

Model-Driven Engineering: Automatic Code Generation and Beyond

John Klein
Harry Levinson
Jay Marchetti

March 2015

TECHNICAL REPORT
CMU/SEI-2015-TN-005

Software Solutions Division

<http://www.sei.cmu.edu>



Copyright 2015 Carnegie Mellon University

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Department of Defense.

References herein to any specific commercial product, process, or service by trade name, trade mark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by Carnegie Mellon University or its Software Engineering Institute.

This report was prepared for the
SEI Administrative Agent
AFLCMC/PZM
20 Schilling Circle, Bldg 1305, 3rd floor
Hanscom AFB, MA 01731-2125

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

This material has been approved for public release and unlimited distribution except as restricted below.

Internal use:* Permission to reproduce this material and to prepare derivative works from this material for internal use is granted, provided the copyright and "No Warranty" statements are included with all reproductions and derivative works.

External use:* This material may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other external and/or commercial use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

* These restrictions do not apply to U.S. government entities.

Carnegie Mellon® is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

DM-0001604

Table of Contents

Acknowledgments	vii
Executive Summary	ix
Abstract	xi
1 Introduction	1
1.1 New Opportunities	1
1.2 Software Acquisition and MDE Challenges	1
1.3 MDE Tool and Process Risks	2
1.4 Structure of This Report	3
2 Overview of Model-Driven Software Engineering	4
2.1 Model-Driven Engineering and the Software Development Lifecycle	4
2.2 MDE Is Tools <i>and</i> Methods	6
2.3 MDE Presents Both Opportunity and Challenges	7
3 Acquisition Strategy Implications	8
3.1 Artifacts, Data Rights, and Licenses	8
3.2 Design Review Scope and Timing	8
3.3 Impact on Program Risk	9
3.3.1 Maintainability (Part of Sustainment)	9
3.3.2 Certification (Including Cybersecurity, Safety, and Airworthiness)	10
3.3.3 Cybersecurity Assurance	11
3.3.4 Runtime Portability (Part of Sustainment and Open System Architecture)	11
3.3.5 Runtime Performance	12
3.3.6 Usability of Generated User Interfaces	12
4 Selecting and Evaluating MDE Tools	13
4.1 Planning the Evaluation	13
4.2 Establishing the Criteria	14
4.3 Collecting the Data	15
4.4 Analyzing the Data	15
4.5 Interpreting Vendor Responses to the Data Collection Questionnaire	15
4.5.1 Questionnaire Part 1 – Demographics	15
4.5.2 Questionnaire Part 2 – Licensing and Delivery	16
4.5.3 Questionnaire Part 3 – Modeling	17
4.5.4 Questionnaire Part 4 – Target Environment	18
5 Conclusions	20
Appendix A Tool Evaluation Criteria	21
Appendix B MDE Tool Vendor Self-Assessment Instrument	28
References	34

List of Figures

Figure 1:	MDE Uses Models as Primary Artifacts (After diagram by Rumpe [Brambilla 2012a])	5
Figure 2:	The PECA Process [Comella-Dorda 2004]	13

List of Tables

Table 1:	Evaluation Criteria – Product Engineering Risk Area	21
Table 2:	Evaluation Criteria – Development Environment Risk Area	24
Table 3:	Evaluation Criteria – Program Constraints Risk Area	26
Table 4:	Self-Assessment Instrument for MDE Tools	29

Acknowledgments

We thank all of the vendors who participated in the pilot application of our questionnaire. In particular, we thank representatives of Integranova (<http://www.integranova.com>) for their responses and comments.

Executive Summary

Acquisition executives, in domains ranging from modernizing legacy business systems to developing real-time communications systems, often must deal with the following challenge:

Vendors claim that by using model-driven engineering (MDE) tools, they can generate software code automatically and achieve extremely high developer productivity. Are these claims true?

The simple answer is, yes, the state of the practice can achieve productivity rates of thousands of function points and millions of lines of code per person-month using MDE tools for automatic code generation. But the complicated reality is that MDE consists of more than code generation tools; it is a software engineering approach that can affect the entire lifecycle from requirements gathering through sustainment. An acquirer must consider this approach in the context of a particular system acquisition. Aligning MDE methods and tools with the system acquisition strategy can improve system quality, reduce time to field, and reduce sustainment cost. On the other hand, when MDE methods and tools do not align with the acquisition strategy, using them can result in increased risk and cost in development and sustainment.

This report focuses on the application of MDE tools for automatic code generation in the context of the full system lifecycle from development to sustainment. Acquisition programs in government or large commercial enterprises have unique concerns. These acquirers have the challenge of selecting contractors to develop their systems. Then, the tools and processes selected by the contractors and developers have direct impact on the software quality concerns of the acquirer, who often has little influence on the selection of these tools and processes. Thus, the acquirer's process for selecting performers must include evaluating both the development team and the development methodology and tools in the context of the system acquisition strategy. This report provides guidance for selecting, analyzing, and evaluating MDE tools when acquiring systems built using these software development tools and processes.

In Section 2, we first define some terminology of MDE methods and tools and then explain the basic approach of using MDE for automatic code generation. While this report focuses primarily on MDE tools—specifically, tools for automatic code generation—we emphasize that MDE consists of both tools *and* methods. MDE tools are based on engineering methods and techniques and provide automated assistance to engineers using those techniques. An organization should conduct tool categorization and assessment only after it has gained an understanding of software engineering methods and has determined which methods it will adopt. Acquirers are concerned with the full system lifecycle, and they need confidence that the development methods will enable the system to meet the functional, quality, cost, and schedule objectives for both development and sustainment throughout the life of the system. The challenge of selection, evaluation, and use of MDE tools is to support the execution of one or more methodologies from beginning to end. The decision to adopt MDE methods for automatic code generation and the selection of appropriate supporting tools cannot be made in isolation.

Section 3 discusses how the use of MDE tools for automatic code generation can affect acquisition strategy. An acquisition strategy includes identifying the artifacts and data rights to acquire and the artifacts to evaluate at each program decision point. An MDE approach requires securing

the necessary data rights and licensing for the tools, models, generated code, runtime libraries, frameworks, and other supporting software; otherwise, sustainment and evolution become more difficult. The acquirer must also review and evaluate appropriate artifacts at the right time in the acquisition cycle; however, acquirers using MDE approaches may need to expand the evaluation scope and criteria to account for using the model not just to represent the architecture for communication among stakeholders but also to generate the executing software. The acquisition strategy also defines the approach to managing program risks, including risk identification and mitigation. MDE approaches can introduce new program risks, including maintainability, certification, cybersecurity assurance, runtime portability, runtime performance, and interface usability risks.

Section 4 provides guidance on selecting and evaluating MDE tools in the context of risks to an organization's system acquisition effort. We use the PECA (Plan, Establish, Collect, Analyze) method, as described by Comella-Dorda and colleagues [Comella-Dorda 2004]. We explain how to use the PECA method to set up an MDE tool evaluation that will consider the MDE benefits and risks described in Section 3. To this end, we discuss a vendor self-assessment questionnaire that we piloted with two MDE tool vendors. We also provide examples of how to interpret vendor responses to the data collection questionnaire, illustrated with responses received during a pilot application of the questionnaire with several tool vendors. The questionnaire will help acquirers collect information about a vendor's demographics and approaches to licensing and delivery of products and services, modeling representations, and transitioning products and services to the program's target environment. Two appendices to this report provide the questionnaire and a risk-driven framework for evaluating tools with cross-references to the vendor questionnaire that relate to acquisition concerns.

Section 5 offers conclusions and additional areas of investigation. While MDE promises to improve the efficiency of developing, delivering, and sustaining software, these benefits come with some challenges: An acquirer must select an MDE process appropriate to its system requirements, domain characteristics, and stakeholders' needs. The acquirer must also evaluate contractors' and developers' selection of the appropriate tool set to implement the development, integration, and testing of the MDE solution. In addition, acquirers must address sustainment concerns from the beginning of the acquisition process. Research on the promise of MDE is ongoing. Additional areas of investigation include fully mapping the acquisition strategy to areas and questions that are relevant to using an MDE methodology, using domain-specific languages as opposed to the standards-based Unified Modeling Language, integrating multiple models, measuring the economics of MDE, and using MDE to achieve correctness by construction.

Abstract

Increasing consideration of model-driven engineering (MDE) tools for software development efforts means that acquisition executives must more often deal with the following challenge: Vendors claim that by using MDE tools, they can generate software code automatically and achieve high developer productivity. However, MDE consists of more than code generation tools; it is also a software engineering approach that can affect the entire lifecycle of a system from requirements gathering through sustainment. This report focuses on the application of MDE tools for automatic code generation when acquiring systems built using these software development tools and processes. The report defines some terminology used by MDE tools and methods, emphasizing that MDE consists of both tools *and* methods that must align with overall acquisition strategy. Next, it discusses how the use of MDE for automatic code generation affects acquisition strategy and introduces new risks to the program. It then offers guidance on selecting, analyzing, and evaluating MDE tools in the context of risks to an organization's acquisition effort throughout the system lifecycle. Appendices provide a questionnaire that an organization can use to gather information about vendor tools along with criteria for evaluating tools mapped to the questionnaire that relate to acquisition concerns.

1 Introduction

1.1 New Opportunities

Quite often, acquisition executives, in domains ranging from modernizing legacy business systems to developing real-time communications systems, have to deal with the following challenge:

Vendors claim that by using model-driven engineering (MDE) tools, they can generate software code automatically and achieve extremely high developer productivity. Are these claims true?

The simple answer might be, “yes, the state of the practice can achieve productivity rates of thousands of function points and millions of lines of code per person-month using MDE tools for automatic code generation.” The complicated reality is that MDE consists of more than code generation tools; it is a software engineering approach that can impact the entire lifecycle from requirements gathering through sustainment. While one can make broad generalizations about these methods and tools, it is more useful to consider them in the context of a particular system acquisition. Aligning MDE methods and tool capabilities with the system acquisition strategy can improve system quality, reduce time to field, and reduce sustainment cost. On the other hand, when MDE methods and tools do not align with the acquisition strategy, using them can result in increased risk and cost in development and sustainment.

We have focused this report on the application of MDE tools for automatic code generation, but we will position this activity in the context of the full system lifecycle, from concept development through sustainment.

1.2 Software Acquisition and MDE Challenges

Firms such as Gartner, Forrester, and IDC assess and analyze MDE technology for commercial IT developers and providers.¹ In contrast, this report focuses on the unique concerns that arise in acquisition contexts such as the Department of Defense (DoD), other government organizations, or large commercial enterprises. The tools and processes selected by the developer and contractor have direct impact on the software quality concerns of the acquirer, but often the acquirer has little direct influence on the selection of these tools and processes. This report highlights the challenges and provides guidance for selecting, analyzing, and evaluating MDE tools when acquiring systems built using these software development tools and processes.

