26th Annual INCOSE International Symposium (IS 2016)
Edinburg, Scotland, UK, July 18-21, 2016

Using Visual Diagrams and Patterns for Consistent and Complete Requirements

David Lempia
Rockwell Collins
david.lempia@rockwellcollins.com

Bill Schindel
ICTT System Sciences
schindel@ictt.com

Terry Hrabik
Rockwell Collins
terry.hrabik@rockwellcollins.com

Stewart McGill
Rockwell Collins
stewart.mcgill@rockwellcollins.com

Mike Graber
Rockwell Collins
mike.graber@rockwellcollins.com

Abstract. This paper describes a visual and textual requirements representation based upon Model-Based Systems Engineering (MBSE) requirements patterns. This underlying representation is a rethinking of the concepts behind systems requirements. It takes into account the subjective stakeholder value, the life cycle modes, objective technical behavior and it combines the best of agile User Stories with MBSE patterns.

This underlying MBSE requirements pattern improves the quality of conversations between key stakeholders of the system. Agile User Stories begin the conversation by focusing on the problem domain, addressed by MBSE pattern features. MBSE patterns then continue the conversation for the technical requirements.

The content and relationships between the key MBSE patterns capture the product learning of the organization for use on future applications of similar products. The complexity of re-use and re-application is reduced by configurability. Organizations re-apply this product learning in new products resulting in lower costs and increased quality.

Introduction

Systems engineering has a lengthy history of attention to requirements. (Walden et al 2015) In spite of that history, there is evidence for the need of significant additional improvement to related requirements practice and outcomes. There is also significant related change occurring across industry sectors. Requirements are not “settled business”.

Work across the industry continues on the agile front to apply specific techniques to more effectively discover stakeholder requirements and guide subsequent development. Systems Engineers steeped in earlier methods sometimes ask whether agile methods can apply for systems engineering in general, without generating an “information debt” that results in life cycle sustainability challenges. “Technical debt” is an agile engineering term used historically to designate promised end product capabilities not yet created, in the spirit of historical agile engineering emphasis on working product over written product documentation (Leffingwell 2011). “Information debt” is a term originating in the INCOSE 2015-16 Agile Systems Engineering Life Cycle Discovery Project, designating the other side of the balance: lack of information needed to fulfill the life cycle support needs of the product, per (Schindel and Dove 2016).

The advance of Model-Based Systems Engineering (MBSE) practice adds still other capabilities as well as questions. Some aspects of MBSE demonstrate promise for improved representation effectiveness, engaging visual cognition and communication, as well as stronger connection to historical technical practices. However, there is valid criticism that models can be overly technical and may even create a communication gap between stakeholders and the developers. The MBSE

Requirements Pattern is a general model based solution that helps to bring a regular predictable ordering of requirements for a recurring system.

Thus it is reasonable to believe that leveraging the pattern processing capabilities of the human brain to recognize visual patterns would further advance the state of requirements development and system representation (Mattson 2014). Even though humans demonstrate these talents from an early age, we are still learning how to fully harness them in representing systems. The development of visual patterns can provide a framework for not only communication, but also analysis. Without patterns that capture what we already know, we (our collective enterprises) spend time re-learning requirements and design for each project that we (someone in the enterprise) already learned on a past project.

There is increasing recognition of the need to manage intellectual assets. (Sherey, 2006) This includes making better use of what has been learned and focusing efforts on areas where knowledge is lacking. As system complexity and pace grow while resources are constrained, it can reasonably be asked whether current "requirements reuse" strategies are adequate.

Addressing these challenges and changes requires a fresh look at how requirements are represented, discovered, and re-applied, on a more integrated basis. First, how requirements of different types are represented is continuing to change. We briefly review prominent historical aspects of the practice and literature, and then provide a synthesis and integration using currently available MBSE approaches. This includes both Stakeholder Requirements and Technical Requirements.

Second, this paper discusses how requirements for software, hardware, or otherwise (both Stakeholder and Technical) are discovered. This includes related challenges, recent practices aided by MBSE and agile techniques, including increased use of visual methods and lessons from physical science.

The third subject is related to learning at team and enterprise levels, so that the fruits of discovery are effectively applied over and over. MBSE Patterns range from top level system patterns to detail level single requirements prose micro-structure. Beyond the practical application of MBSE Patterns this area also has important visual aspects that aid the individual human learners and practitioners in better understanding the system as a whole.

We close with a summary of the implications of these practices and trends for enterprise processes, infrastructure and related conclusions. While this paper is focused on system requirements, other references illustrate the impact of the same approach on other processes, such as verification (Cook and Schindel 2015) and longer-term learning accumulation and feedback (Schindel 2011).

Representation of Requirements: Review of Historical Practices and Literature

Traditional Requirements

The major cause of bad requirements is said to be that people do not know how to write good requirements (Hooks 1993), implying that traditional processes are sufficient if mastered. The natural question to ask is, what is a good requirement? Hooks characterizes a good requirement as a requirement with the characteristics of necessary, attainable, and verifiable. The most common issues with writing requirements according to Hooks are bad assumptions, including implementation, inclusion of operational details, the use of incorrect terms, using incorrect sentence structure or bad grammar, missing requirements, and over-specification. To teach good requirements writing, Hooks lists each one of these requirement characteristics and shows an example that violates this characteristic and one that fulfils the characteristic.

Problem Frames

Engineers create systems that solve problems in the real world. The problem space goes well beyond the product being developed (the "machine") to include anything of relevance in various "problem domains" (Jackson, 2000). Jackson reserves the word "requirements" for "desired phenomena" in the problem domains, and the scope of a requirement may extend beyond the machine to include other elements in the problem domain. Therefore, in order for engineers to fully understand the requirements, relevant "domain descriptions" must also be captured. A complete problem description of use to developers therefore consists of three classes of information: requirements, domain descriptions and specifications of the machine at its interfaces. In this paper, we differentiate between requirements and specifications using the terms "stakeholder requirements" and "technical requirements". For examples of requirements and specifications, see Jackson's book.

Most engineers are very familiar with design patterns, which look inward toward the solution space to create a taxonomy of possible solutions. In contrast, "problem frames" looks outward toward the problem space and creates taxonomy of problems. By decomposing problems commonly encountered by software engineers, Jackson provides an initial framework for recognizing classes of problems. This allows the engineer to draw on past experience associated with that class of problems to more quickly and completely specify the machine at its interfaces and choose an appropriate set of design patterns (Jackson 2000).

