“Are we there yet?” Assessing Quality in Model Based Systems Engineering

Matthew Hause
Atego
5930 Cornerstone Court West, Suite 250, San Diego, CA 92121
Matthew.Hause@Atego.com

Abstract. How do I know that my model is of good quality?” This question is frequently asked when developing models. This is true whether developing Yourdon, IDEF0, UML, SysML, UPDM, or Lego models. The purpose of the model is to fulfill a need or to answer questions. Consequently, before starting the modeling exercise, one has to determine the questions that need to be answered and assessment criteria. Possibilities include requirements traceability, performance modeling, simulation, behavioral modeling, human factors, etc. Other factors will also affect the scope of the model. Does the model need to capture the complete development lifecycle or product lifecycle? This paper will first examine the concept of quality in general and as applied to modeling. Next we will look at the different reasons for modeling, and some of the corresponding characteristics of quality that can be applied to ensure that the model is of “good quality”.

Introduction

“How do I know that my model is of good quality?” This question is frequently asked when developing models, and has often been asked of the author when assisting on projects.. This is true whether developing Yourdon, IDEF0, UML, SysML, UPDM, or other modeling notations. To answer this question one needs to understand the purpose of the modeling exercise. One of the basic rules of modeling is that the purpose of the model is to fulfill a need or to answer questions. Consequently, prior to starting the modeling exercise, one has to determine the questions that need to be answered. Otherwise the model is unlikely to serve any useful purpose. For example, a model can be created from a requirements specification in order to clarify the requirements, remove inconsistency, communicate them more effectively, and ensure that they are complete, coherent, and consistent. Another model can be created from the same requirements specification that provides a means of analyzing the requirements in order to specify a system or system of systems that will satisfy or realize the requirements. A third model may implement the requirements specification. This is often the case for software based systems using automatic code synchronization, (ACS) or Model Driven Architectures (MDA). There are a number of other possibilities such as performance modeling, simulation, behavioral modeling, human factors, etc. Other factors will also affect the scope of the model. Does the model need to capture the complete development lifecycle of the intended system – requirements through to implementation and installation? Does the model need to capture the complete lifecycle of the product – from initial concept to development, installation, maintenance and finally decommissioning? As each model has been created for a different purpose, each will have its own set of criteria for determining its level of quality. Without first defining the assessment criteria, it will be difficult to determine whether the model has

achieved its purpose. The modeling exercise could continue indefinitely. In this paper, we will first examine the concept of quality in general and as applied to modeling. For clarity there will be a brief summary of SysML in order to clarify the types of concepts that can be modeled using SysML and the typical applications for which it is used. Next we will look at the different reasons for modeling, and some of the corresponding characteristics of quality that can be applied to ensure that the model is of "good quality". The different concepts will be illustrated with worked examples based on over 30 years of experience of modeling and implementing complex systems.

What is Quality?

There are many different ways of looking at quality: from a general point of view, from a systems development point of view, etc.

Dictionary definitions

The problem with quality is that it means different things to different people. A standard dictionary definition of quality is the following: "1. A distinguishing characteristic or attribute. 2. The basic character or nature of something. 3. A feature of personality. 4. Degree or standard of excellence. 5. High social status. 6. Musical tone color. 7. Excellent or superior; a quality product." (Collins, 1993). Definitions 4, 5 and 7 define quality as a something desirous to be obtained. The others are more neutral and describe quality as something that can be either good or bad. A set of thesaurus synonyms is similar listing: character, sort, tendency, excellent and goodness (Roget, 1987). When dealing with systems and models, the question of quality is almost always how to attain the appropriate level for the task at hand. Consequently, it is necessary to define the level of quality required at each stage of the development process, what constitutes good quality, as well as what constitutes bad quality, and so on.

Quality in Industry

The American Society for Quality (ASQ) describes itself as a "global community of experts and the leading authority on quality in all fields, organizations, and industries." The ASQ defines quality as a "subjective term for which each person or sector has its own definition. In technical usage, quality can have two meanings: 1. the characteristics of a product or service that bear on its ability to satisfy stated or implied needs; 2. a product or service free of deficiencies. According to Joseph Juran, quality means 'fitness for use;' according to Philip Crosby, it means 'conformance to requirements.'" (ASQ, 2011)

Quality in Systems Development

In systems and software development, quality also has a variety of meanings. In his Planguage concept glossary, Tom Gilb describes quality as "a scalar attribute reflecting 'how well' a system functions. Examples include Availability, Usability, Integrity, Adaptability, and many others." (Gilb, 2010) He then goes on to list numerous attributes of quality such as:

- Quality levels are capable of being specified quantitatively (as are all scalar attributes)
- Quality levels can be measured in practice
- Quality levels can be traded off to some degree; with other system attributes valued more by stakeholders
- When quality levels increase towards perfection, the resources needed to support those levels tend towards infinity

The above points emphasize that quality can be quantified and that the process owners need to understand how and to what extent quality requirements will be realized during the development process. Brooks (1995) describes the relationship between quality and

productivity restating Capers Jones advice to “focus on quality and productivity will follow.” However, he then goes on to warn that productivity drops as one pursues extreme quality, restating the warning about pursuing the right level of quality. “How do I know that my model is of good quality?”

Quality in Models

The phrase "Essentially, all models are wrong, but some are useful" has been attributed to Professor George E. P. Box, (1987). This is appropriate because modeling by its very nature creates an incomplete replica of the problem or system. (A complete replica would of course be reality itself.) Modeling systems is based on abstracting the characteristics of a domain of interest (e.g. the problem or the solution) to the modeler and ignoring the others. In computer science, abstraction is the mechanism and practice of factoring out details so that one can focus on a few concepts at a time (Illingworth, 1991). Depending on the level of abstraction, and the focus of concern, there will be different viewpoints that can be applied to the system. In addition, each level, and each viewpoint will have its own unique elements. For SysML, some viewpoints will provide information to be allocated to several engineering domains (e.g. hardware, software, procedural or mechanical etc.). UML can be used to model the software aspects of the system. For modeling centered on a particular domain, modeling will involve “abstracting the abstraction” to take into account the information that is appropriate to the viewpoint and the domain. Additional information, application of standards, constraints, etc., will then be added to the model. In other words, systems, software and hardware engineers will all look at a problem from their own points of view.

