintained and developed by multiple organizations, 6) never have been modeled, 7) have been changed many times with limited subsequent regression testing, and 8) not be adequately documented. Each of these, and other conditions, contributes to an associated growing "technical debt" that represents the complexity, scale and cost of maintaining and modernizing a legacy system[2]. For modernization projects there are three basic choices: 1) rewrite from scratch, 2) buy and adapt an off-the-shelf software package or 3) modernize the current application. Industry studies have shown that return on investment, costs, success rates and stakeholder satisfaction are significantly better for tool-driven modernization projects[3]. For modern systems and architectures that are fully modeled, MBSE standards, tools and processes should enable efficient methods for maintaining and modernizing while minimizing technical debt. The "elephant in the room" that has not been effectively addressed is the huge investment that most organizations have in valuable legacy software codes that are not modeled and, in many cases, not adequately documented. For such systems modernization would typically require detailed manual analysis and refactoring, or, detailed manual design modeling. Manually redeveloping and modernizing valuable mission critical software assets using classic development tools is too costly and time consuming. Fortunately, there are new standards and powerful model-driven, rule-based, tools designed to enable automated modernization methods. These capabilities give organizations much better options, reducing risk and cost, when modernizing billions of dollars of mission critical legacy software.

1.2 Model Based System Engineering (MBSE)

To meet acquisition and system engineering design challenges, system architects and engineers are beginning to use MBSE standards, tools and methodologies. These capabilities enhance communications, automation, analytics, and decision making. MBSE, as envision by INCOSE, is multidisciplinary enabling the effective integration of each design discipline required to deliver a complete product (e.g., mechanical, electrical, software, hydrodynamic, hardware, software, information assurance, etc.)[4]. The recent release of Object Management Group's (OMG) Unified Profile for Department of Defense Architecture Framework (DODAF) and Ministry of Defence Architecture Framework (MODAF) (UPDM)[5], with its implicit support for NATO's Architecture Framework (NAF), will improve the availability and interoperability of MBSE tools and processes. UPDM does not, however, include specific improvements that will directly assist in modernizing existing source code.

1.2.1 Model-Driven Architecture (MDA)

OMG has developed a number of important technical MBSE standards for the modeling of software intensive systems including; 1) Unified Modeling Language (UML) with Object Constraint Language (OCL), 2) System Modeling Language (SysML), 3) Business Process Modeling Notation, and now 4) UPDM. These OMG standards enable model-based system representation, analysis, and code generation independent of the target software implementation technology. Thus, OMG technical standards enable functional system requirements to be modeled and maintained in an abstract form. These models can then be transformed for a particular target environment (e.g., hardware suite, architecture pattern, software language, information exchange protocol). OMG calls this "top-down" process MDA, i.e., build models and transform them into executable code.

1.2.2 Architecture Driven Modernization (ADM)

MDA does not directly help to modernize legacy applications and architecture that have not been modeled. Interest in this area developed within the OMG in 2003 with the subsequent formation of the ADM Platform Task Force (ADMPTF)[6]. This paper summarizes work done by the ADMPTF to formalize legacy software application and architecture representation, independent of the platform it runs on, the language it is written in, or length of time it has been in production. Today, OMG's ADM specifications enable information sharing and

powerful tools for maintaining and evolving legacy source code. ADM is a "bottom-up" process, i.e., formal software design models are derived from source code. Subsequent modernization steps then take advantage of new automated analysis and transformation methods. Note that an ADM model is not the traditional AF model, rather conceptually it is similar to a detailed SysML design model.

1.3 Architecture Definition and Evaluation

ADM standards, tools and processes conceptually complement the traditional MBSE architecture definition and evaluation discipline. ADM methods enable legacy application and architecture definitions to be recaptured as formal abstract models preserving functional semantics, business rules, behaviors and coding conventions. This type of logical and conceptual modeling enables subsequent analysis by formal analytics, as well as documentation and code generation using generic methods, i.e., not specific to the original source language. Thus, ADM enhances architecture definition and evaluation in multiple ways, it:

- captures legacy code as models from which new executable architectures can be defined,
- extends the type and scope of MBSE tools available to developers and architects,
- builds on other formal, open, OMG modeling languages,
- defines new formal modeling methods and technologies for architecture evaluation, and
- enables MBSE methods to be applied to the legacy system modernization process.

2.0 OMG ADM FORMALIZATION

The OMG ADMPTF has built industry consensus on the requirements for, and types of, legacy software modernization. These are captured in a set of reference ADM scenario use cases. The ADMPTF has also developed and published the required technical model-based representation standards needed to support its ADM scenarios. These standards are a set of meta-models that facilitate the collection, analysis, refactoring and transformation of existing software systems. As OMG standards they facilitate interoperability among industry modernization tools. Using them, tool vendors may support selected aspects of the ADM process, and seamlessly exchange system data and application meta-data among tools and across the phases of the modernization process. A given modernization project may involve more than one scenarios and use multiple ADM modeling standards.

ADMPTF industry members have taken-up these standards, developed tools and demonstrated real-world successful ADM results showing dramatic cost and time reductions. In this section we will quickly review the OMG ADM scenarios and technical standards.

2.1 ADM Scenarios

An ADM modernization initiative, or project, will typically be done in a series of model-driven phases. Each phase accomplishes a type of modernization task, e.g., a language-to-language conversion, data structures and data base redesign, etc. The ADMPTF defined a dozen types of modernization tasks, referred to as scenarios[7]. To complete an initiative and accomplish a set of modernization objectives may require performing more than one scenario. Formal software design models, initially generated by ADM tools from the legacy source code, are refactored and transformed during the modernization scenarios by various tools. Appendix 1 describes each of the following twelve ADM scenarios.

