Model of Models Methodology: Reuse Your Architectural Data

Aleczander Jackson
Modern Technology Solutions, Inc.

Barrion Palmer
Modern Technology Solutions, Inc.

Daniel Cobb
Modern Technology Solutions, Inc.

One of the primary yet underutilized benefits of Model Based Systems Engineering is reuse. Most users of Model Based Systems Engineering lack the understanding of the process involved to develop a project tree that enables use of project data in other projects to facilitate that multi-dimensional reuse. Although the benefits speak for themselves, few practitioners fully grasp the modeling techniques, and rigor, necessary to facilitate a model that is fully traced, modular, and reusable. The Model of Models methodology satisfies these needs, enabling Digital Engineering on the project while realizing significant savings to project time and cost.

BACKGROUND

Digital Engineering (DE)

Definition

Digital Engineering (DE) is defined by the *Department of Defense's (DoD) Digital Engineering Strategy* (2018, p. 3) as "an integrated digital approach that uses authoritative sets of system data and models as a continuum across disciplines to support lifecycle activities from concept through disposal." DE is a major initiative being undertaken by the US Department of Defense to "modernize how the Department designs, develops, delivers, operates, and sustains systems." The need for this undertaking is to obtain the following five expected benefits: informed decision making/greater insight through increased transparency; enhanced communication; increased understanding for greater flexibility/adaptability in design; increased confidence that the capability will perform as expected; and increased efficiency in engineering and acquisition practices.

Digital Engineering Strategy

Within the *Digital Engineering Strategy* (2018) the concepts for digital engineering are further refined into sets of processes that must be undertaken by organizations to obtain digital compliance. However, the key take-away from the strategy is that the current mechanisms used to architect systems, namely document-based engineering techniques, are responsible for significant slow-down in the acquisition and development processes which necessitates the move to model-based techniques for system design, and true

digital engineering will involve the interconnection and synchronization of data across different engineering tools so that the "authoritative source of truth" of the data will be established without conflict from the conceptual design to the detailed design of the system. Developing systems in this manner will establish a "digital thread" that can be "pulled" in any linked engineering tool which will enable better change analysis as changes down-stream can be automatically realized up-stream by the tooling, and impacts that may be caused by changes to up-stream artifacts can quickly be visualized.

Systems Engineering

System

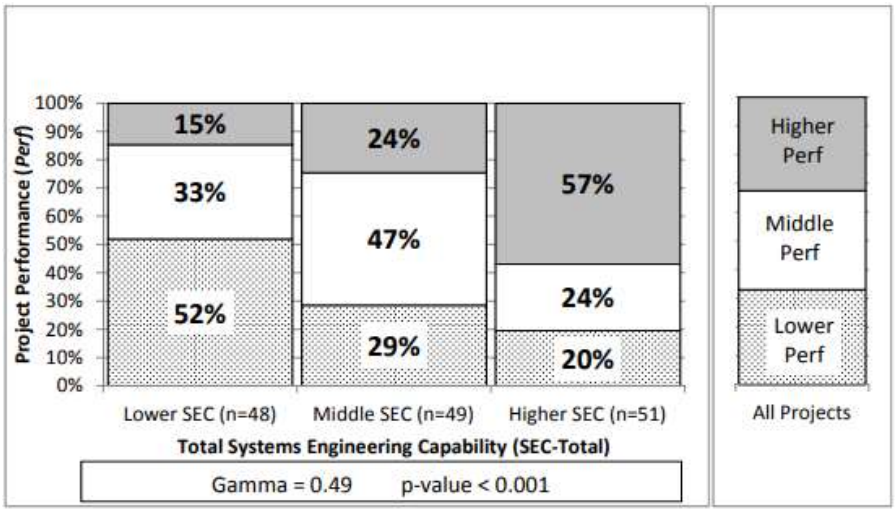
A system has numerous accepted definition but for this concept it can best be understood as "a purposeful collection of interrelated components, of different kinds, which work together to deliver a set of services to the system owner and users" (Sommerville, 2016, p. 772). Systems are made of subsystems which are used to enable the system to perform the functions assigned to it. Subsystems may be composed of their own subsystems, components, defined in this paper as either lower-level systems which will be procured or the combination of subsystems and components. Another important concept within Systems Engineering is a System of Systems (SoS), which is simply a system composed of other independent but related systems that, when brought together into an SoS, delivers greater performance than the combination of individual systems can output alone.

SE Definition

The *INCOSE Systems Engineering Handbook* (2015, p. 11) defines systems engineering as "an interdisciplinary approach and means to enable the realization of successful systems." Eisner (2008) defines systems engineering as "an iterative process of top-down synthesis, development, and operation of a real-world system that satisfies, in a near optimal manner, the full range of requirements for the system." And finally, the Federal Aviation Administration (2006) defines systems engineering as "a discipline that concentrates on the design and application of the whole (system) as distinct from the parts. It involves looking at a problem in its entirety, considering all the facets and all the variables and relating the social to the technical aspect."

Project Performance

FIGURE 1
PROJECT PERFORMANCE VERSUS SE CAPABILITY



From the definitions of systems engineering, we come to realize that systems engineering is a discipline that analyzes systems as compositions of functioning parts. Performing this kind of engineering has become essential due to the progressively increasing complexity of systems and the interconnectedness of the systems crossing multiple engineering disciplines.

The value that systems engineering provides is that it has emerged as an effective way to manage complexity and change (Systems Engineering Handbook, 2015). In addition, as Olivier Casse writes in *SysML in Action with Cameo Systems Modeler* (2017, p. 4), benefits obtained from systems engineering include "increase in quality by implementing the verification and validation principle as early as possible". Further analysis performed by Elm and Goldenson (2012) have shown that around 15% of organizations with a lower Systems Engineering Capability (SEC) delivered higher levels of project performance versus 52% of delivering lower levels. Compare that to organizations with a higher SEC, which 57% produced higher project performance while only 20% delivered lower levels.

FIGURE 2
COST AND SCHEDULE VS SE EFFORT

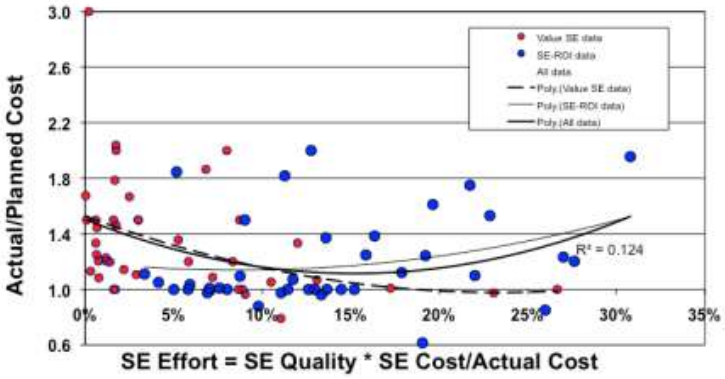
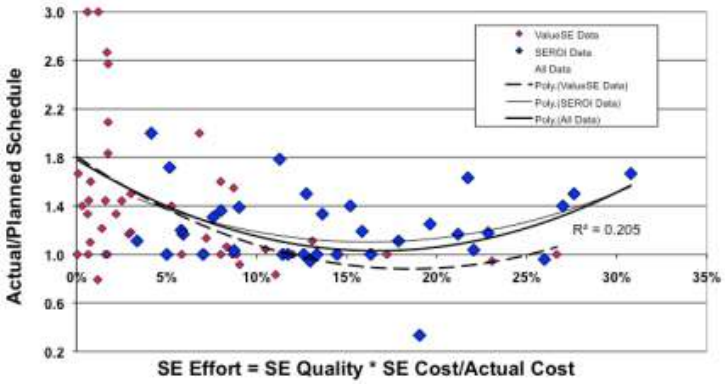


Figure 19. Cost overrun vs. systems engineering effort



In a similar fashion, Eric Honour (2013) revealed during his study that increasing the percentage of SE within projects results in better success up to an optimum level, above which greater SE reduces performance. His studies showed that the optimum SE level of effort was 14% of the total program cost. Eric was also able to determine that adding SE to a project with no SE activities provides a 7:1 return on investment for each unit cost of SE where at the median level, the return is at 3.5:1. The evidence clearly supports that providing thoughtful, quality systems engineering to a project tends toward producing more favorable results.

