Architecture-driven modernization has a long history, but it has faced many challenges as it has been deployed and redeployed at numerous organizations. Any time there is a lack of standards within a given industry, it opens the door to confusion, and this has a direct bearing on the adoption rate of a given set of solutions. This is particularly true in the case of modernization.

While an executive buyer or even casual deployment team of modernization solutions is unlikely to spend much time thinking about standardization issues, standardization does matter. Having seen companies go through modernization efforts over the decades and across various organization units, one can quickly see that the lack of synergy, coordination, and interoperability across modernization tools results in silo-based projects that cannot cross individual project or organizational boundaries. Numerous issues have complicated adoption of modernization over the years that can be tied directly back to the historic lack of industry standards such as:

- No consistent message from modernization service and tool providers as to the breadth of capabilities that modernization can provide.
- Conflicting and often wrong-headed value propositions in terms of what modernization can deliver to the enterprise bottom line.
- Lack of a clear understanding of the disciplines and the specific role of tools in various stages of modernization projects.
- Inability to perform assessments beyond the bounds of an individual system because tools cannot interoperate with other tools that support different platforms and languages.
- Repeated start-up and assessment efforts every time a new project team or business unit begins a modernization effort.
- Inability to muster any momentum for modernization programs due to the fragmented concepts and strategies that dominate the marketplace.
The bottom line is that the historic lack of standards has been responsible for proliferating misinformation and the inability of management to create a cohesive modernization strategy. Fortunately, a task force was formed under the international, not-for-profit standards body called the Object Management Group (OMG). The OMG is responsible for the creation and maintenance of a number of standards including UML, BPMN, and a host of other standards in a variety of technical and business domains.

Standards are created by member organizations that submit a standard in response to a task force request. Members then evaluate the submissions and pass them as a standard. Various levels of oversight and review are involved and, in a number of cases, the work is coordinated with other standards groups such as the Open Group or International Organization for Standardization (ISO). The modernization standards discussed in this chapter involve current standards, work in progress, and planned standards as defined by the OMG modernization roadmap.

MODERNIZATION STANDARDS OVERVIEW

In 2003, the OMG Architecture-Driven Modernization (ADM) Task Force was formed by a number of companies including major hardware suppliers, systems integrators, independent software providers, and customer organizations. The ADM Task Force crafted and issued a multi-stage modernization roadmap that established a plan for a series of modernization standards.\(^1\)

OMG standards focus on creating a common view of information or metadata that a given tool gathers, manipulates, and shares with and through other tools. Within the context of modernization, this means defining a common view of all of the artifacts that comprise existing software environments. Because every modernization tool gathers and stores information about existing software environments in different ways, there is no expectation that every vendor has to change every tool repository to conform to a common metamodel.

When it comes to exchanging or sharing this information with other tools, however, a common view of existing systems metadata becomes very important, and this is what the ADM standards are meant to address. For example, consider project planning that requires an analysis of several large systems containing overlapping functionality. These systems are running on a mainframe and several distributed environments. They are written in COBOL, IBM Assembler, C, and Java using different data structures and user interfaces. Two tools have been brought in to examine these systems, compare overlapping data usage and business logic, and create a consolidation plan.
Traditionally, several approaches have been used to accomplish this analysis. One involved running a tool across as much as the tool could read and reviewing the rest of the systems manually. This time-consuming approach resulted in incomplete and inaccurate results. A second approach involved running the same tool and totally ignoring the rest of the applications as out of scope. This second approach created an incomplete view of the problem and the solution, leading to a failed or canceled project. The last approach involved running two separate tools and attempting to reconcile reports and inquiries manually, across highly incompatible views of these systems. Each approach was time consuming, incomplete, and error prone.

