Reuse in Systems Engineering

Gan Wang, *Member, IEEE*, Ricardo Valerdi, *Member, IEEE*, and Jared Fortune, *Member, IEEE*

***Abstract*—Reuse in systems engineering is a frequent but poorly understood phenomenon. Nevertheless, it has a significant impact on system development and on estimating the appropriate amount of systems engineering effort with models like the Constructive Systems Engineering Cost Model (COSYSMO). Practical experience showed that the initial version of COSYSMO, based on a "build from the scratch" philosophy, needed to be refined in order to incorporate reuse considerations that fit today's industry environment. The notion of reuse recognizes the effect of legacy system definition in engineering a system and introduces multiple reuse categories for classifying the four COSYSMO size drivers—requirements, interfaces, algorithms, and operational scenarios. It fundamentally modifies the driver counting rules and updates its definition of system size. It provides an enabling framework for estimating a system under incremental and spiral development. In this paper, we present: 1) the definition of the COSYSMO reuse extension and the approach employed to define this extension; 2) the updated COSYSMO size driver definitions to be consistent with the reuse model; 3) the method applied to defining the reuse weights used in the modified parametric relationship; 4) a practical implementation example that instantiates the reuse model by an industry organization and the empirical data that provided practical validation of the extended COSYSMO model; and 5) recommendations for organizational implementation and deployment of this extension.**

***Index Terms*—Cost estimation, metrics, reuse, systems engineering.**

I. Introduction

ALMOST all systems have a legacy. Today, more often than not, systems are developed based on an evolution of previous systems. New releases of software are developed by modifying and enhancing a previous release. Similarly, new generations of airplanes, ships, and automobiles are developed by improving the functionality and performance of previous models. In many situations, product lines are managed through incremental improvement of previous system definitions. In other situations, existing systems are modernized through technology insertions and obsolescence management. Although system-level examples where previously developed components and capabilities have been leveraged can generally be easily identified, the current literature fails to address how such instances of reuse at the system-level should be appropriately quantified [22]. In other domains, such as software, reuse is a much better documented concept [16], [19].

When a "new system" is developed, it commonly uses existing components, proven functionality, or established architecture. Across industry, we have witnessed ever increasing trends of COTS-integration in system development and ever diminishing endeavors of constructing something completely new from a "clean slate". However, incorporating disparate elements together into a developing system is often no easy task, as the integration of legacy elements can negatively impact project resources, sometimes substantially [11]. Knowing this, addressing reuse from a systems perspective, particularly the impact on the systems engineering effort required throughout the life cycle, is critical to accurately estimating the development cost of a system.

The concept of reuse has been broadly studied, from near-philosophical discussions in terms of technological inventions [9] to useful techniques used in semiconductor fabrication design process [12]; from knowledge management perspective [13] to manufacturing and service activities [7]. Perhaps the most extensive effort to date has been the focus on software reuse [16], examining issues ranging from methods and techniques [8], [17], processes [5], [16], [20], to productivity and economic impacts [4], [5], [10], [23], and to social and behavior phenomenon in reuse [2], [15]. COCOMO [4], [5], for example, captures reuse quantitatively by characterizing source lines of code (SLOC) in terms of process and design maturity in developing new software and provides a practical approach for estimating new development with the benefit of such a reuse. Systems engineering, as a relatively young field in engineering, has yet made cognitive steps in formulating its own reuse strategy. As the result, the reuse has been mostly opportunistic and little guidance has been offered to proactively leverage the practice. In fact, methods are only emerging to quantitatively evaluate systems engineering activities and characterize the economic impact and productivity [6], [31]. As systems engineering content rapidly increases in developing a system, more practical guidelines are urgently needed.

One method for estimating the amount of systems engineering effort required for a project is the Constructive Systems Engineering Cost Model (COSYSMO) [25]. Developed at the University of Southern California with the support of a consortium of academic, industry, and government organizations, COSYSMO is a parametric model for estimating the systems engineering and integration effort required for the conceptualization, design, test, and deployment of software and hardware systems and executing projects that develop such a system. The model provides a parametric relationship that estimates systems engineering effort under nominal schedule, in person months, based on four size drivers—system requirements

Manuscript received November 03, 2009; revised April 06, 2010. Date of publication June 28, 2010; date of current version September 01, 2010. This work was supported in part by BAE Systems, the Industrial Affiliates of the Center for Systems and Software Engineering, and the Consortium members of the Lean Advancement Initiative.