MDE tools for automatic code generation cannot be considered in isolation: In all of software engineering, there is a tight coupling between the *system domain* (e.g., business system, command and control system, or avionics system), the *methods* used throughout the system lifecycle, and the *tools* used to support the chosen methods. Furthermore, acquirers such as the government have the challenge of selecting contractors to develop their systems. This selection process includes evaluating the development team along with the development methodology and tools in the context of the system acquisition strategy.

This coupling leads to acquisition evaluation questions such as the following:

¹ Access analysis of MDE at <http://www.gartner.com>, <http://www.forrester.com>, and <http://www.idc.com>.

- Do the engineering process and associated development tools match the desired acquisition strategy?
- Do the tools support the developer's software development methodology?
- Are the code generation tools capable of integrating with other development and management tools to support measurement and monitoring of development progress?
- Will the selected development methodology with its associated tools be available and compatible for the expected lifecycle of the system?

These questions might apply to an acquisition that does not include MDE for automatic code generation. But because MDE development tools have a larger influence on system qualities (e.g., performance, security, and modifiability) than traditional software development tools such as compilers, the scope of the acquisition evaluation must be adjusted to include attention to tooling.

While the categorization, assessment, and selection of methods and tools are related, the issues are sufficiently different that they should be treated separately. Tool assessment and selection should be conducted by an organization only after it has gained an understanding of software engineering methods and has decided which methods it will adopt [Firth 1987a].

Finally, the artifacts produced by an MDE software development process are different from those of a traditional software development process. Thus, the acquisition strategy and plan must identify the appropriate artifacts and address the timing and subject of the artifact evaluations.

1.3 MDE Tool and Process Risks

The use of MDE technology for automatic code generation impacts the quality attributes of the software produced. Fundamentally, MDE tools should be treated as commercial, off-the-shelf (COTS) products, and they bring many of the same risks. We provide three examples here and analyze the effect of MDE on these and other quality attributes more broadly later in this report.

- **Maintainability:** Sustainment and evolution projects using MDE models and automatic code generation may be more efficient and have a shorter and less expensive development cycle, compared to traditional approaches. However, using them creates a dependency on the MDE tools to manipulate the model and generate code, and introduces risks associated with the continued availability of the tools and compatibility of new releases of the tools. Transitioning away from an MDE approach to perform sustainment and evolution by directly changing the automatically generated code can be time consuming and expensive, because the automatically generated code is usually not structured for human readability and understanding.
- **Certification:** Certification approaches that rely on source code inspection or analysis may have difficulty operating on automatically generated code, which is not structured for human readability and is often larger in volume (lines of code) than manually generated codebases. Furthermore, the automatically generated code may depend on runtime libraries developed by the tool vendor, for which there is no source code or documentation available. Finally, the automatically generated code and dependent libraries introduce new supply chain vulnerabilities that must be considered.

Portability to new hardware and software infrastructure: Certain MDE automatic code generation approaches, such as those based on the Object Management Group’s Model-Driven Architecture® (OMG MDA®) standards, promise easy portability to new hardware and infrastructures. However, the software models used in these approaches can be as complex as the code developed in a traditional “code-driven” solution, thus reducing the development and sustainment benefits of MDE. On the other hand, other MDE automatic code generation approaches use simpler models that more closely match the target system domain, offering fast development and evolution at the expense of runtime portability. Failure to identify and mitigate the risks introduced when using MDE for automatic code generation impacts initial development cost, schedule, and quality and has significant impacts on other lifecycle processes and on long-term sustainment of the system.

1.4 Structure of This Report

Earlier Software Engineering Institute (SEI) reports have addressed the categorization and evaluation of software engineering methods, tools, and products [Firth 1987a, 1987b; Wood 1988; Comella-Dorda 2004]. This report discusses how MDE methods and tools fit into the guidance, taxonomies, and frameworks presented in these reports, which are still generally applicable.

In Section 2, we define the terminology used by MDE methods and tools. Section 3 discusses how the use of MDE for automatic code generation impacts acquisition strategy. Section 4 provides guidance on selecting and evaluating MDE tools in the context of risks to an organization’s system acquisition effort.

2 Overview of Model-Driven Software Engineering

2.1 Model-Driven Engineering and the Software Development Lifecycle

In practice, several terms are used to identify software development approaches that center on models as primary artifacts (in contrast to traditional processes that use source code as the primary artifact). These terms include [Mittal 2013]

- model-driven engineering (MDE)
- model-driven development (MDD)
- model-driven system/software engineering (MDSE)
- model-based engineering (MBE)
- model-based system/software engineering (MBSE)
- model-driven architecture (MDA)

The only one of these terms that has a precise definition is MDA, which is defined by a set of standards promulgated by the OMG [OMG 2003].

In this report, we use the term *MDE* to refer descriptively to a software development approach that treats models as the primary artifacts created and used by software lifecycle processes. The enabling tools and technologies include a broad spectrum of capabilities that may provide value for developers, acquirers, and end users.

The topic is complicated by the different perspectives, motivations, and pathways that are taken in considering and adopting the approach [Selic 2008, Whittle 2014]. For example, MDE can be positioned as a solution to

- platform independence [OMG 2003]
- software reuse [Greenfield 2004]
- creating trustworthy software [Weigert 2006]
- architecture analysis of runtime qualities [Feiler 2012]
- bridging from problem definition to solution synthesis [Pastor 2007]
- implementation of a formal specification [Davies 2014]
- “grand unified theory” of software engineering [Diskin 2012]

Except for the last item, these are all possible motivations for DoD acquisition to consider using MDE.

Finally, the topic is also complicated because the term *model* can be defined in many ways. We will adopt the general definition given by Brambilla and colleagues, who define a model as

a simplified or partial representation of reality, defined in order to accomplish a task or reach an agreement on a topic [Brambilla 2012b]

This definition is similar to that used by the OMG MDA specification [OMG 2003], ISO/IEC/IEEE standard for architecture documentation [ISO 2011], and DoD Architecture Framework [DoD 2010].

As depicted in Figure 1, an MDE approach uses models directly, rather than using source code, as the basis for most software engineering tasks, including

- rapid prototyping
- static analysis (architecture, completeness, correctness)
- dynamic analysis via executable models
- documentation
- refactoring and transformation
- generation of executable code
- automated testing

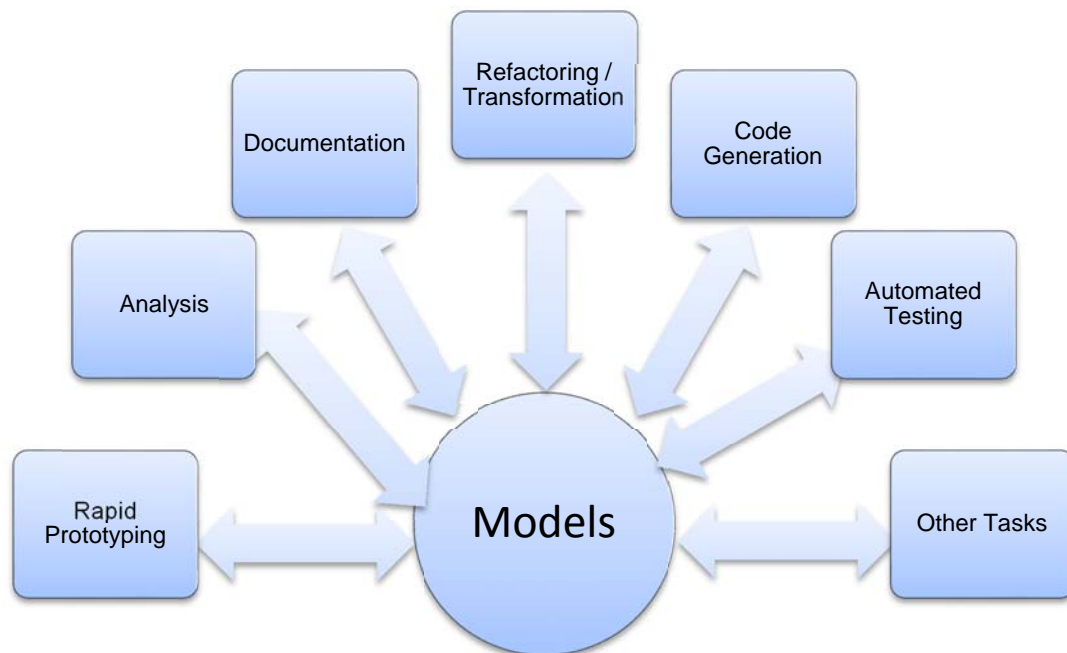


Figure 1: MDE Uses Models as Primary Artifacts (After diagram by Rumpe [Brambilla 2012a])

It is important to recognize that there is no single model of a system; instead, there can (and should) be multiple models. Models provide leverage by enforcing separation of concerns: Each model is a simplified view of the system focused on the concerns related to one of these software engineering tasks, and it omits details about the system that are unnecessary to accomplish that task. For example, a model concerned with analyzing performance might need to represent only the best case and worst case execution times for a calculation. On the other hand, a model concerned with analyzing safety and liveness might need to represent only the external resources needed by the calculation and the order in which the resources are acquired and freed. Finally, a model concerned with generating code for the calculation might represent only the target platform and the exact algorithm to be used. In this way, models help us complete each task more efficiently, often by allowing us to use tools to automate the task.

Since each model addresses only a subset of our concerns, we usually need several models to perform all of the required software engineering tasks. When we have multiple models, we have to maintain consistency between models and the implementation, consistency among the models

across tasks, and consistency among the models throughout the lifecycle to correctly and completely represent our system.

Each model is represented using a modeling language that has a graphical representation, a textual representation, or both. Typical modeling languages include the standards-based Unified Modeling Language (UML) and Architecture Analysis and Design Language (AADL) and proprietary languages such as the Integranova Model Execution System (M.E.S.).² Generally, each MDE tool supports development and analysis of models using only one modeling language (although a tool may import and export other languages), so the selection of tool and modeling language are tightly coupled.

2.2 MDE Is Tools *and* Methods

While this report focuses primarily on MDE tools and, more specifically, tools for automatic code generation, it is important to remember that tools are based on engineering methods and techniques and provide automated assistance to engineers using those techniques. An organization should conduct tool categorization and assessment only after it has gained an understanding of software engineering methods and has determined which methods it will adopt. While the categorization and assessment of methods and tools are related, the issues are sufficiently different that they should be treated separately. The issues that arise in understanding, evaluating, and using particular MDE methods will be discussed in subsequent SEI publications, where they can be addressed in depth.