The OMG ADM standards now in place and being created by the ADM Task Force are meant to address the challenges facing modernization analysts who have had to struggle around piecemeal solutions to significant IT architecture challenges. These standards allow tools to share captured information, which benefits organizations looking for alternatives to traditional IT strategies or seeking to use modernization to augment those strategies. Consider that:

- Analysts and project planning teams can gain a comprehensive view of the overall application and data architecture to ensure that an accurate understanding of the as-is architecture is incorporated into strategic plans.
- Projects may not be able to rely on a single tool where they span application areas, organizational units, or even business boundaries.
- One tool cannot analyze every platform or language.
- Non-IT (e.g., desktop) systems can be examined using specialty tools, and this information can be incorporated into the bigger planning picture.
- Visualization features in one tool can be used on information captured by another tool.
- Additional tools can be applied to visualize application and data architectures in new and unique ways that could include simulations or other advanced analysis techniques.
- Existing IT architecture metadata can be imported into refactoring and/or transformation tools and, ultimately, into forward engineering tools to help automate redesign and redeployment.
- Modernization standards allow organizations to begin modernization projects knowing that tool interoperability across vendors provides migration options should a given vendor tool be removed from the market.
- Standards provide organizations with assurance that they are investing not just in individual tools but in a coordinated industry approach.

OMG standards enable tool interoperability across languages, platforms, vendor organizations, business units, and IT disciplines. One beneficiary of this interoperability concept is the enterprise that must leverage these
tools to modernize IT environments and synchronize these modernization efforts with the evolution of business models and forward engineering paradigms. Collectively, these standards create a set of building blocks for modernization-based alignment of business and IT architectures.

Figure 3.1 shows the role of various OMG standards across business and IT domains. A number of standards support the evolution and transformation from existing business and IT architectures to target business and IT architectures. As more organizations understand the need to synchronize transformation efforts across multiple architectures, the demand will grow for vendor tools to interoperate across business and IT domains, as-is and to-be architectures, and the various tools that function within and facilitate the evolution and transformation of these environments.

The backdrop for Figure 3.1 is the modernization domain structure introduced in Chapter 1. Figure 3.1 depicts several metamodels within the IT domain that build upon each other in terms of their ability to reflect different views of the existing IT architecture. The Abstract Syntax Tree Metamodel (ASTM) standard provides the most granular view of the IT architecture and supports the highly automated transformation from existing languages and platforms to target languages and platforms. This type of transformation can be accomplished with limited or no involvement of the business architecture, although we recommend engaging business teams to ensure that you built support and funding for these projects.

The Knowledge Discovery Metamodel (KDM) standard is a multipurpose metamodel that represents all aspects of the existing IT architecture. The KDM is the baseline interchange metamodel and facilitates tool interoperability for any tool that captures or utilizes information about the existing IT architecture.
Modernization Standards Overview

Additional modernization standards that provide different views of the existing IT architecture include the Systems Assurance Evidence Metamodel (SAEM), the Pattern Recognition Metamodel, and the Structured Metrics Metamodel (SMM). These standards will be explained in more detail throughout the remaining sections of this chapter. Basing each of these subsequent ADM standards on a common metamodel view of existing IT architectures eliminates reanalysis and misinterpretation of existing IT environments throughout iterative assessment and modernization project cycles.

Modernization paths through application, data, and business architectures require synchronization of IT and business domain, and this has been historically absent from modernization tools and projects. The goal is to enable highly coordinated transformations of business and IT architectures and ensure that business effectively drives change through IT application and data architectures.

Figure 3.1 shows how business modeling standards, which continue to evolve just as modernization standards evolve, support the mapping across business architectures and between business and IT architectures. For example, there have been selective mappings established between the existing software artifacts within KDM and rule definitions defined within the OMG Semantics for Business Vocabulary and Rules (SBVR) standard. SBVR defines business semantics and the rules associated with those semantics. While this is an important interface, it is not a complete picture of the business architecture. Business units, capabilities, processes, customers, partners, and value chains, along with semantics and rules, collectively define the business architecture.

The standards needed to fully represent the business architecture, along with how these standards interrelate, are still evolving through active efforts by the OMG Business Architecture Working Group. As business standards evolve, modernization standards can be synchronized to ensure that business architectures and IT architectures evolve in a synchronized fashion so that IT can be more responsive to short- and long-term business requirements.