G. Wang is with BAE Systems, Reston, VA 20190 USA (e-mail: gan.wang@baesystems.com).

R. Valerdi is with the Engineering Systems Division, Massachusetts Institute of Technology, Cambridge, MA 02139 USA (e-mail: rvalerdi@mit.edu).

J. Fortune is with the Center for Systems and Software Engineering, University of Southern California, Los Angeles, CA 90089 USA (e-mail: fortune@usc.edu).

Digital Object Identifier 10.1109/JSYST.2010.2051748

(REQ), system interfaces (INT), system algorithms (ALG), and operational scenarios (SCN)—and adjusted by fourteen effort multipliers, which capture the product and project environment and complexity factors. COSYSMO defines a sizing quantity called "system size" from a weighted sum of the four size drivers. The estimating relationship is shown in (1) as follows:

$$\mathrm{PM_{NS}} = A \cdot \left(\sum_k (w_{e,k}\Phi_{e,k} + w_{n,k}\Phi_{n,k} + w_{d,k}\Phi_{d,k}) \right)^E \cdot \prod_{j=1}^{14} \mathrm{EM}_j \quad (1)$$

where

$\mathbf{PM}_{\mathrm{NS}}$	effort in person months (nominal schedule);
A	calibration constant derived from historical project data;
k	{REQ, IF, ALG, SCN};
$\mathrm{w_x}$	weight for "easy," "nominal," or "difficult" size driver;
Φ_x	quantity of "k" size driver;
E	represents (dis)economies of scale;
EM	effort multiplier for the j_{th} cost driver. The geometric product results in an overall effort adjustment factor to the nominal effort.

Being the first of its kind, the model has, in a very short period of time, caught the attention of the systems engineering community, industry and academia alike. It has demonstrated the potential to bridge a long-time gap between system complexity and its corresponding systems engineering effort estimate. Organizations have made various attempts to pilot the model and apply it to practical applications [21], [28].

Early application of the Constructive Systems Engineering Cost Model (COSYSMO), however, has revealed that the model did not recognize the concept of reuse in systems engineering [27], [29]. It assumes that all of its four size drivers—system requirements, system interfaces, system algorithms, and operational scenarios—are new entities when sizing a system. In other words, the model is based on a "build from scratch" philosophy and assumes all systems are developed from a "clean slate".

This differs from how systems are typically built today since requirements for a new system may be "adopted" from an existing system. Furthermore, some of the new system's requirements may be "modified" from a prior system. Moreover, the evolution of system requirements over the system life cycle may result in "deleted" requirements from the initial configuration baseline. The same situations may apply to the other three size drivers—interfaces, algorithms, and operational scenarios. As a result, the calculated system size does not reflect reusing these system elements and, consequently, can result in inaccurate estimate of systems engineering effort required to realize such a system. This problem intensifies when dealing with the incremental and spiral development.

Therefore, we propose incorporating the concept of reuse for estimating the size of a system, in order for COSYSMO to more accurately estimate the systems engineering effort.

II. Development Approach

When defining the concept of reuse for COSYSMO, the following basic principles hold true.

1) COSYSMO is an open model, developed by the community and for the community. Users are free to change and/or extend the model.
2) On the other hand, it is beneficial for the industry to agree on the basic definition, relationship and parameters to better communicate basis of estimates.

COSYSMO has been developed as an open model by the community of academia, industry and government for the use of the general public. This implies that everyone is free to adopt, modify, and/or extend the model. In fact, it is intended for organizations to adapt to their own engineering processes and business models, and to develop local, tailored estimating tools. In defining reuse as another aspect of the model, it should not, in any way, restrict or hinder individual applications or adaptations of this model. In fact, it should help to facilitate such a local implementation.

On the other hand, similar to other cost estimating models, COSYSMO provides a common method for the industry at large to measure and communicate basis of estimate and productivity. It is important that the same basic definitions are consistently understood and applied. This includes definition of terms and nomenclatures, the parametric relationship, and the guidelines for measurement.