- Application Portfolio Management
- Application Improvement

- Language-to-Language Conversion
- Platform Migration
- Non-Invasive Application Integration
- Services Oriented Architecture Transformation
- Data Architecture Migration
- Application & Data Architecture Consolidation
- Data Warehouse Deployment
- Application Package Selection & Deployment
- Reusable Software Assets / Component Reuse
- Model-Driven Architecture Transformation

2. 2 ADM Modeling Standards

The OMG ADMPTF has developed and published a set of standard ADM semantics and metamodels for capturing, sharing, and visualizing application and architecture data, metrics and metadata. Appendix 2 describes each of the following seven ADM modeling initiatives.

- Knowledge Discovery Meta-Model (KDM) Package
- Abstract Syntax Tree Meta-Model (ASTM) Package
- Analysis Package
- Metrics Package
- Visualization Package
- Refactoring Package
- Target Mapping & Transformation Package

3.0 ADM MBSE PROCESS AND PRACTICE

ADM concepts, technical standards and tools extend traditional MBSE. The ADM process uses tools to automate many documentation, analysis, and transformation tasks. Automation is necessary but not sufficient. The ADM process also relies on two types of subject matter expertise and participation; 1) functional expertise, 2) refactoring and architecture expertise. The first is typically provided by the customer, and the second, by an ADM service provider using their tool set and methodologies.

The following discussion reflects ADM MBSE processes and practice at The Software Revolution, Inc (TSRI), a company founded by one of the authors (Newcomb). A more complete discussion of ADM practices along with case studies of successful ADM projects is given fuller explication in [8]. In this section we examine how ADM standards are used in practice in state-of-the-art tools and processes.

3.1 Levels; Code, Functional Design and Architecture

The scope of modernization scenarios, shown in Appendix 1, reflects the ADMPTF's recognition of the range of modernization needs. The scope and richness of the ADM technical standards, shown in Appendix 2, reflect the engineering and architectural need to perform modernization at multiple system engineering levels, the code level, functional design level and architecture level. The ADM technical standards provide this basis for unified

modeling, analysis, documentation, refactoring and transformation. They enable ADM tool vendors to developed tool suites that address a range of modernization scenarios and system engineering needs.

3.2 ADM Process Discipline

How ADM is practiced will vary from organization to organization and may vary from project to project. Some may choose an informal approach, using ADM tools to probe and better understand a legacy system before beginning traditional manual refactoring and recoding. In many cases this approach is not sufficient, a formal automated approach is required in order to:

- guarantee that the modernized system, or component, exhibits precisely equivalent performance and behavior when compared to the legacy system, or component (and thus avoid the expensive of a new certification or broader system testing),
- reduce the time and cost of modernization by reducing labor,
- address the challenge of large projects with significant accumulated technical debt,
- fully document the "to be" system pedigree based on the trusted "as is" system, and
- migrate to a model-based software functional baseline for future modernization work.

In these, and other cases, a formal, repeatable, rule-driven, automated ADM methodology should be used. The ADMPTF scenarios and standards enable a formal approach. For skilled practitioners, most projects can achieve 100% automation levels, meaning that a set of expert-defined model transformations can be executed against the legacy system model to fully refactor and transform it into the objective system model. This capability introduces economies of scale, supports customization to client specific requirements and enables an iterative, agile and rapidly repeatable process. With a model-based, rule-driven, automated tool framework, every step in the modernization process, every modeling object ever created by the process, and each transformation rule applied can be tracked end-to-end to ensure quality and completeness. The same models, formal rules, and tracking enable the development of comprehensive visual presentations and integrated documentation that can aid inspection, understanding and validation by users.

3.3 Commercial Practice

TSRI has a twenty year corporate history, and its technology can be traced back to the author's (Newcomb) work at Boeing's Artificial Intelligence Lab and the USAF Knowledge Based Software Assistant program in the mid 80s. TSRI has developed Janus Studio™ an ADM tool suite capable of full formal, automated, ADM modernization workflows. It employs advanced artificial intelligence technologies to apply automated expertise with machine-mediated precision, uniformity and accuracy. Through mathematically rigorous application of transformation rules to code-level, design-level and architecture-level models, Janus Studio automatically modernizes without the loss or distortion of functionality while improving application quality and enhancing performance.

In Figures 1 and 2 the numbered "black dots" are referred to in the text using "(x)" notation. Black dot #X, (X), refers to the same thing in Figure 1 as in Figure 2. Figure 1 shows an overview of Janus Studio in context. On the left, are the legacy systems (1) implemented in a wide variety of programming languages that can be parsed (2). On the right are shown some of the target environments and programming languages that can be generated (11). In the middle Janus Studio, incorporating (top to bottom), key ADM processes (6), the derived and transformed system design model (i.e., Intermediate Object Model, TSRI's internal ASTM+KDM model representation) (3, 5, 10) and the interactive design and transformation documentation (i.e., Blueprints) (7, 8).