Model-Based Engineering

“Model-based Systems Engineering (MBSE) is the formalized application of modeling to support system requirements, design, analysis, verification, and validation activities beginning in the conceptual design phase and continuing through-out development and later lifecycle phases.” (INCOSE, 2007). Put simply, modeling is at the heart of all aspects of the development effort, covering the complete lifecycle, and has a direct effect on any generated artifacts. The acronym we are using, MBE, encompasses both systems and software development.

SysML and UML

The Unified Modeling Language (UML). The Object Management Group (OMG) (1999) specification states, “UML provides system architects working on object analysis and design with one consistent language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling”. UML consists of class, use case, component, deployment, state machine, sequence, timing, activity, package, communication, composite structure, interaction overview and instance diagrams. For more information on UML see (Fowler, 2004, OMG, 2007a).

The Systems Modeling Language (SysML)

In March 2003, the OMG issued a Request for Proposal (RfP) for a customized version of UML suitable for Systems Engineering written by the OMG Systems Engineering Domain Special Interest Group (SE DSIG). Friedenthal, Burkhart, (2003) gives early history on the development of the UML for SE RFP. The customization of UML for systems engineering supports modeling of a broad range of systems which may include hardware, software, data, personnel, procedures and facilities. The goal is to provide a “standard modeling language for systems engineering to analyze, specify, design and verify complex systems, intended to

enhance systems quality, improve the ability to exchange systems engineering information amongst tools and help bridge the semantic gap between systems, software and other engineering disciplines" (OMG SysML, 2003). For the sake of brevity, and the fact that the UML specification is 1000 pages and the SysML specification is 300 pages, we will not attempt to describe the two languages in detail. For more information on SysML, see OMG, (2007b) and Hause, (2006a). For more information on UML, see OMG, (2007a).
SysML added two new diagrams which are the parametric and requirements diagrams that will be explored later. In addition, the class and composite structure diagrams were modified to include elements to model logical and physical block structures for systems engineers. These are the block definition diagram and the internal block diagram. For further information see (OMG, 2007b, Friedenthal, 2008, Holt, 2008, Korff, 2008). Figure 1 shows the four pillars of SysML, namely structure, behavior, requirements and parametrics.

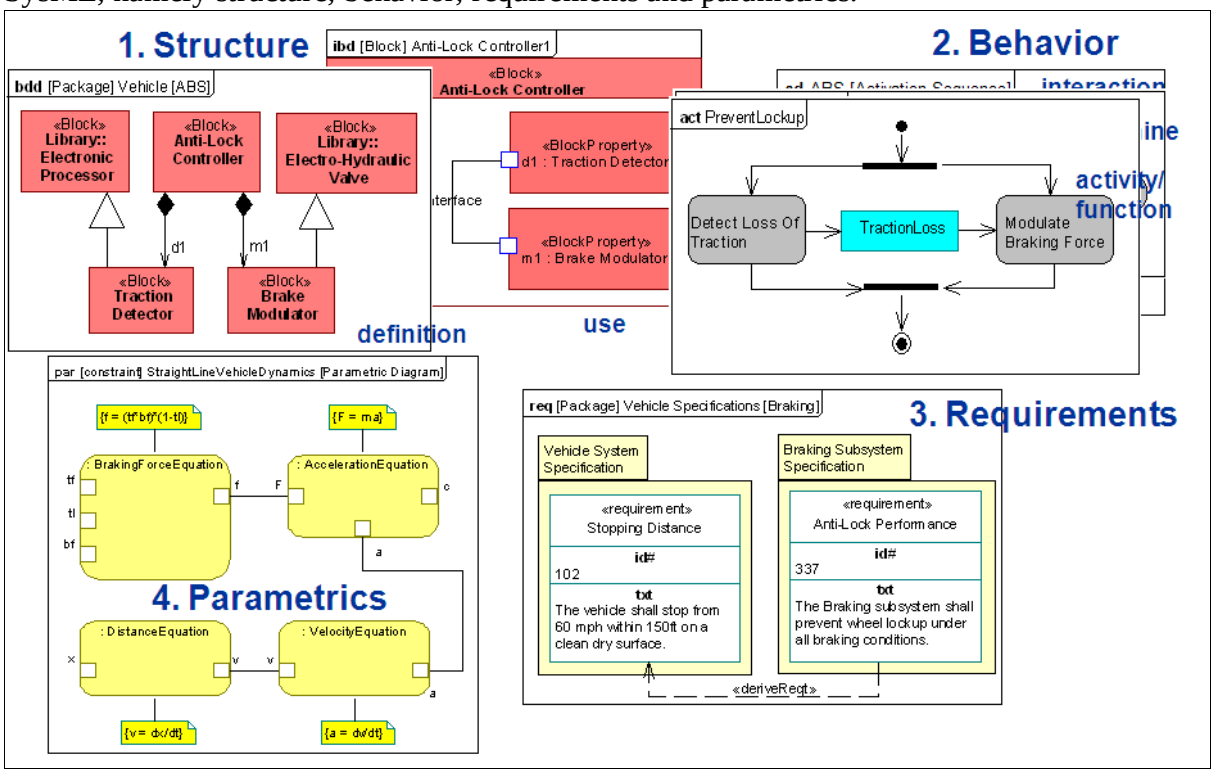

Figure 1. The Four Pillars of SysML

Examples of Model-Based Systems Engineering

The following are examples of some of the criteria to determine model quality. As this is a short paper, they have not been exhaustively developed, and are simply exemplars of types of quality that can be associated with models.