Four Variable Model

The Four Variable Model (FVM), see Figure 1, provides a framework to differentiate between system and software requirements and forms a basis for requirements patterns (Parnas and Madey 1991). In this model, the systems context requirement (REQ) monitors (MON) and controls (CON) system level environmental variables (NAT). In the extended software context the requirement (REQ') monitors (MON') and controls (CON') software level variables. The INPUT and OUTPUT translate the monitors MON to MON' and controls CON' to CON. This basic schema can be extended to include constraints, interactions, interfaces and modes, applicable to both stakeholder and technical requirements (Lempia and Miller 2009).

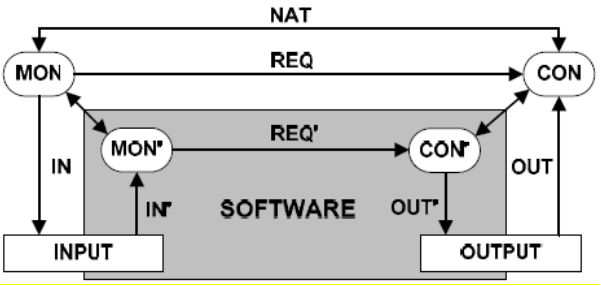

Figure 1. Four-Variable Model

Agile Requirements

Scaled Agile development treats requirements and acceptance in a fundamentally different way than traditional requirements for large software projects (Leffingwell 2011). The primary measure of success is no longer a large batch of requirements ready to pass off to a development team, but working software developed in small batches. Backlog items are broken down using investment themes, epics, features, stories, acceptance criteria, and non-functional requirements. See Figure 2.

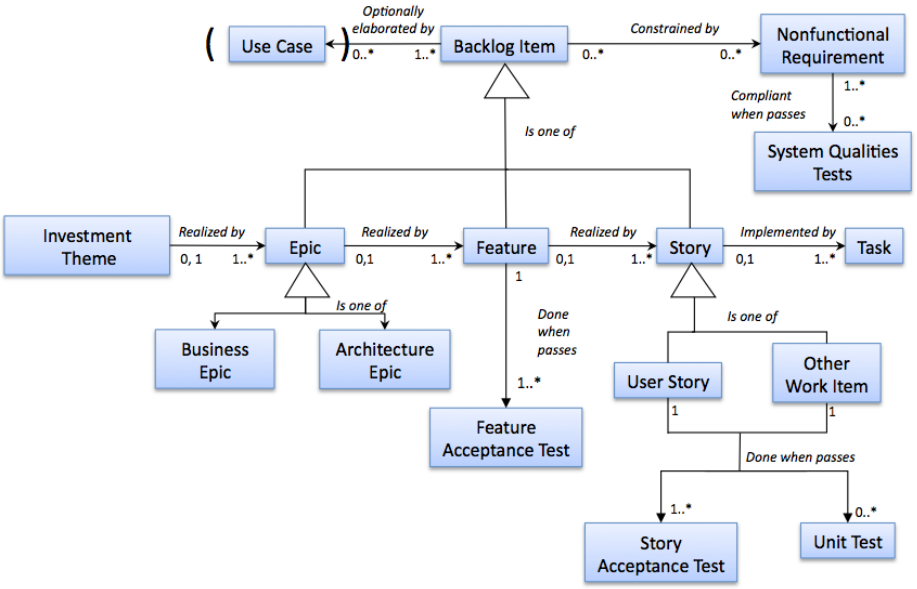

Figure 2. Agile Enterprise Backlog Model

The Epic, Feature, Story, and Acceptance Criteria are written using a textual pattern that includes the stakeholder, wants, and rationale. As an example, features and user stories are in the form of an expression: As a <user role> I want <activity> so that <business value>. Part of the User Story is the Acceptance Criteria that defines how the team will know that the intent of the story has been achieved. The Acceptance Criteria are in the form of: Given <some context>, when <some action is carried out>, then <a particular set of observable consequences should obtain>. These features and stories will be discussed later.

Nonfunctional Requirements describe system attributes such as security, safety, maintainability, performance, and reliability. These are persistent qualities and constraints of the system, and apply to any and all backlog items throughout the model (Leffingwell 2011).

Textual Requirements Grammatical Patterns

A textual requirements grammatical pattern uses the concept of "structured natural language" to help write more consistent, unambiguous, and verifiable requirements. Boeing has identified four different types of patterns for writing requirements, functional/performance, design, environmental, and suitability. Using this pattern, Boeing is able to measure and improve the quality of requirements (Carson 2015). The detailed pattern has four cases:

- Functional/Performance - The AGENT shall FUNCTION in accordance with INTERFACE-OUTPUT with PERFORMANCE [and TIMING upon EVENT TRIGGER in accordance with INTERFACE-INPUT] while in CONDITION.
- Design - The AGENT shall [exhibit] DESIGN CONSTRAINTS [in accordance with PERFORMANCE while in CONDITION].
- Environmental - The AGENT shall [exhibit] CHARACTERISTIC during/after exposure to ENVIRONMENT [for EXPOSURE DURATION].
- Suitability - The AGENT shall exhibit CHARACTERISTIC with PERFORMANCE while CONDITION [for CONDITION DURATION].

*MBSE: S*Models and S*Patterns, and Underlying Representations*

Model-Based Systems Engineering (MBSE) has been a major initiative of INCOSE and the systems engineering community for years, with involvement by thought leaders, practitioners, and commercial tool suppliers (INCOSE OMG Initiative 2015). MBSE shifts the representation of

systems information from prose and informal diagram semantics to explicit and frequently visual formal data structures. MBSE also seeks to improve understanding and communication, reduce ambiguity, and exploit visual cognition of humans and automation capabilities of information technology. INCOSE's Board of Directors has stated an objective to transition to a future where modeling is the norm in systems engineering. There are currently a number of different specific MBSE methodologies (Watson 2015) and modeling languages (SysML 2015; Dori, 2002; IDEF 2015).

