Making the Case for Quality Metrics for Conceptual Models in Systems Engineering

Ronald E. Giachetti
Department of Systems Engineering
Naval Postgraduate School
Monterey, CA USA
Email: regiache@nps.edu

***Abstract*—The adoption of model based systems engineering takes models and puts them front and center to support all systems engineering activities. The majority of the models are static models describing some aspects concerning the structure of the system. This paper addresses the question of what makes a good model. The paper discusses modeling languages and how quality is defined and measured in software engineering where much more work has been done. We propose developing metrics based on the systems engineering activity models to supplement quality metrics borrowed from software engineering.**

I. Introduction

Systems engineering is largely performed by engineers interacting with models. Engineers create, analyze, and communicate system design ideas and information via models. Engineers have a long history of using models to support their work. What is new is the desire to formalize the use of models in engineering and elevate them as the main means for the representation, storage, and communication of system information. We term this Model-Based Systems Engineering (MBSE).

As models assume a more central and critical role in the design and development of systems, the community needs to consider the question of what makes a good model? The question of what makes a model good has largely been unasked in the systems engineering community in spite of the trend toward model-based systems engineering.

Addressing the goodness of models is important because of several reasons. First, the engineering community has a diverse set of models available, often to represent the same phenomenon. It is likely some models are better than others at representing a particular system aspect. Second, it is likely the models available to us influence how we think about systems [1]. Ferguson [2] makes a strong case that engineers think nonverbally as evidenced by the use of drawings ranging from sketches to more formal blueprints. These drawings shape our thinking about systems. No doubt models also shape our thinking. Lastly, understanding the goodness of models can help tool developers to improve the tools we use in systems engineering.

The remainder of the paper addresses the question of what makes a model good and how can we assess it? The paper first reviews the literature on model verification, validation, and quality metrics. We consider the closely related work done in software engineer and assess whether its applicable to systems engineering. We propose a quality model to analyze the quality of systems engineering models. Our work contributes a set of quality metrics based on the pragmatics of the modeling language in terms of its support for systems engineering activities.

II. Model Based Systems Engineering

Model based systems engineering (MBSE) uses models to support the entire systems engineering process. The models are computer interpretable, which means all the model data is stored in a database and available for reuse in multiple models or views. Advocates of MBSE purport it provides many benefits to communication, efficiency, effectiveness, and traceability.

Engineers face a decision about which modeling language to use in MBSE. Currently, popular available choices include SysML, LML, DoDAF as well as others built into tools such as Vitech Core's schema. Our focus on the goodness of models starts with understanding the structure of modeling languages.

III. Modeling Language

A modeling language consists of a set of modeling constructs capable of representing relevant domain concepts, a description of the construct's attributes, and the interrelationships between them [3]. The modeling language has a grammar, also called syntax, describing the rules for forming valid combinations of the constructs. A modeler uses the modeling language to create models as combinations of constructs according to the language's rules in order to represent concepts or actual artifacts. Ideally, a model construct has an isomorphic mapping to a real-world artifact.

It is possible in languages such as SysML to build a model that is correct per the rules but suffer from ambiguous semantics. Many researchers seek highly formal syntax and semantics in order to avoid such ambiguity because the models in MBSE need to be computer interpretable and exchangeable between software tools [4] [5]. Ontologies offer an approach to formalize a modeling language for systems engineering because they define the concepts and relationships in a domain [6] [7].

We design modeling language for a purpose, and the criteria to evaluate the language must be with respect to its purpose.

Most work on assessing the goodness of models involves verification and validation of simulation models. In this case the criteria of what constitutes "good" is clear because we want to use models to analyze and/or predict the performance of a system. The concept of model validation is not suited to the large number of system architecture models forming a part of MBSE. SysML structure diagrams (block definition diagram, internal block diagram, and package diagram)and requirement diagram are examples of models where traditional notions of validity do not apply. Yet these models are important to MBSE, and the community should be asking how do we know they are good?

How engineers use modeling languages may be markedly different than the goals of formal notation and semantics for computer interpretation and interoperability. Ralph Johnson says, from a relativistic perspective, "Architecture is a social construct" because it depends on what the system developers think is architecturally significant for inclusion [8]. Others have expressed similar thoughts and findings. Bucciarelli [9] argues there is no single universal and shared truth that all those on a multi-disciplinary design team agree to. It is via a myriad of means including sketches, verbal, graphs, and negotiation that designers are able to communicate about the design. A recent study of how engineers communicate about system functions found practitioners use multiple definition of function almost simultaneously [10]. Such studies suggest that engineers are able to handle ambiguity and it might even be useful during the system design phase. This argues against too much formalization in a modeling language because it might constrain the expressiveness of the engineering.