As IT architectures undergo modernization, part of that evolutionary process is to transform existing architectures into target architectures that are typically defined using a different set of disciplines and paradigms than were used on existing application and data architectures. For example, target architectures are now being crafted using a Model-Driven Architecture® (MDA), as defined by the OMG. According to the OMG Web site, MDA “provides an open, vendor neutral approach to the challenge of business and technology change.”

MDA is supported through the use of the Unified Modeling Language (UML), another OMG standard. According to OMG,
UML, along with the Meta Object Facility (MOF), also provides a key foundation for OMG’s Model-Driven Architecture®, which unifies every step of development and integration from business modeling, through architectural and application modeling, to development, deployment, maintenance, and evolution.

UML has been widely deployed in development tools and is an ideal target for standards-based mapping between as-is and to-be IT architectures through the alignment of ADM-based standards and MDA and UML. Note that we are not advocating the use of UML as a key communication tool for business but only as a requirement and specification facility.

Mapping between the modernization standards, business standards, and forward engineering standards addresses a significant industry problem. In the past, the forward engineering industry largely ignored existing software assets and this has resulted, in part, in the industry failures highlighted in Chapter 1. There is a gap between modernization and Greenfield- and COTS-based initiatives. Interoperability across modernization, business and development standards, and paradigms breaks down these barriers and facilitates increased automated tool support for business architecture/IT architecture alignment efforts.

In the absence of modernization standards and tools, there is no glue in place to synchronize the transition across business and IT domains and as-is and to-be architectures. This gap, along with the historic absence of an integrated set of business modeling standards, has placed an impossible burden on organizations that need to address business and IT challenges while dealing with aging and complex architectures.

Therefore, modernization interoperability standards, which have been developed by the ADM Task Force and deployed in commercial modernization tools, are essential to the successful adoption of forward engineering paradigms such as MDA and the ability to align existing IT architectures with evolving business modeling paradigms. The next sections describe the ADM standards that have been deployed, are being deployed, or are in various planning stages.

**KNOWLEDGE DISCOVERY METAMODEL**

The KDM established a metamodel that allows modernization tools to exchange application metadata across applications, languages, platforms, and environments. This metamodel provides a comprehensive view of as-is application and data architectures, but was not intended to represent systems at the most granular levels possible. (The ASTM, on the other hand, addresses detailed metadata below the procedure level.)
The KDM describes representations of the as-is application and data architecture along with information that can be derived from an as-is view. The KDM was developed based on a number of key requirements and

- Represents principal artifacts of existing software as entities, relationships, and attributes
- Is restricted to the artifacts of the existing software environment
- Maps to external artifacts with which the software interacts through other standards and metamodels
- Consists of a platform- and language-independent core but is extensible to support other languages and platforms
- Defines a single unified terminology for knowledge discovery of existing software assets
- Is represented using UML class diagrams
- Utilizes an XMI interchange format for vendors to import and export tool-specific metadata out of and into their toolsets
- Supports every type of platform and language environment through various means and describes the physical structure of software architectures and the logical structures where appropriate
- Facilitates tracing artifacts from a logical structure back to physical artifacts

Figure 3.2 identifies each of the varying views of the existing IT architecture represented by the KDM. For example, Figure 3.2 shows how KDM depicts an “execution” perspective as well as a “build” perspective of applications. The execution perspective exposes software artifacts from a runtime view to depict how the system flows, how transactions are triggered, and how execution artifacts interact. The build view, on the other hand, depicts system artifacts from a source, executable, and library viewpoint. Other perspectives include design, conceptual, data, and scenario views.

The KDM offers a vehicle for modernization teams, armed with standards-based technologies to establish and maintain a complete and comprehensive view of application and data architectures. This ability provides an enhanced capability to understand, refactor, and transform these environments.

KDM supports a variety of languages, database structures, middleware, teleprocessing monitors, and platforms in a common repository. Where a certain technology cannot be represented, a tool vendor can extend KDM to include language- or platform-specific entities and relationships. A special mention should be made of the portion of the KDM called “Micro KDM.” The Micro KDM allows vendors to exchange statement-level representations of software programs.