One of those methodologies, described by the INCOSE MBSE Patterns Working Group in (Schindel et al 2015), formally separates the underlying conceptual representation semantics from the semantics of model views. This facilitates use of popular modeling languages and tools while separately reconsidering the question of underlying concepts necessary to Systems Engineering in any case. For example (Schindel 2005) illustrates that the whole notion of technical requirements statements can be re-conceptualized as a form of "transfer function", opening new ways to understand requirements while still using a form of natural language prose that becomes part of the MBSE model. The underlying representation semantics are summarized by the S*Metamodel, of Figure 3. An S*Model is any MBSE model that satisfies the S*Metamodel, no matter what modeling language it uses. An S*Pattern is an S*Model of a system family (product line, platform) that can be configured to describe a specific configuration instance, itself an S*Model. Notice the break between High-Level and Detail Level Requirements is different than the break between Stakeholder World Language and Technical World Language in figure 3 below. Examples are in this paper and in (Peterson and Schindel 2013; Schindel 2011).

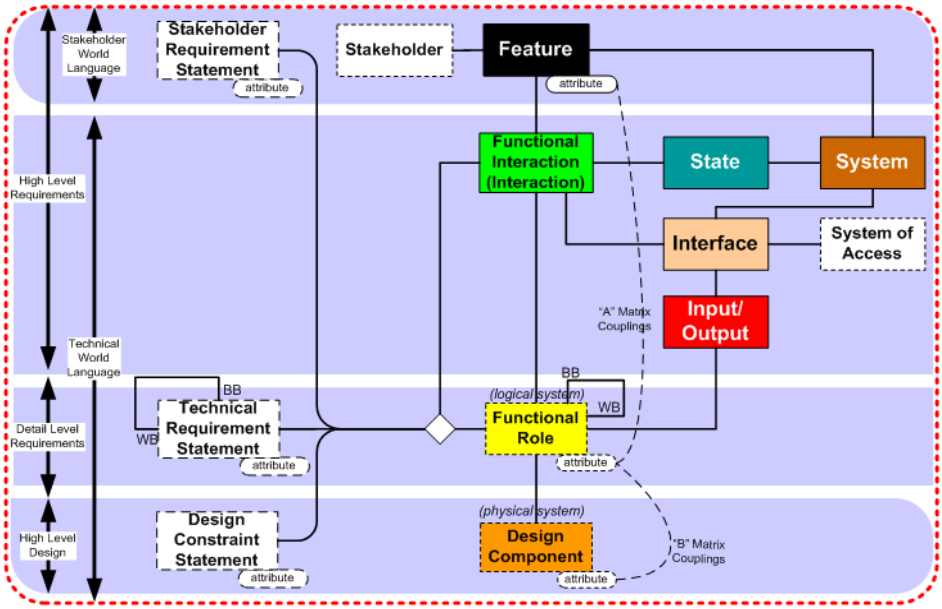

Figure 3**. Summary of S*Metamodel**

Initial Discovery of Requirements, Using MBSE Views

Initial Discovery of Stakeholder Requirements

Consider a team that must write requirements for a remotely piloted rover vehicle. The rover vehicle is a real physical vehicle used in a robotic competition. The team wants to initially identify the stakeholder requirements of the rover vehicle. This section illustrates initial discovery (potentially

contributing to a re-usable pattern), and a later section illustrates configuration from an existing pattern, taking advantage of earlier discoveries.

Using the initial project request, the team identifies the probable classes of stakeholders (shown in Figure 5) and creates an initial system domain diagram (shown in Figure 4, and also known as an external system environment or context diagram). Note that the concept of "stakeholder" (any entity with an interest stake in the system of interest – for example, a company shareholder) and external "actor" (any entity directly interacting with the system of interest—all shown in Figure 4) are not the same. Even though these groups may overlap—not all stakeholders are direct actors, and not all direct actors (e.g.,. the Power Grid in Figure 4) are stakeholders.

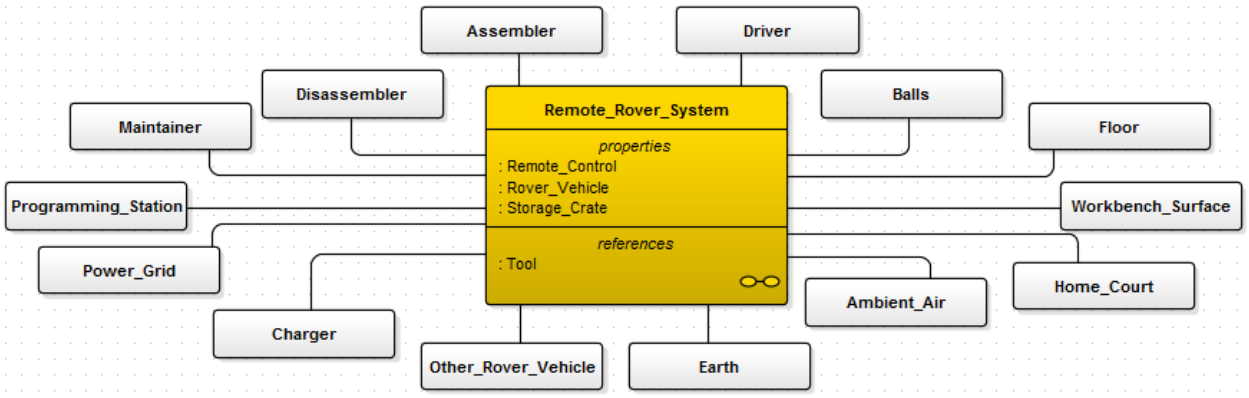

Figure 4. System Domain Diagram

The team identifies stakeholder advocates who will represent the stakeholder classes. The role of such advocacy is called by different names in various methodologies (for example, agile), but in all such cases is a key role in what follows. Typical advocates are representatives of regulatory bodies, quality department representatives, or business managers, who speak on behalf of stakeholders such as at-risk production workers, human users of the product, or company shareholders.