Problem

However, as with anything, Systems Engineering has room for improvement. As Deligatti describes in *SysML Distilled* (2014, p. 2), systems engineering is "more expensive than it needs to be as you incur a significant percentage of total life cycle cost maintaining that disjoint set of artifacts. And if you don't pay that cost, the artifacts become inconsistent and obsolete". To elaborate on Deligatti's statement, document-based systems engineering comes with several problems, because the artifacts are disjoint; when changes occur, it is often a manual and intensive effort to update all artifacts to reflect the change. SE becomes a search and find of sorts for correcting translation errors and identifying inconsistencies by hand across engineering disciplines. Adding to the tedious effort of making a change throughout all artifacts, running an impact analysis, to understand the total cost associated with a change, has become challenging due to the disjointedness of the artifacts, often leading to errors that propagate into the design. However, there is a solution to this challenge which lies in Model Based Systems Engineering.

Model Based Systems Engineering (MBSE)

Definition

One major component of Digital Engineering is a foundation based on model-based systems engineering (MBSE). The *INCOSE Systems Engineering Vision 2020* (2007, p. 15) defines MBSE as "the formalized application of modeling to support system requirements, design, analysis, verification, and validation activities beginning in the conceptual design phase and continuing throughout the development and later life cycle phases". The process of performing MBSE as Friedenthal, Moore, and Steiner describe it in *A Practical Guide to SysML* (2015, p. 17), is "the output of the systems engineering activities is a coherent model of the system (i.e. system model) that is part of the engineering baseline, and the emphasis is placed on defining and evolving the model using model-based methods and tools."

Benefits

Benefits of using model-based systems engineering include "increase(d) precision of the system specification and design resulting in reduced downstream errors; improve(d) traceability between system requirements, design, analysis, and verification information to enhance system design integrity; improve(d) ability to maintain and evolve the system specification and design baseline throughout the system lifecycle; support(ed) reuse across projects; and provision of a shared understanding of the system to reduce miscommunication among the development team and other stakeholders" (Friedenthal & Oster 2017, p. 2).

Language

FIGURE 3
PILLARS OF MBSE

MBSE is built on three main pillars, i.e. a language, a tool, and a method, see Figure 2 created by Loyd Baker. The languages are the graphical modeling languages that are used to construct diagrams of elements and define what the model is "saying".

There are a few MBSE languages in the market, and each has specific purposes for its use. This paper focuses on the Systems Modeling Language (SysML) which was jointly developed by the International Council on Systems Engineering (INCOSE) and the Object Management Group (OMG). Due to it being registered with the OMG, it is one of the few MBSE languages that are "standardized" in the market which enables data transfer between tools based on common language artifacts and diagrams.

Tool

The second pillar, the tool, utilizes the language within a software context to visualize the system. There are several tools in the market, some utilizing SysML, others with their own custom languages that do not fully comply with a standardized language. Although a non-standardized language can enable a "stove-piped" MBSE capability, data integration will always be at the forefront in a language that is standardized and shared across many platforms such as SysML. Many tools will provide APIs to export data; however, if they are compliant with a language like SysML, a standard export into the XML Metadata Interchange (XMI) format must be a capability of the tool. This enables the data from SysML tools to be shared across many tools, reducing the dependence on any one vendor, and ensuring that the right tools for the right job are used.

It is important to remember that not all tools are the same; some tools are better than others and add enhanced features to increase productivity of the workforce, and because time is money, the more time your engineers save due to the tool-set you provide them, the more productive they become which increases their return on investment.

Method

The third pillar, selection of the method(s), is arguably the least understood. While finite languages and tools exist on the market, the number of processes and methodologies that can be applied to a modeling effort are only limited by knowledge and understanding. Thankfully there are proven methodologies that can be adapted to best fit an organization's needs throughout the process.

A methodology is not a one-size-fits-all solution. Methodologies are a set of methods used to produce a specific output. Different methodologies use different techniques to produce potentially different outputs, even for the same problem. Just like when comparing object-oriented programming techniques to functional, the first question to ask before selecting the language of use is "What are you trying to accomplish?" Until that fundamental question can be answered, do not proceed, because this leads to engineering blindly which is arguably a waste of an organization's time and money. Instead, define the expected outputs of an effort, then decide which "tool" in the engineer's "toolbelt" is a best fit to accomplish the task in the most efficient manner possible. This analogy directly applies to methodologies: selectively choose the one on each program that best meet the needs of that program. However, just like integrating a language like C++ with Java, methodologies do not necessarily integrate cleanly with one another so ensure that consideration is factored into the methodology-making decision.

DIGITAL ENGINEERING

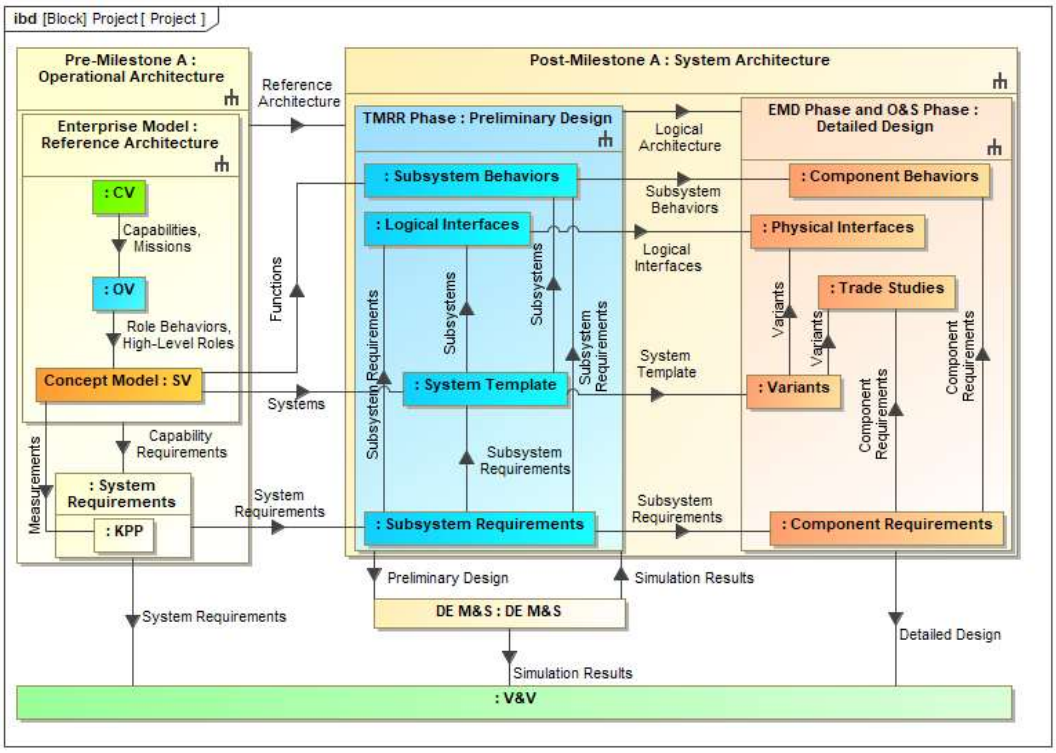
Concept

The concept of digital engineering as defined by the US Office of the Secretary of Defense (OSD) focuses on the interconnection of data models across disparate tools. The vision focuses on an integrated set of engineering tools where data can be linked, if not synchronized, across the various tools. The outcome of this effort is what is referred to as a "digital thread" which represents a fully linked set of data that spans multiple tools and engineering disciplines enabling the analyst to view the entire data integration. The value of this goal is that if something must change, then the entire impact of such a change across all tools can be

realized rapidly, enabling faster decision making that prevents uninformed/expensive decisions from being made.

However, for the digital thread to be accomplished, specific methodologies in the respective tools must be followed to ensure that there are no traceability gaps. Unfortunately, the existence of gaps within the traceability of digital artifacts will manifest themselves as gaps within the digital thread, leading to breaks within the thread, and reducing its usefulness. Therefore it is not enough that a methodology is implemented; the chosen methodology must take into account tools and processes that enable that traceability being described above.