KDM tool compliance can be achieved by a vendor tool at multiple levels as shown in Figure 3.2. One important point regarding compliance with KDM and other ADM standards is that for tools to be KDM compliant, they do not
need to apply any internal changes to their products or to tool repositories. The KDM is an external representation that can be imported into and exported from using the agreed-upon standard XMI formats. Compliance, therefore, is a matter of mapping internal tool repository representations to the KDM format for purposes of importing or exporting metadata to or from another toolset. This is also the case for other ADM standards.

KDM was adopted as an OMG standard in 2007 and has undergone revisions since that time. KDM is also slated to be adopted as an ISO standard in early 2010. ISO works with the OMG on various standards. KDM, along with ASTM as described in the next section, is the foundation for various other modernization standards. When an organization considers various modernization technologies or tools, they should ask about KDM compliance.

### ABSTRACT SYNTAX TREE METAMODEL

This ASTM was established to represent software at a very granular level of procedural logic, data definition, and workflow composition. ASTM can provide this granular level of information to KDM to augment the KDM view of a system. As a standard, ASTM can stand alone and supports tools geared at the complete, functionally equivalent refactoring and transformation of a system from one platform and language environment to a target platform and language environment.

Some background on the concept of the abstract syntax tree (AST) helps clarify the purpose of the ASTM. ASTs are models of software that represent software artifacts using data structures that represent the types of language constructs,
their compositional relationships to other language constructs, and a set of direct and derived properties associated with each language construct. The AST is derived by analyzing software artifacts and provides a way to create a representation of those software artifacts.

An AST is an extensible, formal representation of the syntactical structure of software that is amenable to formal analysis techniques. It is possible to traverse the AST and reconstruct the “surface syntax” of the system or reconstitute it in textual form from the abstract structures. While the use of AST structures for the abstract representation of the structure of software has become an accepted practice for modeling software, the format of AST structures and the mechanisms for representation and interchange of AST models were not standardized prior to the formalization of the ASTM.

AST interchange, via the ASTM, facilitates exchanging software models in standard formats among tools. The ability to freely exchange software models between tools provides organizations with the ability to use advanced model-driven tools for software analysis and modernization. The ASTM has these attributes:

- ASTM is language and platform independent, but can be extended as needed.
- ASTM uses XMI formats for tool-based metadata exchange.
- Generic Abstract Syntax Tree Metamodel (GASTM) represents a generic set of language modeling elements common across numerous languages. Language Specific Abstract Syntax Tree Metamodel (SASTM) represents particular languages such as Ada, C, FORTRAN, and Java.
- Proprietary Abstract Syntax Tree Metamodel (PASTM) expresses ASTs for languages such as Ada, C, COBOL, etc., modeled in formats inconsistent with MOF, the GSATM, or SASTM.

Figure 3.3 represents the structure of the ASTM, including the SASTM, GASTM, and PASTM.

The ASTM standard, which was officially adopted as a standard in 2009, will continue to evolve as it expands to support more languages and environments. It will also support the evolution of other ADM standards as they change and mature.

**PATTERN RECOGNITION**

The Pattern Recognition Metamodel (PRM) is a work-in-progress effort by the ADM Task Force and is initially focused on structural, architectural, and design patterns and anti-patterns that provide essential information about software. Patterns and anti-patterns provide qualitative information about existing software systems, but this concept can also be extended into the business domain of standards because the PRM is being established as a generic standard.
Patterns and anti-patterns are based on the analysis and abstraction of the metadata initially represented by the KDM, ASTM, or alternative sources where applicable. The PRM standard will include

- Pattern libraries describing structural, architectural, and design patterns that define opportunities to retain or transform aspects of the existing IT architecture
- Anti-pattern libraries describing structural, architectural, and design anti-patterns that signal certain refactoring and/or transformation scenarios
- A metamodel capable of representing the patterns and anti-patterns defined within the patterns library

Initial use of the PRM will be in the area of software quality and systems assurance. In either case, a given pattern or anti-pattern may signal that a system is of high or low quality. Patterns also signal that the software or business structures have desirable qualities that should be retained or reused in target architectures. For example, if an application contains viable, reusable business logic that is identifiable, segregated, and reusable, planning teams may determine that this system is an ideal candidate to be directly transformed into a UML-based environment with little or no refactoring.
A second patterns example involves highly rationalized, clearly defined data definitions, shared across several applications, that could serve as a basis for extracting and deriving a baseline data model that reflects the business data required to move to the target data architecture. In this example, little or no refactoring of redundant, inconsistent data definitions is required and the existing data definitions can be directly transformed using data transformation and modeling technologies and techniques. This pattern may also be reflected in metrics that signal high-quality software.