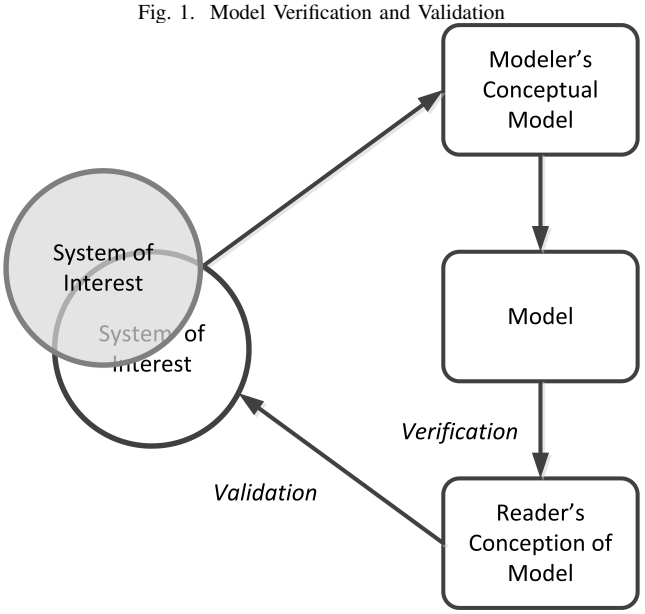
IV. Engineering Models

An engineering model is a representation of a real system. A wide variety of models are used in systems engineering making it necessary to offer a classification because the quality issues are different for each type of model. Broadly we classify a model as analytical, computational, or conceptual. Analytical models are math models and may be deterministic or stochastic. Physics-based models are an important subclass of analytical models in which the model is based on equations describing the underlying physics of the system. Computational models exploit the power of the computer to do many calculations and simulate system behavior. Many subclasses of computational models exist including discrete event simulation, agent-based simulation, and continuous simulation models. Some computational models are used to approximate physical processes such as finite element analysis, computational fluid dynamic, and other similar models.

A conceptual model represents concepts and the relationships between concepts in a visual format. Many of the system engineering architectural models are conceptual models including SysML requirements models, functional flow block diagrams, SysML diagrams, flow charts, and most all DoDAF models. Conceptual models are useful artifacts in system development programs because they facilitate communication of complex ideas among the many program stakeholders. For example, a block diagram showing how the system is partitioned into subsystems and the interactions between those subsystems is a valuable tool to communicate design intent. A primary advantage of conceptual models over natural language is the visualization is a more efficient way to communicate complex ideas. Other benefits, and the motivation for MBSE, include a more formal language that is computer implementable, ability to connect multiple models in a central repository available to all team members, and the availability of tools for automating various types of analysis or verification since the models are computer readable.

The traditional perspective of model quality is of model verification and validation. A modeler examines a system of interest (SoI) and through the modeling activity creates a conceptual model consisting of the modeler's assumptions, intended purpose, model constructs, and their meanings. The modeler then constructs the model in the computer or graphically. Verification is the activity of determining whether the model as constructed accurately represents the conceptual model. Validation is the activity of determining whether the constructed model represents the SoI to a degree sufficient for the model's purpose. Figure 1 illustrates the process. MBSE is done in teams, and models are a medium of communication, which adds the issue of interpretation of a model such that it does not reflect the modeler's intent. Figure 1 allows for this condition by showing the SoI as two different elements: one observed by the modeler and one as interpreted by the model user.

Fig. 1. Model Verification and Validation



Verification and validation of models has been defined in the context of analytical, physics-based and simulation models [11] [12]. One of the most accepted validation technique and the one that gives the greatest confidence is statistical comparison of the model's output data with actual data. The

verification and validation approaches are not possible for descriptive models. Descriptive models are different because there is no output data to compare with actual data.

Some researchers leverage the accepted verification and validation methods for simulation models by converting SysML behavioral models into executable models such as Markov chains or simulation models and then verify and validate the simulation model [13]. Another approach is checking the model to find errors oftentimes using formal logic with strong language syntax and semantics [15] [14]. Such techniques ensure model verification, but not necessarily validation, although they can determine whether a structure or behavior sequence is sound. Moreover, these techniques are used for the SysML behavioral models only, and do not address the issue with respect to the structural models.