One initial discovery method (from agile practices) is identification of Features from the stakeholder representative(s). Figure 5 illustrates Features associated with the game, electrical recharge and maintainability feature of the rover vehicle. In the underlying S*Metamodel, these Features are the primary modeled modularization of stakeholder value or trade space. They are later used for (1) configuration of the pattern for specific applications, (2) trade-space evaluation to support consideration of alternatives and decisions, (3) Failure Mode and Effects Analysis (FMEA) and other risk analyses, to understand impacts, as well as any other Systems Engineering process in which stakeholder value plays a part. Notice that Features defined in the S*Metamodel are compatible with Features defined in the scaling of agile [Leffingwell]

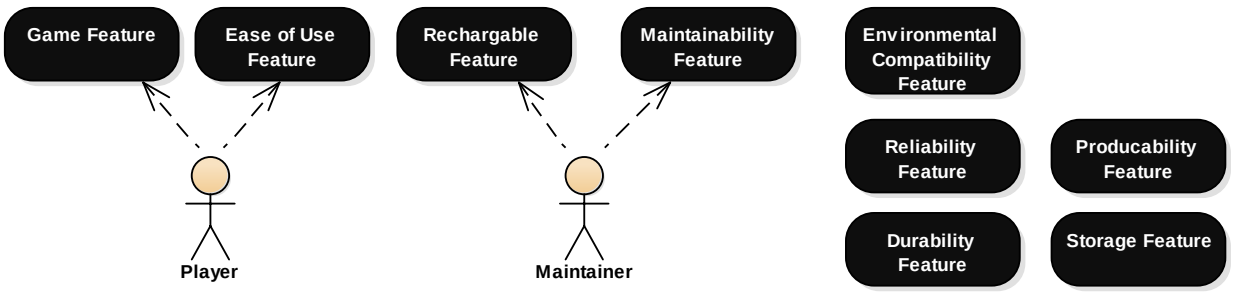

Figure 5. Stakeholder – Features Diagram

Features are chunks of stakeholder value, described in language and concepts of stakeholders, with attributes that further quantify that value in the samestakeholders' nomenclature. Accordingly, Features are frequently subjective (e.g. "Ease of Use" in Figure 5), but are nevertheless a formal part of the S*Model, where they are next associated with Functional Interactions.

In this paper we examine the Game Feature and the Rechargeable Feature to demonstrate how to use diagram views to create or select requirements. In the Game Feature, the Player drives the Rover across the Game surface. In the Rechargeable Feature, the Maintainer recharges the Remote Vehicle System.

Whereas Features represent value-laden subjectivity, Functional Interactions are value-free objective expressions of real physical interactions that will occur between the remote rover and its external actor domain (the value of a Functional Interaction is embodied in the features that it supports). Features support the validation of the system; Functional Interactions support the verification of the system. The association of Features and Interactions establishes a bridge between the subjective stakeholder world and the objective technical world, defining which Interactions are expected to deliver the values described by Features. The discovery of Technical Requirements is thus associated with Interactions, as described in the next section.

Initial Discovery of Technical Requirements

As explained in (Schindel 2005), all Technical Requirements (whether functional or non-functional) can be viewed as a generalization of "transfer functions" associated with external physical Interactions. As further described in (Schindel 2013a), this moves the problem of discovering Requirements to the problem of discovering Interactions — the context for all the laws of the physical sciences (Schindel 2016). MBSE is a major visual help in that discovery process, because there are three MBSE views that provide independent discovery paths into that Interaction space, helping us explore and define it:

1. Feature-Interactions View: This view helps us discover potentially new Interactions based on the values perceived by Stakeholders. (If User Stories are used, these are associated with the same Features, so those Stories help to drive this view.)
2. State-Interactions View: This view helps us discover new Interactions based on situations (as in use cases or system modes) that may be encountered in time.
3. Actor-Interactions View: This view helps us discover potentially new Interactions based on who or what (from the external actor set) is interacting with the system.

This approach is independent of specific MBSE modeling language or methodology, as long as they are capable of representing the minimal S*Metamodel concepts (Schindel 2011). The power of using these three views together is that they all lead to the same underlying set of Interactions, but will nearly always have different visual and mental impacts to remind us of Interactions that were missed in viewing the other two views. The team therefore iterates on this set until the same set of Interactions (shown in Figure 6) has been associated with States (shown in Figure 7) and Features and Actors (shown in Figure 5).

For each Interaction, a set of Technical Requirements can be discovered, or configured from a pattern, if that Interaction is already modeled in the pattern. A later section covers the pattern configuration case, and the following illustrates the initial visual discovery and representation of technical requirements for a given interaction.

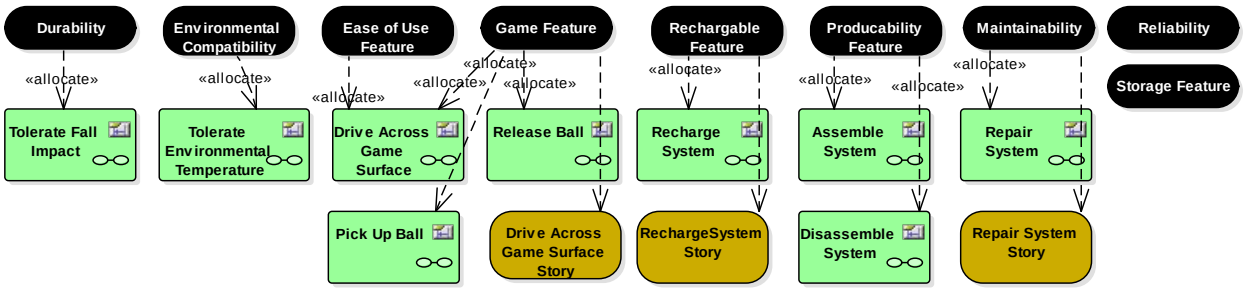

Figure 6. Features-Interactions View

To understand the time domain portion of the requirements, a state machine model used in the S* Model State Interactions View is created to look at the life-cycle and operational states of the rover system. See Figure 7 below. The state model shows the state of the system of interest. The game day begins with the system in storage (Being Stored). The system is then assembled (Being Assembled) and configured (Being Configured) for game play and waits to operate in the Waiting to Operate state. The game is played in the Operating state. When each game is done, the system returns to the Waiting to Operate state. During a game, the system can be discharged, damaged, or can fail. A discharged system is maintained (Being Maintained) and returns to the Waiting to Operate state. Damaged and failed systems are repaired (Being Repaired). When the game play is completed, the system is disassembled (Being Disassembled) and stored (Being Stored). An interaction called Recharge System ~~(Figure 9)~~ is identified for the Being Maintained state. See Figure 7 below. Additional interactions need to be discovered.