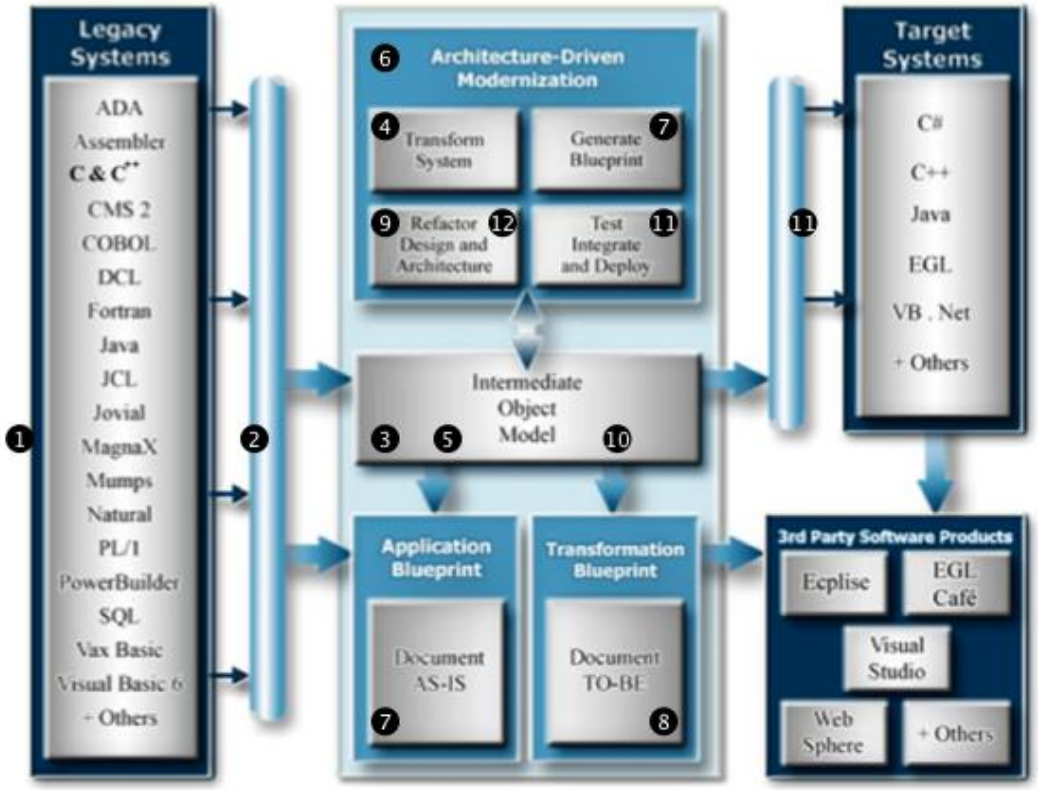

Figure 1: TSRI's Janus Studio™

Figure 2, and the following sections, relate back to Figure 1 and present the Janus Studio modernization process in more detail. Two powerful TSRI proprietary tools drive key Janus Studio processes and activities. Transformation are carried out by JPGEN™, a parser generator using an external grammar specifications (e.g., for each source programming language), and JTGEN™ a pattern recognition and pattern inference rule engine. At various stages these tools parse, transform and generate code and models.

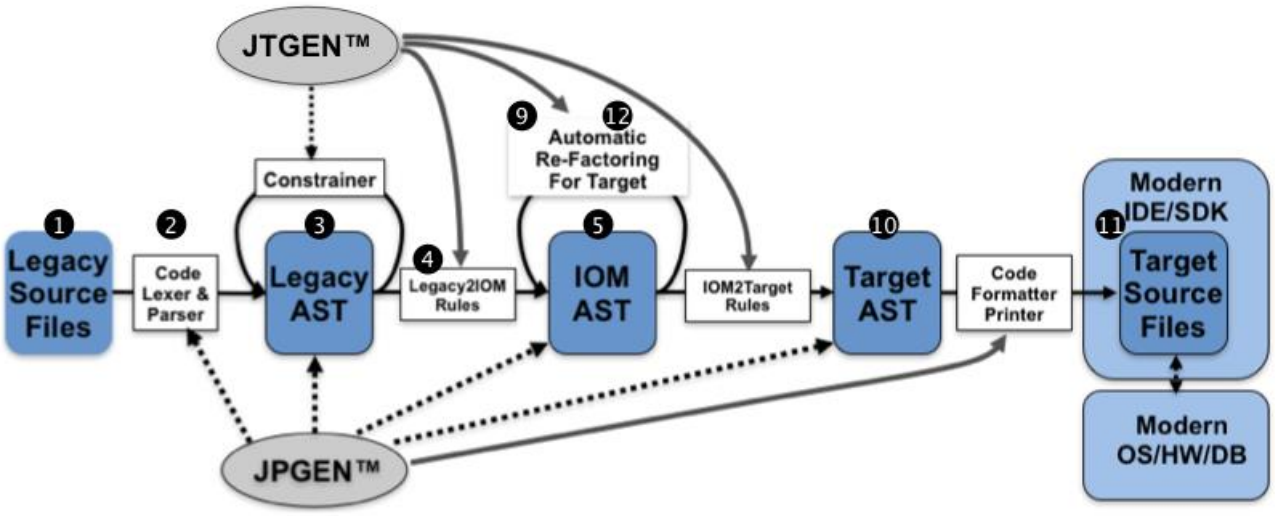

Figure 2: TSRI's Janus Studio™ Model-Based Modernization Process

3.3.1 Derive A Software Design Model From Legacy Source Code

All ADM projects start by parsing (2) the legacy source code (1) to derive a software design model (i.e., legacy AST, an instance of an ASTM model)(3) that is then transformed (4) into an IOM (5) for internal processing. This preserves the "as is" model and provides a workspace for modernization transformations.