It remains unclear whether verification and validation are appropriate means to think about the quality of static models describing system structure or architecture. In these cases, we should be asking what is the purpose of the model and how well does the model serve that purpose? To answer the question we need criteria. The next section discusses how the software engineering community has addressed the issue.

V. Quality Metrics

The software engineering community has given much more attention to the quality of conceptual or descriptive models. Models of software serve multiple purposes, but in the end they must meet formal requirements for being able to convert models to correct and reliable executable code. Table I shows the criteria from three sources side-by-side. Paige et al. [16] adopt previously debated principles of programming languages to modeling languages for software engineering. They observe that modeling languages are essentially no different than programming language, they are designed and consequently there are goals for the language. The goodness of the language is how well it enables us to develop systems. They derive principles of what constitutes a good modeling language and use the principles to evaluate UML. Many others in software engineering have identified and discussed quality criteria within that domain. Friedenthal et al. [17] is one of the few to discuss quality attributes in the context of modeling in systems engineering. Their analysis is in the context of SysML, but there is nothing unique about SysML preventing the criteria's use for other modeling languages. In another article, Friedenthal and Burkhart [18] list without much discussion criteria of a modeling language as being expressive, precise, communicative, support efficient and intuitive model construction, interoperable, manageable, usable, and customizable. The software community coalesced on the international standard ISO/IEC 9126, which defines six quality attributes and a process model to evaluate software using the quality attributes.

Lindland et al. [19] observe there are many lists of quality attributes often well argued for, but lacking an overall framework. The authors present a framework based on semiotics to classify the quality aspects into syntactic (adherence to the language rules or syntax), semantic (the meaning and relevance of the concepts within a problem domain) and pragmatic (understandability of a model by stakeholders). Of interest herein is the semantic and pragmatic goals. The semantic goals is whether the modeling language can represent concepts in the domain and the completeness of the representation. The semantic goal is how ontologies are evaluated, to what degree do they represents concepts in a domain? In our view, syntax and semantics correspond to traditional notions of verification and validation. Pragmatics is the new dimension because it involves the human users of the model in definition of quality.

TABLE I
Quality Criteria

Paige	Friedenthal	ISO/IEC 9126
Simplicity	Purpose well-defined?	Functionality
Uniqueness	sufficient scope	Reliability
Consistency	model fidelity	Usability
Seamlessness	completeness relative to scope	Efficiency
Reversibility	well-formed	Maintainability
Scalability	internal consistency of the model	Portability
Supportability	understandability of the model	
Reliability	accurate or valid model of domain of interest	
Space economy		

Pragmatic quality is the degree to which users understand the models. Suitable means to measure pragmatic quality would be experiments with human subjects or analysis of model constructs support for the engineering activities.

The quality criteria can be highly subjective and open to interpretation. For example, Vaneman [20] finds SysML to be arduous to use and therefore falling short of being usable. No doubt, there are many others who would find SysML to be highly usable. A second caution about using quality metrics is it assumes a generic purpose for all modeling languages. What is missing is the evaluation of the language with respect to its stated purpose. Lastly, any evaluation will be difficult because system modeling languages have many purposes as well as many unattributed uses beyond their design intent.

VI. Differences between Software and Systems Engineering

Before adopting quality metrics from software engineering to systems engineering, we will examine how the two disciplines are different. The artifact of software engineering is a conceptual object, i.e., software, without any physical manifestation. In software engineering it is possible to have code generators that automate the step of converting a conceptual model into code. For example, a UML class model shows the software's classes, attributes, and how the classes are related. The implementation of the model in a programming language such as Java will have classes, attributes, and relationships corresponding to the model. In fact, the isomorphism is an important property of the mapping. The programming language is equivalent to a modeling language except at a

different level of abstraction. Two implications of the relationship between conceptual model and computer programming code is first we can partially automate code generation with benefits of efficiency and better quality. Second, the fact that model and program are both conceptual objects helps explain why software engineers thought of adopting principles for programming language design to modeling language design.