Acquirers are usually concerned with the full system lifecycle, and they need confidence that the development methods to be used will enable the system to meet the functional, quality, cost, and schedule objectives for both initial development *and* sustainment throughout the life of the system. The challenge of selection, evaluation, and use of MDE tools is to support the execution of one or more methodologies from beginning to end. Some of the concerns include the following:

- How well does the MDE tool support the method in exposing flaws in the requirements and support the creation of appropriate documentation?
- Does the tool support the method in validation of requirements early in the lifecycle?
- How does the tool support the method in description and validation of the quality attributes expected in the system?
- Can the MDE tool support the design, development, implementation, and sustainment of the system throughout its lifecycle?
- Does the MDE tool support the partially complete representations of the system design and implementation to enable appropriate communication between all stakeholders of the system?
- Does the MDE tool support the human parts of the engineering process, including creativity, innovation, ease of implementation, and identification of defects?
- Does the tool allow changes to be easily incorporated through all levels, from design to implementation, into the target system in a disciplined and uncomplicated manner?
- How can the design method best build on previous work done by others, including reuse and salvaging of parts at all stages of representation?

² Access details about these modeling languages at <http://www.uml.org>, <http://www.aadl.info>, and <http://www.integranova.com/integranova-m-e-s>.

The decision to adopt MDE methods for automatic code generation and the selection of appropriate supporting tools cannot be made in isolation. Since there is no single model of a system, and different models may be supported by different tools, the acquirer and developer must be concerned about consistency among the models and interoperability among the tools.

2.3 MDE Presents Both Opportunity and Challenges

MDE approaches to software development promise to improve the efficiency and effectiveness of delivered systems in a number of areas:

- rapid fielding of business systems, including legacy modernization – Several tool vendors target this domain with tools optimized for these types of systems.
- rapid prototyping for technology demonstration – Use of MDE for quick development to demonstrate capability and feasibility avoids the sustainment and evolution challenges outlined in this report.
- portability/open architecture – Using MDE approaches to separate functionality from platform-specific concerns (using OMG MDA standards, for example) can lead to greater portability across platforms.
- correctness by construction – Use of MDE approaches to compose systems from validated components or certified components can reduce cost and risk.

Domain-specific MDE tools provide high leverage in the near term for rapid fielding and rapid prototyping, where agility and time to fielding are high priorities and where portability and other lifecycle concerns are lower priorities. While MDE can be used to achieve the portability and modular substitution of components necessary for successful open architecture approaches, we see MDE as part of a broader architecture and ecosystem solution that is taking shape. Use of MDE to achieve “correctness by construction” is achievable today in limited cases such as small scale or very narrow domains and is an area for further research.

3 Acquisition Strategy Implications

An acquisition strategy includes identifying the artifacts and data rights to acquire and the artifacts to evaluate at each program decision point. The acquisition strategy also defines the approach to managing program risks, including risk identification and mitigation [DAU 2014]. The use of MDE for automatic code generation has several implications for an acquisition strategy.

3.1 Artifacts, Data Rights, and Licenses

Development tooling is usually not a significant concern when acquiring software-intensive systems, but it can be a significant concern when acquiring a system developed using an MDE approach, particularly when using MDE for automatic code generation. In MDE-developed software, the models are the primary development artifacts, embodying the software architecture design and component designs and ultimately driving the automatic code generation. Ideally, all sustainment and evolution of the software will also use an MDE approach, which requires data rights and the necessary licensing for the tools, models, and generated code. When only the automatically generated source code is acquired, without the models used to generate the code, then sustainment and evolution are more difficult because the automatically generated code is usually not structured for human readability and comprehension.

In addition, automatically generated code often requires the use of specific runtime libraries, frameworks, and other supporting software. The licenses, data rights, or both for this additional software must be taken into account as the development and sustainment strategy of the system is developed. The tools used to model and generate the code are another set of COTS products that the acquisition strategy and plan must also include.

Finally, although models may be easier to comprehend than source code, the acquisition strategy should ensure that appropriate accompanying documentation is provided.

3.2 Design Review Scope and Timing

The acquirer must review and evaluate appropriate artifacts at the right time in the acquisition cycle. For example, in an approach using MDE for automatic code generation, the software architecture documentation may consist of a subset of the code generation model, along with accompanying documentation to provide context and design rationale. The software architecture should be evaluated early in the design process, as discussed by Bergey and Jones [Bergey 2013]; however, the evaluation scope and criteria may need to be expanded to account for the use of the model not just to represent the architecture for communication among stakeholders but also to directly generate the executing software. Furthermore, if the tool includes or depends on particular runtime support software (as might be typical for a domain-specific MDE tool, for example), then the scope of evaluation must also include that software.

The architects may have developed other models, in addition to the code generation model, to address other concerns, such as performance analysis. If this is the case, then the architecture evaluation scope must include assessing consistency among the various models.

Finally, reviewers may need to use the MDE tools to view the models—exporting the model into a generic format such as Portable Document Format (PDF) files may not provide the visual resolution and the ability to efficiently navigate through the model. Tool availability and access to the network where the model is stored become issues that the acquirer must address in planning the evaluation.

3.3 Impact on Program Risk

An MDE approach promises such things as automatic code generation, improvement of cost and schedule, reduction of technical risk by enabling early analysis, and the ability to demonstrate capabilities and validate requirements by using executable models or rapid prototypes.

On the other hand, MDE approaches can introduce new risks. This section identifies some risk areas that should be considered in the specific context of a particular acquisition program. Note that there is some overlap in the risks identified: A particular risk may relate to multiple acquisition concerns.

3.3.1 Maintainability (Part of Sustainment)

The use of MDE for automatic code generation introduces a *development time dependency* on the tool chosen to support the process. The chosen tool is used to create and modify the model, which is then processed to generate the code. Unlike traditional source code, which can be created and modified by many different tools, the state of the market for MDE tools is that, in most cases, a model can be edited and modified only by the tool that created it, and changing tools may require rebuilding the entire model. This dependency has several implications:

- The tool vendor must remain in business and continue to support the tool.
- The tool vendor will continue to evolve the tool. Assessing that new versions of the tool are compatible with the program's modeling needs introduces new sustainment costs.
- The tool vendor may evolve the tool in ways that are not compatible with the program's use of the tool. For example, the vendor may change or drop features used in the model, such as file formats or modeling language constructs. This introduces the need to make changes to the model purely to maintain compatibility with the tools.
- The tool may depend on other software (operating system, file system, or other third-party software), which may change in ways that break the tool. This can be mitigated by adopting virtualization technology that makes it possible to take a "snapshot" of the development environment, including many of the dependent software packages, and allow dependable execution of that snapshot at a (much) later date. However, the acquisition plan must account for storing and archiving the snapshots as deliverable artifacts.

The tool generates code that executes at runtime, introducing a *runtime dependency* on the tool. While this is similar to the dependency on the source code compiler in a traditional development approach, the transformation of model to source code (by the MDE tool) can carry more responsibility for functionality and quality than the transformation from source code to object code (by a compiler). In addition, the automatically generated source code may still need to be compiled, so risks associated with the compiler still exist. The runtime dependency on the MDE tool has several implications:

- The tool vendor will continue to evolve the tool. Assessing that the code generated by new versions of the tool is compatible with the program’s functional and quality needs introduces new sustainment costs.
- The tool may generate incorrect or unsuitable code. Finding and repairing this code may be difficult because the automatically generated code is usually not structured for human readability and comprehension. Furthermore, the repairs may need to be redeveloped or, at a minimum, reinserted every time new code is automatically generated.
- The tool vendor may modify the tool so that the code generated by the tool no longer satisfies the program’s requirements. For example, interoperability may be impacted if new versions of the tool generate code with different performance characteristics or a different approach for handling runtime errors. Even if the intention is to make the generated code “better”—such as making the code faster or using a standard error handling approach—modifications to the tool may break the system.
- The automatically generated code may have runtime dependencies on other software (e.g., a target operating system, database, or Java® virtual machine [JVM]). These dependencies may require introducing new packages into the system’s runtime environment, or there may be version conflicts between the version of a package that the automatically generated code depends on and the version that other software in the system depends on.

3.3.2 Certification (Including Cybersecurity, Safety, and Airworthiness)

Certification authorities assess design information (e.g., models and analyses) but typically also rely on testing and source code inspection.

Certification issues when MDE is used for automatic code generation can arise from the regulatory agencies and the regulations and policies that must be satisfied. For example, the DO-187C criteria for “design assurance level A” software requires modified condition/decision coverage (MC/DC) testing. This criteria applies to both the target code and the tools that generate the code. The high cost of certifying the code generation tools falls on the tool vendor, so few vendors have performed this testing. As of the date of this report, only the SCADE tool suite from Esterel Technologies³ has been certified to DO-187B criteria. A related issue is the processes used by the regulatory agency. For example, the FAA inspects both manually generated and automatically generated source code.

Using an MDE approach for automatic code generation has several implications for both initial certification and recertification during sustainment:

- The automatically generated code may not be compatible with the certification authority’s testing procedure. For example, the generated code may not provide the test data insertion and monitoring expected by the certification authority.
- The certification authority may not be able to efficiently or effectively inspect the generated code. The automatically generated code is usually not structured for human readability and comprehension.

³ Access information about the SCADE suite at <http://www.esterel-technologies.com/products/scade-suite>.

- The generated code may depend on runtime libraries supplied by the tool vendor. There may be no documentation for these libraries, and the source code for the libraries may not be available for inspection.
- During sustainment, if the model is modified, it may not be possible to identify which of the generated code modules are affected, preventing incremental recertification.

3.3.3 Cybersecurity Assurance

Previous sections of this report have discussed how the MDE tools for automatic code generation introduce development time and runtime dependencies. These dependencies have several implications for cybersecurity assurance:

- As cybersecurity policy and practices evolve, the tool may not generate compliant code. For example, the tool may not generate code that is compatible with required authorization mechanisms, access control policies, or encryption practices. It may be possible to work around some types of code generation deficiencies by restructuring the model, but other deficiencies may be impossible to overcome.
- The automatic code generation process is a new source of supply chain vulnerability. The program must assure that the generated code is free from malware, “back doors,” and other types of vulnerabilities.
- The automatically generated code may depend on runtime libraries supplied by the tool vendor. These libraries are yet another new source of supply chain vulnerabilities. There may be no documentation for these libraries, and the source code for the libraries may not be available for inspection.
- The automatically generated code may have runtime dependencies on other third-party software (e.g., target operating system, database, or JVM), which must be included in cybersecurity assurance assessments and sustained (i.e., security patches applied) for the life of the system.