3.3.2 Assess and Document the "As Is" Software Design Model

The derived IOM (5) is evaluated (6) in this stage for completeness, dependencies, various metrics, other analytics and presented to the user as an interactive online "as is" design document (7). TSRI refers to this package as an Application Blueprint which serves as an initial detailed baseline for defining the objective "to be" architecture. These two perspectives enable the definition of a transformation plan that is captured as a Transformation Blueprint (8).

3.3.3 Develop a Transformation Roadmap to Achieve the "To Be" Software Design

During this phase the IOM is rewritten (9), in accordance the transformation plan, into new object-oriented, platform-specific, code (10). If required, this transformation also couples the generated code to the target database, and resolves or converts internal and external interfaces to existing replacement services or new development APIs on the target platform middleware. At this point the code has been architecturally modernized and should demonstrate functional equivalency, validating the transformed IOM (11).

3.3.4 Incrementally Refactor Software Design Model and Test

In this phase, the validated IOM is incrementally refactored (12) using rule-driven pattern-matching transformations and incrementally revalidated using regression testing. These steps are aimed at improving the modernized system's design, architecture, maintainability and enhancing its performance. Semi-automated, automated refactoring and custom developed refactoring operations are carried out against the IOM to generate redesigned code modules and re-architected application tiers.

3.3.5 Generate and Document the Final "To Be" Modernized Source Code

During the finalization phase, the latest code baseline is merged into the modernization process where transformations developed, applied and validated in the earlier phases are applied to the final release (1-12). Delta areas are identified for final regression tests. At this point, the modernization objectives have been achieved and a "to be" architecture documentation package can be generated containing the final source code (11) and an Application Blueprint (7) for the "to be", now "as is" modernized system.

3.4 ADM Projects Successfully Completed

TSRI has accomplished over 100 successful ADM modernization projects during the last ten years. Table 1 characterizes a sample of the projects. Some of these projects are described in great detail in [8] and at the TSRI company web site [9]. For the Thales Air Traffic Management system project which modernized critical components of the European Air Traffic Management System, one of the authors (Newcomb) was recognized with the 2011 International Steven's Award for software development methods [10]. The award recognized contributions that "advanced the application of artificial intelligence- and rule-based technologies for reverse engineering, architecture reassessment, testing, and redevelopment, providing innovative products and services for reliable reengineering of mission-critical and high assurance systems".

Also available at [9] are Application Blueprints, for over 250 open source projects (e.g., Apache Tomcat). The Application Blueprints are a open source resource for visualizing the IOM model, its data and metrics and how it is linked to project source code.

Table 1: Sample TSRI ADM Projects

	Integrator	System	Code	SLOC	TTC
Civilian	SAIC & Open Source	Fileman, VistA Pilot and	MUMPS to Java	2.5M	6 mo
	Health Care Insurance Co	OpenVista Cross Provider System	PowerBuilder & Magna X to Java	1.2M	3 mo.
	AMDOCS	Billing System	COBOL to C	5.1M	7mo.
	Thales Air Systems	French Air Traffic Management	Ada to Java	495K	12 mo.
	Thales Air Systems	Nordic Air Traffic Management	Ada to Java	541K	9 mo
	Thales Air Systems	Australian Air Traffic Management	Ada to Java	638K	9 mo
	Unisys	NY State Dept. of Criminal Justice	COBOL Documentation	308K	2 mo.
	NEA	Grant & Business Systems	COBOL to C++	656K	7 mo.
	SAIC	Veteran's Health Administration	MUMPS to Java	300K	4 mo.
	State of OR	Employee Retirement System	COBOL to C# .Net	250K	4 mo.
	State of WA	Off. of Super of Public Instruct.	COBOL to C# .Net	191K	5 mo.
	TriGeo	Sim v4.0 (Internal Product)	Java Docs & Re-Fact.	370K	2 mo.
	EDS	Proof-of-Concept	P/L 1 to Java	50K	7 mo.
	CSC	Bureau of Immigration	COBOL to C++	17K	3 mo.
Military	CGI/Stanley	Advanced Field Artillery Tactical Data System	Ada to Java	5.5M	9 mo.
	Boeing	ALCA - Czechoslovakia	Jovial to C++	9K	2 mo.
	LMCO	P-3C	Ada - C++	656K	14 mo.
	ITT	BMEWS - Cobra Dane	Ada/Fortran - C++	380K	8 mo.
	Raytheon	Satellites	Ada/Fortran - C++	284K	5 mo.
	L-3	VTT	Ada - C++	77K	3 mo.
	LMCO	SAC Strategic Planning System	Ada - C++	40K	2 mo.
	DSR	E-2C ACFT	Ada - C++	20K	5 mo.
	USAF	CAMS	COBOL Docs	1M	6 mo.
	NGC	REMIS	COBOL - C++	400K	7 mo.
	Dyncorp	WCSRS	COBOL - C++	90K	5 mo.
	ITT	BMEWS - ROSA	Fortran/C Docs	2M	4 mo.
	Raytheon	Patriot Missile	Fortran - C++	200K	6 mo.
	Litton PRC	Strategic Air Command	Fortran - C++	50K	4 mo.
	Raytheon	WDAC	Fortran - C++	40K	1 mo.
	SAIC	EOSS	VAX Basic - Java	38K	5 mo.
	TRW	MILSTAR	Jovial to C++	143K	1 mo.
	USAF	F-16 Decis	Jovial to C++	50K	4 mo.