However, the above two principles could inherently be conflicting with each other. Free adaptation of the model could lead to individual interpretations, while over-restriction of the model definition could limit its application. Care must be taken to strike a fine balance to preserve both of the above principles, so that it does not over-constrain its application across the industry and, at the same time, preserve the integrity of the model.

The approach taken was to aggregate the organizational definitions complementarily based on a "minimum common denominator" strategy. The reuse model has been piloted in several organizations in the aerospace industry through their individual implementation of the COSYSMO model. With the benefit of periodic interactions, we found that the organizations essentially implemented the same basic model definition with minor differences, mostly with preferred choice of words. For the industry definition, the following guidelines were agreed upon.

A) Establish a minimum set of reference categories for each of the four size drivers, so that organizations can either directly apply and/or expand to additional reuse categories.
B) Provide minimum common-denominator definitions, so that organizations can use, refine, and/or instantiate as appropriate to fit their operational needs.

Goal "A" indicates that we define only those reuse categories that all stakeholders can agree on as the common denominator for the community, and these categories are intended as the reference for individual implementations. In other words, the intent is to always maintain this set of categories, but one may add other categories, generally as subdivisions, if appropriate.

Goal "B" states that the definitions are intentionally structured with the minimum common denominator language so that organizations can either directly use or, if necessary, may refine and substantiate the definitions to fit their operational use.

Several industry-level round-tables and workshops have been conducted under the stated guidelines to achieve the community agreement over a period of more than a year that involved stakeholders from the major aerospace organizations, commercial companies, and government agencies. While some of these workshops are in person, others are via teleconferences. The participants are from ten's of attendees in a conference to a few key stakeholders on a telecom. Topics range from discussions on terminologies to quantitative exercises to define weights, as described in Section V. As a result, the following reuse extension has been defined.

III. The COSYSMO Reuse Extension

The approach taken to represent the system size is analogous to that used to represent software code size in which there are several categories of code, including new and different levels of reuse, as well as deleted [3], [18], [24]. This is motivated by its legacy—COCOMO II [5]. In the case of software, the size of the code is often represented as Equivalent New Source Lines of Code (ESLOC). ESLOC is computed as the weighted sum of the new, the reused, the modified, and the deleted code. Similarly, we define equivalent requirements" (eReqs) in COSYSMO as a function of the weighted sum of the new, reused, modified, and deleted requirements in a system.

The COSYSMO reuse model consists of five categories for counting its size drivers, termed: *new*, *modified*, *deleted*, *adopted*, and *managed*. The quantities of the four COSYSMO size drivers, i.e., number of requirements, number of interfaces, number of algorithms, and number of operational scenarios, may be classified into one of five categories below.

1) *New*: Items that are completely new.
2) *Modified*: Items that are inherited, but are tailored.
3) *Deleted*: Items that are removed from a system.
4) *Adopted*: Items that are incorporated unmodified. Also known as "black box" reuse.
5) *Managed*: Items that are incorporated unmodified and untested.

As an example, a requirement can be *new*, which no precedence can be found for the system to be developed. New items are generally unprecedented and may be associated with a low level of familiarity. A requirement may be *modified* in the sense that a heritage element it is associated with is reused, but needs a limited level of modification or tailoring for the element to be fully incorporated in the new system. A requirement may also be *adopted* where the indicated functionality and performance has previously been developed and can therefore be incorporated without any changes. This is commonly referred to as "black box" reuse since a realized requirement is literally copied and remains unchanged. A requirement that is considered *managed* when an associated element is incorporated as a turn-key component with minimal development effort, except for engineering management. Please note that testing in this context refers to the formal and complete system test procedures and that the word "untested" means bypassing such formal steps. *Deleted* requirements are those that are already in the legacy system or architecture design, but need to be removed from the current system definition based on customer need or contractual commitments.

It is important to note that the *modified* category can span a wide range of possible effort. Modification may entail a simple change or a complete revamp of the entire system architecture. The intent of the *modified* category is to capture those elements that involve tailoring or interface-level changes only, with no changes to the interior architecture. Therefore, those items that are inherited but require a significant amount of architectural or implementation-level changes should be counted as *new*.