3.3.4 Runtime Portability (Part of Sustainment and Open System Architecture)

Portability concerns manifest as a desire to execute the automatically generated code in several environments, each comprising different hardware and software infrastructure, or the concerns may manifest as a desire to change the system’s hardware and software infrastructure over time. The implications are the same in either case:

- The tool may not generate code compatible with the desired hardware and software infrastructure environment.
- The automatically generated code for different environments may deliver different functionality, deliver different levels of a quality attribute, have different runtime dependencies on third-party software, or have some combination of these variations. For example, an MDE tool may generate a graphical user interface for one environment while it generates only a command line interface for another environment. Or throughput and latency can differ across the environments. These differences may be due to intrinsic differences in the capabilities provided by each environment or to technical and business decisions embodied in the tool itself. Such variation results in either reduced flexibility in selecting a target environment or higher sustainment and certification costs because the software is materially different across the environments.

3.3.5 Runtime Performance

The automatically generated code must satisfy the system's runtime throughput, latency, concurrent request processing, and other performance quality requirements. While use of an MDE approach may provide early confidence that these requirements can be met, if one or more of these requirements change, there is a risk that the automatically generated code may not satisfy the new requirement. In some cases, restructuring the model may result in generated code with better performance; in other cases, limitations in the automatically generated code may prevent the system from ever meeting the new requirements.

3.3.6 Usability of Generated User Interfaces

In some system domains, such as business systems, the MDE tools may generate user interfaces as part of the automatically generated code. The generated user interfaces may support functions such as system configuration and administration, system monitoring, and end-user activities.

These generated interfaces have several implications for usability:

- An advantage of using an MDE approach is that user interfaces can be generated and evaluated early in the initial development phase, improving confidence that the first version of the system will satisfy functional and usability requirements.
- However, as requirements evolve, there is a risk that the generated interfaces do not meet the new functional and usability requirements. To satisfy the new requirements, significant architecture changes may be needed, for example, by removing the user interfaces from the automatically generated part of the system and integrating traditionally developed user interface code with the automatically generated code.

4 Selecting and Evaluating MDE Tools

Section 2 described the MDE tool environment in general and some implications of using MDE, focusing on the software development process decisions. Section 3 noted some of the acquisition challenges related to adopting an MDE methodology for automatic code generation. This section assumes that the program has decided to use an MDE approach for automatic code generation and that the next step is to evaluate and select an appropriate MDE tool.

The MDE tool selection process is just like selecting COTS products for any other domain. Various COTS evaluation methods are available: We will use the PECA method, as described in a report by Comella-Dorda and colleagues and shown in Figure 2 [Comella-Dorda 2004, Section 4]. The next four subsections will explain how to use the MDE benefits and risks described previously to set up an MDE tool evaluation, which includes a vendor self-assessment questionnaire that we piloted with two MDE tool vendors.

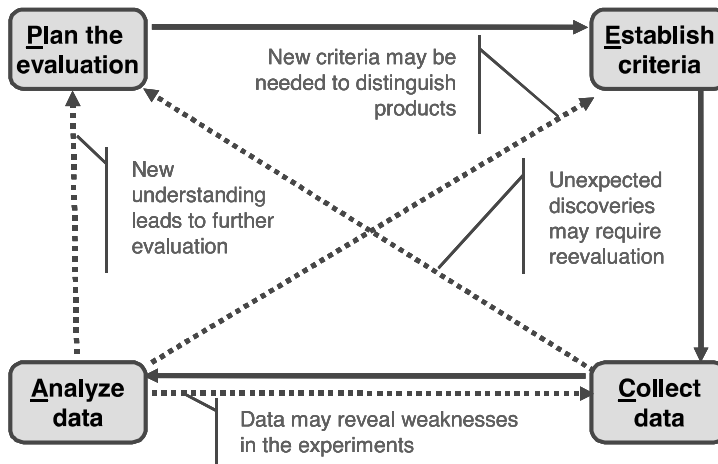


Figure 2: The PECA Process [Comella-Dorda 2004]

4.1 Planning the Evaluation

As described in PECA, planning the evaluation involves creating an evaluation team, defining the team's goals, identifying the stakeholders, and then defining the approach for making the final decision. Considerations for the decision approach include how much effort to apply to the evaluation, a basic strategy for selection, and what filters can be used to limit the number of products to evaluate. For MDE tool selection, acquirers should also consider the following:

- **system requirements:** Before selecting the development methodology and tools to support the methodology, the acquirer needs a clear understanding of the system to be developed. The size of the system and features to be developed will help define the selection criteria.
- **domain:** The type of system and its unique domain characteristics are very important to the decision for the appropriate software development methodology. In addition, determinations about the domain will help define the size and scope of the system to be developed.

- **stakeholders:** The development team's experience and skill set will be critical to the success of the overall system development. These factors will influence the software development methodology selected and determine how much additional training the team will need to use the methodology and the selected supporting tools.

4.2 Establishing the Criteria

The acquirer must establish criteria with which to decide whether a particular tool for automatic code generation is suitable for a specific system acquisition. To establish criteria, we use a risk taxonomy developed by Carr and colleagues to ensure that all relevant acquisition strategy concerns are covered [Carr 1993].

This risk taxonomy has three main sections:

- product engineering, which covers activities that create a system that satisfies the specified requirements and customer expectations. Risks in this area generally arise from requirements that are technically difficult to achieve, inadequate requirements and design analysis, or poor design and implementation quality. These are listed in Table 1 in Appendix A.
- development environment, which includes risks related to the development process and system, management methods, and work environment. These are listed in Table 2 in Appendix A.
- program constraints, which cover risks that arise from factors external to the project. These are listed in Table 3 in Appendix A.

The risk taxonomy provides a checklist to ensure that all potential risks are considered. Most projects will not have risks in *every* area of the risk taxonomy. However, if a program does have a risk in one of the risk areas, then that risk may cause particular acquisition concerns in the context of using MDE for automatic code generation. For example, if requirements stability is a program risk, then in the context of using MDE for automatic code generation, the abilities to operate on partially complete models and to provide tool support for model refactoring become important. Also, tool features for stakeholder communication and integration between the code generation tools and requirements management tools can mitigate some of the impact of this risk. As another example, if design difficulty is a program risk area, then the ability of the tool to automate simulation or analysis can help mitigate that risk by demonstrating design sufficiency early in the development cycle.

To establish selection or evaluation criteria for the MDE automatic code generation tools, the program should first scan the risk taxonomy and identify those risk areas that apply to the project. Each risk creates one or more acquisition concerns, which may refine the program risk or indicate how certain tool features or capabilities might help mitigate the risk.

Note that there is some repetition in the acquisition concerns columns of the tables in Appendix A because multiple program risks may create the same acquisition concern. For example, the availability of training in the capabilities and use of the tool impacts risks areas of development system familiarity for the initial development team, maintainability, management experience, and customer (acquirer) experience.

Finally, the acquisition concerns must be positioned in the specific program context. For example, continuing the training availability example, there will be a specific need to train X number of

people, within Y days of contract award, at a cost of less than Z dollars. As another example, going back to the previous example of requirements stability risk, there will be a specific need to integrate the MDE tool with the particular requirements management tool (and version and platform) that the program will use.

4.3 Collecting the Data

Appendix B contains a questionnaire to be completed by MDE tool vendors, to provide data needed to make the tool selection decision. We piloted an early version of the questionnaire with several tool vendors. The version of the questionnaire presented here includes improvements from that pilot application, along with additional questions suggested by the participating vendors.

The questionnaire structures the data from the vendors about the tools to allow easier analysis and comparisons.

4.4 Analyzing the Data

The PECA report lists various methods that can be used to consolidate, visualize, and then analyze the collected data [Comella-Dorda 2004]:

- Consolidation
 - All-to-Dollars Technique
 - Weighted Aggregation
- Analyzing
 - Sensitivity Analysis
 - Gap Analysis
 - Cost of Fulfillment

The strengths and weaknesses of each approach are described in the PECA report.

We recommend starting with Gap Analysis. Using the risk taxonomy, an acquirer can trace gaps back to their impact on acquisition concerns and program risk areas. For those gaps that are deemed significant, the acquirer can extend the Gap Analysis by considering Cost of Fulfillment and explore the relationships between significant gaps using Sensitivity Analysis. Each technique is discussed in more detail in the PECA report [Comella-Dorda 2004, Section 5].

4.5 Interpreting Vendor Responses to the Data Collection Questionnaire

This section provides examples of how to interpret vendor responses to the data collection questionnaire, in the context of a qualitative evaluation such as a Gap Analysis. The responses are drawn from the actual vendor responses received during the pilot application of the questionnaire. This discussion does not address every item in the questionnaire but highlights areas where the pilot application produced interesting and illuminating results.

4.5.1 Questionnaire Part 1 – Demographics

The market for MDE tools is evolving rapidly, and vendor business models and priorities are changing rapidly. The vendor responses to the questionnaire pilot indicated that some vendors' business includes providing custom software development services enabled by their code generation products along with licensing the code generation products as a separate offering. If this is the

case, then special attention is needed for responses about customer base, reference customers, and market share to determine if the response applies to the entire business (development services plus tool products) or just to the separate tool products.

Responses to questions about initial product release date and product revision history provide insight into the product maturity and feature stability. Generally, consistent support for the acquirer's target environment and feature set is desirable.

We found the responses to the question asking for a "product overview" to be enlightening. One vendor indicated that its product was "aimed at the model-driven development of IT systems typically characterized by being architected in a three-layered fashion (Presentation...Business Logic...Persistence Layer)." This tool would be a possible candidate for developing or modernizing a business system but would probably be a poor candidate for a real-time embedded system. On the other hand, another vendor indicated that its product is a "general purpose environment to capture software system behavior in a high-level model and generate application code....The tool suite has so far been used to generate applications in the telecommunications, automotive, and information system domains." While this tool might be used to generate code for a business system, it seems to provide less domain-specific leverage than the first example. However, this tool might be a good candidate for developing a real-time embedded system.

4.5.2 Questionnaire Part 2 – Licensing and Delivery

The vendor responses to the questionnaire pilot indicated that the tools are often licensed and delivered in two parts: a design client that is installed and executed by each modeler or developer and a transformation or code generation server. Some vendors offer the server in a software-as-a-service (SaaS) model, where the vendor hosts the transformation/code generation service and developers access it over the Internet. Vendors may also offer an on-premises installation of the server components, which would be necessary to satisfy security classification or concerns about protecting intellectual property. If the servers will be installed on-premises, then special attention is needed to identify the prerequisite IT infrastructure and staff skills and training to install and sustain the server components of the tool.

In assessing answers to questions about the availability of training, pay particular attention to the assumptions that the vendor makes about the customer's prerequisite knowledge. Some vendors may offer comprehensive training in MDE that includes the use of their tools, while other vendors may offer only more focused training on the specifics of their tools. For example, one vendor assumes "a basic familiarity with computer science," a prerequisite that most developers might meet but that other stakeholders, who will need to use parts of the tool to certify or evaluate the design, might not.