3.4.1 ADM Cost

The cost of performing a specific modernization project will depend on the type and amount of system engineering and architecture work required in the ADM scenario(s). Regardless, it is has been TSRI's experience that automated modernization projects, performed using Janus Studio, on average are 1/10th the cost of similar tasks done by manual methods. Similarly, the time to accomplish the task is greatly reduced, typically by 50%. In more extreme cases, where the technical debt and the size of the source code package are greater, the modernization savings, on a per source line of code basis, can be even greater due automation. These initial benefits may be compounded in the future. The ADM process produces not only a new documented source code baseline but also a formal software design model that can continue to be used for future system analysis, testing, development and modernization activities.

4.0 ADM IMPACT

ADM standards, tool technologies, and model-based processes enable a new range of software MBSE. ADM methods build on traditional structured software analysis and metrics for assessment of information systems, but, go beyond to enable automated refactoring that transforms the behavioral, structural and interaction properties of the system at the code, design and architectural levels. Perhaps surprisingly, modernization projects have the lowest risks and highest rates of return for the lowest investment compared to build or buy alternatives[3]. This is in part because, formal ADM methods aid in controlling code scope, complexity and quality which helps to avoid other costs and risks associated with major redevelopment efforts.

The thoughtful reader will have already begun to consider how such capabilities could impact traditional systems development and maintenance processes. For example, because functional enhancements can be more easily undertaken on modernized software, software improvement projects should incorporate two phases; application and architecture technical refresh, followed by functional improvements. Other considerations might include; what additional skills will developers and managers require, how will ADM fit with AF modeling and detailed system modeling, and how might developers blend ADM and MDA methods?

For those developers comfortable working manually to construct and refactor code, ADM methods and models may be too formal and abstract. For those architects already laboring to develop AF models that capture system business processes, business rules, enterprise semantics, capabilities, etc., ADM models may seem too detailed and not linked to operational views and mission architectures.

Standardized ADM tools and processes are relatively new to acquisition managers and system integrators, so new in fact that traditional acquisition practices, as well as widely adopted software life-cycle management disciplines, have yet to recognize them. Regardless, specialty modernization tools and experts have been around for some time. So we might ask, if ADM skills, tools and processes are really so novel should organizations draw on external expertise when necessary, and in the interim, rely on traditional software development and MBSE methods? Or is it time to develop internal ADM skills and begin to adopt ADM methods for critical legacy software systems? Before answering too quickly, consider that in addition to improving current technical system engineering and modernization processes, ADM methods may actually necessitate changing traditional software business processes, project roles and client –supplier relationships. It may also alter the way software engineering tools are used in support of software design and build processes.

Consider that ADM's modeling abstraction enables new processes. An IOM (i.e., ASTM and KDM representation for any source code language) enables a single comprehensive set of analytic methods to be uniformly applied to legacy systems composed of many languages (e.g., the authors are exploring how ADM can help modernize a maritime, mission critical, tactical system currently implemented using Ada, C, C++ and Java software components). Thus, a customer organization could apply a standard ADM analytic evaluation package to all delivered source code.

Last, consider that ADM models and methods might enable new technical and business processes not previously possible. New semi-automated ADM applications might enable a functional SME to link ADM design models to AF models, formally documenting top-to-bottom the "as is" executable architecture. Additional ideas are discussed below.

4.1 Periodic Modernization Events

All organizations developing or using software must eventually deal with legacy software accumulated technical debt. And yet, the need to adopt ADM methods may not become clear until traditional software methods and costs are no longer effective or affordable due to the escalating costs of the accumulated technical debt (perhaps detected as ubiquitous patching, endless debug cycles and degraded code structure and quality). The high risk associated with greenfield redevelopment of a system that is functionally sufficient, but technically a basket case, may also become a catalyst for trying ADM or adopting ADM methods. Overall cost might be greatly reduced, delayed or avoided if ADM refactoring tools and methods were used routinely in the software life cycle. The promise is compelling, but the technology, skills and process are unfamiliar. Like MBSE, internalizing ADM skills, tools and processes will take time and investments and so many organizations choose to initially, or periodically, hire external ADM expertise. This has multiple initial benefits in that it can be an effective and less risky way to; 1) modernize a critical system, 2) save time and money, and 3) learn about ADM technology and methodologies. This is what many government and industry organizations have been doing.

4.2 ADM Enhanced MBSE Process

The periodic reliance on external ADM expertise is likely necessary for most organizations but is no substitute for an internal MBSE process that can effectively address modernizing critical systems. While it is possible to create ADM "islands", through periodic initiatives, a larger challenge and opportunity exist for those that can integrate ADM concepts and capabilities into their enterprise MBSE business processes. Generalizing, the more frequently modernization is undertaken, the lower the technical debt that must be addressed in each cycle, the

lower the risk, and the easier the revisions. The more frequently ADM methods are used the more they should be performed in-house as a part of a normal software portfolio maintenance and modernization process.

In what ways might ADM standards, tools and methods interplay with, and enhance, typical MBSE business processes? As has already been discussed, the AF top-down modeling approach and the ADM source code based bottom-up approach result in different types of models. Traditional MDA models, methods and tools are more similar to ADM, but again, differ in some important ways. Each adds value, each complements the others. How these related but different approaches can be combined to support enterprise MBSE requirements and practice is still an evolving technical and commercial story and will vary in practice from organization to organization. Importantly, ADM can uniquely address the critical challenge of modernizing and maintaining legacy software, a challenge that we know consumes a majority of many military software acquisition budgets. For budgetary reasons alone ADM is compelling, but ADM enhanced MBSE and business processes could offer further benefits.