Systems engineering is different because our conceptual models represent physical artifacts. Systems engineering conceptual models do not in general have any isomorphism with the real objects. In fact, a gap remaining in MBSE is bridging the divide between the conceptual and mostly descriptive models of system architecting and the physics-based and computational models for analyzing system designs. Tool providers such as Rational System Architect have developed code generators to turn conceptual activity models into executable simulation models. The linking of the conceptual models to physics-based and other analyitcal models is largely incomplete [21]. As a result we do not have the same connection from model to artifact that software engineering does. Part of model quality should assess whether our design models lead to functioning systems. After all, a challenge for design is how do designers know whether an artifact conforming to the specifications they design will perform as intended? [22]

One of the major system modeling language efforts has been SysML, which borrows heavily from UML. A result of borrowing heavily from UML is adopting the underlying concepts and implied methods that guided the development of UML

A. Modeling Language Goals

SysML is designed to provide a "... simple but powerful constructs for modeling a wide range of systems engineering problems. It is particularly effective in specifying requirements, structure, behavior, allocations, and constraints on system properties to support engineering analysis." [23]. SysML was derived by committee from UML 2.0 with one of its goals to reuse UML as much as possible.

The Lifecycle Modeling Language (LML) states the following goals [24]

1) to be easy to understand by engineers and other stakeholders
2) to be easy to extend
3) to support both functional and object-oriented approaches
4) to support activities in the entire system life-cycle
5) to support both evolutionary and revolutionary system changes to system plans and designs over the lifespan of the system.

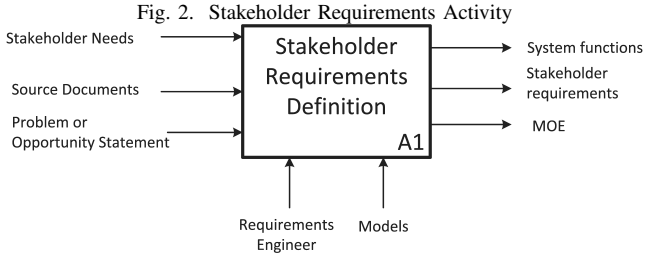
In neither case do the language developers state any quality attributes of how well the language serves the goals. Also, in both cases the statement of purpose is very broad in both cases. UML discusses the early development phase activities and LML says all activities in the life-cycle.

VII. Quality Evaluation via Process Model

The primary consideration for the quality of a model should be how well does the model support the systems engineering activities. Each activity has certain information needs, and the quality of the modeling language is whether it satisfies those information needs. The measurement process should identify for each activity the required information entities and attributes. Additionally, the modeling language should support integration among all the activities primarily through traceability.

Figure 2 shows a systems engineering activity and the inputs and outputs according to INCOSE's definition of the stakeholder requirements activity. The measure of quality for the modeling language is how well does it support the activity? The quality measures need to qualify how well the model supports the output by further elaborating what are the attributes of the outputs, and whether they are represented in the models.

Fig. 2. Stakeholder Requirements Activity



In the case shown, one of the main outputs is stakeholder requirements. A requirement is a declarative statement of what the system needs to do using particular phraseology such as "the system shall ..." In a US Department of Defense program, we need to specify threshold and objective values for the requirement. Looking at the set of requirements, we need to be able to represent the relationships between the requirements. The life-cycle view tells us we want to support traceability of the requirement to its source and later to how it is implemented by the system. We also need to know how the requirement is validated and verified. In summary, the process perspective of evaluating the quality of the modeling language examines how well the modeling language supports the process and its activities.

The other quality metrics from the software domain let us evaluate how good is the modeling language in terms of its usability, scalability, portability, and the other quality aspects. These are qualities deemed important for any modeling language.

An important quality metric, especially for engineering of systems of systems is the interoperability of the models. Model interoperability has long been a challenge in engineering and is a well studied problem. Much of the work is in the simulation domain, but the concepts are applicable to all types of engineering models. The challenge is greater than the exchange of data, it also requires understanding the

epistemology underlying the models. One such approach is the levels of interoperability model [25].

VIII. Proposed Quality Framework

We propose a quality framework that adapts the quality metrics from software engineering, but also adds quality metrics from a pragmatic process perspective. Adopting the quality metrics from software engineering is reasonable because modeling in both disciplines involves conceptual models and we share many of the same goals of understandability, etc. For example, Bonnema argues communication is one of the most important purposes of architecture models [26]. However, for systems engineering models to be useful and to derive the benefits of MBSE, our models must support the systems engineering process. For this reason, an important quality metric is how well a set of models supports each systems engineering activity. One approach is to take the activities as defined by INCOSE and check against them. However, this also suggests an approach whereby the MBSE method is defined with activities that take advantage of particular model characteristics while mitigating where models fall short. Certainly, further research is warranted to explore these issues surrounding model quality.