Anti-patterns, which are admittedly more likely to arise in existing software systems, include structural, architectural, or design-based patterns that are problematic or minimally questionable from a variety of qualitative perspectives. Anti-patterns are signs of poor software quality. Examples of anti-patterns include:

- Cloned or redundant data structures that are inconsistently defined across a system or multiple systems
- Cloned or redundant business logic, distributed across a system or systems
- Syntactically inactive source code (can never be executed)
- Semantically inactive source code (cannot be activated based on conditional logic)
- Diagnostically defective logic that violates modular programming standards and likely works in unanticipated ways including:
  - Unintended recursive structures
  - Logic that runs past the end of the program
  - Logic that exits from a routine prematurely, leaving active addresses that could trigger an unintended return to another address
  - Buffer overflows or data area overlays that could wipe out program data unintentionally
- Business logic that is highly intertwined with implementation-dependent logic
- Highly coupled data access patterns that provide input to target data architecture redesign efforts or performance tuning of existing structures
- System workflow and execution patterns that expose portions of business process flows

Note that identification of a given pattern or anti-pattern could result in triggering multiple actions. For example, highly coupled data access patterns could trigger a performance tuning effort in the near-term and a data architecture redesign effort over the long term. The pattern and anti-pattern libraries will likely continue to grow over time.

A major benefit of the PRM is that it provides the vehicle through its pattern and anti-pattern libraries to standardize terms for problematic software and
business issues inherent in IT and business architectures. It is often difficult to articulate if a system is in good shape or bad shape, or whether modernization options could make a difference. In addition, if a system does have useful and valid functional logic, which could be determined by recognizing a pattern that links software logic occurrences to business capabilities, then reuse of that logic could be valuable under certain modernization scenarios.

Structural views, as represented within the KDM or ASTM, provide some insights into modernization requirements. The KDM and ASTM as a foundation for deriving patterns and anti-patterns, however, can provide much more significant insights into refactoring and transforming IT architecture. When KDM is mapped to certain business models, these insights into potential transformation opportunities expand even further.

Finally, patterns and anti-patterns defined in a given PRM patterns library, whether related to IT architecture, business architecture or a collective view of both architectures, can be described in several ways. One way to communicate these patterns is through a direct mapping to the Structured Metrics Metamodel (SMM) standard that the ADM Task Force has developed.

**STRUCTURED METRICS METAMODEL**

One of the best ways to quantify various aspects of existing IT architectures is the use of metrics. In our experience, what organizations do not know about their application and data architectures could fill volumes. Executives and most managers are not very interested, however, in looking at reports or diagrams of their IT environment. Yet these individuals are the ones who must understand, buy into, and fund IT projects. They also have the power to incorporate and drive modernization disciplines into traditional IT projects. The use of metrics can provide project teams and management with the hard facts needed to gain executive attention, build executive support, and procure funding for modernization projects.

Historically, software metrics have focused on a very narrow slice of the IT architecture. For example, the McCabe Cyclamate Complexity metric, which is useful when attempting to determine program testing and maintenance difficulty, provides somewhat limited strategic value to modernization architects and planning teams.

A second metric that has been emphasized to a fault is the amount of syntactically inactive source code within a system. This overemphasis on this simple construct is, in our opinion, due to the limited imaginations of tool providers and service teams. Most modernization tools can find and handle this type of dead code whereas other constructs are harder to analyze, identify, and address. This has led tool builders to focus on this single metric, which in turn
has taken the focus off of a much larger set of issues that need to be addressed including system level anti-patterns; poor design; and fragmented, redundant architectures.