4.2.1 Process Improvements

Military systems acquisition relies on the government developing requirements and the contractor determining how to best meet those requirements. Too often this leads to a "black box" system. The government gets a capability but not the detailed system and architecture design and implementation knowledge (i.e., AF and SysML models) that will be essential for maintaining and modernizing the system over its lifetime. Most often today, source code and various software description documents are all that is provided. Further, a contractor may not be understand the detailed design characteristics of "black boxes" provided as government furnished equipment (GFE). To some degree these issues are the result of programmatic decisions and the typically high costs charged for detailed design documentation. The lack, or loss, of insight and understanding as to what is within our systems, and how they work, limits the government's ability to oversee system acceptance, certification, and maintenance. Arguably, it can have negative effects on information assurance, security, current and future costs, etc. These limitations are a part of the technical debt that eventual must be paid.

ADM standards (i.e., ASTM and KDM) provide an important architectural definition and evaluation innovation by enabling the formal representation of any functional design or architecture in a technology independent manner. It enables ADM-derived military system design models to be managed and evaluated in a single, consistent manner, independent of the system, age, technology or developer. This approach, while dependent on the sophistication of the legacy source code parser, enables us to envision a useful new ADM repository for understanding and managing software portfolios.

4.2.1.1 Standardized ADM Portfolio Documentation and Metrics

Migrating legacy source code portfolios to a common formal representation held within a single ADM repository would be a step forward for most organizations. Such a repository could meet the needs of various ADM MBSE workflows. It would enable a standardized, unified (i.e., support across system-of-systems code), formal MBSE approach to modeling, transformation planning and execution, documentation generation, access and version control. This type of repository could be used to generate standard, formal, detailed GFE design and code packages for industry during contract competition and system development. It creates a strategic resource for preserving and managing system design and implementation knowledge over time and across inter-related systems.

4.2.1.2 ADM-based Enhanced Design Analytics

The software portfolio ADM repository makes possible the uniform and automated application of system, and enterprise-specific, metrics and ASTM/KDM analytics to evaluate performance, complexity, vulnerability, compliance, assurance and other functional and non-functional design characteristics. A uniform, detailed, set of analytics for characterizing an entire software portfolio at the code, functional design and architectural levels

along with a design models and generated documentation would greatly aid the customer in understanding, comparing and managing the quality and complexity of mission critical software. These analytics could be applied within and between systems (e.g., checking interoperability by matching data types on both sides of an intersystem interface). Again a portfolio-level approach creates a strategic resource for preserving and managing system design and implementation knowledge over time and across inter-related systems.

The automated generation of ADM-enabled analytic assessments could become an integral step in a software (re)engineering practice, rather an afterthought applied just prior to a final gateway review. Consider further that automated refactoring methods in some ADM tools (i.e., CodeHawk™ from the Kestrel Technology and Coverity SAVE®) are already being used to identify and remove common weaknesses, defects and vulnerabilities. These methods benefit from the work on quality assurance metrics (e.g., Common Weakness Enumeration (CWE), Common Attack Pattern Enumeration and Classification (CAPEC)) and can enable routine quality and information assurance design checks as an integral part of every maintenance or function point assessment in the software development life cycle.

4.2.2 New ADM Scenarios

ADM standards and tools could provide potentially valuable new methods of tackling familiar, difficult and expensive problems. This section discusses what might one day be new ADM scenarios. The concepts discussed are already technically supported. Additional formal semantics, or profiles, could help to further standardize and broaden these potential ADM scenarios.

4.2.2.1 ADM-based Enhanced System Testing

As previously discussed, ADM tools and techniques can be used to transform one system model design pattern to another while preserving a system's/model's semantics and behavior. Consider also that model patterns can be created for software "test points". A "recording" test point might; 1) record data or state information, 2) capture and record user interactions, 3) log exceptions and errors, etc. Test points can then be introduced into a system design model by pattern matching transforms (e.g., *find* [user input] *change to* [user input + write log entry]). Using MDA methods, a model with rule-inserted test points can be used to generate an instrumented source code package. In this way it becomes practical to overlay on a system, or across a system-of-systems, a consistent, unified, test harness. Using such techniques, ADM standards and tools not only support static software evaluation but can enable robust and holistic run-time testing.

Similar ADM methods could also provide improved automated testing capabilities. Consider that ASTM & KDM design models can be adorned with metadata, including requirements data (e.g., response time), and analyzed for state and input constraints (e.g., from this state there are only four selection options [1..4]). From this, and other model information, a semi-automated tool could construct constrained test scripts to automate system/user interaction testing. As with the recording test points, "data inject"/"user interaction" test point models could be defined and overlaid on the system model creating a programmatic interface supporting the defined test scripts.

At this point we have used deep model evaluation knowledge to generate test scripts and apply test point model transformations to control and instrument software testing. At run time a test script could generate inputs, receive real-time recorded responses, and evaluate if the results are within acceptable limits. A similar simpler type of model-driven testing is available in many integrated development environments. In this case, however, the abstract design model and the ability to identify design patterns would make it possible to instrument the run time system and system-of-systems at the code, functional and architectural levels. When testing is complete a version of the source code, without the test points, would be generated.