Ability to Communicate

As stated above, the "L" in SysML stands for language. The purpose of a language is to communicate thoughts, feelings, ideas, concepts, etc., but above all to communicate. Consequently, to communicate effectively one must always consider the audience or group of people with which one wishes to communicate. Consider the diagram in Figure 2.

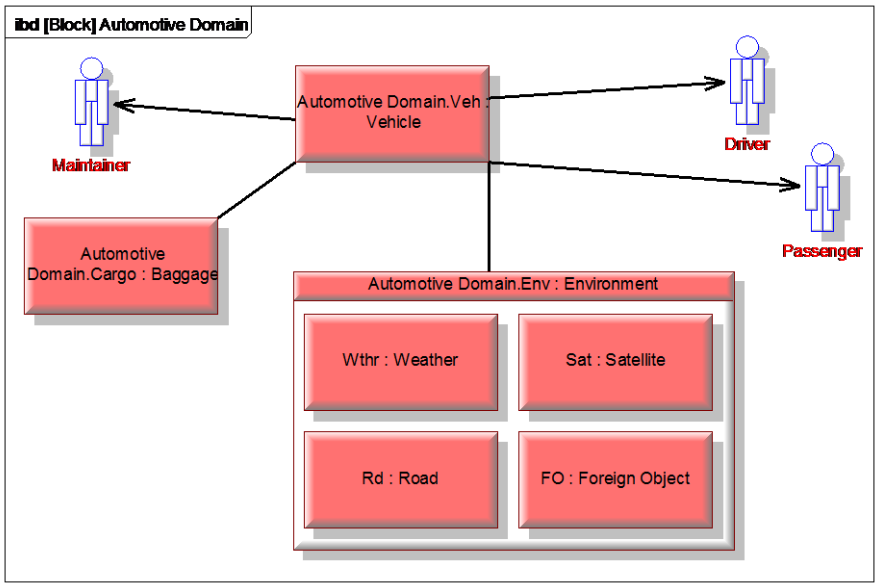

Figure 2. Automotive Domain SysML Internal Block Diagram (IBD)

Figure 2 contains the main elements of the automotive domain such as the vehicle, driver, passenger, maintainer, cargo, and environmental elements such as the weather, satellites, the road, and foreign objects. It is not effective however in quickly communicating these ideas. The user has to read the labels in the boxes to understand what each element is. If the purpose of the diagram is to quickly communicate these ideas to a non-technical audience, it will be difficult to capture their attention as the diagram just looks like boxes with lines between them. Figure 3 contains the same elements however; the boxes have been replaced with icons representing these elements. As a consequence, it is much easier to communicate the concepts involved to a non-technical audience. Even a technical audience can benefit from using icons such as mechanical devices, electrical and electronic components, etc. Libraries of these icons can be associated with stereotypes as well as an asset library to promote consistency, communication, and reuse.

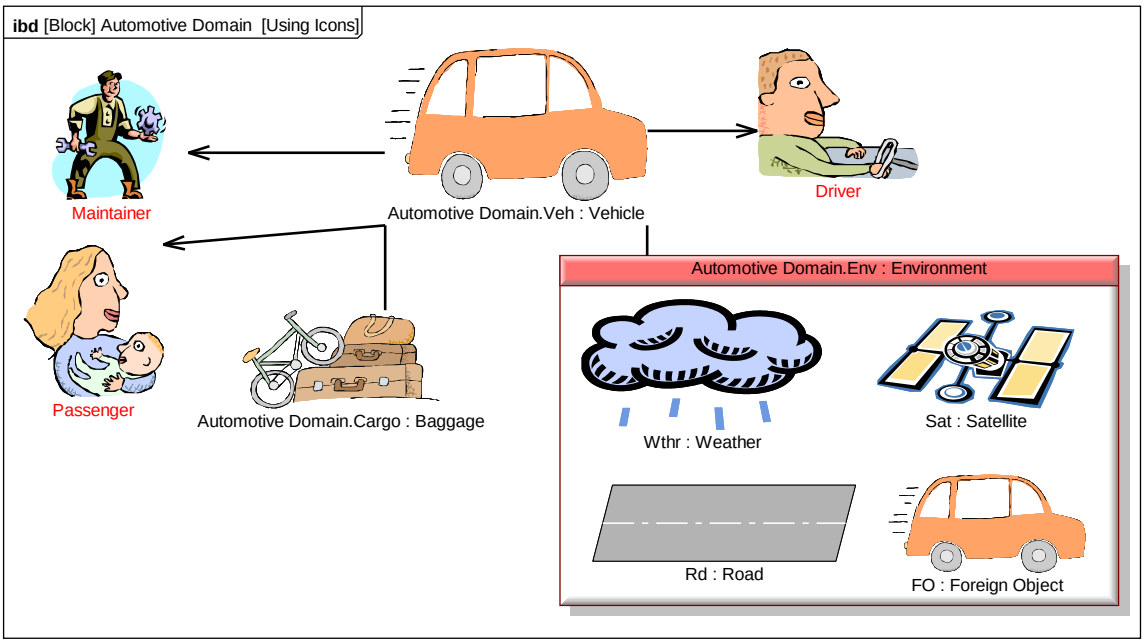

Figure 3. Automotive Domain IBD Using Icons.

Requirements Traceability

According to the ISO 9000 the definition of quality is the degree to which the inherent characteristics of process, product or system meet "The Requirements" is the quality of process, product or system, irrespective of the sub-classification or sub-categorization of "The Requirements". Quality is, therefore, a question of degree. As a result, the central quality question is: How well does this set of inherent characteristics comply with this set of requirements? In short, the quality of something depends on a set of inherent characteristics and a set of requirements and how well the former complies with the latter. According to this definition, quality is a relative concept. By linking quality to requirements, ISO 9000 argues that the quality of something cannot be established in a vacuum. Quality is always relative to a set of requirements. (Praxiom, 2010)

Consequently, demonstrating traceability to requirements within the model is essential. The SysML requirement diagram captures requirements hierarchies and the derivation, satisfaction, verification and refinement relationships. The relationships provide the capability to relate requirements to one another and to relate requirements to system design models and test cases. The «copy» relationship is used to show reuse of a requirement within a different requirement hierarchy. The «rationale» concept can be used to annotate any model element to identify supporting rationale including analysis and trade studies for a derived requirement, a design or some other decision. The requirement diagram provides a bridge between typical requirements management tools and the system models. Figure 4 shows a typical example of a requirements diagram.

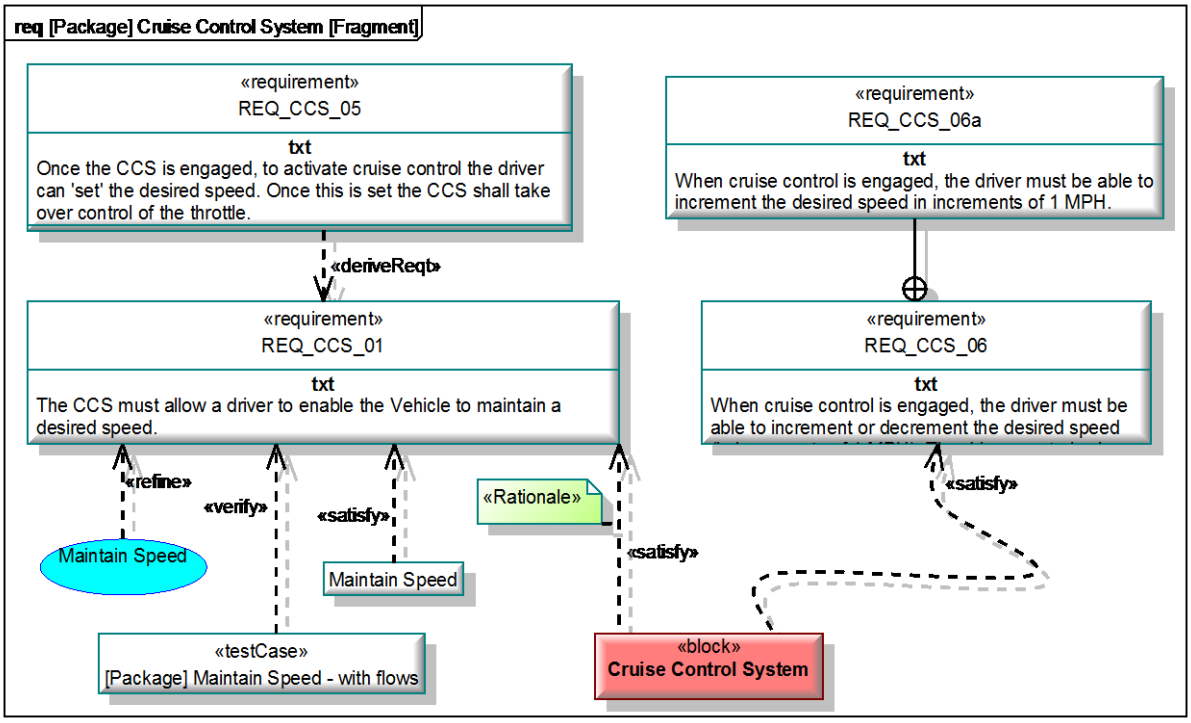

Figure 4. Requirements Diagram Fragment.

The requirements diagram shown above shows a requirement hierarchy along with satisfaction, derived, and other relationships. Quality tests for requirements can be as simple as testing to make sure all requirements are satisfied. Additional checks can specify that all satisfaction relationships are documented with a satisfaction argument or rationale, and that they are verified using a SysML test case. Even more complex checks can examine requirements hierarchies to examine aggregation relationships, as well as ensuring that all model elements can trace back to a requirement. As requirements can be categorized as functional,

non-functional, etc., tests can be made to ensure that the model element's type satisfying a requirement is appropriate. This is only possible because the requirements are integrated into the SysML/UML model.

SysML Parametrics

The parametric diagram represents constraints on system parameter values such as performance, reliability and mass properties to support engineering analysis. Parametric diagrams are used to describe constraints on system properties to support engineering analysis. In order to support this type of modeling a ConstraintBlock has been introduced into OMG SysML. A ConstraintBlock defines a set of parameters and one or more constraints on the parameters. By default, these parameters are non-directional and so have no notion of causality. These ConstraintBlocks are used in a parametric diagram to constrain system properties. ConstraintBlocks may be used to express mathematical equations such as 'F=m•a' and 'a = δv/δt', or statistical values and utility functions such as might be used in trade studies. Based on the reusable concept of a block new ConstraintBlocks can be built by reusing more primitive ConstraintBlocks such as basic mathematical operators. SysML also defines a model of value types that can have units and dimensions and probability distributions. The value types are used to type properties of blocks.

The Parametric Diagram is a specialized variant of an internal block diagram that restricts diagram elements to represent constraint blocks, their parameters and the block properties that they bind to. Both parameters and properties may be represented as small "pin-like" boxes to help make the diagrams more scalable. An example of the Parametric Diagram is shown in Figure 5.

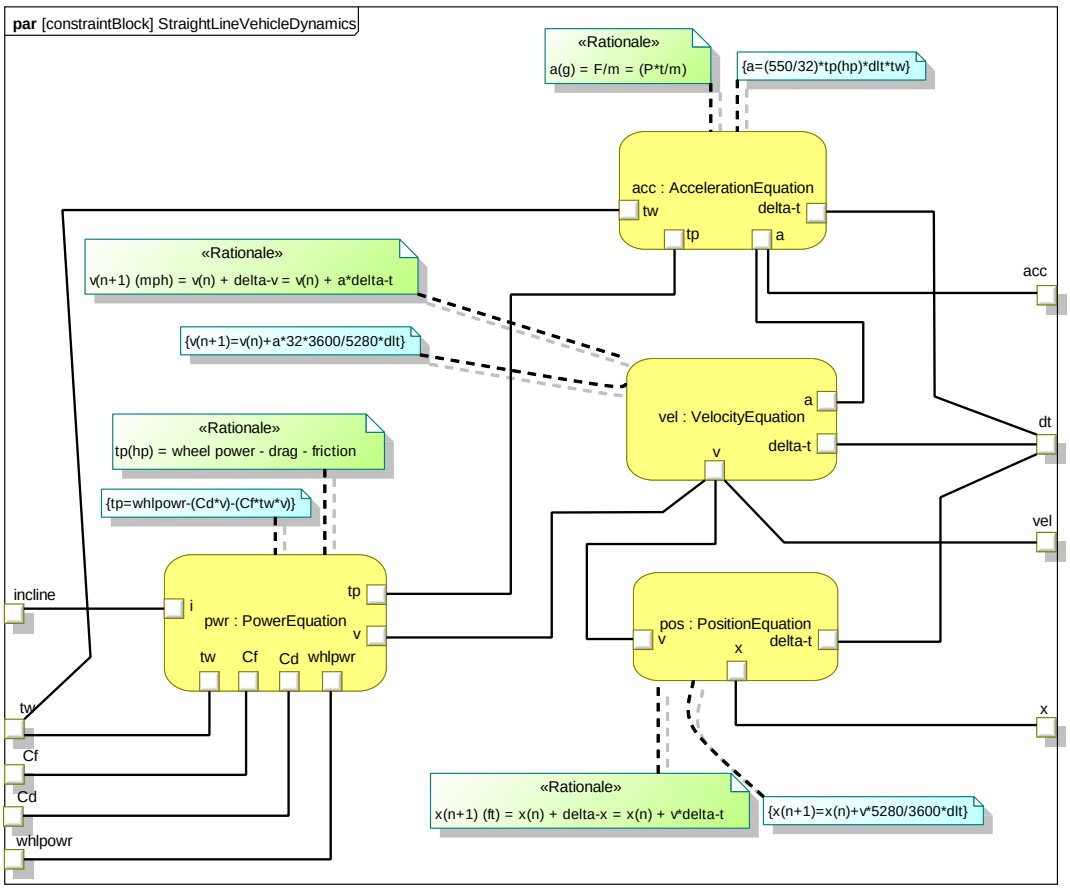

Figure 5. Parametric Diagram for a Braking Force Equation

Figure 5 shows a simplified braking force equation for automobile. The purpose of the diagram is to evaluate the total mass, braking force, tire friction, position, ABS duty cycle, etc. to determine how long it will take a car travelling at speed to come to a complete stop. In this case the quality of the model will be its ability to evaluate whether the car will meet certain safety standards. The purpose of the model is to determine whether the proposed car design itself will be of sufficient quality.

Measures of Effectiveness

A measure of effectiveness (moe) represents a parameter whose value is critical for achieving the desired mission cost effectiveness. It will also be assumed that the overall mission cost effectiveness can be determined by applying an objective function to a set of criteria, each of which is represented by a measures of effectiveness. Figure 6, which is taken from the SysML specification (OMG, 2007b), shows Fuel Economy, Quarter Mile Time, Zero to 60 time, Payload Capacity, and Unit Cost. The overall cost effectiveness for each alternative may be defined by an objective function that represents a weighted sum of their moe values. For each moe, there is a separate parametric model to estimate the value of operational availability, mission response time, security effectiveness and life cycle cost to determine an overall cost effectiveness for each alternative. It is assumed that the moe's refer to the values for system alternative HSUV alt1. (OMG, 2007b)

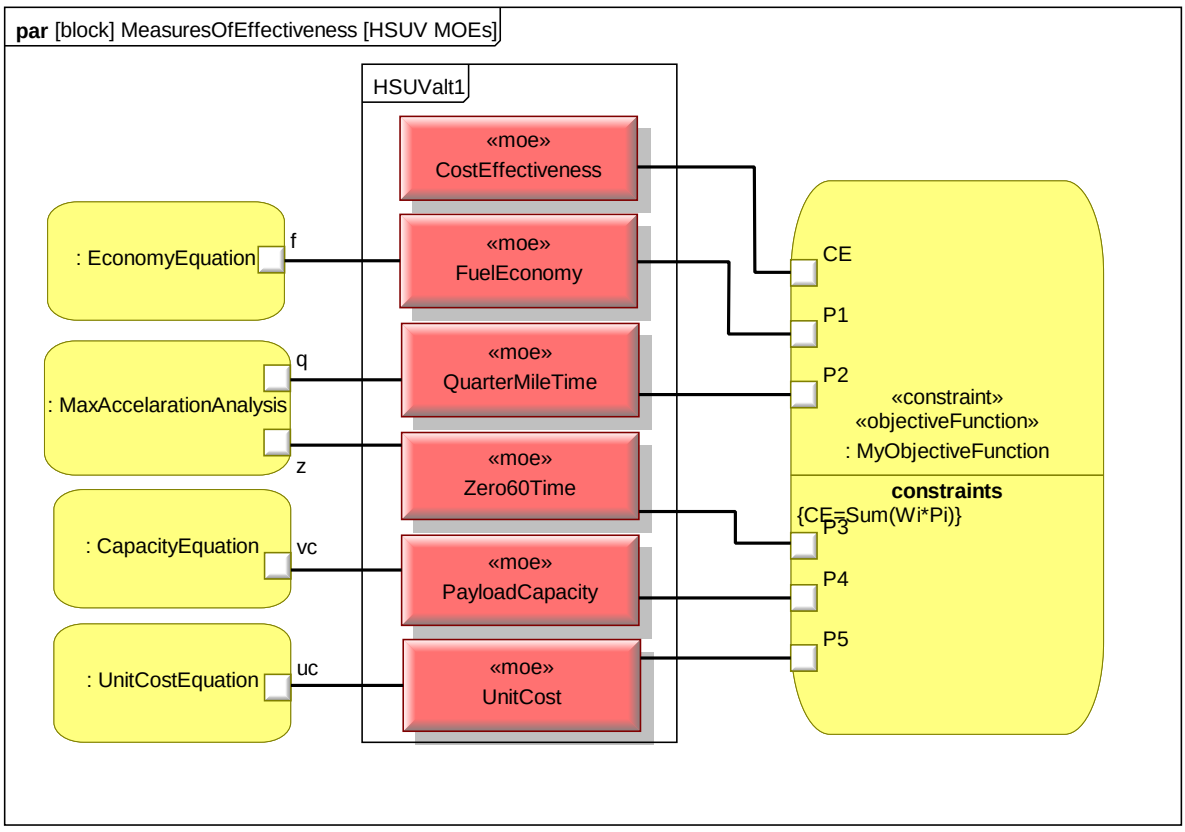

Figure 6. Measures of Effectiveness.

Trade-off Analysis

The parametric diagrams shown in the previous sections and can be used to support trade-off analysis. In Figure 6 the constraint block defines an objective function to compare alternative solutions. The objective function constrains measures of effectiveness or merit and includes a weighting of utility functions associated with various criteria used to evaluate the alternatives.

Properties bound to parameters of the objective function may have probability distributions associated with them that are used to compute expected or probabilistic measures of the system. (OMG, 2007b) In this way, the optimum solution can be determined for the system.

Model Execution

UML and SysML models can of course be executed. The foundational UML or fUML initiative has been formed to define an execution semantic for both activity and state diagrams. Currently, execution semantics have been determined by individual tool vendors and often involved integration of particular programming languages. For models whose ultimate goal is generated code this is perfectly acceptable and often preferable. fUML specifies a language independent means of executing models. (OMG, 2009) This is a more ideal solution for systems engineers and system architects who may not be familiar with programming languages. Regardless of how it is done, execution of the model against a pre-defined set of criteria can determine correct functionality, performance, timing, error handling and can help to validate use interfaces. Models built purely for simulation can help users determine the ultimate look and feel of the eventual system. The extent to which they can do this effectively determines the level of quality of the model.

Style, Standards, and Visualization

In the same way that there are coding and documentation standards, there needs to be modeling standards. These can be as simple as checking to make sure that all required fields have been filled in. For example use cases should contain a full use case description, pre and post conditions, intent and alternate courses. More complex tests would ensure that use cases have been elaborated to sequence diagrams, trace to or refine functional requirements and that the use case text follows the company standard PDL. Quality checks for the class model for which code will be generated would verify that those classes, operations, attributes, etc are named according to the coding standard, Complexity metrics can also be checked such as the number of attributes, associations, operations, level of inheritance, etc. The McCabe complexity metrics are the best example. (McCabe, 1976) Some examples checks for SysML models are ensuring that all activities have been allocated to structural elements, item flows and port types are consistently typed, and that logical or abstract elements have been allocated to concrete or physical ones.

Quality checks can also examine the complexity of the diagram such as the "7 plus or minus 2" rule, (originally described by George A. Miller) to ensure that diagrams are readable. More complex checks can ensure that state diagrams do not contain dead-end states and activity diagram paths can be executed in a deterministic fashion. As part of the adoption of MBE, a style guide should be produced by the process owner with examples to ensure a consistent approach to MBE.

Integrating Quality Into The Process

The Importance of process. A good starting point for defining a process or integrating these concepts into an existing process is the Object Oriented Systems Engineering Methodology (OOSEM). This has been successfully adopted by several major companies. For more information, see Lykins, et al, (2000) and other information available at the OOSEM website http://syseng.omg.org . It is imperative that a well-defined process be specified elaborating how quality checks fit into the overall process, whether they are suggested or mandatory, and how updates, modifications, variations, dispensations, etc. will be handled. As with any process improvement initiative, the most important thing is to start where you are with your existing process, figure out where you would like to be, and determine how you are going to arrive at your destination incrementally whilst ensuring that improvement can be measured. There is an old saying, "If you don't know where you are, a map won't help." Consequently, it

is important that process metrics already exist prior to starting a process improvement initiative. Otherwise, it will be impossible to determine whether MBSE techniques are in fact improving things.

Quality Assurance

Tom Gilb describes Quality Assurance (QA) as “the generic name for any set of activities, which have as their primary or partial intent, or effect, to influence (‘assure’) the quality levels of a product or process.” (Gilb, 2010) There is also the assumption that modifying the process will affect the quality of the product that is being delivered. Consequently, for MBE projects it is essential that quality criteria for models be included in the process.

The Iterative nature of Quality.

For each stage of the process it is necessary to consider the inputs, activities performed during that stage, the roles of the people involved, and intended recipients of the outputs of that stage. In fact this is true for both the process/project as a whole as well as for each sub-activity in the process at all levels. This pattern is fractal in nature. A fractal is "a rough or fragmented geometric shape that can be split into parts, each of which is (at least approximately) a reduced-size copy of the whole," a property called self-similarity. (Mandelbrot, 1982). For each activity or task in the process, the level of quality needs to be determined prior to starting in order to determine the degree or extent of the task. For example, early on in the process a rough draft of the concepts will suffice to enable the team to agree on the way forward. If the intended system is a safety critical implementation, this rough draft will need to be developed to a great level of detail before it will achieve its intended goal. Figure 7 contains an example systems and software development process.