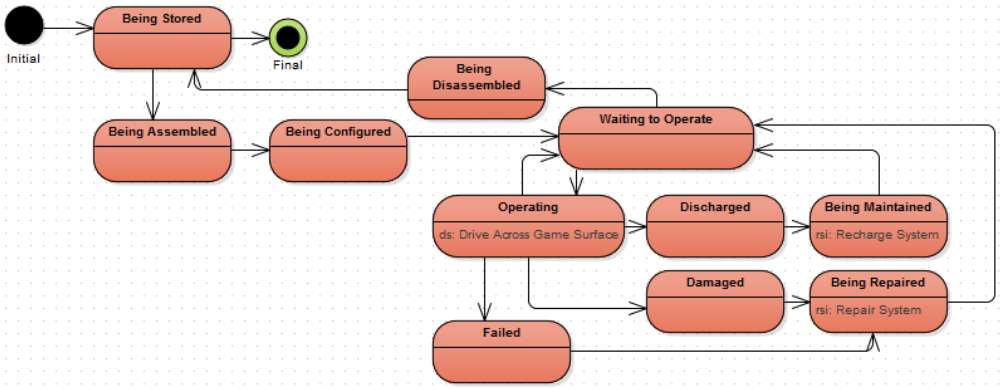

Figure 7. State-Interactions View

All requirements (even those called “non-functional”) for the system of interest occur in behaviors that are visible during external interactions. One interaction can result in many requirements. Figure 8 shows a list of functional interactions captured.

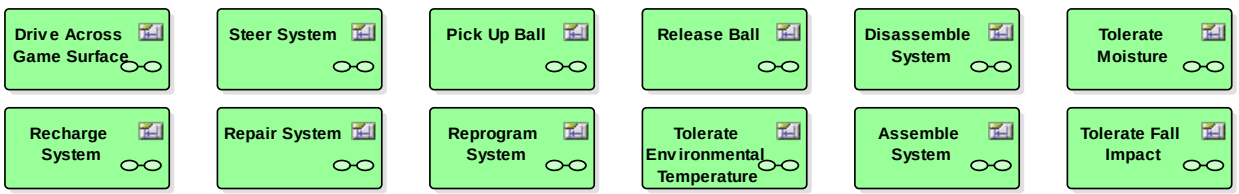

Figure 8. Functional Interactions

The Functional Interactions model shows physical interactions (exchange of forces, energy, mass flows, or information) between two or more objects from the domain model. Each interacting object plays a Functional Role, describing its behavior in the Interaction. Behaviors describe input-output

relationships of the interacting objects. (Schindel 2005) These relationships are expressed in Technical Requirements, and can be parametrized by attributes. For example, the time it takes to calculate or the Minimum_Connect_Force, with a value of 0.5 lbs.

To help understand Functional Interactions, consider the example of a Drive Across Game Surface functional interaction derived from the Game Feature and from the Operating State. The Drive Across Game Surface functional interaction shows information that flows between the player, remote, and rover. See Figure 9 below.

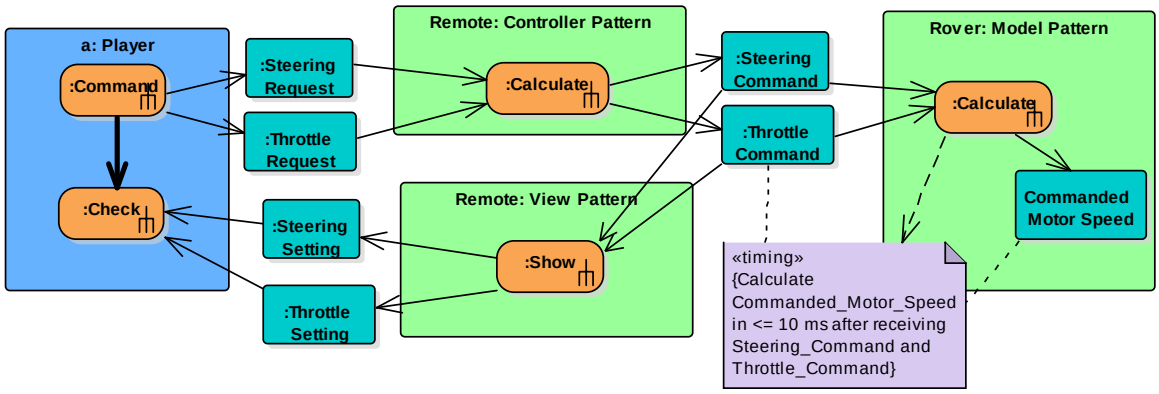

Figure 9. Drive Across Game Surface Functional Interaction

The Player requests Steering and Throttle using a Remote. The Remote and Rover are modeled using a Model-View-Controller pattern. The Remote Controller Pattern calculates a Steering and Throttle command for both the Remote View Pattern and the Rover Model Pattern. The Rover Model Pattern Commands the Motor Speed. The View Pattern shows the Steering Setting and the Throttle Setting. Finally, the Player checks the Steering Setting and the Throttle Setting.

Each behavior in the System (Remote_Rover_System) results in one textual requirement. The following templates are used as a guide to assist in writing the textual requirement from the graphical view:

If [**TRIGGER**],
the **OBJECT** shall **BEHAVIOR**
[(**OUTPUT_DATA** with the **OUTPUT_DATA_ATTRIBUTE**(s))(s)]
[**PERFORMANCE_CONSTRAINT**]
[using (**INPUT_DATA** with the **INPUT_DATA_ATTRIBUTE**(s))(s)]
[while in **STATE** state].

This template is based upon the Four Variable Model described by Parnas. The behavior is the transfer function between the inputs and outputs. For the Drive_Across_Game_Surface functional interaction, the resulting requirements are:

- The Remote_Control_Pattern shall Calculate Steering Command and Throttle_Command using Steering_Request and Throttle_Request while in the Operating state.
- The Remote_Control_Pattern shall Show Steering_Setting and Throttle_Setting using Steering_Command and Throttle_Command while in the Operating state.

The Rover_Model_Pattern requirement adds a performance pattern overlay. This pattern overlays the calculate section of the pattern with the timing constraint. The resulting requirement is:

- The Rover_Model_Pattern shall Calculate Commanded_Motor_Speed in <= 10ms after receiving Steering_Command and Throttle_Command while in the Operating state.

As a second example, consider the Recharge System interaction derived from the Rechargeable Feature and the Being Maintained state. See Figure 10 below.