4.2.2.2 ADM-based Integration

Already, developers are able to use SysML modeling to capture and evaluate detailed interactions among subsystems and between systems. If MDA techniques are used to generate a system's source code from its SysML models then the system assessments might hold-up at run time (assuming no manual coding). These model-based evaluations provide an early opportunity to identify and correct integration and interoperability issues. They do not substitute for integration testing. The system integrator is still responsible for bringing together the many source code pieces and ensuring they interoperate to meet functional and non-functional requirements.

As has been hinted at earlier, ADM methods might also change the way system integration is performed. Consider an approach, where the integrator receives each subsystem source code package and adds them to the ADM repository where they are modeled, evaluated, and documented. The integrator's proprietary analytics could be applied to the project, as could the government's analytics-based acceptance test criteria. The analytics can check for completeness, correctness, compliance, etc. The integrator could apply transformations to adjust subcontractor code, system or architecture design, if required. At this stage the integrator could apply test points, as described above, in a holistic manner to code from every subcontractor. The integrator's, instrumented, model-generated system software package could then be tested. Keep in mind that this code is not the code delivered by the subcontractors! Instead, the integrator has a groomed, instrumented, source code package generated from an evaluated and documented set of integrated models. The advantages of this type of integration could be many.

Just as the integrator could use this technique so could the government customer to expose, evaluate, test, and document delivered products. One day the contractors modernizing legacy systems might deliver ADM type executable design models, with code generation a necessary but pro forma step.

4.2.2.3 ADM support for Architecture Framework Views

As has been mentioned, ADM ASTM and KDM system design models are not typical AF or SysML views. Regardless, it should be possible to create meaningful semantic associations between the system design models and architecture models. This might be done using a guided, semi-automated, characterization and allocation of the source code generated system design models. Automated pattern matching profiles could perform, as is already done, design abstraction and modeling. Such a tool would still require functional and architectural subject matter expertise to add system and architecture specific semantics and metadata to the derived patterns. Such a MBSE process working from the bottom-up could greatly accelerate capturing authoritative, detailed, "as is" architecture products for mission critical legacy software and architectures.

5.0 SUMMARY AND RECOMMENDATIONS

Modernizing valuable legacy mission critical systems and architectures is very expensive and routinely required for many reasons! To deal with the "elephant", the crushing technical debt associated with maintaining and modernizing military legacy software systems, ADM methods should be adopted by acquisition and contractor organizations. ADM standards and methods enable powerful new types of MBSE tools and processes for computer-aided legacy software portfolio evaluation, documentation, refactoring, testing, and life cycle management. Real-world experience with ADM has routinely demonstrated dramatic cost and time reductions while improving software quality. The adoption of these new tools and techniques will require investment and likely require adaptation at the technical, business process and organizational levels. ADM project initiatives are expected to have a high return on investment. Enhanced ADM MBSE scenarios may be more disruptive but could also result in even greater legacy software life cycle improvements and savings.

REFERENCES:

[1] Information Technology: Agencies Need to Strengthen Oversight of Billions of Dollars in Operations and Maintenance Investments, Government Account Office Report 13-87 (GAO-13-87), October 2012

[2] http://en.wikipedia.org/wiki/Technical_debt

[3] Modernization: Clearing a Pathway To Success, The Standish Group, 2010

[4] http://www.incose.org/

[5] http://www.omg.org/spec/UPDM/

[6] http://adm.omg.org

[7] http://adm.omg.org/ADMTF_Overview_Ulrich.pdf

[8] Information System Transformations: Architecture Driven Modernization Case Studies by William Ulrich and Philip Newcomb (TSRI CEO), Morgan Kaufman, 2010 (OMG Press) (429 pgs.)

[9] http://www.tsri.com

[10] http://reengineer.org/stevens/

APPENDIX 1: ADM SCENARIOS

The ADMPTF has defined a dozen base modernization tasks, referred to as scenarios[7]. In combination, the scenarios address various analysis, refactoring, consolidation, migration and redesign initiatives that an organization may need to pursue.

I. Application Portfolio Management

Application systems and architectures are mission critical business assets and, as such, form a portfolio that requires analysis, documentation and management. This scenario captures and exposes technical and functional meta-data on an organization's systems using systems analysis tools to augmented expert analysis. The resulting knowledge base on existing systems is central to the effective and ongoing ability to manage information systems as organizational assets

II. Application Improvement

The application improvement scenario is typically comprised of several modernization tasks. This scenario is performed to improve the robustness, integrity, quality, consistency and/or performance of applications. Using analysis tools, functional and architecture experts seek to; correct program or system flaws, improve source code structure, rationalize and standardization data definitions, improve user interfaces, and other system refactoring tasks. This scenario involves no architecture transformation tasks, that is, it focuses on improving the application in its current architecture context.

III. Language-to-Language Conversion

This scenario is performed to migrate source code from one language to another. This may be driven by a variety of factors, but does not involve a redesign of the application functionality beyond that which is required by the language change itself. There are direct language-to-language software development transformation tools. In the ADM scenario, the source language is used to generate a formal design model in KDM and ASTM (see Appendix 2) which is then transformed into the desired target language.

IV. Platform Migration

Systems will over the course of their life need to migrate from one platform to another. This is often driven by hardware or software (e.g., operating system or middleware) obsolescence or a strategic change to new standards or products. This scenario does not include functional or data redesign unless essential to the platform migration. It would be combined with a language-to-language conversion if required.