At a program or module level, a more useful metric would be the number of occurrences and the whereabouts of semantically inactive code, which indicates that there is convoluted logic embedded within an application that should not be transformed wholesale into target architectures but should be selectively extracted, analyzed, and either discarded or reused accordingly.

Consider some of the other metric categories that have been ignored but provide valuable insights into architecture and planning teams. These include basic environmental counts such as job steps, system-wide data definition replication, program-to-program nesting levels, percentage of batch versus online interfaces, number and percentage of business capabilities in a current system that map to a target architecture, or the mapping of data usage between the current and target architecture. These are just a few examples of useful metrics that are mostly ignored in practice during assessment projects. This gap can be attributed to the lack of agreed-upon foundation “metrics library” to support quality analysis, modernization assessment, planning, and deployment.

Using metrics such as these, a wide variety of modernization scenarios can be more effectively planned and estimated, staffed accordingly, and deployed with a higher degree of confidence. Software metrics are required to assess the qualitative and quantitative aspects of systems and serve as a foundation for scoping and estimating modernization projects. Metrics can also provide a much broader set of benefits. The SMM, which was officially adopted in 2009, is the vehicle for supporting software quality, systems assurance, green computing, and business architecture. The SMM enables interchange of metric metadata and helps define representative metrics as metadata or metadata attributes within various models. The SMM standard is based on:

- A metrics metamodel that enables the interchange of measures across a wide spectrum of domains
- Metric libraries that support selective domain categories including modernization, systems assurance, business domain analysis, and other areas

The metamodel is in place and will likely be adjusted as it is deployed into the marketplace. Similarly, the metrics libraries will continue to evolve based on the usage of the SMM standard.

Figure 3.4 depicts a summary-level view of the SMM where a measure is the thing numerically identified and a measurement is the result of the process to quantify a given measure. An observation represents contextual information such as the time of the measurement and the identification of the measurement
tool. This structure provides a way for various tools that produce or utilize metrics to import and export those metrics across a wide variety of domains. These domains are defined by the metric libraries to be associated with the SMM over time.

The SMM has established an initial framework of metric categories to be incorporated into the modernization metrics library. These categories support software quality analysis and modernization planning. They include

- Environmental Metrics (e.g., number of screens, programs, lines of code.)
- Data Definition Metrics (e.g., number of data groups, overlapping data groups, unused data elements.)
- Program Process Metrics (e.g., Halstead, McCabe.)
- Architecture Metrics (e.g., number of batch workflows, number of discreet online workflows, and depth of hierarchical database hierarchies.)
- Functional Metrics (e.g., capabilities supported by a given system, business data as a percentage of all data, capabilities in current system that map to capabilities in target architecture.)
- Quality/Reliability Metrics (e.g., failures per day, meantime to failure, meantime to repair.)
- Performance Metrics (e.g., average batch window clock time, average online response time.)
- Security/Vulnerability (e.g., breaches per day, vulnerability points.)
These metric categories represent a comprehensive set of proven industry metrics that provide analysis, planning, and deployment support for modernization projects. These are subset examples of one view from a modernization metrics library that is posted on the Internet. Additional metrics libraries will continue to emerge for software assurance, development standards, business modeling, and other domains that arise within the standards community. The SMM metamodel was designed to be flexible enough to provide a foundation for these and future metric libraries.

Views of the existing architecture, as defined within KDM or ASTM models, provide the baseline for a foundational set of ADM metrics. KDM provides a good source for a wide variety of structural metrics. The previously identified environmental metrics category is a good example of the kind of metrics that can be easily derived from the KDM. Other standards will ultimately serve as an additional source of potential metrics. The ADM Pattern Recognition initiative at OMG, in particular, will provide another level of metrics that support more qualitative analysis of existing IT environments. Several of the previously discussed metric categories, such as the architectural metrics, reflect patterns and anti-patterns. The SMM will serve as the foundation for these new standards as they evolve.

To raise the degree of sophistication and support that metrics can provide to modernization projects, we must seek to derive metrics not just from one standard, but from a combination of standards. For example, one previously referenced metric category was called functional metrics. Within this category, the ability to derive the number of capabilities supported by existing systems that need to be retained within the design of the target architecture requires metamodel mapping between ADM and business domain standards. Business metamodel mappings are being considered through the efforts of the OMG Business Architecture Working Group.