For each size driver, three additional complexity levels are defined in COSYSMO: *easy*, *nominal*, and *difficult*. These levels are invariant in the context of reuse. Depending upon the point of view, one may consider that there are three levels of complexity within each category of reuse. Or, alternatively, within each level of difficulty, there can be five categories of reuse. Conceptually, the two notions—*categories of reuse* and *levels of difficulty*—form a two dimensional classification framework for size drivers that provide adequate level of granularity in determining system size. Therefore, the estimating relationship incorporating reuse is expressed as (2)

$$\mathrm{PM}_{\mathrm{NS}} = A \cdot \left[\sum_k \left(\sum_r w_r (w_{e,k}\Phi_{e,r,k} + w_{n,k}\Phi_{n,r,k} + w_{d,k}\Phi_{d,r,k}) \right) \right]^E \cdot \prod_{j=1}^{14} \mathrm{EM}_j \qquad (2)$$

where

$\mathbf{PM}_{\mathrm{NS}}$	effort in person months (nominal schedule);
A	calibration constant derived from historical project data;
k	{REQ, IF, ALG, SCN};
r	{New, Modified, Deleted, Adopted, Managed};
w_r	weight for defined degrees of reuse;
w_x	weight for "easy," "nominal," or "difficult" size driver;
Φ_x	quantity of "k" size driver;
E	represents diseconomies of scale;
EM	effort multiplier for the j_{th} cost driver. The geometric product results in an overall effort adjustment factor to the nominal effort.

IV. Modified Size Driver Definitions

With the introduction of reuse into its parametric relationship, the COSYSMO size driver definitions require their amendments. As an example, the original definition for algorithm contains the verbiage: "This driver represents the number of *newly defined or significantly altered* functions..." [25], which directly conflicts with the concept of reuse such as "adopted."

TABLE I
RATING DEFINITIONS FOR NUMBER OF SYSTEM REQUIREMENTS DRIVE

Easy	Nominal	Difficult
- Simple to implement	- Moderately difficult to implement	- Complex to implement or engineer
- Traceable to source	- Can be traced to source with some effort	- Hard to trace to source
- Little requirements overlap	- Some overlap	- High degree of requirements overlap

TABLE II
RATING DEFINITIONS FOR NUMBER OF SYSTEM INTERFACES

Easy	Nominal	Difficult
- Simple	- Moderate complexity	- Complex protocol(s)
- Uncoupled	- Loosely coupled	- Highly coupled
- Strong consensus	- Moderate consensus	- Low consensus
- Well behaved	- Predictable behavior	- Poorly behaved

These definitions must be reexamined to resolve the inconsistencies.

These proposed changes in definitions were elaborated with stakeholders at the COSYSMO working group meeting at the Practical Software and Systems Measurement User Group Meeting in Denver, CO, in July 2007. As the result, the amended COSYSMO size driver definitions, consistent with the reuse extension, are given as follows.

A. *Number of System Requirements (Table I)*

This driver represents the number of requirements for the system-of-interest at the system level or the level of "sell-off" to customer, which may include derived requirements at the same level. The quantity of requirements includes those related to the effort involved in system engineering the system interfaces, system specific algorithms, and operational scenarios. Requirements may be functional, performance, feature, or service-oriented in nature depending on the methodology used for specification. They may also be defined by the customer or contractor. Each requirement must have systems engineering effort associated with it such as verification and validation (V&V), functional decomposition, functional allocation, etc. System requirements can typically be quantified by counting the number of applicable "shalls" in the system or marketing specification.

B. *Number of System Interfaces (Table II)*

This driver represents the number of shared physical and logical boundaries between system components or functions (internal interfaces) and those external to the system (external interfaces). These interfaces typically can be quantified by counting the number of unique external and internal system interfaces among ISO/IEC 15288-defined [14] system elements at the system level for the system-of-interest.