V. Non-Invasive Application Integration

The non-invasive (i.e., does not change an underlying system) application integration scenario support modernizing the user front-ends of business systems, typically upgrading to a browser-based graphical user interface supported by a middleware software technology. Users gain a modern interface experience while the core system functionality, data structures and other interfaces remain mostly unchanged. The scenario may precede the SOA migration scenario.

VI. Services Oriented Architecture Transformation

The services oriented architecture (SOA) scenario may require deep refactoring and modularization of existing monolithic application functionality as well as alignment to established architectural patterns and services. This would typically include the segregation of application and data architectures. Business logic should be separated from the user interface and data access mechanisms. A full SOA transformation will impact both application and back-end systems. ADM standards and tools aid in this complex set of transformations by

helping to identify and track relationships between the physical system, program functionality, data usage and user interfaces.

VII. Data Architecture Migration

A data architecture migration moves one or more data structures from a non-relational file or database to relational data architecture. Many times this is done using a "quick and dirty" approach, leaving users with performance, reliability and data accessibility problems. Pitfalls include ignoring business requirements, sidestepping relational design techniques, not incorporating related or redundant data in the project, not utilizing qualified data analysts or treating the project as a straight conversion effort. This scenario shuns the quick and dirty approach.

VIII. Application & Data Architecture Consolidation

In this scenario, duplicative and complementary systems and data are consolidated. This simplifies the organizations software portfolio and reduces costs. The scenario may aid an organization by producing single authoritative data sources, thus, improving quality and better managing access. The actual semantic harmonization and alignment of data from multiple sources may be challenging, but this scenario creates a context within which to gain better control of enterprise data.

IX. Data Warehouse Deployment

This scenario consolidates organizational and business data in an enterprise repository with services that can be used to protect, extract, analyze and transform the data as required for business purposes.

X. Application Package Selection & Deployment

This scenario aids an organization with third party application selection and integration decision making. ADM tools can be used to compare functional requirements and assist with determining which new application capabilities should be implemented, integrated, discarded or updated. It also outlines how existing systems are to be retired, integrated or retooled to work with a package.

XI. Reusable Software Assets / Component Reuse

Modernization includes a reuse scenario for critical software applications and services. Reuse can improve productivity and ensure consistency across an enterprise. Redundancies can impose significant hidden costs in application and data structures across software portfolios, and thus, identifying opportunities for reuse can create current and future savings. ADM tools can help locate these opportunities across the software portfolio and support the refactoring needed to achieve the desired simplifications.

XII. Model-Driven Architecture Transformation

Most legacy software was created by hand, perhaps within the context of an integrated development environment that assisted the developer in managing his or her code base. Transforming from a manual code development environment to a model-driven development environment requires the appropriate tools and developing formal design models for existing applications and architecture. This scenario could be accomplished using traditional MBSE tools from the top down but in most cases this is not an option. This scenario relies on ADM modeling tools to transform existing hand-crafted applications into models. These formal models can then be transformed to generate new versions of familiar applications.

APPENDIX 2: ADM MODELING STANDARDS

The OMG ADMPTF has developed a set of standard ADM semantics and metamodels for capturing, sharing, and visualizing application and architecture data, metrics and metadata as required by the ADM scenarios[7]. This appendix gives a short characterization of these standards.

Knowledge Discovery Meta-Model (KDM) Package

The KDM provides a foundation for all other ADM standards by establishing a meta-model that spans system artifacts and the relationships among those artifacts. It enables modernization tools to exchange application meta-data and interoperate independent of computing platform and operating system. The KDM meta-model provides a comprehensive view of application structure and data, down to the procedural level supporting software modernization, IT portfolio management, software assurance and legacy software knowledge discovery.

Abstract Syntax Tree Meta-Model (ASTM) Package

ASTM builds upon the KDM to represent software below the procedural level. ASTM itself serves as a universal high-fidelity gateway for modeling code at its most fundamental syntactic level. The ASTM respects the scope of the KDM and the UML for modeling the semantics of higher-level software concepts and it therefore includes only low-level semantics that are closely associated with code (namely code location, scope, reference, and type). ASTM extends KDM to fully represent applications and facilitate the exchange of granular meta-data. ASTM supports a direct 1-to-1 mapping of all code-level software language statements into a single low-level software semantic model. This mapping provides a framework for a powerful, formal, high-fidelity, invertible, single representation of any software language application code (achieved by linking ASTM with concrete syntax specifications).

Analysis Package

The ADMPTF Analysis Package extends the structural representations in the KDM and ASTM by representing additional levels of analytical or behavioral representations of systems in certain meta-models. In this way, analysis results can be stored and shared in association with the system under analysis. Another ADM analysis specification is the Software Assurance Evidence Meta-model (SAEM) for collecting, developing, evaluating, communicating, and managing software assurance evidence.

Metrics Package

The Metrics Package standardizes meta-model representation and sharing of various types of metric information (e.g., structured metrics, automated function points) in association with the system under analysis.

Visualization Package

The Visualization Package provides meta-model(s) that support the visualizations of existing system structures, data usage and behavior. ADM visualization can be used to depict application meta-data stored within the KDM as appropriate for planning and managing modernization initiatives.

Refactoring Package

The Refactoring Package meta-model describe the ways that KDM can be used to transform and refactor applications. This includes code, functional and architectural rationalization and modularization to improve existing applications.

Target Mapping & Transformation Package

ADM Transformation defines mappings between the KDM and ASTM and target models. This standard supports mappings and transformations that may occur between an existing application model and an MDA environment.