FIGURE 4
ARCHITECTURAL DEVELOPMENT PROCESS



This paper introduces a novel methodology to solve this problem: the Model of Models Methodology (MOM) for MBSE. MOM was created to increase inheritance across the model and secondly to ensure a process that enables full traceability across the model artifacts. This traceability ensures that the digital thread can be established throughout the model, so when the model is integrated with other engineering tools, there will be no breaks in the thread across the MBSE portion of the tool chain.

Figure 4 provides a glimpse into the Architectural Development Process where the flow-down of the modeling artifacts is identified relative to a system executing the DoD Acquisition Life Cycle. Although MTSI provides a formal training in MBSE through a SysML Fundamentals Course, not everyone involved with the MOM process will need to undertand the language to appreciate the capabilities that the methodology provides. MOM begins with a reference architecture that produces the system-of-systems (SoS) view of a particular operation in which the various systems that must exist in order for a particular mission to be executed are identified. This results in systems requirements which feed the system template, which is a logical definition of the class of system, providing all functions and components that would be true across all potential implementations (variants) of the system. Using the concept of inheritance, the detailed logical template will provide a black-boxed or closed system view, which can then be broken into

specific variants of the component to produce variants of the overall system. From this point, a set of trade studies will be conducted in the system to determine the best configuration of available components which will create optimal set of final solutions.

At all phases, capturing the test cases early and upfront allows the verification of system requirements later in the system life cycle. Additionally, for specific programs that need to connect to complex physics-based simulation tools, preliminary design data will be fed into formal modeling and simulation tools which provide results back into the architecture for further refinement.

MODEL OF MODELS METHODOLOGY

Introduction

The Model of Models (MOM) methodology is a process that can be used to produce a reusable, modular, and fully-traced MBSE architecture. The methodology was derived by the SMEs from MTSI based on significant analysis of commercial organizations in which they consulted to analyze the best mechanism that could be followed to maximize reuse of artifacts across an organization. MOM is based on the Object-Oriented Systems Engineering methodology (OOSEM) but is tailored to abstract and depict at a logical enterprise level the key processes affecting major lifecycle threads such as acquisition, program management, engineering (architecture development), sustainment, etc. MOM also introduces benefits at the system/component level by facilitating reuse through the development of system/component meta-models (aircraft, ships, tanks, vehicles, widgets, etc.) that capture the key structure and behavior and serve as the templates from which variants are derived/designed/built.

The benefits gleaned from reuse are substantial time and money savings. Most organizations are in the business of developing a family of products and cannot afford to start every project with SE from scratch. Instead of rearchitecting systems each time the organization must use a component in a new project/program, they can instead refer to the definition stored in a common component library which enables the architectures of components, subsystems, and even full-blown systems to be reused across many different projects and programs. This reuse substantially and effectively reduces the upfront SE workload, exponentially increasing time and cost savings in the architectural development portion of a program. See appendices A and B for information on how this concept is leveraged with aircraft and tank architectural development.

The primary benefit of a modular architecture is that it enables plug-and-play with components. Each component is made of many data element types, months to years of engineering information for that component, and standard requirement and verification objectives for those components. Using this concept, we can create models of all possible variants in a short time period, each based on and traced up to the logical architecture, which enables rapid trade study analysis as each of these variants can be compared against one another in an almost instantaneous fashion to test which are best to suite the user's needs.

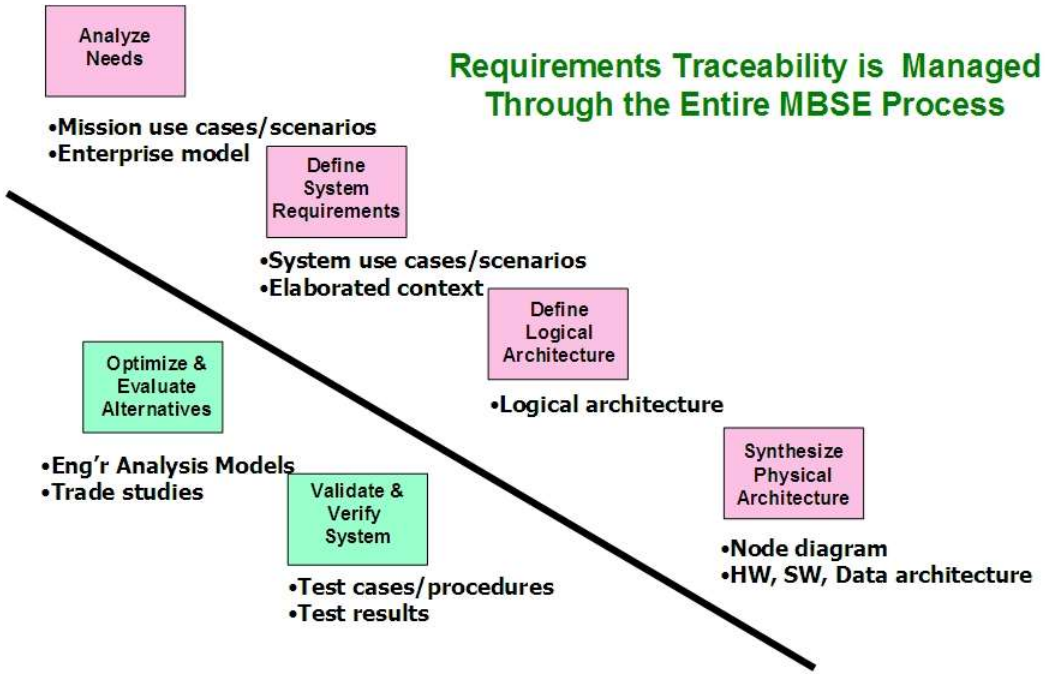
Finally, the benefits observed from full traceability is that it enables full impact analysis. For instance, if a system-level requirement must change, we are able to follow the tree of derived requirements from the system requirement to show them as all being potentially impacted from the source system requirement. This enables informed decision-making for both impact and timing of a change in the system model to have the lowest-possible impact while gaining the needed feature/benefit.

Object Oriented Systems Engineering Method (OOSEM)

Definition

"OOSEM is a top-down, scenario-driven process that uses SysML to support the analysis, specification, design, and verification of systems. The process leverages object-oriented concepts and other modeling techniques to help architect flexible and extensible systems that can accommodate evolving technology and changing requirements" (Friedenthal et al. 2015, p. 417).

FIGURE 5
OOSEM PROCESS



The OOSEM Process, depicted above in Figure 5, is built on six main aspects: analyze needs, define system requirements, define logical architecture, synthesize physical architecture, optimize and evaluate alternatives, and validate and verify system. The processes to the right of the line are in a waterfall-style fashion going in order from the analyzation of needs down into synthesizing the physical architecture in the standard order of development, namely understand the concept first, define the functions the system will perform, and then piece together the system that provides those functions in order to meet the requirements. To the left of the diagonal line we find "Optimize & Evaluate Alternatives" and "Validate & Verify System" which are intended to be performed in parallel with the steps to the right of the line.

OOSEM is an excellent process for development of systems that are object-oriented in nature such as a large system-of-systems; it provides a very strict taxonomy-based rigor that must be followed, and due to the top-down nature of the development effort it enables full traceability and improves impact analyses. One disadvantage of these formal approaches is that they are conceptual in nature and should be refined and tailored for each use to include powerful tool capabilities, like MOM's use of component library projects within system projects to enable reuse. Another disadvantage of strictly using the formal methodologies is they tend to lead to monolithic models which cause the model to become extremely large, making navigation and data management of the model arduous.

Problems Associated With Monolithic Models

As mentioned, one of the disadvantages of traditional MBSE methods is that they lead to the creation of monolithic models. These monolithic models are difficult to navigate and manage, which often leads to the creation of duplicate or conflicting elements. A duplicate element in an MBSE architecture contains separate data and must be managed independently, which can lead to model inconsistency if one of the duplicates is modified. Worse than duplicate elements, conflicting elements are a critical problem in a model because if two elements conflict with one another, it becomes impossible for the system architecture to meet the requirements imposed upon it, leading to invalid system architectures, which are then carried into development. Developing the invalid system architecture leads to late-term development rework which is costly and will most likely require a schedule slippage or missing feature in the delivered system.