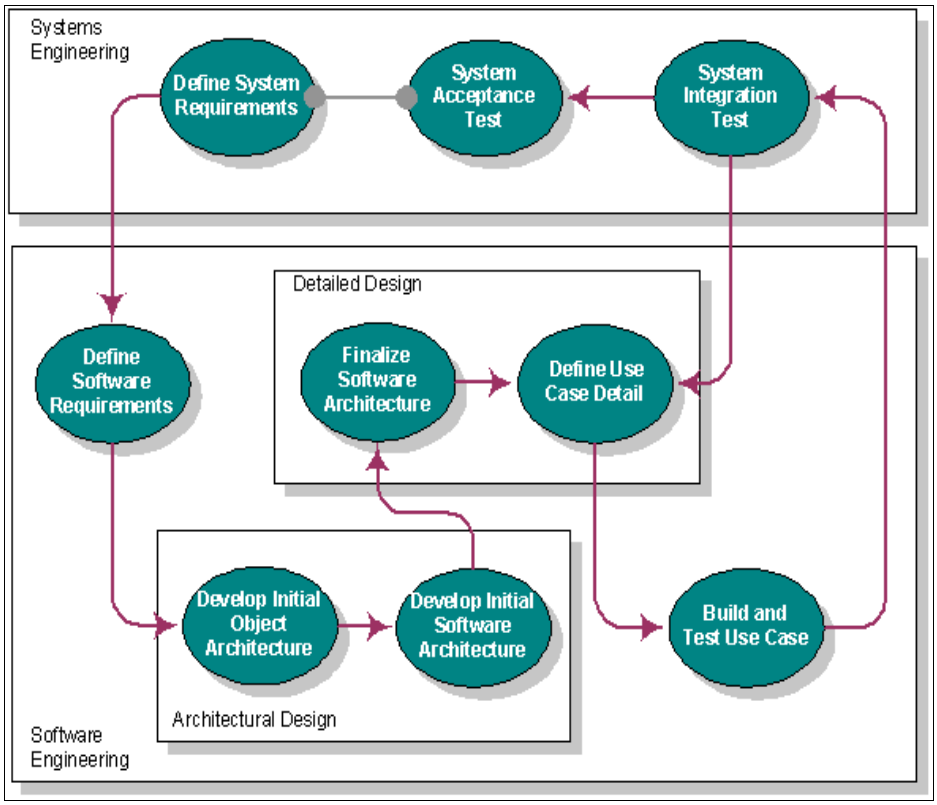

Figure 7. Example Integrated Systems and Software Development Process.

Each of these elements can of course be broken down further. Figure 8 shows the detail for the Define System Requirements task.

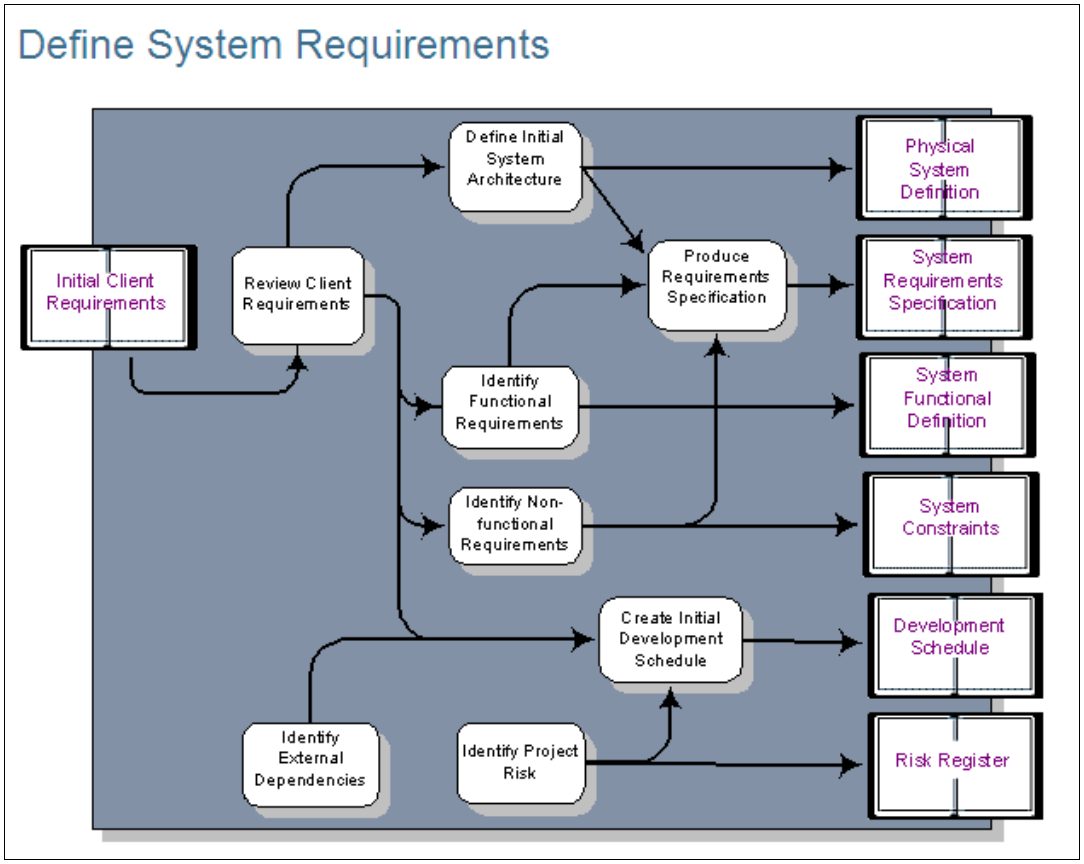

Figure 8. Define System Requirements.

As part of a well-documented process, each quality and assessment criteria need to be defined for each activity, input and output. These will also need to be refined over many projects and customized for each project. To do this effectively, this needs to be automated as much as possible.

Automated Quality Assurance

Jim McCarthy states that "QA's principal function is to continually assess the state of the product so that the rest of the team's activities can be properly focused." (McCarthy, 1995) For this to take place, QA checks and criteria need to be as automated, transparent, and painless as possible. Just as no C programmer should submit his or her code for review without having it go through Lint, no system designer should submit a design for review without submitting it to a quality check. In order for this to work effectively, this needs to be automated as much as possible. To continue with the software metaphor, manually checking all of the tests provided by Lint would take a very long time, be relatively error-prone, and would suffer from subjectivity. Integrated Model-Based Quality Assurance requires integrated tools. These should provide summary views (often called dashboards) and detailed views, auto-correction of specific types of errors, configurable modeling standards, visualization of errors, and configurable and user-defined reviews to ensure that your model is complete, consistent, and correct. Without this level of automation, it will be difficult to enforce quality standards across an organization.

Conclusion

This paper has looked at a variety of different criteria for quality in model-based development. We have demonstrated some of the ways that quality can be defined and evaluated in a model.

In order for this to support the organization, it is necessary that the entire process, standards, and development lifecycle be taken into account. It must also be automated to work effectively. However, as always people are still required to ensure that a complete, consistent, and coherent model is understandable, communicates its intent effectively, and satisfies the customer's true requirements.

References

1. American Society for Quality, definition of quality available from http://asq.org/glossary/q.html, accessed May, 2011.
2. Box, George E. P.; Norman R. Draper (1987). Empirical Model-Building and Response Surfaces, p. 424, Wiley. ISBN 0471810339, Page 424.
3. Brooks, Frederick P., 1995, The Mythical Man-Month, Essays on Software Engineering Anniversary Edition, Addison-Wesley Publishing company,
4. Collins Paperback English Dictionary, 1993, Harper Collins Manufacturing, Great Britain.
5. Czarnecki, Eisenecker, 2002, Generative Programming: Methods, Tools and Applications, Published by Addison Wesley.
6. Fowler, Martin, 2004, UML Distilled, Third Edition, Addison-Wesley,
7. Friedenthal, S., Moore, A., Steiner, R. Practical Guide to SysML: The Systems Modeling Language, Morgan Kaufman September 2008
8. Gilb, Tom, Planguage Concept Glossary, available online at http://www.gilb.com/tiki-page.php?pageName=Competitive%20Engineering%20Glossary (Accessed March, 2010). A complete list of his numerous publications is available at this site.
9. Hause, M.C., 2006a, The Systems Modeling Language - SysML, Sept 2006, INCOSE EuSEC Symposium, Edinburgh, 2006 Proceedings.
10. Hause, M. C., 2006b, Cross-Cutting Concerns and Ergonomic Profiling Using UML/SysML, INCOSE International Symposium Orlando, Florida, Proceedings.
11. Holt, J., Simon Perry, S., SysML for Systems Engineering, IET Publications, 2008
12. Illingworth, V., Glaser, E.L., Pyle, I.C., 1991, Oxford Reference Dictionary of Computing, Oxford University press, Walton Street, Oxford, 1991.
13. International Council on Systems Engineering (INCOSE), Systems Engineering Vision 2020, Version 2.03, TP-2004-004-02, September 2007
14. Korff, A., Modellierung von eingebetteten Systemen mit UML und SysML, von Spektrum Akademischer Verlag Taschenbuch - 13. June 2008
15. Lykins, H., Friedenthal, S., And Meilich, A. Adapting UML For An Object-Oriented Systems Engineering Method (OOSEM)", Tenth Annual Int Symp INCOSE proceedings. Minneapolis, Mn, USA, July 16-20, 2000.
16. Mandelbrot, B.B. (1982). The Fractal Geometry of Nature. W.H. Freeman and Company. ISBN 0-7167-1186-9.
17. McCabe, Thomas J., IEEE Transactions On Software Engineering, Vol. Se-2, No.4, December 1976
18. McCarthy, Jim, 1995, Dynamics of Software Development, Microsoft Press, Washington.

19. Object Management Group (OMG), 1999. OMG Unified Modeling Language Specification (draft). [online] Available from: http://www.omg.org [Accessed April 2001].

20. Object Management Group (OMG), 2005a, UML Testing Profile for UML 2.0, v1.0, formal/05-07-07 (full specification)

21. Object Management Group (OMG), 2007a. Unified Modeling Language: Superstructure version 2.1.1 with change bars ptc/2007-02-03. [online] Available from: http://www.omg.org [Accessed September 2007].

22. Object Management Group (OMG), OMG Systems Modeling Language (OMG SysML™), V1.0, 2007b, OMG Document Number: formal/2007-09-01, URL: http://www.omg.org/spec/SysML/1.0/PDF, Accessed November, 2007

23. Object Management Group (OMG), 2009, Semantics of a Foundational Subset for Executable UML Models (FUML), available online at http://www.omg.org/spec/FUML/Current

24. Praxiom, Quality Management Definitions in Plain English, available at http://www.praxiom. com/iso-definition.htm

25. Roget Thesaurus, 1987, Penguin Books, published by the Longman Group, UK

Biography

Matthew Hause is Artisan's Chief Consulting Engineer, the co-chair of the UPDM group and a member of the OMG SysML specification team. He has been developing real-time systems for over 30 years. He started out working in the Power Systems Industry, and has been involved in Process Control, Communications, SCADA, Distributed Control, military systems and many other areas of systems development. His roles have varied from project manager to developer. His role at Artisan includes mentoring, sales presentations, standards development and training courses. He has written a series of white papers on project management, Systems Engineering, architectural modeling and systems development with UML, SysML and Architectural Frameworks. He has been a regular presenter at INCOSE, the IEEE, BCS, the IET and other conferences. Matthew studied Electrical Engineering at the UNM and Computer Science at the University of Houston.