Questions in Part 2 also address the target runtime environment. One vendor responded that its tool generates code that assumes a "POSIX compliant operating system," with customizations available for many other operating systems. If the target environment is not POSIX compliant, which is likely, then the acquirer must pay special attention to the customizations needed to execute in the desired target environment: what is the complexity of the customization, who maintains the customization, and how will the customization be sustained over the lifetime of the system? The answers to these additional questions may or may not indicate program risks.

Another vendor response to this question indicated that its tool generates “generic source code for a complex application on a variety of platforms/stacks.” In this case, a later response in Part 4 provided a complete enumeration of the supported operating systems.

4.5.3 Questionnaire Part 3 – Modeling

Vendor responses about model representation may highlight their use of the UML standard. Acquirers in a program using an MDE approach must recognize that out-of-the-box UML does not have the semantic precision needed to perform automatic code generation. A UML *profile* (a well-defined set of extensions to the core UML standard) is needed to close the semantic variation points and ambiguities. Vendors should report that they support a standard profile (e.g., Recommendation ITU-T Z.109, which is a profile that focuses on distributed systems) or have defined a proprietary profile. Responses claiming to generate code from “standard UML” require further explanation, which will probably uncover the implicit use of a proprietary profile.

Another important issue in model representation is the availability of both graphical and textual representations. Purely graphical representations have scalability limitations—there are limits to how many elements can be legibly represented on a single page or screen, and while form-based data entry can accelerate initial model creation, it can be cumbersome for maintenance of the model. In contrast, it can be more difficult to see the big picture using a textual representation, but availability of operations such as pattern-matching search and replace can make model maintenance more efficient. Generally, easy and interchangeable use of both types of representation is desirable.

Model export is important for interoperability with other tools, such as tools for model analysis or tools used by certification authorities, and to create artifacts needed by stakeholders for activities like design reviews. All of the vendors in our pilot supported export of an Extensible Markup Language (XML) representation of the model. The XML Metadata Interchange (XMI) is an OMG standard for representing UML models in XML, and vendors frequently mentioned it in responses on the questionnaire. If stakeholder communication is important, then other export formats such as Microsoft® Word-compatible .doc files or PDF files are desirable, and we found that not all vendors reported that they supported these formats. Part 3 of the questionnaire also addresses the creation of documentation artifacts for stakeholder communication—there is some overlap between “export” and “documentation”—but some vendors consider them distinct capabilities, so the questions are separate but the responses should be viewed together. In both cases, note that the export and documentation features may be add-on features that must be acquired separately from the base tool.

Part 3 of the questionnaire also addresses tool capabilities for model simulation or execution. Depending on the size and type of system being developed, it may be desirable to simulate or execute the model directly to assess requirements interpretation and sufficiency or to demonstrate design feasibility. Not all vendors in our pilot supported this capability, so if it is needed, responses in this part of the questionnaire will provide insight into the fidelity of the model execution and specific features for debugging. On the other hand, if the system and target environment are conducive to rapid installation and configuration, then generation and deployment of the actual system may be an effective approach, so a lack of capability in this area may not represent an important gap for the tool.

Some MDE tools provide a capability to generate tests that can be executed on the model, the generated code, or both. For example, one of the vendors in our pilot responded that its tool supports generating test suites (including branch coverage) from developer-created UML use case maps. The tool can execute the test suite on the model, or it can generate test code to execute the test suite on the generated code. An acquirer should evaluate such a response to ensure that the generated test code is compatible with any limitations in the target environment (e.g., is an interactive console interface required?). An alternative to automatically generated test suites is to leverage the tool capabilities to create additional model elements to drive and execute developer-defined tests. This approach requires more work for test design but may be the only approach compatible with target environment constraints.

Support for “round-trip engineering” was a hot-button issue in our pilot. Round-trip engineering is a capability that allows manual changes to the generated code to be reflected back into the model. All vendors in our pilot took the position that the model is the primary artifact: The generated code should not be manually changed, and any changes to the generated code are not reflected back into the model by their tools.

All of the vendors reported that they support using a typical revision control system or software configuration management system to allow multiple developers to work on the model concurrently. If the project has a specific requirement for a software configuration management (SCM) tool, the acquirer should review compatibility with the tool vendor. A model may comprise multiple files, which may be tightly coupled. Although an SCM system allows independent versioning of coupled files, this independence may cause problems in the tool if the contents of the multiple files are not consistent. One vendor reported having a repository capability built into the tool, which provides versioning and access control at the system and model levels, not at the file level.

The ability to create reusable templates for parts of a model is an advanced tool feature but is particularly desirable for creating larger systems and for governing the design of larger systems. Only one of the vendors provided this capability. If this feature is not available in the tool, the development team can work around this with additional processes.

Finally, the questionnaire asks the vendors to report typical developer productivity when using their tools to construct systems. Responses ranged from 2–10× improvement to 25× improvement, compared to traditional development approaches. Acquirers using this questionnaire to screen vendors might want to reword this question to reflect the type of system being built by the program and any relevant constraints. The vendor’s response to this question is just a starting point for assessing development costs—code generation is only one of many software engineering activities.

4.5.4 Questionnaire Part 4 – Target Environment

The first questions in Part 4 focus on the transformation approach: Tools may generate source code ready to be compiled, packaged, and deployed into any suitable environment; however, some tools generate code for deployment only into a specific environment. All of the vendors in our pilot fell into the first category, but in our survey of the market, we found at least one example of vendor-produced code that could be executed only in that vendor’s platform-as-a-service (PaaS) cloud.

For tools that generate source code, the target languages must be compatible with the rest of the system and tools. The supported languages are usually aligned with the tool's intended application domain. For example, one vendor responded that its tool, intended to create business system applications, supported only C# and Java, which are typical for that type of system. Another vendor, whose tool is intended to generate embedded and distributed software applications, supports C, C++, Java, and C#. If a tool does not generate source code in a language needed for the acquirer's system, that *may* indicate that the tool's intended use is not in the appropriate application domain.

If the acquirer's system will run in an enterprise application framework, such as Java Enterprise Edition (JEE) or a particular cloud environment, then automatic generation of additional deployment artifacts is desirable. For example, one vendor responded that its tool can target specific JEE application servers such as WebSphere and Weblogic as well as cloud environments such as Microsoft Azure and Amazon Web Services®.

Finally, while the generated code has dependencies on specific operating systems and databases, most of the vendors responded that their generated codes will run on a broad set of commonly used platforms. In evaluating responses to this question, carefully assess the exact products and versions supported by the tool vendor for alignment with your system architecture and evolution plans. Also, it may be helpful to review the vendor's release history (in Part 1 of the questionnaire) to assess how its tool support has tracked the updates to particular operating systems and databases of interest to the program.

5 Conclusions

While MDE promises to improve the efficiency of developing, delivering, and sustaining software, these benefits come with some challenges. An acquirer must select an MDE process appropriate to its system requirements, domain characteristics, and stakeholders' needs. This process may involve working to change an organizational culture used to focusing on code rather than models. In addition, the acquirer must evaluate contractors' and developers' selection of the appropriate tool set to implement the development, integration, and testing of the MDE solution. The acquirer may also need to work with certifiers accustomed to evaluating different types of artifacts than those produced in an MDE acquisition. And significantly, acquirers must address sustainment concerns from the beginning of the acquisition process.

Guided by the criteria for evaluating tools in Appendix A, an acquiring organization can use the vendor self-assessment questionnaire in Appendix B to collect information about a vendor's demographics and approaches to licensing and delivery of products and services, modeling representations, and transitioning products and services to the program's target environment. Together, they will help an organization evaluate risks in product engineering, the development environment, and program constraints, ensuring that potential risks across all relevant acquisition strategy concerns are covered.

Further research is needed on the benefits and risks of using MDE approaches for developing and acquiring software. Additional areas of investigation include fully mapping the acquisition strategy to areas and questions that are relevant to using an MDE methodology, using domain-specific languages as opposed to the standards-based UML, integrating multiple models, measuring the economics of MDE, and using MDE to achieve correctness by construction.

Appendix A Tool Evaluation Criteria

The tables in this appendix list all of the risk areas in our risk taxonomy [Carr 1993]. Table 1 shows the risk areas related to product engineering, which include requirements, design, code and unit test, and specialty engineering concerns. Table 2 shows the risk areas related to the development environment, which include development and management tools and processes. Finally, Table 3 shows risk areas related to the program constraints, including resources, contract constraints, and interfaces with related programs.

As noted in Section 4, this risk taxonomy provides a checklist to ensure that an acquirer considers all possibly relevant issues. Most programs will not have concerns in *every* risk area; however, if a program has concerns in a particular risk area, then the tables provide guidance on how those risks impact acquisition concerns in the context of using MDE tools for automatic code generation.

A few detailed risk areas in the taxonomy (for example, “Management Experience”) have no direct relationship to the MDE tool, and these are marked “Not applicable.”

The rightmost column of each table provides cross-references to the questions in the vendor questionnaire that relate to the acquisition concerns. The full questionnaire is provided in Appendix B.

Table 1: Evaluation Criteria – Product Engineering Risk Area

Risk Area	Potential Acquisition Concerns Related to MDE Tools for Automatic Code Generation	Questionnaire Reference
Requirements		
Stability	<p>Responding to requirements changes may necessitate operating on partially complete models and performing refactoring or rework on models.</p> <p>Communication with stakeholders is particularly important to resolve requirements issues, so tool features that support this become more important.</p> <p>Interfaces between the software modeling tools and the requirements management tools promote co-evolution of requirements and software.</p>	<p>3.3.4</p> <p>3.5.2, 3.5.2.1</p> <p>3.5.1</p>
Completeness	In addition to the concerns noted above about requirements stability, the ability to execute or simulate execution of the model can help validate requirements completeness.	3.2.1.1, 3.2.1.2, 3.2.1.3, 3.2.1.4, 3.2.1.5, 3.2.1.6, 3.4.1, 3.4.2.1
Clarity	<p>The ability to execute or simulate execution of the model can help validate interpretation of requirements.</p> <p>Stakeholder communication features also help with validating interpretation of requirements.</p>	<p>3.2.1.1, 3.2.1.2, 3.2.1.3, 3.2.1.4, 3.2.1.5, 3.2.1.6, 3.4.1, 3.4.2.1</p> <p>3.5.1, 3.5.2, 3.5.2.1</p>
Validity	The ability to execute or simulate execution of the model can help validate correctness and necessity of requirements.	3.2.1.1, 3.2.1.2, 3.2.1.3, 3.2.1.4, 3.2.1.5, 3.2.1.6, 3.4.1, 3.4.2.1
Feasibility	<p>The ability to execute or simulate execution of the model can help demonstrate feasibility of requirements.</p> <p>The ability to perform analysis of the model for qualities such as latency, throughput, and consistency can help demonstrate feasibility of requirements.</p>	<p>3.2.1.1, 3.2.1.2, 3.2.1.3, 3.2.1.4, 3.2.1.5, 3.2.1.6</p> <p>3.4.1, 3.4.2.1</p>