A common feature that several MBSE tools provide is a teamwork environment in which multiple team members can work on the same model at the same time. This is a great feature, except for when a model becomes too large for the infrastructure to sustain as each artifact and relationship is an element that is loaded into memory, and each team member working on the model is also consuming memory. As the model gets larger and as more members are working on the same project at the same time, the server hosting the model will become increasingly overloaded leading to development slow-downs and inefficiencies as features like committing the model to the server can in some cases take several hours, consuming a majority of the working day.

Finally, monolithic models create overreach. Even with the best of intentions, if a team member makes an unofficial update to the model which contains errors, a single source of failure is created as the model has no way of checking for its own logical fallacies, i.e. it is only as smart as the engineers who input the data. If all the data lives in a single model, it becomes more challenging to add permissions to the specific model sections, enabling such users to voluntarily "contribute" to the effort without official training and understanding of model best practices, and often times cause errors that must be repaired at a later date.

These issues can be extrapolated into a very simple formula provided below:

$$Result = Disaster = Shelf - Ware = Failure! \quad (1)$$

The failure of an implementation of MBSE has created cultures of distrust, resistance, and objection to the implementation of MBSE within many organizations. The method chosen can lead to inevitable problems that will affect the entire organization. As such, a better method must be chosen.

Justification for MOM

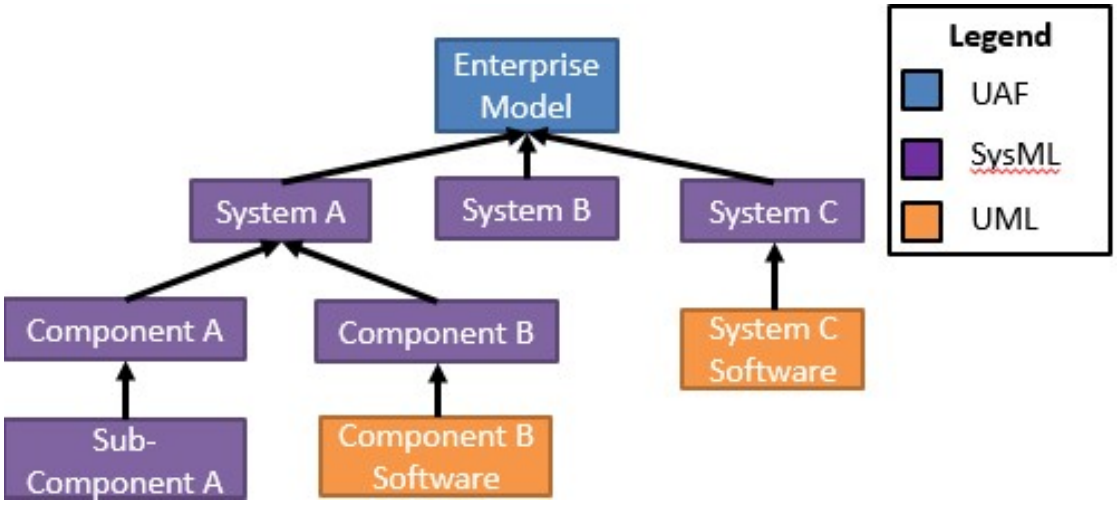
In addition to avoiding monolithic models, there are other observations the research team has made during their experience as consultants and contractors. The first is that most complex systems are system-of-systems, defined by ISO/IEC/IEEE (ISO, 2019) as a "set of systems or system elements that interact to provide a unique capability that none of the constituent systems can accomplish on its own." Second, few are responsible for the full development of a system from the concept to production. This leads to architecting a system down to a finite level where components are acquired from component manufacturers. If a component is defined earlier as a lower-level systems that will be acquired without the need for full-decomposition to compose subsystems, then they will often be captured at the black-box level which is used to place "the main focus on how (the System of Interest) interacts with the environment without getting any knowledge about its internal structure and behavior" (Aleksandraviciene & Morevicius, 2018, p. 12).

The observation made above is that there is a point where systems engineers produce variants from different combinations of components that will be evaluated against one another. This process is referred to as trade-study analysis and is performed to determine which variants will best meet the requirements of the system. What is interesting to note is that this concept is recursive in nature as the architects of the components for the system develop their component as a system and will use components of their own. This continues until getting down to the fundamental level where one is placing resistors or cutting rods.

The Methodology

Reiterating from previous sections, the Model of Models methodology was created for the purposes of reuse, modularity, and full-traceability of MBSE model data. In addition to these fundamentals, the qualitative benchmarks used to evaluate MOM are ease of use and a "natural" feel. MOM should not be overly complex but rather be straight-forward to understand and it flexible for many different applications and should naturally map to the way a system is constructed. The result of these goals increases navigability of the architecture which improves model management and the "correctness" of the model.

FIGURE 6
MODEL OF MODELS

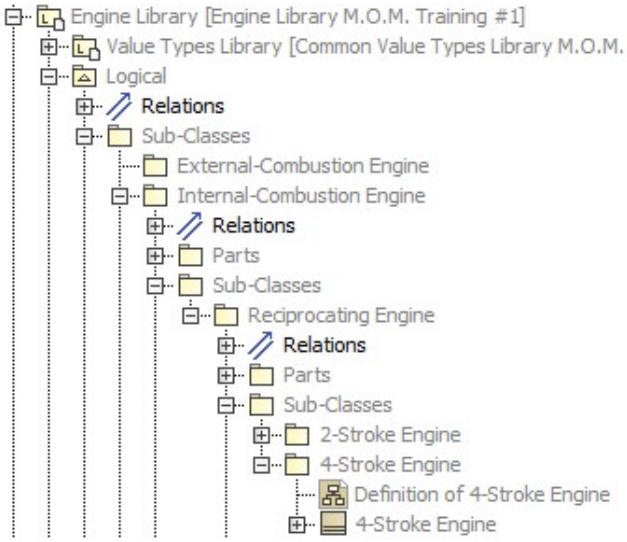


This leads to the concept. Namely, the Model of Models methodology focuses on the creation of component library projects which feed components into higher-level projects recursively. This creates a hierarchy where the "model of models" is the highest level in the overall system model, in the case of Figure 6, the "Enterprise Model", which is used to show the integration of multiple systems into mission scenarios. This Enterprise Model will load system A, B, and C definitions from library projects dedicated to those specific types of systems. Within the Enterprise Model, variants will be created in the SoS level and most analyses will be performed to determine which combination of system variants will provide the highest success rates in mission execution. The same process is performed with System B where variants of Component A and B are evaluated to develop the optimal system design, which will enable the system to meet its requirements, leading to better mission performance. The process iterates at the component model level using sub-component models, and can even apply this principle to allow software architectures to be deployed within the systems to analyze the impact of specific software designs on the functionality of the system.

One thing to notice is that each of the sub-projects provides elements to the higher-level project. First, this is done to circumvent a common limitation of MBSE tools preventing circular references, which means that a project uses a project that uses itself. Secondly, the choice to inherit up ensures that the higher-level systems have full visibility into their fully system hierarchies. This supports full analysis of each system and helps protect intellectual property (IP) in that only the acquirer of components can see the component definitions, so suppliers have less to worry about IP leakage to their competitors.