IX. Conclusion

This paper discussed the issues surrounding determination of what makes a good model. The article found there is little discussion within the systems engineering community of what makes a good model. Measurement is one of the first steps towards better understanding a phenomenon, and the measurement of quality of models in systems engineering would help the field. In the related field of software engineering there is a lot of work and even a standard for measuring the quality of a model. This article proposes to adopt many of the quality attributes from software engineering but to also use the systems engineering process model to identify the information requirements for performing model-based systems engineering.

References

[1] Giachetti, Ronald E. Evaluation of the DoDAF Meta-model's Support of Systems Engineering. *Procedia Computer Science* 61 (2015): 254-260.

[2] Ferguson, Eugene S. *Engineering and the Mind's Eye*. MIT press, 1994.

[3] Guizzardi, Giancarlo. On ontology, ontologies, conceptualizations, modeling languages, and (meta) models. *Frontiers in artificial intelligence and applications* 155 (2007): 18.

[4] Nutting, Joseph W. Examination of modeling languages to allow quantitative analysis for model-based systems engineering. Diss. Monterey, California. Naval Postgraduate School, 2014.

[5] Reichwein, Axel, and Christiaan JJ Paredis. Overview of architecture frameworks and modeling languages for model-based systems engineering. *ASME 2011 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*. American Society of Mechanical Engineers, 2011.

[6] van Ruijven, L. C. Ontology for systems engineering as a base for MBSE. *INCOSE International Symposium*. Vol. 25. No. 1. 2015.

[7] Jenkins, Steven, and Nicolas Rouquette. Semantically-rigorous systems engineering using SysML and OWL. (2012).

[8] Fowler, M., Who needs an architect? *IEEE Software*, (2003) 2-4.

[9] Bucciarelli, Louis L. Between thought and object in engineering design. *Design studies* 23.3 (2002): 219-231.

[10] Eisenbart, B., Gericke, K., and Blessing, L. T. (2016). Taking a look at the utilisation of function models in interdisciplinary design: insights from ten engineering companies. *Research in Engineering Design*, 1-33.

[11] Kleijnen, Jack PC. Verification and validation of simulation models. *European journal of operational research* 82.1 (1995): 145-162.

[12] Sargent, Robert G. Verification and validation of simulation models. *Proceedings of the 37th conference on Winter simulation*, 2005.

[13] Debbabi, Mourad, et al. *Verification and validation in systems engineering: assessing UML/SysML design models*. Springer Science and Business Media, 2010.

[14] Giammarco, K., Auguston, M., Well, you didn't say not to! A formal systems engineering approach to teaching an unruly architecture good behavior, *Complex Adaptive Systems Conference*, November 13 - 15, 2013, Baltimore, MD

[15] Rodano, Matthew, and Kristin Giammarco. A formal method for evaluation of a modeled system architecture. *Procedia Computer Science* 20 (2013): 210-215.

[16] Paige, Richard F., Jonathan S. Ostroff, and Phillip J. Brooke. Principles for modeling language design. *Information and Software Technology* 42.10 (2000): 665-675.

[17] Friedenthal, Sanford, Alan Moore, and Rick Steiner. *A practical guide to SysML: the systems modeling language*. Morgan Kaufmann, 2014.

[18] Friedenthal, Sanford, and Roger Burkhart. Evolving SysML and the System Modeling Environment to Support MBSE. *Insight* 18.2 (2015): 39-41.

[19] Lindland. O,I., Sindre, G., Slvberg, A.: Understanding Quality in Conceptual Modeling, *IEEE Software* 11(2), pp. 42–49 (1994)

[20] Vaneman, Warren K. Enhancing model-based systems engineering with the Lifecycle Modeling Language. *Systems Conference (SysCon)*, 2016 Annual IEEE. IEEE, 2016.

[21] Beery, Paul T. A model-based systems engineering methodology for employing architecture in system analysis: developing simulation models using systems modeling language products to link architecture and analysis. Diss. Monterey, California: Naval Postgraduate School, 2016.

[22] Galle, Per. Foundational and instrumental design theory. *Design Issues* 27.4 (2011): 81-94.

[23] OMG Systems Modeling Language, version 1.4.

[24] Lifecycle modeling language (LML) specification, December 1, 2015.

[25] Wang, Wenguang, Andreas Tolk, and Weiping Wang. The levels of conceptual interoperability model: applying systems engineering principles to modeling and simulation. *Proceedings of the 2009 Spring Simulation Multiconference*. Society for Computer Simulation International, 2009.

[26] G. Maarten Bonnema, Communication in Multidisciplinary Systems Architecting, *Procedia CIRP*, Volume 21, 2014, Pages 27-33.