Risk Area	Potential Acquisition Concerns Related to MDE Tools for Automatic Code Generation	Questionnaire Reference
Precedent	References from other customers who have used the tool to create systems with similar characteristics to this system (e.g., size, complexity, specific capabilities, target platforms, an performance) demonstrate the suitability of the tool for this system.	1.1.5.2, 1.1.5.3
Scale	<p>Limitations on the size or complexity of the model that can be represented, analyzed, or transformed by the tool will limit the scale of the system that can be created.</p> <p>The ability to support parallel model development and code generation by multiple teams is usually necessary to create large, complex systems.</p> <p>The ability to incrementally develop models for parts of the system and compose those parts into larger models is usually necessary to create large, complex systems.</p>	<p>3.1.4</p> <p>3.3.2.1, 3.3.2.2</p> <p>3.3.4</p>
Design		
Functionality	The ability to execute or simulate execution of the model can help demonstrate correctness of functionality.	3.2.1.1, 3.2.1.2, 3.2.1.3, 3.2.1.4, 3.2.1.5, 3.2.1.6, 3.4.1, 3.4.2.1
Difficulty	<p>The ability to execute or simulate execution of the model can help demonstrate design sufficiency.</p> <p>The ability to perform analysis of the model for qualities such as latency, throughput, and consistency can help demonstrate design sufficiency.</p>	<p>3.2.1.1, 3.2.1.2, 3.2.1.3, 3.2.1.4, 3.2.1.5, 3.2.1.6</p> <p>3.4.1, 3.4.2.1</p>
Interfaces	<p>The ability to execute or simulate execution of the model can help demonstrate correctness of external system interfaces.</p> <p>If only parts of the system will be automatically generated, while other parts will be developed using traditional approaches, the interfaces between these two types of software must be designed and developed.</p>	<p>3.2.1.1, 3.2.1.2, 3.2.1.3, 3.2.1.4, 3.2.1.5, 3.2.1.6, 3.4.1, 3.4.2.1</p> <p>3.5.2.2, 4.1, 4.1.1, 4.1.2, 4.2, 4.3</p>
Performance	The ability to perform analysis on the model for performance qualities such as latency and throughput can help demonstrate design sufficiency.	3.2.1.1, 3.2.1.2, 3.2.1.3, 3.2.1.4, 3.2.1.5, 3.2.1.6, 3.4.1, 3.4.2.1
Testability	The tool should generate code that exposes internal states and interfaces needed to test the generated software.	3.2.1.1, 3.2.1.2, 3.2.1.3, 3.2.1.4, 3.2.1.5, 3.2.1.6, 3.3.1, 3.3.1.1, 3.5.2.2, 4.1, 4.1.1, 4.1.2, 4.1.3, 4.2, 4.3
Hardware constraints	<p>The tool must be able to generate code that will execute in the current and any anticipated target environment.</p> <p>The generated code must be sufficiently efficient (in terms of processor, memory, network, disk, and other resource utilization) to operate in the target environment.</p>	4.1, 4.1.1, 4.1.2, 4.1.3, 4.2, 4.3
Non-developmental software	Any runtime packages, libraries, or other software required to execute the generated code must be known, compatible with the current and future target environments, and able to be certified for use in those environments (e.g., information assurance, weapons safety, and flightworthiness)	2.2.1, 4.2, 4.3
Code and Unit Test		
Feasibility	<p>The ability to execute or simulate execution of the model can help validate feasibility of the generated code.</p> <p>If only parts of the system will be automatically generated, while other parts will be developed using traditional approaches, then analysis to determine feasibility of the interface is needed.</p>	<p>3.2.1.1, 3.2.1.2, 3.2.1.3, 3.2.1.4, 3.2.1.5, 3.2.1.6, 3.4.1, 3.4.2.1</p> <p>3.5.2.2, 4.1, 4.1.1, 4.1.2, 4.2, 4.3</p>

Risk Area	Potential Acquisition Concerns Related to MDE Tools for Automatic Code Generation	Questionnaire Reference
Testing	The tool should generate code that exposes internal states and interfaces needed to test the generated software.	3.2.1.1, 3.2.1.2, 3.2.1.3, 3.2.1.4, 3.2.1.5, 3.2.1.6, 3.3.1, 3.3.1.1, 3.5.2.2, 4.1, 4.1.1, 4.1.2, 4.1.3, 4.2, 4.3
Coding and implementation	The tool must be able to generate code that will execute in the current and any anticipated target environment. The generated code must be sufficiently efficient (in terms of processor, memory, network, disk, and other resource utilization) to operate in the target environment.	4.1, 4.1.1, 4.1.2, 4.1.3, 4.2, 4.3
Integration and Test		
Environment	The generated code may have runtime dependencies on third-party software or software provided by the tool vendor. This software must be compatible with the integration environment. The code generated by the tool must expose interfaces using technology bindings (e.g., languages, protocols, and standards) that are compatible with the integration environment.	2.2.1, 4.3 3.2.1.1, 3.2.1.2, 3.2.1.3, 3.2.1.4, 3.2.1.5, 3.2.1.6, 3.3.1, 3.3.1.1, 3.5.2.2, 4.1, 4.1.1, 4.1.2, 4.1.3, 4.2, 4.3
Product	The tool should generate code that exposes internal states and interfaces needed to test the generated software.	3.2.2, 3.3.1, 3.3.1.1, 3.5.2.2, 4.1, 4.1.1, 4.1.2, 4.1.3, 4.2, 4.3
System	The tool should generate code that exposes internal states and interfaces needed to test the generated software.	3.2.1.1, 3.2.1.2, 3.2.1.3, 3.2.1.4, 3.2.1.5, 3.2.1.6, 3.3.1, 3.3.1.1, 3.5.2.2, 4.1, 4.1.1, 4.1.2, 4.1.3, 4.2, 4.3
Engineering Specialties		
Maintainability	Data rights must be acquired for the models, tools, and other technology needed to operate on the model and generate code. Training and support must be available to enable the sustainment organization to gain the knowledge and skills needed to sustain the software.	2.1.1, 2.1.2, 2.3.2 2.3.1, 2.3.1.1, 2.3.3, 2.3.4
Reliability	The generated code must be correct and robust. Any runtime packages, libraries, or other software required to execute the generated code must be correct and robust.	4.1 2.2.1, 4.3
Safety	The generated code—along with any runtime packages, libraries, or other software required to execute the generated code—should be compatible with safety certification tools and processes.	3.3.1, 3.3.1.1, 3.5.1, 3.5.2, 3.5.2.1, 3.5.2.2, 4.1, 4.1.1, 4.1.2, 4.1.3, 4.2, 4.3
Security	The generated code—along with any runtime packages, libraries, or other software required to execute the generated code—should be compatible with security certification tools and processes.	3.3.1, 3.3.1.1, 3.5.1, 3.5.2, 3.5.2.1, 3.5.2.2, 4.1, 4.1.1, 4.1.2, 4.1.3, 4.2, 4.3
Human factors	If the generated software includes generated user interfaces, these must be assessed for usability.	
Specifications	The ability to execute or simulate execution of the model can help validate feasibility and interpretation of the specifications. Stakeholder communication features also help with validating interpretation of specification.	3.2.1.1, 3.2.1.2, 3.2.1.3, 3.2.1.4, 3.2.1.5, 3.2.1.6, 3.4.1, 3.4.2.1 3.5.1, 3.5.2, 3.5.2.1

Table 2: Evaluation Criteria – Development Environment Risk Area

Risk Area	Potential Acquisition Concerns Related to MDE Tools for Automatic Code Generation	Questionnaire Reference
Development Process		
Formality	Not applicable	
Suitability	<p>As noted earlier, there is a tight coupling between the selected development process and the MDE tools used to support that process. Many tools are more or less compatible with particular development processes. To some extent, all of these evaluation criteria are related to this taxonomy topic.</p> <p>References from other customers who have used the tool to create and sustain systems using a similar development process can demonstrate the suitability of the tool.</p>	1.1.5.2, 1.1.5.3
Process control	The tool must provide mechanisms for maintaining consistency of modeling, analysis, and generation at the scale required to develop and sustain the system (e.g., multiple teams, multiple connected sites, and multiple contractors).	3.3.2.1, 3.3.3.2
Familiarity	Not applicable (this refers to the familiarity of the development and sustainment teams with the processes; tool familiarity is addressed below).	
Product control	<p>An update to the tool may necessitate repeating model analyses, repeating code generation, and repeating test, integration, and certification. Tools that are rapidly evolving may put a strain on the development process.</p> <p>If the tool is delivered as a service, then configuration control of the tool is provided by the vendor.</p>	1.1.1, 1.1.3.1, 1.1.3.2, 1.1.4.1, 1.1.4.2 2.1.1, 2.1.2
Development System		
Capacity	The tool's modeling, analysis, and code generation environment will require the development and sustainment organizations to deploy particular platforms and prerequisite software.	2.1.3
Suitability	The tool's modeling, analysis, and code generation environment must be compatible with security and other standards of the development and sustainment organization.	2.1.3
Usability	<p>References from other customers who have used the tool to create systems with similar characteristics to this system (e.g., size, complexity, specific capabilities, target platforms, and performance) demonstrate the usability of the tool for this system.</p> <p>Integration of the tool with upstream (e.g., requirements management) and downstream (e.g., integration or certification) tooling improves usability.</p>	1.1.5.2, 1.1.5.3
Familiarity	If the development and sustainment teams are not familiar with the tool, training and support must be available to enable the organizations to gain the knowledge and skills needed to develop and sustain the software.	2.3.1, 2.3.1.1, 2.3.3, 2.3.4
Reliability	The tool should be mature and stable, as demonstrated by a tool release stream that maintains continuity of features while adding incremental enhancements and compatibility with underlying platform changes.	1.1.1, 1.1.3.1, 1.1.3.2, 1.1.4.1, 1.1.4.2
System support	If the vendor cannot provide support for the tool over the life of the system, then the model will have to be migrated to a new tool. This decomposes into two issues: the vendor's long-term corporate health and the vendor's continued support for the tool product.	1.1.1, 1.1.3.1, 1.1.3.2, 1.1.4.1, 1.1.4.2, 1.1.5.1, 1.1.5.2, 1.1.5.3, 1.1.5.4, 1.2.4, 1.2.5.1, 1.2.5.2