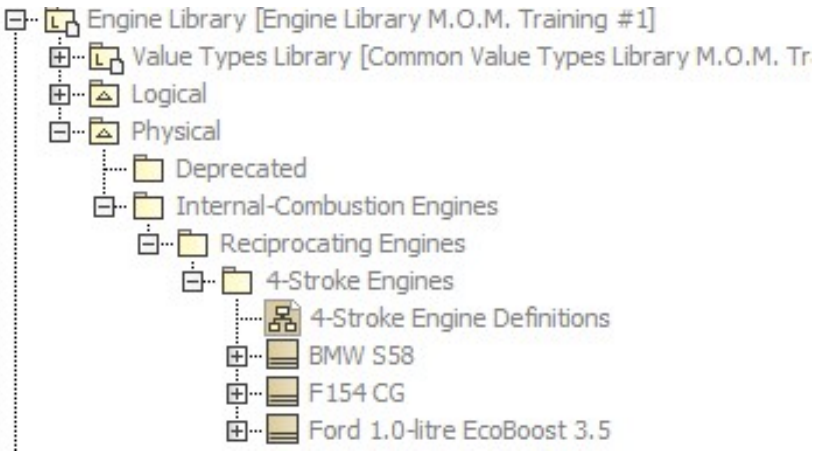
MOM Library Layout

FIGURE 7
LOGICAL SECTION OF LIBRARY



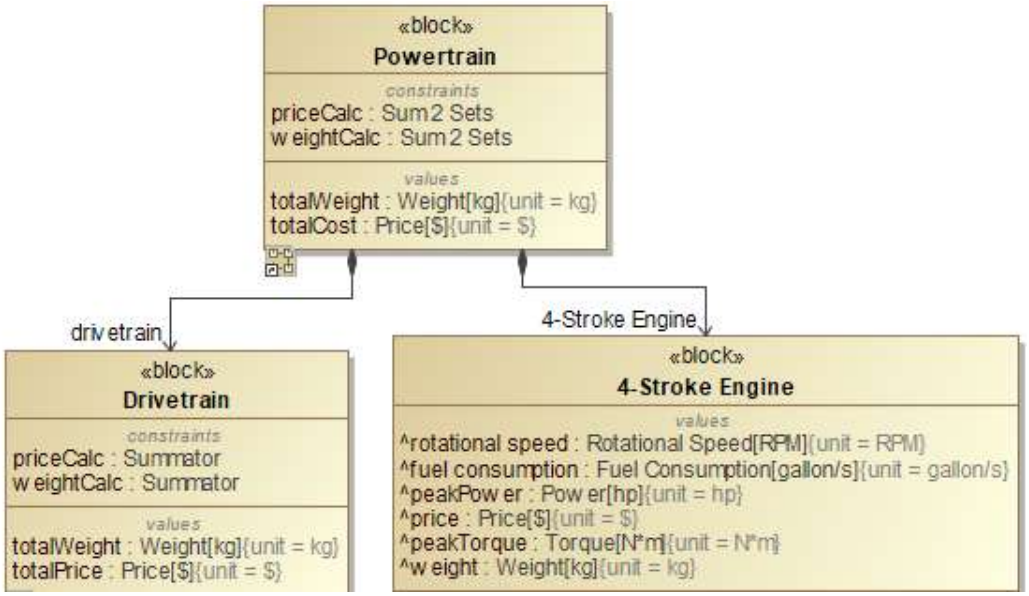
The Model of Models methodology uses a recursive taxonomy layout. We define the highest-level classification of the library artifact, which is shown in the case of Figure 7 as an engine. At this level, we all parts are created that will be true for all subclasses of the system. Creating the next tier of classification, shown above as "External-Combustion Engine" and "Internal-Combustion Engine", will provide the parts and functions that are true about each subclass. Continuing through this recursive process gets to the lowest level of taxonomy, which will inherit and refine the parts provided from the higher levels.

FIGURE 8
PHYSICAL SECTION OF THE LIBRARY



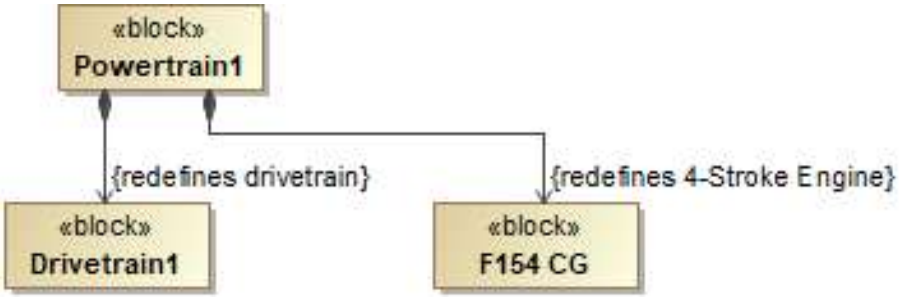
One we have finished the logical taxonomy to the lowest classification for the type of component needed, in this case the "4-stroke engine", the next step is creation of variants. The model structure in the "Physical" section of the library mirrors the logical, and at the lowest level captures the definitions of the specific variants of the components. In this example, the variants being captured for the "4-stroke engine" are the "BMW S58" the "F154 CG" and the "Ford 1.0-litre EcoBoost 3.5".

FIGURE 9
USE OF THE LOGICAL COMPONENT



Once the components are placed in the library, we either return to the model of the system (or SoS) we are creating, or iterate through to creation of the model for this system. For the logical section of the model, we model down to the logical component's usage, in our example the use of the "4-Stroke Engine" above, which we pull in from the used engine library project. We perform the operation of creation of component libraries, modeling of the full taxonomy down to the type of element we wish to use in our particular system, the creation of the variants we will evaluate, and the use of these logical elements in the logical section of our system model for each component that we need to use. The value being, once this taxonomy has been defined, assuming the model is placed in a shared location, anybody can use these reusable components for all future programs which will significantly reduce the amount of work required to create or enhance new systems.

FIGURE 10
USE OF VARIANT ENGINE



Our final step in our system model is to go through the physical system model, which inherits its architecture from the logical system template, and redefine each of the logical components with physical components from the library. Redefinition is done to "rename, augment, constrain, or override the redefined members in the context of instances of the specializing classifier," (Fakhroutinov, 2015) which in layman's

terms means that we are replacing the logical definition with a specific variant. We perform this for all components in a subsystem to create a subsystem variant. We redefine each subsystem in a system variant with subsystem variants to create the system variant. As can be seen, this operation is intensive, so why would anybody consider this process? The return on investment of this intensive process is that it creates full traceability throughout the system architecture from concept through to the specific variant implementations. Another major benefit from this process is that any changes to the library components to add more specificity will propagate to each of the system models in which they are used, which will provide higher fidelity in simulations, enabling better understanding when verifying that the architecture will satisfy the requirements imposed upon it.

CONCLUSION

The Model of Models methodology was developed as an improvement on OOSEM. It was built to provide reuse, modularity, and enable full traceability. The sub-goals of MOM are for the process to be easy, repeatable, recursive, and to "feel natural". The benefits this method provides are significant time savings as soon as model elements are reused from the component libraries, reduction in difficulty of performing impact analyses, and finally it enables ease of variant creation due to inheritance from the rigorous taxonomy observed. The full traceability provided by MOM fully enables the core principles of Digital Engineering ensuring that no gaps within the digital thread arise from the MBSE environment.

ACKNOWLEGEMENTS

This paper and the implementation without it would not have been possible without the support from MTSI's senior leadership: Kevin Robinson (CEO), Russ Wolfe (VP of Engineering), Tim King (SVP of AMD), and Dana Larkins (PM of Digital Engineering). In addition, thank you to Joy Newlin and Dennis Susol for your analysis and improvements to the existing methodology. Thank you to Sarah Cates, Neil Patel, Joseph McCreless, David Fields, Sami Rodriguez, and Leigh Mariotti who proved MOM's success by taking a leap of faith and implementing it on their high-priority government projects. Thank you to past training, mentorship, and leadership from Barry Papke and Saulius Pavalkis who initially trained me to become a competent Systems Engineer.

A special thank you to Sanford Friedenthal, who's coaching, mentorship, and feedback has improved the outlook of the team guiding us to develop this method. A special thank you to Gary Duncanson, previous CEO of No Magic, who took a chance in hiring me, giving me the opportunity to incubate with some of the top MBSE practitioners in the world. A special thank you to my wife Mary, who enabled me to grow rapidly in this field while taking care of the children and the home. And finally, the most special thank you to Robert Saunders, Assistant Department Head of Electrical Engineering at University of Arkansas, for being a close colleague, friend, mentor, motivator, and helping me develop the spirit to always do things right the first time.

REFERENCES

Aleksandraviciene, A., & Morkevicius, A. (2018). *MagicGrid Book of Knowledge*. Kaunas, Lithuania: Vitae Litera.