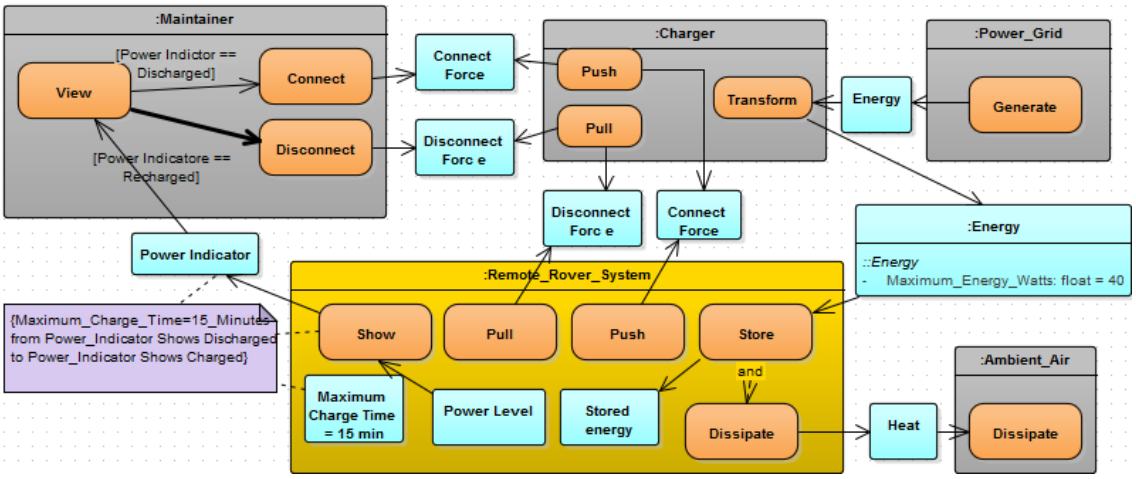

Figure 10. Recharge Function Interaction Diagram

In this Functional Interaction the Maintainer looks at the Power Indicator to see that the Remote_Rover_System is Discharged. The Maintainer then acts to connect a Charger into the Remote_Rover_System. The Power_Grid energy is transformed by the Charger and is stored in the Remote_Rover_System. Heat is dissipated into the Ambient_Air. It takes a Maximum_Charge_Time of 15 minutes to charge the Remote_Rover_System. For the Recharge functional interaction, the resulting requirements are:

- The **Remote_Rover_System** shall **Store Stored_Energy** and **Dissipate Heat** using **Energy** with the **Maximum_Energy_Watts**=40 while in **Being Maintained** state.
- The **Remote_Rover_System** shall **Show Power Indicator** with **Maximum_Charge_Time_Minutes**=15 from **Power_Indicator** Shows **Discharged** to **Power_Indicator** shows **Charged** using **Power_Level** while in **Being Maintained** state.
- The **Remote_Rover_System** shall **Push Connect_Force** with **Maximum_Connect_Force_lbs**=0.5.
- The **Remote_Rover_System** shall **Pull Disconnect_Force** with **Maximum_Disconnect_Force_lbs**=0.5.

Requirements Re-Use, Using MBSE Patterns

Effective initial discovery of requirements, the subject of the previous section, is less impactful if those discoveries have to be repeated in future projects, as if they were new discoveries. This issue is larger than a concern over repeating the labor of a successful discovery process. A common story heard across enterprises (Nolan, Pickard, et al 2015) is the much higher cost of “escapes” (in this case, a missed or misunderstood requirement), later corrected and only then realized to be re-discovery of what was already known elsewhere in the enterprise. The resulting project delay, rework, and opportunity cost of such re-discovery far outweigh the simple cost of repeating an earlier successful discovery process.

Accordingly, this paper is equally concerned with how already-discovered requirements can be applied without incurring the delays, costs, and risks of their re-discovery. In one sense, this might be seen as the familiar systems engineering subject of “requirements reuse”, but we summarize here the requirements part of a more general attack on a larger problem. The INCOSE Patterns Working Group (INCOSE MBSE Initiative Patterns WG 2015) is concerned with an extension to Model-Based Systems Engineering (MBSE) referred to as Pattern-Based Systems Engineering (PBSE). The related methodology, summarized in (Schindel et al 2015), is about the accumulation (including improvement), management, and application of recurring system patterns. This is

applicable to requirements, designs, risks, verification, tests and other aspects as an integrated whole, represented within configurable models called S*Patterns.

In this approach, the project-specific models described in the previous ("discover") section are treated as configured instances (project-specific configured S*Models) of more general models (recurring S*Patterns). Both the individual configured system models and the more general recurring system patterns conform to the underlying S*Metamodel of Figure 3.

Not specific to any modeling language or toolset, the S*Metamodel assures that a minimum set of explicit model concepts (also used in the previous discovery section) are expressed, and provides a consistent framework for expressing both recurring patterns and their project-specific configurations. The underlying S*Metamodel ideas can be mapped into any modeling language and COTS toolset. As a result, Stakeholder Requirements, Technical Requirements, and many other recurring aspects of systems can be incrementally accumulated in an S*Pattern and repeatedly configured for use in individual projects. Refer to Figure 11. The recurring model elements are not treated as a library of loose small-scale components, but are instead integrated into the system model as an explicit expression of recurring holistic patterns. The paradigm of their re-use is thus not based on "search" but instead on configuring the general pattern, driven by Stakeholder Features and cascading through the rest of the system space of requirements, design, test, and other aspects.

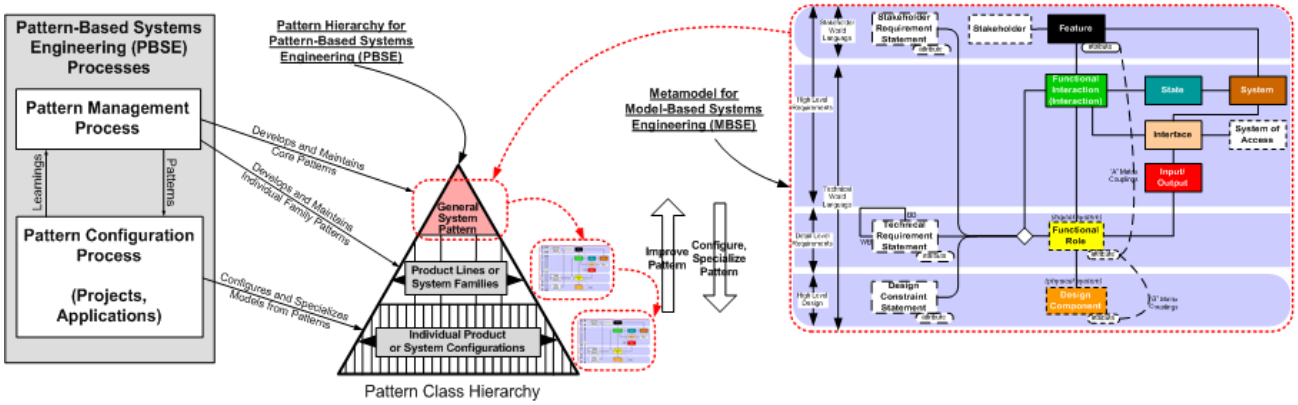

Figure 11. **S*Patterns Are Re-usable, Configurable S*Models**

In this paper, we will confine that process to an illustration of configuration of Stakeholder Features and Technical Requirements. It is also important to note that in real project situations, it is common to see a combination of both the requirements pattern configuration process (re-using what is known), and the requirements discovery process (learning new things), described in the previous section. The S*Metamodel spans both these activities.