Risk Area	Potential Acquisition Concerns Related to MDE Tools for Automatic Code Generation	Questionnaire Reference
Deliverability	The modeling, analysis, and code generation tool must be available for the life of the system, or the program will have to migrate the model to different tooling. Any customization or configuration of the tools must be included in the delivery.	1.1.1, 1.1.3.1, 1.1.3.2, 1.1.4.1, 1.1.4.2, 1.1.5.1, 1.1.5.2, 1.1.5.3, 1.1.5.4, 1.2.4, 1.2.5.1, 1.2.5.2 3.3.6
Management Process		
Planning	Integration of the MDE tools with project management tools and dashboards is desirable.	3.5.2
Project organization	If the tool requires particular project staff resources to operate or manage the environment, then these should be reflected in the project organization.	2.1.3
Management experience	If the development and sustainment teams' managers are not familiar with the tool and its use, training and support must be available to enable the organizations to gain the knowledge and skills needed to develop and sustain the software.	2.3.1, 2.3.1.1, 2.3.3, 2.3.4
Program interfaces	Integration of the MDE tools with project management tools and dashboards is desirable.	3.5.2
Management Methods		
Monitoring	Integration of the MDE tools with project management tools and dashboards is desirable.	
Personnel management	Certification or other formal demonstration of proficiency in the use of the tool may be part of the development and sustainment organizations' personnel management practices.	2.3.1, 2.3.1.1, 2.3.3, 2.3.4
Quality assurance	Several issues arise in planning and executing architecture and design reviews: If the generated code requires any runtime packages, libraries, or other software to execute, this additional software must be included in the review scope. Reviewers must have appropriate access to the models. While the tool may support export to a format such as PDF documents, this format may not be usable by reviewers. Interactive viewing and navigation of the model may be needed.	2.2.1, 4.3 3.5.1, 3.5.2, 3.5.2.1
Configuration management	The tool must provide mechanisms for maintaining consistency of modeling, analysis, and generation at the scale required to develop and sustain the system (e.g., multiple teams, multiple connected sites, and multiple contractors).	3.3.2.1, 3.3.3.2
Work Environment		
Quality attitude	Not applicable	
Cooperation	Not applicable	
Communication	Not applicable	
Morale	Not applicable	

Table 3: Evaluation Criteria – Program Constraints Risk Area

Risk Area	Potential Acquisition Concerns Related to MDE Tools for Automatic Code Generation	Questionnaire Reference
Resources		
Schedule	Developer productivity is measured differently when using automatic code generation approaches.	3.3.5
Staff	If the development and sustainment teams are not familiar with the tool, training and support must be available to enable the organizations to gain the knowledge and skills needed to develop and sustain the software.	2.3.1, 2.3.1.1, 2.3.3, 2.3.4
Budget	The tool, including any optional features (e.g., import, export and integration with other tools), along with the environment to execute the tool, must be acquired. If the generated code requires any runtime packages, libraries, or other software to execute, this additional software must be acquired and sustained. Developer productivity is measured differently when using automatic code generation approaches.	2.1.1, 2.1.1.1, 2.1.2 2.2.1, 2.2.1.1, 4.3 3.3.5
Facilities	The tool and the generated software must execute within any facilities constraints of the program (e.g., classified enclaves).	2.1.3, 4.1, 4.1.1, 4.1.2, 4.1.3, 4.2, 4.3
Contract		
Type of contract	Not applicable	
Restrictions	MDE tools that generate code become part of the software supply chain. If the generated code requires any runtime packages, libraries, or other software to execute, this additional software becomes part of the software supply chain.	2.2.1, 4.3
Dependencies	The use of MDE tools for generating code creates a dependency on the tools themselves.	
Program Interfaces		
Customer	If the acquiring team is not familiar with the tool and with MDE development practices, training and support must be available to enable the organizations to gain the knowledge and skills needed to successfully acquire and sustain the software.	2.3.1, 2.3.1.1, 2.3.3, 2.3.4
Associate contractors	The tool must provide mechanisms for maintaining consistency of modeling, analysis, and generation at the scale required to develop and sustain the system (e.g., multiple teams, multiple connected sites, and multiple contractors). Stakeholders must have appropriate access to the models. While the tool may support export to a format such as PDF documents, this format may not be usable by stakeholders. Interactive viewing and navigation of the model may be needed. If some parts of the system are created using MDE and others using traditional methods, then specification of data rights and deliverables for each type of software complicates the acquisition, but this is not directly a tool concern.	3.3.2.1, 3.3.3.2 3.5.1, 3.5.2, 3.5.2.1

Risk Area	Potential Acquisition Concerns Related to MDE Tools for Automatic Code Generation	Questionnaire Reference
Subcontractors	<p>The tool must provide mechanisms for maintaining consistency of modeling, analysis, and generation at the scale required to develop and sustain the system (e.g., multiple teams, multiple connected sites, and multiple contractors).</p> <p>Stakeholders must have appropriate access to the models. While the tool may support export to a format such as PDF documents, this format may not be usable by stakeholders. Interactive viewing and navigation of the model may be needed.</p> <p>If some parts of the system are created using MDE and others using traditional methods, then specification of data rights and deliverables for each type of software complicates the acquisition, but this is not directly a tool concern.</p>	3.3.2.1, 3.3.3.2 3.5.1, 3.5.2, 3.5.2.1
Prime contractor	If some parts of the system are created using MDE and others using traditional methods, then specification of data rights and deliverables for each type of software complicates the acquisition, but this is not directly a tool concern.	
Corporate management	Not applicable	
Vendors	All of these concerns pertain to managing vendor risk related to the MDE tool vendor.	
Politics	Not applicable	

Appendix B MDE Tool Vendor Self-Assessment Instrument

Table 4 contains the questionnaire used to collect information about a particular MDE automatic code generation tool. This questionnaire was used for the pilot data collection discussed in Sections 4.3, 4.4, and 4.5. We have also included additional questions suggested by the participating vendors and added after the initial pilot.

Table 4: Self-Assessment Instrument for MDE Tools

Question Number	Section	Subsection	Question Title	Subquestion Title	Question
1.1.1	Demographics	Product Identification	Product Name		What is the name of the product, suite, or tool set?
1.1.2			Product-Specific Website URL		Link to the web page that describes the product.
1.1.3.1			Current Version	Release Date	When was this version released?
1.1.3.2				Patch Level	What is the current minor version and patch level? When was it released? If possible, provide a link to the Release Notes describing the update.
1.1.4.1			Original Version	Release Date	When was the initial version of the product released?
1.1.4.2				Revision History	Please provide a brief history of the evolution of the product, from the initial version through the current version. Identify the major features or capabilities that were added or removed in each version.
1.1.5.1			Installed Base of Customers	Number	How many customers (organizations and end users) are using your product?
1.1.5.2				List of Selected Customer Organizations	Identify (if possible) customers using the product or systems that have been built using the product.
1.1.5.3				List of Customer References	Identify (if possible) customers that can provide references for the product.
1.1.5.4				Market Share	How would you characterize your product's market share?
1.2.1		Company Information	U.S. Customer Address		Company address
1.2.2			Company Website URL		Company URL
1.2.3			Point of Contact for MDE Tool Product		If we have further questions, whom should we contact? (Name, title, phone number, and email address)
1.2.4			Company History		How long have you been developing MDE tools? What products have you delivered, and how have they changed over time? What experience does your senior technical staff have in producing MDE tools? What software engineering processes do you use for developing your products?

Question Number	Section	Subsection	Question Title	Subquestion Title	Question
1.2.5.1			Company Financials	Overall Financial Metrics	Provide or point to your overall company financial results for the past 3 years. What financial results are you projecting for the next year?
1.2.5.2				Percentage of Gross Revenue Derived from MDE Sales	How does this MDE tool fit into your overall financial picture? What portion of your total revenue is due to this product? Please provide or point to this information for the past 3 years and for the next year.
1.3			Product Overview		
2.1.1	Licensing and Delivery	Development Tool	License		What type of license do you offer? Is there an open source version of the product? How is that licensed?
2.1.1.1				Commercial Pricing	What is your pricing model for commercial offerings? Enterprise license? Node-locked? User-locked? Floating license?
2.1.2			Delivery Model		How is the product delivered? Is the development tool installed on-premises by the customer? Is the development tool delivered in a SaaS? Do you (or a partner) provide development services and use the tool to accelerate your team's work?
			Prerequisite Software		What prerequisite software (operating system, database, etc.) is required for the tool installation and execution?
2.2.1			Runtime Component(s)	Required	
2.2.2		License			If you provide any runtime software, how is that licensed? Free/open source?
				Commercial Pricing	What is the commercial pricing model for the runtime software? Enterprise? Per instance? Other?
2.2.3		Delivery Model			How is the generated software executed? Does the customer load it into the target environment? If so, how do you deliver any necessary libraries or other runtime packages (e.g., media or download)? Do you (or a partner) provide an off-premises execution environment (infrastructure as a service [IaaS] or PaaS)?
2.3.1		Support		Technical Support	
2.3.1.1			Cost		What is the cost for technical support?
2.3.2				What is your model for providing updates?	

Question Number	Section	Subsection	Question Title	Subquestion Title	Question	
2.3.2.1			Software Maintenance and Updates	Cost	What is the cost for a maintenance and update plan?	
2.3.3			Training – Availability		What training is available for your product? Please identify who delivers the training—your company, licensed partners, or other source—and identify approximate costs. Identify training for individuals (online tutorials, books), formal training (online or in-person courses), and coaching, mentoring, or consulting.	
2.3.4			Training – Recommendations		What prerequisite experience do you recommend for users (e.g., technologies, methodologies, projects, or languages)? What is the minimal product-specific training that you recommend for a user to be competent in using the product?	
3.1	Modeling				These questions deal with how models are represented in your product.	
3.1.1.1		Representation	Structural	Object Oriented	What object-oriented models are available in your product, such as class diagrams and component diagrams?	
3.1.2			Behavioral		What behavioral models are available in your product, such as state machines, sequence diagrams, activity diagrams, or action languages?	
3.1.3.1			UML 2.0	MDA Support	Does your tool support OMG MDA constructs such as platform-independent models (PIMs) and platform-specific models (PSMs)?	
3.1.3.2				Extensions	What UML extensions do you support? Do you use domain-specific profiles?	
3.1.4			Limitations		What are the size and complexity limitations for representing models using your tools?	
3.1.5.1			Format	Internal	What format is used for internal (native) model representations?	
3.1.5.2				Export	What export formats are supported (XMI, PDF, etc.)?	
3.1.5.3				Import	What import formats are supported (XMI, etc.)? Are there limitations on what can be imported?	
3.2			Testing			Please answer the questions in this section if your product supports model execution.
3.2.1.1				Model Execution and Debug Support	Diagram Animation	How is the model execution visualized?
3.2.1.2		Breakpoints			Can a developer set breakpoints to stop execution? How are breakpoints specified?	
3.2.1.3		Execution Tracing			Can a developer trace the flow of execution through the model? How is this presented?	
3.2.1.4		Event/Data Injection			Can a developer inject data or events into the model? How is this accomplished?	
3.2.1.5		Value Display			Can a developer monitor and display values of variables and data structures during model execution? How is this accomplished?	
3.2.1.6		Concurrency			How is concurrency represented during model execution?	