We anticipate that the SMM will be used to support business domain and other standards through the use of domain-specific metrics libraries for business architecture, green computing, SOA, and other topical areas. In addition, the OMG ADM Task Force will work to ensure that pattern and anti-pattern libraries defined in conjunction with the PRM will be synchronized with corresponding SMM metric libraries.

Finally, the SMM metrics can be plugged into various project-estimating models for various modernization projects, which offer project planning teams powerful, quantitative and qualitative measures that can be used to establish project timelines and funding. For example, data definition redundancy metrics would facilitate estimating efforts for a scenario to migrate from flat file structures to a relational database. These estimating models for modernization scenarios will continue to evolve as they are deployed internally and in conjunction with modernization tool and service providers.
ADM VISUALIZATION

KDM, ASTM, PRM, and SMM establish an excellent foundation of information about existing IT architectures. Yet the means of visualizing this vast and potentially powerful cross-section of metadata is not addressed in any of these standards. Rather, individual tool vendors define what is to be visualized and how it is to be presented. As a result, there is no consistency in how one tool versus another tool depicts the same information to the users of those tools. One appropriate definition of “visualization” as it applies to the concept of ADM metadata is as follows.

Visualization is the process of representing abstract business or scientific data as images that can aid in understanding the meaning of the data.

An ADM Visualization standard would allow vendors to exchange metadata geared at enriching the visualization of complex, yet essential, relationships and derived analysis buried inside of various ADM metamodel views. Visualization provides the link for modernization analysts between the four baseline standards (i.e. KDM, ASTM, PRM, and SMM), refactoring, and transformation. With no set way to visualize existing IT architecture, modernization analysts could potentially ignore or sidestep issues that need to be addressed in existing systems. A lack of visualization definitions and consistency ultimately leads to project teams ignoring or neglecting the need to address this particular issue, buried within existing systems. This is a common practice within modernization projects.

Ignoring visualization of existing systems metadata handicaps analysis and planning efforts associated with modernization initiatives by propagating inconsistent, misleading, or conflicting views of this important information. ADM Visualization addresses this issue by providing a consistent set of visualizations across platforms, languages, and tools. ADM Visualization, which is only at the discussion stages as of this writing, would support various artifact representations that either exist or will emerge from ADM and related standards. These include, but are not limited to, the following concepts:

- System-to-system interface mappings
- General architectural artifact mappings between various KDM views
- Data-store-to-data-store mappings
- Data redundancy visualizations
- Batch and online workflows
- Data definition mappings at various levels
- Functional decompositions and mappings
- Call and related invocation structures
- Current to target technical, data, and application architecture mappings
- Other mappings and views as may be applicable
ADM Visualization plays an important role in bridging the gap between the current set of understanding-based ADM standards and the refactoring and transformation standards discussed in the following sections. Early research into visualization of existing architectures suggests that there are various 3D and dynamic representations that will ultimately provide new and unique ways to view not only as-is IT architectures but also business architectures.

ADM REFACTORING

As discussed in Chapter 1, “refactoring is any change to a computer program’s code which improves its readability or simplifies its structure without changing its results.” Recognizing refactoring as a unique collection of tool-enabled disciplines isolates the underlying “model transformation” requirements without adding the complexities of architectural transformation to the equation. Model-to-model transformations may be specified within OMG standards.

We should provide some background on the concepts of model transformation to help position the refactoring and transformation discussions that follow. Johan den Haan defines an eight-step approach that includes concepts such as transformation rules, rule application scope, source and target relationships, and other factors. OMG has expressed approaches for this but these transformations generally define moving from an MDA platform-independent model to a platform-dependent model, which is inherently part of the MDA forward engineering concepts. Forward engineering has applied this concept within the standards community; modernization can similarly apply it too.

For example, model transformation in the case of refactoring would involve source-model-to-target-model transformations where the source would involve poorly rationalized data definitions (i.e., highly redundant, inconsistent) and the target would involve rationalized data definitions. Taking this into a standards framework provides modernization teams the ability to use one tool to gather relevant application metadata and, if required, apply a second tool to refactor those representations. It also supports the idea of refactoring across language and platform domains. There are several reasons for isolating the refactoring standard model transformations from the transformation standard model transformations:

1. Refactoring is a bounded, well-defined set of disciplines that are well understood and provide value without radical architecture changes.
2. The complexity of building all modernization options into a single package is impractical based on the diversity and complexity of the transformations involved.
3. While some applications will be transformed to new target architectures, many more will merely require incremental improvements, retaining their application and data architectural integrity.

4. Addressing refactoring as a separate set of disciplines creates options for project teams and tool vendors as to how and when they wish to refactor applications versus transform those applications to new target architectures.

5. Treating refactoring as a separate discipline will reduce the tendency of modernization teams to migrate flawed architectures into new target environments.

Refactoring can be enabled through metamodel transformations. The process involves defining the model-to-model transformations that would occur to certain KDM, ASTM, or other ADM metamodels as necessary to achieve different refactoring tasks. Consider the following points:

- Populated KDM and ASTM models represent views of the existing IT architecture.
- Programs and systems represented by the populated models are, as a general rule, typically riddled with poor structure, diagnostics, and convoluted logic.
- Flaws include a high degree of inconsistency and redundancy, monolithic (vs. modular) programs, spurious logic, or poorly implemented solutions and other factors too long to list here.
- Structural, diagnostic, and design flaws should be corrected if software or the essence of that software is to survive in current or target architectures.
- Moving flawed systems directly into MDAs without refactoring will propagate these issues into those environments.
- A series of transformational passes between views of these populated models would have the capacity to produce refactored results that could generate refactored applications.
- Model transformation-based refactoring provides a powerful, standardized approach to extending the life of countless millions or billions of lines of software while establishing a baseline for subsequent transformation.

The ADM Refactoring standard is still in the discussion stages within the OMG ADM Task Force.

**ADM TRANSFORMATION**

The ADM Transformation standard defines model transformations between the KDM, ASTM, and target architecture paradigms. These target models include,
for example, UML, SOA, and other potential representations. Deployment of this standard will complete ADM Task Force efforts in providing a modernization bridge between existing systems and target architectures.

ADM Transformation defines an encoding set of transformational mappings between other ADM standard metamodels and various target architectures. Model transformations, as it applies to the transformation stage of modernization, is already being pursued through research work and related project initiatives. One such effort explored the transformation of domain-specific models to MDA. Even though this approach was limited to the transformation of Web engineering paradigms to MDA, the general applicability of the concept can be expanded into a more generalized form.

For example, KDM is a language- and platform-independent representation of existing IT architectures. This means that several language environments and platform-specific collections of artifacts may be represented within the KDM. Assuming that the target architecture is MDA, using various UML representations, a series of model transformations would be defined to populate UML-based target architecture representations.

Work along these lines has already been pursued by a consortium of organizations from Europe (MODISCO) that moved C and Java applications into a KDM representation, which was transformed into UML2 representations. This was called the MODELPLEX scenario. These types of efforts are becoming more common; however, transforming existing software environments into target architectures — in the absence of refactoring — will merely result in moving problematic architecture and design constructs into new target paradigms, which will propagate serious architectural problems into UML and other MDA target architectures.

The ADM Task Force is responsible for ensuring that these models are synchronized in such a way that vendors can utilize them in a coordinated fashion. This approach will be beneficial to the organizations pursuing modernization projects. The ADM Transformation standard has no set target date.

SUMMARY

A cross-section of software tool vendors and service providers has emerged to enable the modernization of existing systems. Unfortunately, users and vendors have been working in isolation, often reinventing the wheel. Standardization of modernization metamodels, along with related ADM patterns and metrics libraries, provide vendors, service providers, and the enterprises deploying modernization solutions with a clear roadmap for delivering projects. Standardization of the software modernization process will help businesses reduce the risk of undertaking software improvement initiatives by lessening
the time, risk, and cost of software modernizations, improving the effectiveness of modernization tools and extending the ROI on software development projects.

REFERENCES