Baker, L. (2017). *FUNDAMENTALS OF MODEL-BASED SYSTEMS ENGINEERING WITH SYSML* [Image]. University of Alabama at Huntsville. Retrieved October 9, 2020, from http://cps.uah.edu:8080/print_friendly_MBSE_SysML.pdf

Casse, O. (2017). *SysML in Action with Cameo Systems Modeler.* Kidlington, Oxford, UK: Elsevier Ltd.

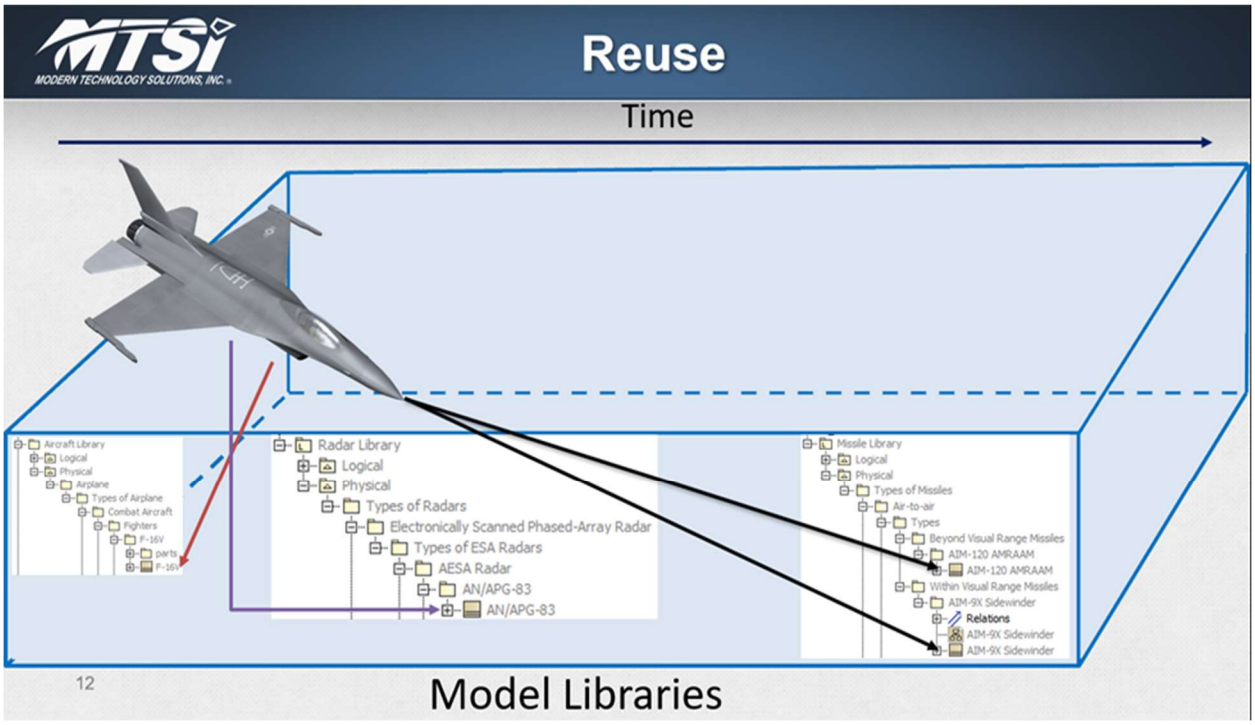
Delligatti, L. (2014). *SysML Distilled.* London, England: Pearson Education Inc.

Department of Defense. (2018). *Digital Engineering Strategy.* Retrieved October 9, 2020, from https://ac.cto.mil/wp-content/uploads/2019/06/2018-Digital-Engineering-Strategy_Approved_PrintVersion.pdf

Eisner, H. (2008). *Essentials of Project and Systems Engineering Management.* Hoboken, NJ: John Wiley & Sons, Inc.
Elm, J., & Goldenson, D. (2012). *The Business Case for Systems Engineering Study: Results of the Systems Engineering Effectiveness Study*. Software Engineering Institute, Carnegie Mellon University. Retrieved October 9, 2020, from https://resources.sei.cmu.edu/library/asset-view.cfm?assetID=34061.
Fakhroutinov, K. (2015, September 9). *Redefining Property in UML*. UML Diagrams. Retrieved October 9, 2020, from https://www.uml-diagrams.org/redefining-property.html#:~:text=Redefining%20Property%20in%20UML,instances%20of%20the%20specializing%20classifier.
FAA. (2006). *Systems Engineering Manual, Version 3.1*. Federal Aviation Administration.
Friedenthal, S., Moore, A., & Steiner, R. (2015). *A Practical Guide to SysML* (3rd Edition). Waltham, MA: Elselvier Inc.
Friedenthal, S., & Oster, C. (2017). *Architecting Spacecraft with SysML*. Createspace Independent Publishing Platform.
Honour, E. (2013). *Systems Engineering Return on Investment.* [Doctoral dissertation, University of South Wales]. Retrieved October 9, 2020, from http://www.hcode/seroi/index.html
INCOSE. (2007). *INCOSE Systems Engineering Vision 2020. 15*. Retrieved October 9, 2020, from https://sdincose.org/wp-content/uploads/2011/12/SEVision2020_20071003_v2_03.pdf.
ISO/IEC/IEEE 21839. (2019). *Systems of Systems (SoS) considerations in life cycle stages of a system*. Geneva Switzerland: International Organization for Standardization.
Sommerville, I. (2016). *Software Engineering*. Harlow, England: Pearson Education Limited.
Walden, D.D., Roedler, G.J., Forsberg, K.J., Hamelin, R.D., & Shortell, T.M. (2015). *Systems Engineering Handbook*. Hoboken, NJ: John Wiley & Sons, Inc.

APPENDIX A

FIGURE 11
INITIAL CREATION OF PROJECTS



We begin the application of the Model of Models methodology with the concept of creation of aircraft. On the left, red arrow, we begin by defining the aircraft. Since we are imagining we have never architected an aircraft before, we must build the logical taxonomy of an aircraft down to the lowest-level "Fighters", which would inherit the full definition of an airplane, the definition of the subclass of "Combat Aircraft" and the definition of the subclass of "Combat Aircraft". This is all contained within the "Logical" model within the aircraft library. We then through architecting a "Fighter" aircraft realize we need a radar and a missile. For the radar library we follow the similar taxonomy definition process, first defining in the logical model all subsystems and components common to all radars, further specifying this generic definition into the electronically scanned phased-array radar, which is further defined into the "AESA Radar". Finally, for the Missile Library the same process is followed, first defining the taxonomy from the missile down to the subclass of "Air-to-Air" missile down to the lowest-level subclasses of "Beyond Visual Range Missiles" and "Within Visual Range Missiles". The lowest-level logical components are used within the lowest-level aircraft, i.e. a block named "Fighter" composes an "AESA Radar" block, a "Beyond Visual Range Missile" block and a "Within Visual Range Missile" block.

Once we've setup the taxonomy, we proceed into variant definition. In the example, above, the first variant of a fighter aircraft is the F-16V. This aircraft uses the specific kind of AESA radar, which is the AN/APG-83 as well as the two missiles, the AIM-120 AMRAAM and the AIM-9X Sidewinder. The AN/APG-83 is a kind of AESA radar, and as such inherits the definition of an AESA Radar from the logical block contained in the logical model. The AIM-120 AMRAAM is a variant of a "Beyond Visual Range Missile" and as such will inherit from the corresponding logical block. The AIM-9X Sidewinder is a variant of a "Within Visual Range Missile" and likewise will inherit from the corresponding definition.

One key concept within MOM is to realize that each component in a component library is most likely composed of parts which can be defined in their own library. This cycle of taxonomy and definition can continue in a recursive fashion, until reaching the necessary depth for a particular project/program, but due to the modular design of the libraries, can be further enhanced at a later date.

FIGURE 12
USE OF EXISTING LIBRARY COMPONENTS

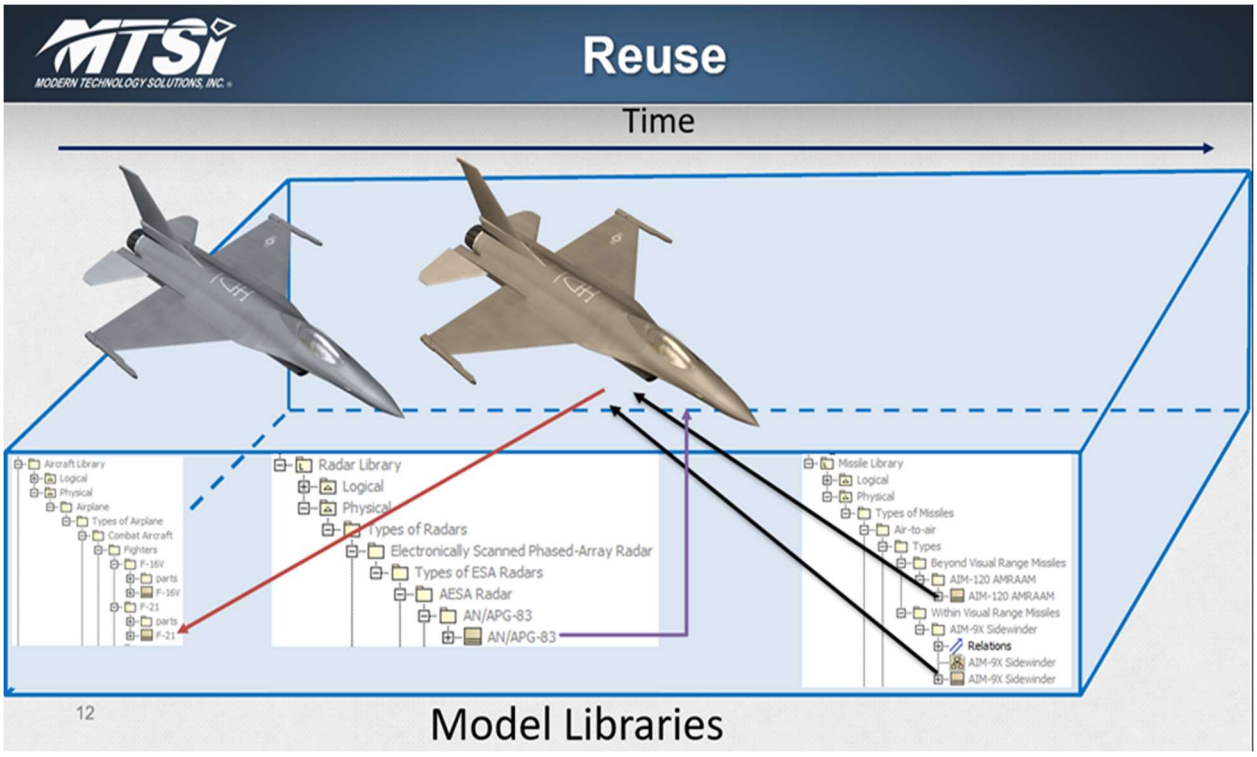


Figure 12 conveys what happens when we receive a task to define an F-21 model. Because the radar library has been previously defined and because the missile library has been previously defined down to the lowest-level logical element needed, i.e. AESA Radar, Beyond Visual Range Missile, and Within Visual Range Missile, we don't have to spend any time on defining the taxonomy again. Also, because the variants of the AN/APG-83, the AIM-120 AMRAAM, and the AIM-9x Sidewinder are also directly used in the F-21 design, no time must be spent developing the models from either of these which removes the architectural development time of those components completely from the project, which ideally reduces the total time the project takes and cost from the development of the F-21 model.

FIGURE 13
PARTIAL USAGE OF LIBRARY COMPONENTS

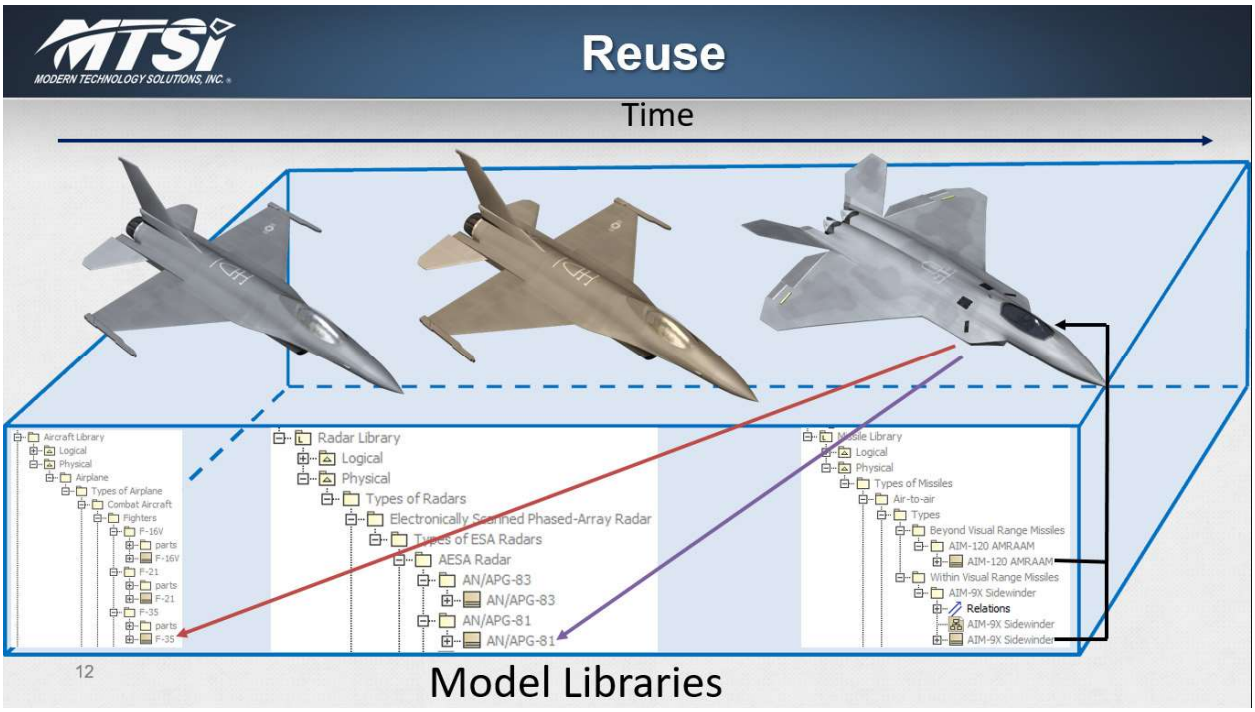


Figure 13 depicts a situation where we receive the task to develop a model for an element that can partially use the defined libraries. In the case of the F-35 it uses a different radar than the AN/APG-83, so we would need to define the AN/APG-81 within our radar library, though since it is an AESA radar, no updates must be made to the logical radar taxonomy. In addition, because the F-35 uses the same AIM-120 AMRAAM and AIM-9X Sidewinder missiles as were previously defined, no time must be spent defining these within our missile library. As such, the only time spent would be capturing the definition of the AN/APG-81 within our architecture, thus continuing to save time and money because of the library method.

APPENDIX B

FIGURE 14
DEFINITION OF TANK

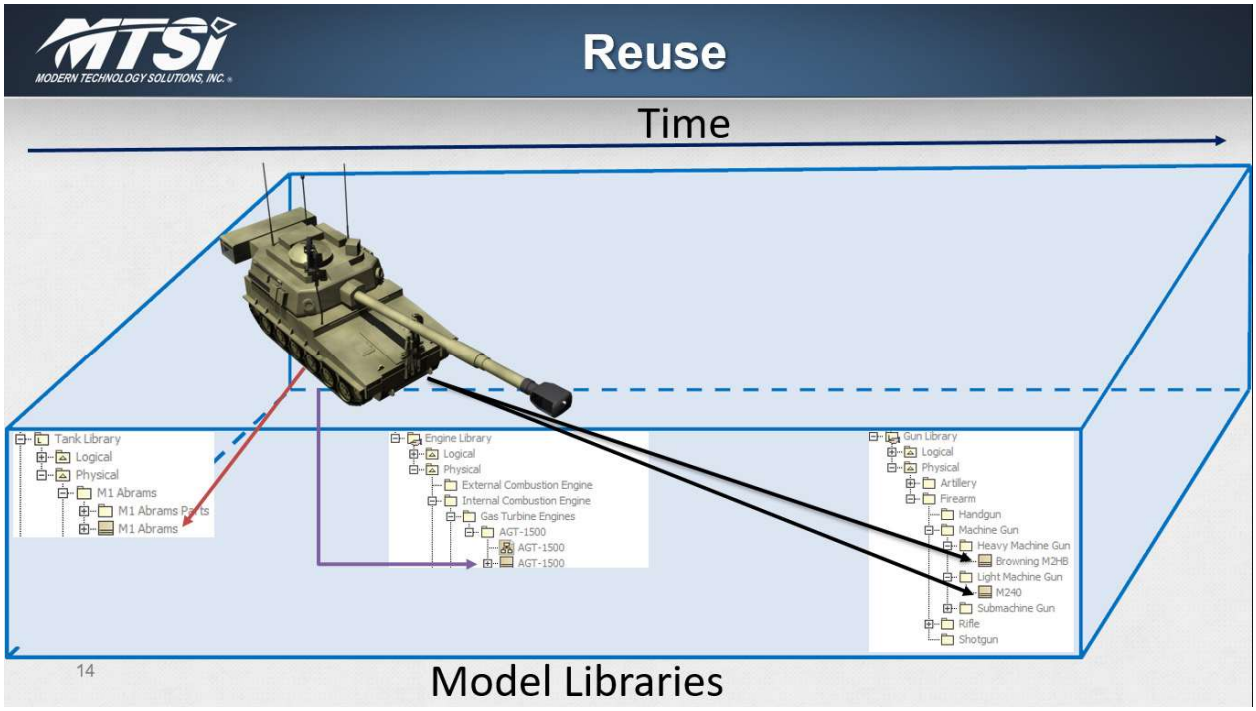
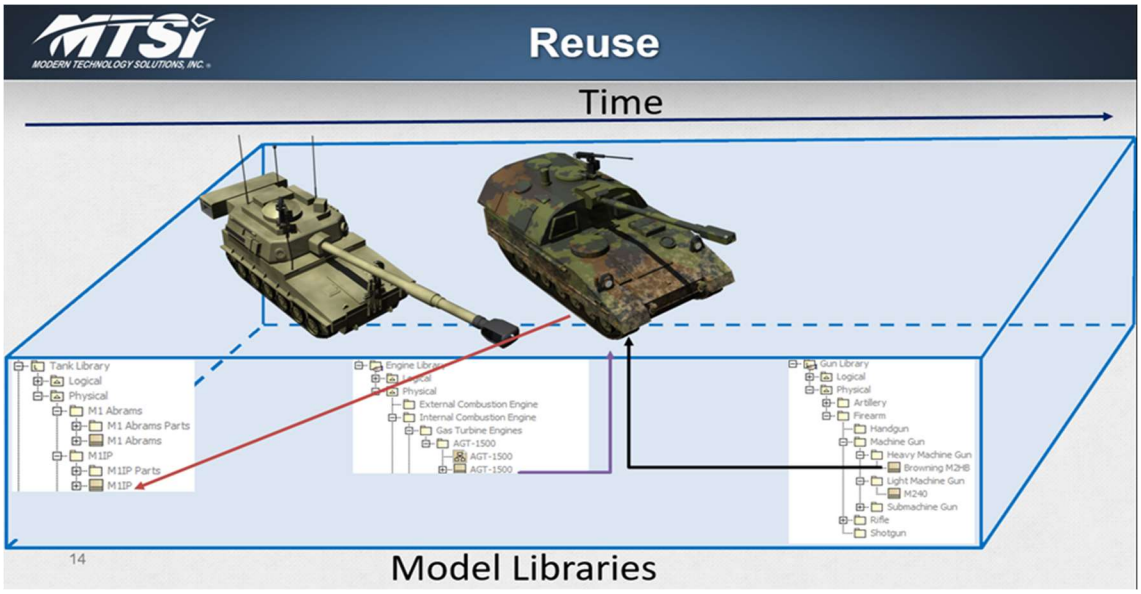


Figure 14 conveys that our MOM process can be used to define any kind system, including tanks. As mentioned in Appendix A, we start with an empty tank library where we'd classify the taxonomy of tanks within the logical model. Each tank must have an engine and a gun, as well as undepicted components, so we continue the thread by defining the logical taxonomy for engines, beginning at the logical definition of an engine itself, the sub-class of "Internal Combustion Engine", and its subclass of "Gas Turbine Engine". We perform the same operation for the guns where we define the logical "Gun" architecture, the subclasses of gun, i.e. "Artillery", "Firearm", and "Rifle". And then we define the subclasses of firearms which include "Heavy Machine Gun" and "Light Machine Gun".

We then define the variants of our system and components. We begin by defining the M1 Abrams tank block which uses the gas turbine engine "AGT-1500", the heavy machine gun "Browning M2HB", and the light machine gun "M240".

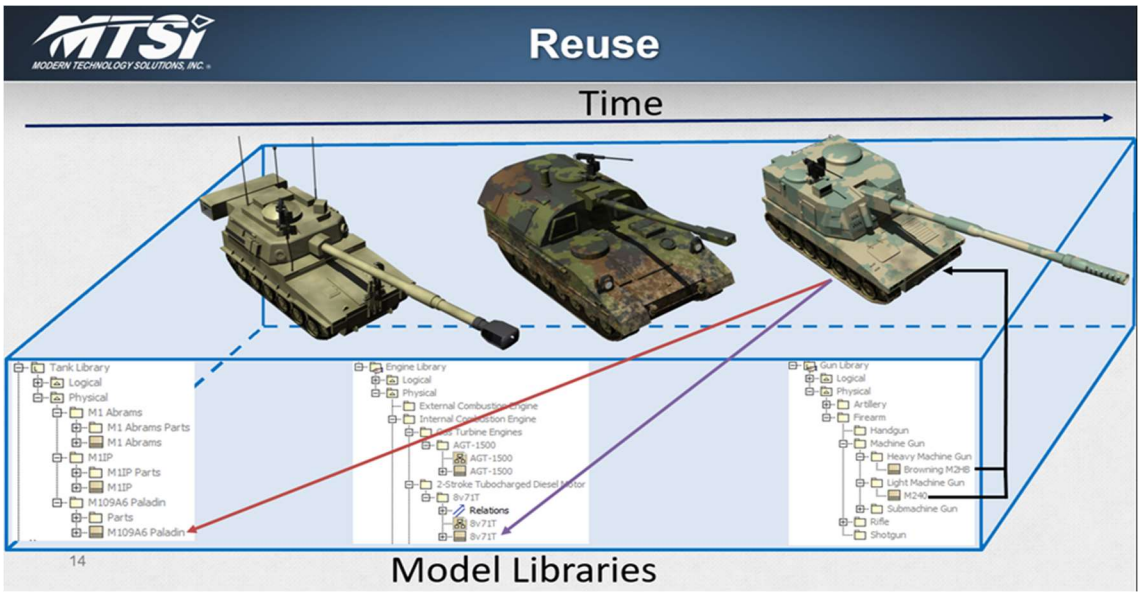
As mentioned in Appendix A, this process is recursive and can be defined through the components making up the engine or the guns, and the components for the components making up the engine and/or the guns, as deeply as needed.

FIGURE 15
REUSE OF ENGINE AND GUN



We then capture another tank, the M1IP, for which we have the convenience of reuse of the AGT-1500 engine and the Browning M2HB gun. In this case, we save time and cost that would be spent on rearchitecting these systems by pulling them straight from our defined libraries.

FIGURE 16
PARTIAL REUSE



Finally, we build the M109A6 Paladin. This tank uses a 2-Stroke Turbocharged Diesel Motor, which would require additional taxonomy definition in the logical model but uses the same guns. As such, we would spend time during the process to define the updates to the taxonomy but would require no additional time defining the guns. As such, we continue to save time through use of the libraries, and capture a new type and taxonomy of engine that will be used in future tank designs that require a 2-stroke turbocharged diesel motor.