Question Number	Section	Subsection	Question Title	Subquestion Title	Question
3.3.1		Workflow	Round-Trip Support		Does your product support "round-trip" development? That is, are changes to the generated code or other generated artifacts automatically reflected in the model?
3.3.1.1				Code Tagging Granularity	Tools that generate code often mark or "tag" sections of the code as generated and controlled by the tool, and other sections outside the tags are editable directly by the developer. Does your product do this? If so, what is the typical scope of the tagged code?
3.3.2.1			Collaboration Support	Teams	What support does your product provide for multiple developers operating on the model?
3.3.3.1			Model Reuse	Discovery of Model from Code	Can your product create a model from existing source code? Describe the process and any limitations.
3.3.3.2				User-Defined Design Pattern	Can users of your product define reusable design patterns or templates and then instantiate these into their models? Describe the process and any limitations.
3.3.4			Model Refactoring		Does your product support "refactoring" (incrementally modifying structure or behavior, usually by splitting or combining model elements)? Describe the processes for refactoring structure (e.g., classes) and behavior (e.g., method splitting). Does your product support operating on a model that is partially completed, for example, a model that has empty placeholders or "skeletons" stubbed in for certain elements?
3.3.5			Developer Productivity		Can you characterize typical developer productivity using your product? Do you have metrics from completed projects?
3.3.6		Tool Configuration		How does the tool persist configuration (config file, registry, database, etc.)? Can full configuration be exported and imported?	
3.4.1		Analysis of the Model	Static Analysis		Does your product support static analysis, such as dependency or complexity analysis? Describe the process and limitations. Does your product produce artifacts useful for specialized analysis such as safety or cybersecurity?
3.4.2.1			Dynamic Analysis	Performance	Does your product support analysis of throughput, latency, or other performance qualities? Describe the process and limitations.
3.5.1		Documentation	Requirements Tracing		Does your product support tracing requirements into model elements? Describe the process and limitations.
3.5.2			Stakeholder Communications		Does your product produce any artifacts focused on communicating with stakeholders, including reports that might be of interest to a project manager? Describe the process and limitations.
3.5.2.1				Model Documentation	How does your product document the model? Where in the model can documentation such as design rationale be attached? What documents can be automatically generated from the model?

Question Number	Section	Subsection	Question Title	Subquestion Title	Question
3.5.2.2				Within Generated Code	How is the generated code documented?
4.1	Target Environment	Transformation Approach			What executable artifacts does your product generate? Do you generate source code that must be compiled? Do you directly generate object code or byte code? Do you generate other artifacts necessary for deployment to an environment such as an application server or enterprise application framework?
4.1.1				To Code	If your product generates source code, which languages and compilers are supported?
4.1.2				To Enterprise Application Framework	If your product generates artifacts for deployment to an enterprise application framework, what target frameworks are supported? Which languages or other bindings (e.g., XML or YAML) are supported?
4.1.3				To VM	If your product generates an executable virtual machine, which formats and hypervisors are supported?
4.2		Target Platforms Supported			If the generated code depends on particular target platforms, please identify the operating system, database, application server, hypervisor, etc.
4.3		Runtime Software Required			Please identify all target environment software that the user must provide, such as operating system, application framework, JVM, libraries, or other packages.

References

URLs are valid as of the publication date of this document.

[Bergey 2013]

Bergey, John & Jones, Larry. “Architecture-Centric Procurement.” Presented at the SEI Architecture Technology User Network (SATURN) Conference. Minneapolis, MN, Apr. 2013. <http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=48061>

[Brambilla 2012a]

Brambilla, Marco. *Model-Driven Software Engineering in Practice - Chapter 1 - Introduction*. Morgan & Claypool, 2012. <http://www.slideshare.net/mbrambil/modeldriven-software-engineering-in-practice-chapter-1-introduction>

[Brambilla 2012b]

Brambilla, Marco; Cabot, Jordi; & Wimmer, Manuel. *Model-Driven Software Engineering in Practice*. Morgan & Claypool, 2012.

[Carr 1993]

Carr, Marvin; Konda, Suresh; Monarch, Ira; Walker, Clay F.; & Ulrich, F. Carol. *Taxonomy-Based Risk Identification* (CMU/SEI-93-TR-006). Software Engineering Institute, Carnegie Mellon University, 1993. <http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=11847>

[Comella-Dorda 2004]

Comella-Dorda, Santiago; Dean, John; Lewis, Grace; Morris, Edwin; Oberndorf, Patricia; & Harper, Erin. *A Process for COTS Software Product Evaluation* (CMU/SEI-2003-TR-017). Software Engineering Institute, Carnegie Mellon University, 2004. <http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=6701>

[DAU 2014]

Defense Acquisition University. *ACQuipedia - Acquisition Strategy*. DAU, 2014. <https://dap.dau.mil/acquipedia/Pages/ArticleDetails.aspx?aid=2338b79c-7d13-4802-aac1-d905c2d95c0a#>

[Davies 2014]

Davies, Jim; Gibbons, Jeremy; Welch, James; & Crichton, Edward. “Model-Driven Engineering of Information Systems: 10 Years and 1000 Versions.” *Science of Computer Programming* 89, Part B (Sep. 2014): 88–104.

[Diskin 2012]

Diskin, Zinovy & Maibaum, Tom. “Category Theory and Model-Driven Engineering: From Formal Semantics to Design Patterns and Beyond,” 1–21. *Proceedings of the Seventh Workshop on Applied and Computational Category Theory* (ACCAT 2012). Tallinn, Estonia, Apr. 2012. Electronic Proceedings in Theoretical Computer Science, 2012. doi: 10.4204/EPTCS.93.1

[DoD 2010]

Department of Defense Deputy Chief Information Officer. *DoDAF Architecture Framework Version 2.02*. U.S. Department of Defense, 2010. <http://dodcio.defense.gov/dodaf20.aspx>

[Feiler 2012]

Feiler, Peter H. & Gluch, David P. *Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis and Design Language*. Addison-Wesley Professional, 2012. <http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=30284>

[Firth 1987a]

Firth, Robert; Mosley, Vicky; Pethia, Richard; Roberts, Lauren; & Wood, William. *A Guide to the Classification and Assessment of Software Engineering Tools* (CMU/SEI-87-TR-010). Software Engineering Institute, Carnegie Mellon University, 1987. <http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=10267>

[Firth 1987b]

Firth, Robert; Wood, Bill; Pethia, Rich; Roberts, Lauren; Mosley, Vicky; & Dolce, Tom. *A Classification Scheme for Software Development Methods* (CMU/SEI-87-TR-041). Software Engineering Institute, Carnegie Mellon University, 1987. <http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=10487>

[Greenfield 2004]

Greenfield, Jack; Short, Keith; Cook, Steve; Kent, Stuart; & Crupi, John. *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools*. Wiley, 2004.

[ISO 2011]

International Organization for Standardization. *ISO/IEC/IEEE 42010 Systems and Software Engineering—Architectural Description*. ISO, 2011.

[Mittal 2013]

Mittal, Saurabh & Martín, José Luis Risco. “Model-Driven Systems Engineering for Netcentric System of Systems with DEVS Unified Process,” 1040–1051. *Proceedings of the Winter Simulation Conference* (WSC '13). Washington, DC, Dec. 2013. IEEE Computer Society Press, 2013.

[Pastor 2007]

Pastor, Oscar & Molina, Juan Carlos. *Model-Driven Architecture in Practice*. Springer, 2007.

[OMG 2003]

Object Management Group. *MDA Guide Version 1.0.1* (Specification omg/2003-06-01). OMG, 2003.

[Selic 2008]

Selic, Bran. “Personal Reflections on Automation, Programming Culture, and Model-Based Software Engineering.” *Automated Software Engineering* 15, 3-4 (Dec. 2008): 379–391.

[Weigert 2006]

Weigert, Thomas & Weil, Frank. “Practical Experiences in Using Model-Driven Engineering to Develop Trustworthy Computing Systems,” 208–217. *Proceedings of IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing (SUTC’06)*. Taichung, Taiwan, Jun. 2006. IEEE Computer Society Press, 2006. doi: 10.1109/SUTC.2006.106

[Whittle 2013]

Whittle, J.; Hutchinson, J.; & Rouncefield, M. “The State of Practice in Model-Driven Engineering.” *IEEE Software* 31, 3 (May/Jun. 2013): 79–85.

[Wood 1988]

Wood, Bill; Pethia, Richard; Gold, Lauren Roberts; & Firth, Robert. *A Guide to the Assessment of Software Development Methods* (CMU/SEI-88-TR-008). Software Engineering Institute, Carnegie Mellon University, 1988. <http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=10609>

REPORT DOCUMENTATION PAGE			<i>Form Approved</i> <i>OMB No. 0704-0188</i>	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.				
1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE March 2015	3. REPORT TYPE AND DATES COVERED Final		
4. TITLE AND SUBTITLE Model-Driven Engineering: Automatic Code Generation and Beyond		5. FUNDING NUMBERS FA8721-05-C-0003		
6. AUTHOR(S) John Klein, Harry Levinson, and Jay Marchetti				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213			8. PERFORMING ORGANIZATION REPORT NUMBER CMU/SEI-2015-TN-005	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) AFLCMC/PZE/Hanscom Enterprise Acquisition Division 20 Schilling Circle Building 1305 Hanscom AFB, MA 01731-2116			10. SPONSORING/MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES				
12A DISTRIBUTION/AVAILABILITY STATEMENT Unclassified/Unlimited, DTIC, NTIS			12B DISTRIBUTION CODE	
13. ABSTRACT (MAXIMUM 200 WORDS) Increasing consideration of model-driven engineering (MDE) tools for software development efforts means that acquisition executives must more often deal with the following challenge: Vendors claim that by using MDE tools, they can generate software code automatically and achieve high developer productivity. However, MDE consists of more than code generation tools; it is also a software engineering approach that can affect the entire lifecycle of a system from requirements gathering through sustainment. This report focuses on the application of MDE tools for automatic code generation when acquiring systems built using these software development tools and processes. The report defines some terminology used by MDE tools and methods, emphasizing that MDE consists of both tools <i>and</i> methods that must align with overall acquisition strategy. Next, it discusses how the use of MDE for automatic code generation affects acquisition strategy and introduces new risks to the program. It then offers guidance on selecting, analyzing, and evaluating MDE tools in the context of risks to an organization's acquisition effort throughout the system lifecycle. Appendices provide a questionnaire that an organization can use to gather information about vendor tools along with criteria for evaluating tools mapped to the questionnaire that relate to acquisition concerns.				
14. SUBJECT TERMS acquisition strategy, automatic code generation, model-driven engineering, risk taxonomy, sustainment, system lifecycle			15. NUMBER OF PAGES 51	
16. PRICE CODE				
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL	