Lockheed Martin MS2
199 Borton Landing Road, P.O. Box 10957
Moorestown, NJ 08057-0927

Motivation for Model Driven Development

John C. Watson

June 21, 2008

Table of Contents

1 Abstract

The goal of this document is to provide a list of motivations for using a model driven approach for developing large, complex software intensive systems and communicate these motivations to those that are relatively new to model driven technology. This document will clearly communicate why we need to embrace these techniques across the industry.

Certainly many of these benefits can apply to all scales of development, but this document will focus on those motivations that are achieved for large system development.

Using the UML/SysML modeling language to capture engineering artifacts foremost and primarily improves our ability to automate our engineering tasks and better share and communicate information relative to document/text based mechanisms. This will be the underlying theme that will thread through the motivations discussed henceforth.

2 Inspiration

As our systems grow in size and complexity, we have all experienced how it is becoming more and more difficult to extend some of these horizons. The amount of information gathered to architect, design, build and maintain these systems has grown significantly.

It may take hundreds of interacting professionals across multiple engineering disciplines, business units and companies to create and maintain one of these large complex systems. The solution to creating larger and larger systems must include better environments to create, communicate and share information across multiple knowledge domains and physically diverse groups of people. The more we enable our abilities to communicate, the more we reduce the risk in our development tasks.

Because of this ever-increasing complexity, we must find better ways to reuse interchangeable components across variant solution domains and systems. Reusing these components will significantly increase our ability to add more capabilities quicker, reduce the development costs, and reduce support costs.

We have found that a common modeling language along with other common modeling related techniques can not only be used to help us specify and build more complex and larger systems, but can also help us specify and share components across multiple domains and disciplines.

These common modeling techniques are not a silver bullet and come with a set of challenges. However, it has been clearly demonstrated to us that this set of cobblestones will allow us to take another step forward.

3 Introduction

One of the fundamental steps in communicating better is to pick a common language, one that is common from the highest levels of abstraction to the lowest level of detailed engineering. This common underlying language is the Object Management Group's (OMG) Unified Modeling Language (UML) and System Modeling Language (SysML).

UML was first introduced in the late 1990's. Its primary goal was to provide a common language to specify, visualize, and document software. UML provides a series of graphic constructs and diagrams that allows us to extend the ability to express a software design by visual means. This can include:

- Behavioral sequences of events and flowcharts.
- The ability to define entities (a class) internal and external to the system.
- Showing relationships between entities, e.g., generalizations ("kind of" relationships), aggregation/composition ("part-of" relationships), and dependency relationships.
- Showing how two entities are associated through interaction.

The System Modeling Language or SysML is an extension of UML to support the specification, analysis, design, verification and validation of a broad range of complex systems. SysML uses much of the existing underlying metamodel of UML. However, it also provides extensions to express domains beyond software. Extensions are listed in the chart below:

Extension	Represents…
Block (an extension of a UML class)	the basic unit of structure in SysML. It can be used to represent hardware, software, facilities, personnel, or any other system element.
Flow	the flow of data, material or energy. Ex: Water flowing though a pipe or current through a wire.
Constraints and Parametric Diagrams	mathematical expressions in the model.
Requirement Entity	a single textual requirement, its attributes and relationships in a model. The model can show how these requirements are refined, derived, and satisfied.

Using the above basic building blocks we have the ability to capture and communicate our ideas and thoughts to express a system's behavior, structure and requirements in a common way.

By studying the system's behavior we can determine "what" the system needs to do to satisfy the needs of the users and requirements of the overall system. From understanding this behavior we can then determine the system's internal architecture and decompose the system into a set of substructures to support this behavior. The system requirements are then allocated to the substructures.

We can then repeat this process by now studying the behavior, structure and requirements of each of the substructures and continue this decomposition process until we have pieces small enough to thoroughly understand and build upon.

Each of these levels of decomposition is captured in UML/SysML models.
Therefore, the models become the focal point to record, review and share factual artifacts of our system across a broad spectrum of engineering and program management disciplines much like existing textual document trees.

The models now provide an additional layer of automation capabilities that allow us to:

- More easily enforce the use of common processes and techniques.
- Enforce the use of a common communication language.
- Automate the process of consistently creating, locating and updating interrelated system artifacts within a large repository.
- Enable the reuse of system artifacts so they're defined once and reused across the full spectrum of engineering disciplines.

4 The Benefits of a Model Driven Approach

When software was limited in scope by the available memory and processor speeds it seemed adequate to entrust the details to a few good folks and hope that they never went away. Today we no longer have that luxury. To build large complex systems, we need to look at these systems from many perspectives and capture the system details so we can not only share this information with a larger audience, but also retain the ability to add new capabilities and support these systems in the future.

The intent of this section of the document explores the benefits of model driven system and software engineering based on our observations and experiences.

4.1 Simplification of Reality

In the general sense a model is defined as a "simplified version of something complex used in analyzing and solving problems or making predictions." This simplification of something can be referred to as an abstraction.

A UML/SysML model allows us to look at a system from wider or narrower views. Certainly if the only visibility of a system we had was by looking at the detailed software code, it would be difficult to understand how a system works and therefore how to resolve software and system issues. If we could pull ourselves up a bit, or hide some details we would see only those more significant design artifacts. This higher abstracted perspective would allow us to see issues that would be too overwhelming at lower layers of detail, the old "forest for the trees" syndrome.

Defining a model's layer of simplification is referred to as establishing a model's vertical context or scope. Therefore, a vertical model context means that a specific abstraction layer is defined, e.g., viewing the system from a 10,000 foot level vs. a 50 foot level. Typical vertical contexts could include a System of System (SoS) view, a system view and a subsystem view. The number and names of these levels may change depending on the size and complexity of the system of interest.

So, how do we interconnect these higher and lower layers of models so that changes in one layer impact layers above and below it?

At each vertical context, a role of the model is to decompose this defined system context into a set of smaller architecturally significant substructures. The conclusion of this decomposition analysis is a derived specification for each of these substructures in the form of behavior, structure and requirements. These specifications are used to drive one or more models on the next lower vertical abstraction layer. The decision of whether this is a "one specification to one lower layer model" or a "many-to-one" relationship is based on the resulting size and complexity of the lower layer models and may also be influenced by existing department organizations. For simplicity sake, let's assume that one specification drives one model; therefore, at the next layer there exists multiple models. This set of models has the same vertical context but now have a defined horizontal context from one model to another.

A horizontal context identifies the model's functional boundaries of responsibility. Defining these boundaries is not only important for clearly defining what functionality is or isn't included in each of these substructures, but also to ensure the interfaces to the peer models are sound.

By establishing both the vertical and horizontal context for each model and interconnecting the models via their substructure specifications you would form a tree models (see Figure 1 – The MODEL TREE). In the same way we decompose our systems into smaller more manageable parts, we also decompose our analysis of the system into an array of interconnected models, i.e., the vertical and horizontal context. These models allow us to subdivide the overall problem space by assigning them to the appropriate engineering disciplines and knowledge domains.

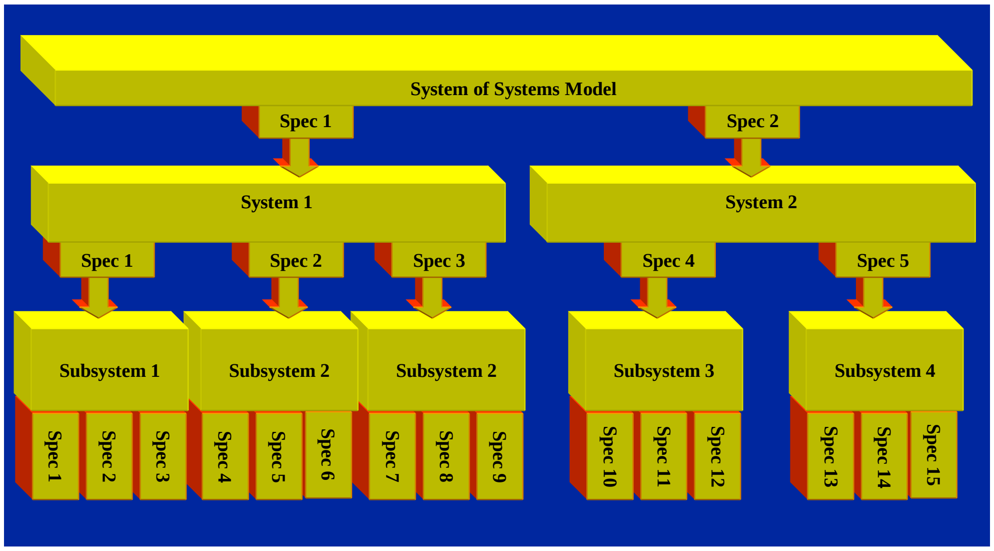

Figure 1 – The MODEL TREE

For example, a large system will typically consist of multiple models hierarchically arranged. At the top of the heap may be a System of Systems (SoS) model. The intent of this model is to move one abstraction layer up from the system of interest and to examine how these peer systems behave in the context of other systems. The role of this SoS model is to capture the analysis for:

- Establishing the boundaries of each individual system within the SoS.
- Defining how multiple systems collaborate to achieve an SoS goal.
- Defining the interfaces used for this collaboration.
- Allocating responsibility to each system.
- Creating a specification for each system that will be further analyzed in the next lower abstraction layer.

For each of the systems, a model is created that is driven from this specification. The role of this model is to decompose the system into a set of subsystems. The role of each of these models is to:

- Analyze the behavior of the system required to achieve the goals established for it by the SoS specification.
- Define the components and their interfaces used for this collaboration.
- Show how multiple subsystems collaborate to achieve these goals.
- Allocate responsibility to each subsystem.
- Create a specification for each subsystem that will be used for the next lower abstraction layer models.

And this pattern could continue decomposing the subsystems into a set of architecturally significant components followed by a model that provides the detailed hardware or software design and implementation of a set of domain related components. This is just an example, the actual tree and the number of layers will be different depending on the departmental organization and complexity of the system. Responsibility of an individual model is generally assigned to an organizational element. As the complexity of the system increases the number of models generally increases both horizontally and vertically.

4.2 Common Language

By using UML/SysML, we use a common language to express and analyze the system of interest. This common language can be shared not only within a specific model, that is horizontally, but also to communicate vertically across hierarchically associated models, that is from one abstract layer of context to another. What does this mean? We can now use a common language from the highest layer of abstraction down to an expression of detailed software design. And in fact we can use this common language to share specification from one product line to another.

This language can therefore be used by all development disciplines including system analysis, system architecture, system specification, software design, and all levels of test.

4.3 Textual and Visual Representation

For most of us our minds can absorb and understand information at a much higher rate when pictures augment the textual information before us. The pictures stay focused in our heads long after the detailed text fades away. By using UML/SysML we can express our thoughts and ideas through both a textual and visual perspective.

Traditionally we have mostly used text to express the specification of the system. The pictures we utilized were free-form in that they didn’t have an underlying set of standard constructs to enforce common ways of visualization. As a result they varied in context, level of detail and expression from one diagram to the next.

UML/SysML defines this underlying set of constructs and refers to it as the metamodel. These same constructs are used to create model elements at each layer and to create a set of visually rich diagrams. Since the same constructs are used at all layers there is a modeling consistency at all the horizontal and vertical contexts, and across multiple product lines.

As model elements are added to a model, a textual explanation can be added to each model element describing more specifically what it is and isn’t. These model elements can include system substructures, functional descriptions, data descriptions, interface descriptions, requirements, etc. Since each of these model elements are defined in a single place within a model and reused on multiple diagrams, the models become a repository for logically cataloging these items along with their textual description. Diagrams alone can still allow too much interpretation to the reader, so when ambiguity arises the reader can drill down to the text to help resolve the issue.

Since UML/SysML models contain both the visual and textual controlled references, excerpts from these models can be used for generating engineering review documents and have the potential for becoming a source for large portions of contractually delivered documents.

Once again, a visual representation is not a new idea. The building industry has long ago established standardized building plans to help translate the architect's and designer's ideas and analysis results so that any building trade company can read, understand and build the building.

4.4 Using Object Oriented (OO) Techniques

Models based on UML/SysML support the ability to utilize Object Oriented (OO) design techniques both in determining the system architecture and the detailed design. The benefit OO techniques provide is the capability to more easily support, enhance and add new capabilities in the future. These techniques can be used in decomposing systems that are focused on software, hardware and mixed systems.

In the software world OO techniques encapsulate both data and functionality within a class. In the System Engineering world, the concept of data is expanded to include any manageable variable, including energy and mass. The responsibility of maintaining a manageable variable is isolated to a single object, or part of the system, be it a software class, a component or, in SysML, a System Block.

OO techniques allow us to avoid the pitfalls the industry has experienced using just functional decomposition. Here the emphasis is on just decomposing the functionally of the system, missing the additional focus on the data or any manageable variable. Without this focus, data can be managed by multiple places in the system. Changing this type of system is difficult since it often means changes throughout the system.

Although the term OO is typically associated to the software industry, the techniques of encapsulation have been known and used extensively in the mechanical and electronic domains. Integrated Circuits propelled the electronics age by using well defined re-usable components with well defined behavior, data and interfaces. But when you think of it, the physical and mechanical world has used these techniques, maybe unknowingly, for centuries, e.g., an engine manages torque.

In addition, the industry has cataloged a number of common OO design and architectural software patterns which can be leveraged when using OO. These patterns help us learn by others experiences, both good and bad.

4.5 The Focus on System Use

Models provide us with an additional dimension of analysis that is often missing or not brought to the forefront in traditional specifications. This dimension is the system behavior. In many traditional specifications the focus is on decomposing and capturing the functionality of the system. These traditional views reference the structure of the system, i.e., the sub-structures, the functionality in each substructure, and the requirements associated to that functionality. What is difficult to find is “when and how” this functionality is used to provide value to an external entity.

In UML/SysML a Use Case is the model artifact used to encapsulate a specific behavioral goal of the system. These goals are based on understanding how an external entity (user, system or human) uses a system service to gain benefit from the system. The Use Case is used to explore this system service and identify what functionality and other external entities are needed to achieve this goal. As we analyze the detailed steps required to reach a goal we are identifying the primitive chunks of functionality required by the system. More importantly, we have tied each primitive functional chunk to a value the system provides to an entity external to the system.

By defining a comprehensive set of Use Cases in the system, we can determine all of the chunks of functionality the system needs and we can see which chunks of functionality are reused in multiple goals. During this process we also allocate each chunk of functionality to a part of the system. Typically this functionality will be updating a manageable variable (data elements in software) also allocated to this part of the system. This process of allocating functionality and assigning these variables to parts of the system provides a well defined responsibility for each part. If at some later point the behavior changes, the changes can be traced to the parts of the system that are affected.

In addition, because a Use Case ties us back to the value the system provides to external entities, we can measure the impact of functional changes to other external entity goals defined in other Use Cases. Ideally all Use Cases that provide a service to an external service should be explored. However, the reality is that time is a precious commodity. Services that must be explored are those that can significantly impact the underlying system architecture, impose a risk to development, or are significant and important to a stakeholder, e.g., the customer.

4.6 Objective View

By using the common language of UML/SysML, these models allow us to more easily share our ideas and thoughts across a large spectrum of engineering disciplines including the disciplines of architecture, design, implementation, deployment and test. Some of these artifacts may even be shared with the customer or other stakeholders. With this exposure and injected feedback from these various disciplines the model allows us to formulate a very objective view of the system’s behavior, structure and requirements.

A diagram may appear to present a simple concept and the tool makes it easy for anyone to create a UML diagram. But, the value is not just in the drawing. The value is in the fact that knowledgeable people have looked at it, questioned it and at this point in time agree that it is correct.

As these model artifacts are reviewed, validated and updated the model becomes a repository of controlled factual system artifacts. So, all of those PowerPoint overheads, notebook sketches and information heard over the wall can be validated, collected and integrated into a single source of objective system information thus improving the quality of sharing and communication system artifacts.

4.7 An Environment to Create

When we're inspired by an idea, whether it's an inspiration to create an oil painting or to develop a technique that doubles a system's throughput, we need the right tool aids to help us capture the idea and help us through the thought process of evolving it to something that is real. This thought process typically includes the ability to quickly and easily jot down and sketch those initial thoughts and not be hampered or distracted by bunny-trials that divert our focus. The initial thoughts are a mere sketch of the idea, so next we need the capability to help shape and refine the idea until it has some foundation. The idea then needs to be shared and communicated to others. These final iterations mature what was once a fleeting idea into its implementation and finally, a reality.

Think of that Erector Set or Lego Blocks we had in our youth. These childhood toys provided the basic constructs and the supporting environment that enabled us to create a simple one room house, or a skyscraper, or a fully mobile and operational "helicopter dump-truck front-end loader." We were only limited by our imagination. And how frustrating was it when we were limited by the number of parts available, but the environment gave us the capability to adapt and quickly adjust our creation to a "three wheel helicopter dump-truck front-end loader."

The UML/SysML models provide this environment and the pallet of tools to realize those inspirations in the system and software world. Because of the underlying building blocks, relationships and constraints defined within the UML/SysML language the environment supports and guides us in quickly sketching our initial thoughts and ideas with little detail. The pallet includes the ability to represent the ideas both graphically and textually in a common language. As our ideas are guided and refined with more and more detail, they can be easily shared and reviewed with other peers and stakeholders. Eventually the ideas mature until they are fully integrated and consistent with existing system or software structures and behaviors. All this can be done within a single automated environment, something difficult to do in the world of independent documents, PowerPoint slides and personal notebooks. When all is done, we have captured the necessary details to test, support and enhance the system in the future.

4.8 Managing Larger Volumes of Engineering Data

Obviously, collecting and maintaining high quality engineering data is essential to making sound decisions, not only during the initial development phases but also when adding future capabilities and providing support. This information not only includes the formal descriptions of the system's structure, behavior and requirements but also includes the engineering analysis artifacts that describe the rational and underlying decisions used to derive this information.

With small systems and small development teams a document based development can be adequate. But now as systems are becoming larger and more complex the amount of information to archive and maintain has increased significantly. This information is naturally very interdependent, so in a document-based environment, when information in one area changes the ability to manage the full impact of the change across the system has proven to be a challenge.

Attempts to use third-party tools to establish manual dependencies, or links, between pieces of this information within and across documents has also proven to become time consuming, difficult to do, very subjective and fraught with human error. As a result these links are rarely created. If they are created it is even rarer that they are maintained. Communicating and sharing document based artifacts across different development environments, developed under different guidelines and maintained with different tools does not help the situation.

Another issue in a document based environment is maintaining the second tier information, those artifacts containing the more detailed engineering analysis information. This information plays a critical role, not only during the initial development phases, but also while evaluating changes for future enhancements and for support. Much of this good work remains buried in our notebooks, files systems or in a series of un-maintained PowerPoint slides and papers. Try as we may, as time moves on and people move on, this information becomes less and less available and out of date.

All of thee issues can result in information across this wide field of resources not being synchronized, both in time and in content, and ultimately can evolve into a lack of confidence of the quality and accuracy of the information.

So, how do we solve these problems of organizing and managing large volumes of interrelated system description data? A solution is to create an Engineering Repository containing multiple vertically and horizontally related UML/SysML models along with other engineering artifacts, including the code. By establishing the appropriate guidelines, these models can be used as an information organization tool and as the glue to provide the interconnectivity with system information established within the models and external to the models.

4.8.1 Organization

The first issue is providing a structure to organize this vast amount of information. Each model provides the ability to hierarchically organize information within. Just as we create folders to organize information hierarchically in our computers, the UML language provides “packages” that are also used as folders in a model to catalog and hierarchically organize information. A large system will typically consist of multiple models hierarchically arranged into a tree of models as described in section 4.1, “Simplification of Reality.” The number of layers and the width at each layer will vary from program to program. However, the important aspect of this organization is that the models are interconnected via model-based specifications.

Finally, we can collect the models and other related artifacts including code, into a single Engineering Repository managed under the umbrella of a configuration management tool. These tools provide a means of:

- Limiting the read and write access to selected items in the repository.

- Maintaining a history of changes to each file in the repository.
- Providing common access and management procedures across all the information in the repository.

How do we capture that invaluable second tier engineering data? The UML/SysML modeling language was designed specifically to allow us to express our ideas and create our engineering artifacts including:

- Sequences of events
- Flowcharts
- Equations
- Requirements
- Structural descriptions
- Trade studies.

Most of the second-tier engineering artifacts can now be created directly in a model in the contexts of other modeling information using a consistent means of expression. Other documents like review results, specifications and other related documents, can be attached via hyperlinks to the most appropriate place in the model organization. These documents are also placed in the Repository and configuration managed like model files and all other documents in the Repository.

4.8.2 Measuring Change

As entities of our system are defined and described in a model, the language also provides the ability to create relationships between these entities. These relationships are created during the natural course of describing the system in the model. For example, in the model we can create a relationship that shows how a part of the system is decomposed into a set of child parts, or we can show how one part of the system is dependent on another part's interface. When a change is needed we can follow these relationships to help us understand and manage the change impact.

Models can also reference and establish relationships between "things" in other models. This allows us to tie multiple models together, establishing dependencies, both horizontally (same abstraction layer) and vertically (multiple abstraction layers) in our model tree and use these relationships to measure the impact of change in both horizontally and vertically connected models.

In the example above, if we decide to change an interface on one of the system's parts we can follow the dependent relationships to understand the impact within the model of interest but also measure the change impact in other models that may use this interface.

4.8.3 Information Integrity

The first level of information integrity comes from the fact that we are using a structured common language across all models. The language is rich enough to allow us to describe what needs to be expressed but provides the discipline of common expression.

The structure of the models, provided by the guidelines, instills the next level of integrity, in that it isolates the areas that provide specification. These areas are managed and changed via a defined change control process that includes engineering reviews and concurrence from the program management team.

However, remember that the growing problem is the expanding size of these systems and insuring the integrity of the resulting large amount of information needed to describe and support the system. Automating the ability to help maintain the integrity of this information will help eliminate the more subtle errors that are difficult for humans to find but easy to create.

Essentially, it is similar to the spell and grammar checker you use in word processing programs. UML/SysML is the governing language across all these models. It is also a well-structured language defining the underlying data types and rules governing the allowable structures and relationships permitted. Most tools supporting UML/SysML contain the ability to programmatically execute a series of consistency and validation checks. Therefore this provides the ability to maximize the integrity, completeness and accuracy of the information in the models.

Finally, maintaining the same look and feel of each model is import to ensure readability and comprehension, thus maximizing communications across all of the horizontally and vertically interconnected models. The problem is that these models will be developed by different organizations, and potentially geographically diverse organizations. Therefore, it is important that a proven set of modeling guidelines and techniques be established early in the program and that reviews and integrity checks measure against these practices.

4.9 The Vocabulary of the System

When we communicate it is not only helpful that we speak the same language but it is also necessary in the technical world that we establish a glossary of defined terms and acronyms. In document driven development, this vocabulary is usually distributed across multiple documents, opening up the possibility that a term or acronym is defined differently in different documents.

In the model driven environment, as terms surface, they are added to a part of the model called the Data Information Model. At that time, a model element is created to represent the term and it is given a textual definition. If necessary, it can be given additional attributes and relationships to other terms and behaviors and can be added to the glossary items to enhance their definitions. As the project progresses some of these terms become the data of the system, some become parts of the system and some remain as glossary items. The modeling technology allows us to share and reuse this glossary across the entire project so that all engineering disciplines are using the same vocabulary definitions.

4.10 System Simulation

Simulation models should be created as early in the development process as practical. These early simulations help us validate the decisions and assumptions made to date. If we find we don’t have adequate margins, we have an opportunity to adjust these assumptions long before a significant investment is made in software coding, purchasing material, hardware manufacturing or system testing. Therefore, simulation is a technique used to mitigate development risk by detecting issues earlier.

The UML/SysML model is used to statically define the system’s performance behavior and structure. For example, we can identify and specify performance critical system behavioral threads and other performance criteria based on the underlying structural assumptions that support these threads. To perform a dynamic simulation, the simulation tool needs this same foundational information. By selectively extracting this information from the UML/SysML model to a simulation tool we can ensure the simulations are using this same objective informational foundation. The simulation tool provides the capability to use this static data from the model and create a simulation one or more of these dynamic threads simultaneously. This allows us to stress the system’s performance capabilities with real world scenarios and beyond, long before significant development monies are spent. As information is updated in the SysML model, either with feedback from simulations or by other analysis, the simulation can be re-run to measure and manage the performance throughout the development cycle.

Today there are industry standardization efforts taking place to standardize how these tools exchange information, making it easier to share this data.

4.11 Maintenance and Training Resource

The models provide a complete resource of information that is invaluable to support and enhance the system. Understanding precisely what was implemented and why is imperative. It is vital to not only accurately determine what needs to change but also to accurately determine how long it will take to make the change. This complete resource of information is also invaluable as a training resource for people new to the program.

In both cases, with multiple abstraction layers available, the starting point for understanding the system doesn’t have to be by examining the details of the code. One can go up a layer or two to understand the bigger pictures first and then drill down.

4.12 Source of Program and Development Metrics

A model is a rich repository of program artifacts. By selectively examining attributes of these artifacts we can collect metrics to determine things such as the complexity of the system at hand, the complexity of the tasks to be accomplished and the progress of those tasks. Since our models reflect the actual progress, metrics can give us a more real-time measure of the size and progress of our programs.

4.13 The Customer's Perception

Certainly it is important for our customers to perceive us as an innovative company. This perception is obtained by our daily interaction on existing contracts and via the proposals we submit. We can portray our innovative approach by adapting to and utilizing the latest and best stable technologies within our products and in the ways we develop and support those products through their lifecycle.

The industry as a whole has shown the ever increasing difficulty in developing large systems in the form of delays and cost overruns. Showing our ability to effectively use model driven development across the full spectrum of engineering disciplines to improve our capacity to execute, deliver and support these systems is a message our customers need to hear.

4.14 Reduced Development Time

Unfortunately, because model-based systems and software development is relatively new, the metrics to support reduced development time claims are difficult to find and if found, difficult to associate to a specific situation. However, we can justify this from a common sense point of view.

The models provide a more effective way to communicate and share objective data, not only across the development team but with other company departments and even with the customer. Therefore, we have mitigated a wealth of risks associated with missing and miscommunications.

Also, the models allow us to see and resolve problems earlier in the devolvement cycle so we have minimized the cost impact and the impact to the development schedule.

And, we have been able to automated tasks such model integrity checks and interchanging data between the model and simulation tools. As a result we can perform these tasks more frequently, we can execute these tasks at all vertical abstraction layers and, we are assured that the information we are using is of the highest quality and consistent across the models and other analysis tools.

4.15 Modeling Can Benefit Legacy Systems

Large complex systems that have been around a while often run into serious problems when attempts are made to fix issues or add capability. The problems stem from the fact that technology has changed and the staff with this detailed knowledge and understanding have moved on. As a result the following pattern often reappears.
First a change analysis is performed and the change appears to be straight forward and the costs are within a sound business decision. Resources are applied and the development begins. Once the detailed work begins, the real underlying complications are exposed. The solution becomes more complex. The costs run through the roof, the product is late and the customer is unhappy. Re-engineering the entire system is too cost prohibitive but doing nothing makes unhappy customers and increases the cost of supporting and enhancing the system.

So, what’s the solution? A more practical solution than re-engineering the entire system may be to isolate an area of the system that is more prone to change or that exhibits the most problems and re-engineer just that part of the system. There are modeling techniques that allow us to isolate and re-engineer them while the remaining parts of the system are unaffected. By using these modeling techniques we have also captured more of the detailed knowledge and trade-offs that are forgotten lost in legacy systems.

5 The Impediments of a Model Driven Approach

The modeling impediments are those actions that have a tendency to reduce or eliminate the benefits of using a model-based approach. The following sections provide a list of some of these impediments.

5.1 Unclear Commitment

Change is hard for all of us, so if we aren’t encouraged we will have a tendency to continue down the comfortable path and avoid the pain to learn new ways. For change to occur it needs to be encouraged starting from the highest level of management and be clearly communicated down through the ranks. It is important for all levels of management to make a full commitment to:

- Understanding, believing and communicating the long term benefits.
- Displaying their willingness to make the investment and enable the change.
- Validating that the cost of change is a smart business decision.
- Clearly communicating that the domain knowledge we possess is the real strength of our organization and by using Model-based Development we can leverage this knowledge more effectively.

5.2 The Misconception that Modeling Adds Cost

The engineering tasks we perform using a model driven approach are the identical tasks we executed before the use of UML/SysML models. The biggest difference is that we have added capabilities to help us automate those tasks and have found better ways to communicate and share significantly more information. Therefore, once the additional modeling skills are acquired the cost associated with using models is actually less because of this automation and added communication. In addition, all the structure, behavior and requirements are now inter-related and aren't on separate disparate diagrams.

From another point of view, if models are not used, the ability to create more and more complex systems becomes significantly more difficult and cost prohibitive.

5.3 Training Out of the Box

Modeling aside, to become a skilled practitioner training should include three things:

1. The basic theory including the language, process and tool training.

2. The "how to" training where this theory is applied via examples and practice for the specific processes, guidelines and techniques that are established for the program and that will be used day-to-day.
3. An apprenticeship where on-the-job training is provided with a mentor. This is where a person practices the trade to develop the skills, understanding and wisdom to make good decisions. A person does not acquire these skills by simply attending a class. The skills and wisdom to use this knowledge comes from practice. Therefore, the mentoring part of UML/SysML modeling training needs to be considered as a crucial step in creating good practitioners and should be in-place before a person is expected to contribute significantly.

5.4 Applying the Right Engineering Discipline

As with all engineering, when we apply good engineering disciplines we get results. Discipline does not come from the right tools or OO techniques. Both are capable of creating disasters. Discipline must come from us. The benefits discussed in Section 4 can only be realized if we apply sound and defined engineering disciplines.

5.4.1 Formal Control Taken Too Early

Areas of the model that provide specification need to be treated in the same way we control and manage all of our source documentation. For these areas, we need to have a formal change management process in place that records the request, measures the true impact to all parts of the system, exposes the changes to all stakeholders for review, and manages the change through the implementation and final approval stages. For model artifacts the review needs to include both the quality of the domain content and the quality of the modeling standards and guidelines that are being used.

However, there will be areas of the models that may not be as stringently controlled. These areas are typically used to suggest changes, sketch new ideas and allow these ideas to grow and mature. They can and should reference controlled model artifacts. But, the level of formality to control these areas needs to be determined by the people that gain the most benefit and should not be overburdened with heavy process steps. As this information matures and become part of the system specification, it needs to then transition to the fully controlled change management process.

5.4.2 Lack of Modeling Processes, Guidelines and Techniques

For large development teams the modeling processes, guidelines and techniques need to be in place and documented before significant modeling begins. These practices will avoid backtracking and assure consistency in content and "look and feel" from one model and domain area to another. They can also help eliminate language dialect and colloquialism differences. All these factors can slow down communications or cause miscommunications.

Establishing these practices take significant time to determine the correct detailed techniques, to work around the tool issues, to prove by practice and to document. Therefore, if it is a fist attempt, the best course is to leverage off of the experience of others by using proven practices.

There will still be domain related practices that need to be established. For these areas it is important to adequately fund and allocate the most experienced modeling skills to these efforts as early as possible. A bad practice can be as damaging as none at all.

And finally, these processes, guidelines and techniques need to be controlled by a change management process to establish the practice baseline and communicate changes to all that are effected.

5.5 The Human Factor and Change

Generally, the technology issues are not nearly as menacing as the human factor issues when it comes to change. People don't like change, and there's good reason. Change means risk. Therefore, this risk has to be managed.

So, what are we asking them to change?

- How we capture and organize our engineering artifacts.
- How to use new modeling tools.
- How to apply OO techniques and move from just a functional decomposition.
- To learn a new language, i.e., UML/SysML, to express their ideas and thoughts

Relative to how we organize the models, interconnect multiple models and include non-model-based artifacts in the Repository, for the most part, once taught, are intuitive. Relative to learning new tools, most engineers go through this regularly. It generally provides some set backs but with the right training it's usually a short-term issue. The last two items however are the big ticket items. Here we're asking the engineers to think about a problem in a different way and asking them to learn a new language to express their thoughts and ideas. It is not just a mater of training; it is a matter of gaining the knowledge, experience and wisdom to feel confident that success can be reached.

After the initial training courses the engineers will begin to grasp and acquire the literacy skills which include:

- Seeing the big picture, as to what the artifacts are.
- Seeing what artifacts are appropriate for each area of analysis and development.
- Being able to begin to read and understand the diagrams.

After the initial courses it is generally not the case to possess the immediate ability to create the diagrams and confidently express their thoughts and ideas. For experienced engineers, the frustration comes from the fact that they are doing the same engineering tasks. They "know what they want to say" in the diagrams but they "just can't say it." If we look back on our own experiences, we should see the same types of patterns. I can read all I want about plumbing, but until I do it a few times, the obvious is not so obvious on the first few attempts. I can take a 6 week Spanish course but until I am forced to practice it, or go hungry, for instance, I will not be able to effectively understand and communicate. Why are SysML and OO techniques any different?

This is further compounded by the fact that this is also new to most of their managers. Engineering Managers generally have a blend of technical expertise and the talent of managing people. During this changing time, when an engineer approaches their manager for help, both are being distracted from the technical issue by the means of how to express it. The solution is to apply the appropriate training and mentoring and, be patient. These combined are probably the best investment an engineering firm can make. That is, take that precious resource of domain knowledge and experiences locked in the heads and hearts of your engineering staff and train them how to share and communicate this knowledge more effectively and to a wider audience.

5.6 UML/SysML Literacy

If you can't speak the language it makes it difficult to communicate. This means everyone involved in the program needs to understand a limited set of modeling terminology and how to read some basic diagrams. Much of what the models present is intuitive but some basic ground rules of understanding need to be established among engineers, program management, and other stakeholders alike.

Not everyone will need to know how to open a model, navigate through the model and make changes. However, if a review document is distributed that contains extractions from a model, those reviewing it need to be proficient at reading and understanding it. For example, if we are deciding what threads of a Use Case are being implemented for a development iteration, the engineering team and program management team need to know what a Use Case is and how to identify a thread. We can solve this problem by establishing a required model literacy course for everyone involved on the program.

6 Summary

As we can see from the benefits mentioned earlier, using UML/SysML models improves our ability to:

- Communicate and share information across a wider field of people and programs.
- Capture and manage larger amounts of interrelated engineering information.
- Provide an environment to help us envision ideas and create solutions.
- Automate the process of collecting, sharing and ensuring the integrity of information.
- Provide the completeness of formal and informal engineering information to ensure our path forward for enhancements and support.

It's ironic, since this is the very reason why many of our customers have asked us to develop their systems, that is, to help them solve similar problems with their information and process management needs.

As with any automated process, when applied correctly UML/SysML modeling will allow us to do our job more efficiently, with better quality and lower costs. The use of a well-structured common language for expressing engineering information opens the door for additional automation opportunities in the future by allowing the models to not only interconnect with each other but to interconnect with other analysis tools.

Many other industries have transitioned to integrated computer aided systems to improve their ability to visualize, create and maintain more complex projects including the building, electrical, mechanical design and electronics industries.

Adopting modeling techniques does not add or reduce the number of system engineering tasks needed to produce a good system. The cerebral activities needed to create and refine our ideas and to develop a good product are the same they have always been. The difference is by utilizing UML/SysML modeling techniques; we have automated our ability to visualize, capture, refine, share and communicate this information.

7 References

Object Management Group, 2007, “OMG Unified Modeling Language (OMG UML), V2.1.2”, http://www.omg.org/spec/UML/2.1.2/

Object Management Group, 2007, "OMG Systems Modeling Language (OMG SysML™), V1.0," http://www.omg.org/cgi-bin/doc?ptc/07-09-01.

MSN, Encarta® Dictionary, http://encarta.msn.com/