Consider first the configuration and re-use of stakeholder requirements. On the surface, we are simply interested in using what we already know about stakeholder interests to re-express them for a specific situation. In PBSE however, this stakeholder pattern area turns out to have deeper significance for the rest of the development process than just stakeholder issues. The configurability of other aspects of the pattern (e.g., technical requirements, design, risks, etc.) is about configuration choices. All those choices involve decisions that are ultimately about the interest of some stakeholder—whether the customer, system operator, company shareholder, distributor, maintainer, manufacturer, or others. Because S*Patterns express all those stakeholder interests in an MBSE Stakeholder Feature model portion of the pattern, configuring this part of the pattern also sets up the

configuration of the rest of the pattern. For example, this is why "feature selection" becomes the "front end" of Product Line Engineering (PLE) processes that use variability models (Pohl et al 2005; ISO/IEC 2013).

Figure 12 illustrates the configuration of Stakeholder Features.

	Mandatory, Optional, or Other Configuration Rule	Populate? (YES/NO)	Feature Name	Feature Attribute Primary Key (PK) Attribute Name	Feature Attribute PK Value #1	Feature Attribute PK Value #2	Feature Attribute PK Value #3	Feature Attribute PK Value #4
5								
6								
7	Mandatory	YES	**Game Feature**	Game Capability	Sense Ball	Navigate	Pick Up Ball	Release Ball
8	Optional	NO	**Rechargeable Feature**	--				
9	Mandatory	YES	**Durability Feature**	Threat Type	Drop Impact	Operational Wear & Tear	Collision	
10	Mandatory	YES	**Ease of Use Feature**	Use Challenge	Steering Control	Ball Handling Control	Learning Time	Help Aids
11	Optional	YES	**Environmental Compatibility Feature**	Environmental Issue	Storage Temperature	Storage Humidity	Water Resistance	Sunlight Tolerance
12	Mandatory	YES	**Maintainability Feature**	Maintenance Capability	Fault Diagnostics	BIST	Part Replacement	
13	Mandatory	YES	**Reliability Feature**	--				
14	Optional	YES	**Storage Feature**	Storage Type	Home Closet	Outdoor	nvas Cover	
15	Mandatory	YES	**Producability Feature**	--		Home Closet Outdoor Canvas Cover		
16								
17								

Figure 12. Configuration of Stakeholder Features

One result of this process, shown in Figure 12, is the configuration of the already known Stakeholder Features for this system. This could be viewed as a "first draft" set of stakeholder requirements for a given project, subject to further discovery with stakeholder representatives, as described in the earlier section on discovery. This approach means viewing the S*Pattern as a continuously improving asset, an MBSE representation of cross-team learning across enterprise projects. This also addresses a challenge of accumulating a proliferation of User Stories sometimes encountered by agile systems engineering teams that find they are organizing them into configurable pattern assets.

Configuring S*Features means populating their instances with different attribute values, and setting those attribute values. Because of the integrated nature of an S*Pattern, this leads directly to configuration of the related system Functional Interactions, Functional Roles, and related Technical Requirements statements. Again, this might be viewed as a "first draft" set of requirements for a specific project, still subject to human review and improvement. However, it is typically a much more complete and consistent first draft than traditionally experienced and available much sooner as a result of exploiting the requirements pattern. Figure 13 illustrates the resulting populated Technical Requirements.

Feature	Interaction	Functional Role	Requirement ID	Requirement
Rechargable Feature	Recharge System	Remote Rover System	RRS-201	The **Remote_Rover_System** shall **Store Stored_Energy** and **Dissipate Heat** using **Energy** with the **Maximum_Energy_Watts**=40 while in **Being Maintained** state.
Rechargable Feature	Recharge System	Remote Rover System	RRS-202	The **Remote_Rover_System** shall **Show Power Indicator** with **Maximum_Charge_Time_Minutes**=15 from **Power_Indicator** Shows **Discharged** to **Power_Indicator** shows **Charged** using **Power_Level** while in **Being Maintained** state.
Rechargable Feature	Recharge System	Remote Rover System	RRS-203	The **Remote_Rover_System** shall **Transmit_Push Connect_Force** with **Maximum_Connect_Force_lbs**=0.5.
Rechargable Feature	Recharge System	Remote Rover System	RRS-204	The **Remote_Rover_System** shall **Transmit_Pull Disconnect_Force** with **Maximum_Disconnect_Force_lbs**=0.5.
Rechargable Feature	Recharge System	Charger	CRG-301	The **Charger** shall **Transform AC_Electrical_Energy**=110VAC to **DC_Charging_ Voltage**=5.0 V and **Charging_Current**=0.2A.
Rechargable Feature	Recharge System	Power_Grid	PGD-151	The **Power_Grid** shall **Generate AC_Electrial_Energy**=110VAC.

Figure 13. Configured Technical Requirements

Conclusions, Implications, and Future Work

Based on the above, leading conclusions, and implications for process and infrastructure include:

1. The combined use of Domain, State, and Feature views provides a visual cognition based means of initial discovery of system external interactions, and of the related system technical requirements.
2. The MBSE patterns capture the learning that organizations do to create products customers want. This captured product learning lowers the future cost of re-application and improves the overall quality of requirements and the system they specify.
3. Stakeholder requirements discovery methods (IE User Stories) fit into a visual modeling method by placing these in a Stakeholder feature space. Using modeling in this trade space allows for better understanding and improved decision making.
4. The same feature space provides a visual representation of modular configuration options and variants, beginning with their stakeholder impact.
5. A hybrid process is the recommended best practice. It begins with what is known (from existing patterns) and includes new discoveries (from the current project). The approach uses User Stories, MBSE, views, metrics and patterns on top of IT resources that manage the patterns across projects.

The relationship between epics, features, user stories, acceptance criteria, and technical requirements in an MBSE context, as well as providing additional reuse examples and benefits of using this method from real-world experience, deserves further exploration.

References

Carson, Ronald S. 2013. "Implementing Structured Requirements to Improve Requirements Quality." *Proceedings of INCOSE 2015 (Seattle, WA, USA).*

Cook, D. and Schindel, W., "Utilizing MBSE Patterns to Accelerate System Verification", in *Proc. of INCOSE 2015 International Symposium*, 2015.

Dori, Dov. *Object-Process Methodology, a Holistic Systems Paradigm*, Springer, 2002

Friedenthal, S. et al, INCOSE/OMG MBSE Initiative wiki: retrieved 2015 from: http://www.omgwiki.org/MBSE/doku.php

Hooks, Ivy. 1993. "Writing Good Requirements", Proceedings of the Third International Symposium of the INCOSE - Volume 2. http://homepages.laas.fr/kader/Hooks.pdf

IDEF Language Reference, retrieved 2015 from: http://www.idef.com/idef0.htm

INCOSE MBSE Initiative Patterns Working Group web site, retrieved 2015 from: http://www.omgwiki.org/MBSE/doku.php?id=mbse:patterns:patterns

ISO/IEC 26550, "Software and systems engineering — Reference model for product line engineering and management", International Standards Organization, 2013.

Jackson, M. 2000. "*Problem Frames: Analyzing and Structuring Software Development Problems"*. Great Britain, Addison-Wesley / ACM Press.

Lempia, D. L. and Miller, S. P. "DOT/FAA/AR-08/32. Requirements Engineering Handbook", Federal Aviation Administration, 2009.

Leffingwell, Dean, *Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise*. Addison-Wesley 2011

Mattson, Mark P. "Superior Pattern Processing Is the Essence of the Evolved Human Brain." *Frontiers in Neuroscience*. Frontiers Media S.A., 22 Aug. 2014.

Nolan, A., Pickard, A., Russell, J, and Schindel, W. "When two is good company, but more is not a crowd", Proc. of INCOSE International Symposium, Seattle, WA, 2015.

Parnas, D. and Madey, J. "Functional Documentation for computer Systems Engineering (Ver. 2), " Technical Report CRL 237, McMaster University, Hamilton, Ontario, Sept 1991.

Peterson, Troy, and Schindel, Bill. "Introduction to Pattern-Based Systems Engineering (PBSE): Leveraging MBSE Techniques", INCOSE IS2013 Tutorial, 2013

Pohl, K, Bockle, G, and van der Linden, Frank. *Software Product Line Engineering: Foundations, Principles, and Techniques*, Springer, 2005.

Schindel, W. "Requirements statements are transfer functions: An insight from model-based systems engineering", Proceedings of INCOSE 2005 International Symposium, (2005).

Schindel, W. and Dove, D., "Introduction to the Agile Systems Engineering Life Cycle Pattern", *Proceeding of INCOSE International Symposium*, 20165b

Schindel, W. "System Interactions: Making The Heart of Systems More Visible", *Proc. of INCOSE Great Lakes Regional Conference*, 2013a.

Schindel, W. "What Is the Smallest Model of a System?", *Proc. of the INCOSE 2011 International Symposium*, International Council on Systems Engineering (2011).

Schindel, W., "Got Phenomena? Science-Based Disciplines for Emerging Systems Challenges", *Proc. of INCOSE 2016 International Symposium*, 2016.

Schindel, W. et al, "MBSE Methodology Summary: Pattern-Based Systems Engineering (PBSE), Based On S*MBSE Models", V1.5.5A, INCOSE Patterns Working Group, retrieved 2015a from INCOSE/OMG MBSE methodologies directory:
http://www.omgwiki.org/MBSE/doku.php?id=mbse:pbse

Sherey, Jason. "Capitalizing on Systems Engineering", Proc. of INCOSE International Symposium, Rochester, NY, 2005.

SysML 2015: "OMG Systems Modeling Language", Version 1.4, retrieved 2015 from: http://www.omg.org/spec/SysML/1.4/Beta/PDF

Walden, D. et al, eds., *Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities*, Fourth Edition, INCOSE, 2015.

Watson, John. INCOSE Directory of MBSE Methodologies: retrieved 2015 from http://www.omgwiki.org/MBSE/doku.php?id=mbse:methodology

Biography

David Lempia is a Principal Systems Engineer in the Engineering Infra-structure & Integrity (EI&I) organization of Rockwell Collins with over 20 years of experience in systems development. He is currently leading Agile Transformation across Rockwell Collins. He is an author of papers on Requirements Engineering Management, Model Based Development, Agile, and Lean.

Bill Schindel is president of ICTT System Sciences. His engineering career began in mil/aero systems with IBM Federal Systems, included faculty service at Rose-Hulman Institute of Technology, and founding of three systems enterprises. Bill co-led a project on Systems of Innovation in the INCOSE System Science Working Group, co-leads the Patterns Working Group, and is a member of the lead team of the INCOSE Agile Systems Engineering Life Cycle Discovery Project.

Terry Hrabik is a Principal Systems Engineer in the Engineering Infra-structure & Integrity (EI&I) organization of Rockwell Collins. He has over 20 years of experience in systems engineering and engineering management, including positions at Tellabs Operations and the MITRE Corporation. For the past 15 years, he has supported several DoD programs in many different roles including Integration and Test Lead, Lead Systems Engineer, and Chief Architect. He is currently the Systems Engineering Lead for the Rockwell Collins School of Engineering.

Mike Graber is a Sr Systems Engineer in the Engineering Infra-structure & Integrity (EI&I) organization of Rockwell Collins. He has over 20 years of experience in systems/software development/testing and project management which includes MCI. He is currently leading the Engineering Project Leadership area for the School of Engineering across Rockwell Collins. Over the last 12 years, he has led several successful projects within the Government Systems business unit as a Technical Project Manager involving Systems, Software, and Hardware disciplines.

Stewart McGill is a Principal Systems Engineer in the Engineering Infra-structure & Integrity (EI&I) organization of Rockwell Collins. He has over 20 years of experience in mechanical/systems engineering and engineering management, including positions at Honeywell, Inc. and the United States Marine Corps. He is currently the Systems Engineering Lead for Strategic Development and Transformation for Engineering Tools.