TABLE III
RATING DEFINITIONS FOR NUMBER OF SYSTEM-SPECIFIC ALGORITHMS

Easy	Nominal	Difficult
-Algebraic	- Straight forward calculus	- Complex constrained optimization; pattern recognition
-- Straightforward structure	- Nested structure with decision logic	- Recursive in structure with distributed control
- Simple data	- Relational data	- Noisy, ill-conditioned data
- Timing not an issue	- Timing a constraint	- Dynamic, with timing and uncertainty issues
- Adaptation of library-based solution	- Some modeling involved	- Simulation and modeling involved

TABLE IV
RATING DEFINITIONS FOR NUMBER OF OPERATIONAL SCENARIOS

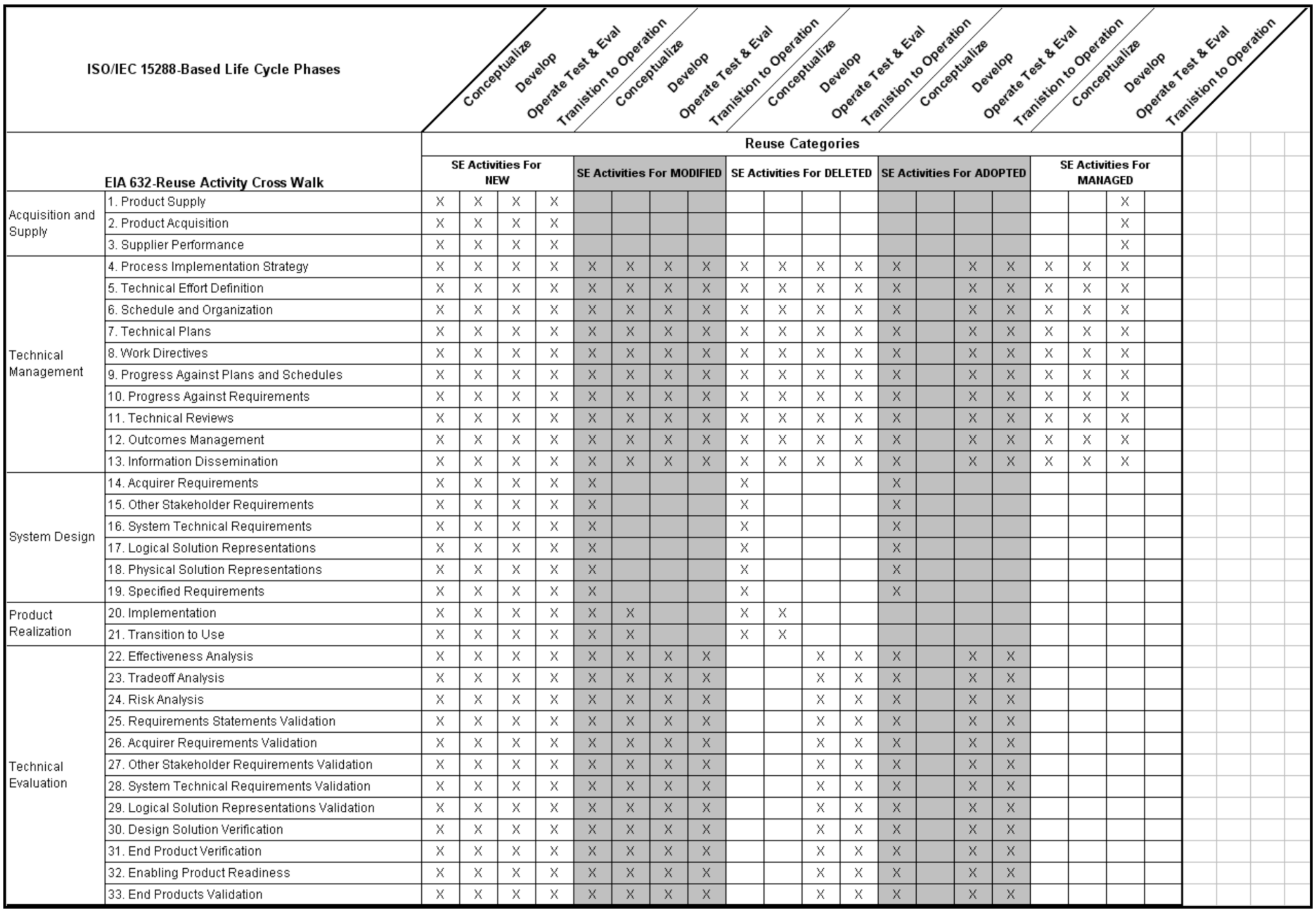
Easy	Nominal	Difficult
- Well defined	- Loosely defined	- Ill defined
- Loosely coupled	- Moderately coupled	- Tightly coupled or many dependencies/conflicting requirements
- Timelines not an issue	- Timelines a constraint	- Tight timelines through scenario network
- Few, simple off-nominal threads	- Moderate number or complexity of off-nominal threads	- Many or very complex off-nominal threads

C. *Number of System-Specific Algorithms (Table III)*

This driver represents the number of mathematical algorithms to be derived in order to achieve the system functional and performance requirements. As an example, this could include a complex aircraft tracking algorithm like a Kalman Filter being derived using existing experience as the basis for the all aspect search function. Another example could be a discrimination algorithm being derived to identify friend or foe function in space-based applications. The number can be quantified by counting the number of unique algorithms needed to realize the requirements specified in the system specification or mode description document.

D. *Number of Operational Scenarios (Table IV)*

This driver represents the number of operational scenarios that a system must satisfy in order to accomplish its intended mission. An operational scenario must be end-to-end and triggered by an operational event. Such scenarios include both the nominal stimulus-response thread plus all of the off-nominal threads resulting from bad or missing data, unavailable processes, or other exceptional conditions. The number of scenarios can typically be quantified by counting the number of use cases or operational modes captured in the user manual, including



ISO/IEC 15288-Based Life Cycle Phases / EIA 632-Reuse Activity Cross Walk		SE Activities For NEW				SE Activities For MODIFIED				SE Activities For DELETED				SE Activities For ADOPTED				SE Activities For MANAGED			
		Conceptualize	Develop	Operate Test & Eval	Tranistion to Operation	Conceptualize	Develop	Operate Test & Eval	Tranistion to Operation	Conceptualize	Develop	Operate Test & Eval	Tranistion to Operation	Conceptualize	Develop	Operate Test & Eval	Tranistion to Operation	Conceptualize	Develop	Operate Test & Eval	Tranistion to Operation
Acquisition and Supply	1. Product Supply	X	X	X	X															X	
	2. Product Acquisition	X	X	X	X															X	
	3. Supplier Performance	X	X	X	X															X	
Technical Management	4. Process Implementation Strategy	X	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X	X	X	
	5. Technical Effort Definition	X	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X	X	X	
	6. Schedule and Organization	X	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X	X	X	
	7. Technical Plans	X	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X	X	X	
	8. Work Directives	X	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X	X	X	
	9. Progress Against Plans and Schedules	X	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X	X	X	
	10. Progress Against Requirements	X	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X	X	X	
	11. Technical Reviews	X	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X	X	X	
	12. Outcomes Management	X	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X	X	X	
	13. Information Dissemination	X	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X	X	X	
System Design	14. Acquirer Requirements	X	X	X	X	X				X				X							
	15. Other Stakeholder Requirements	X	X	X	X	X				X				X							
	16. System Technical Requirements	X	X	X	X	X				X				X							
	17. Logical Solution Representations	X	X	X	X	X				X				X							
	18. Physical Solution Representations	X	X	X	X	X				X				X							
	19. Specified Requirements	X	X	X	X	X				X				X							
Product Realization	20. Implementation	X	X	X	X	X	X			X	X										
	21. Transition to Use	X	X	X	X	X	X			X	X										
Technical Evaluation	22. Effectiveness Analysis	X	X	X	X	X	X	X	X			X	X	X		X	X				
	23. Tradeoff Analysis	X	X	X	X	X	X	X	X			X	X	X		X	X				
	24. Risk Analysis	X	X	X	X	X	X	X	X			X	X	X		X	X				
	25. Requirements Statements Validation	X	X	X	X	X	X	X	X			X	X	X		X	X				
	26. Acquirer Requirements Validation	X	X	X	X	X	X	X	X			X	X	X		X	X				
	27. Other Stakeholder Requirements Validation	X	X	X	X	X	X	X	X			X	X	X		X	X				
	28. System Technical Requirements Validation	X	X	X	X	X	X	X	X			X	X	X		X	X				
	29. Logical Solution Representations Validation	X	X	X	X	X	X	X	X			X	X	X		X	X				
	30. Design Solution Verification	X	X	X	X	X	X	X	X			X	X	X		X	X				
	31. End Product Verification	X	X	X	X	X	X	X	X			X	X	X		X	X				
	32. Enabling Product Readiness	X	X	X	X	X	X	X	X			X	X	X		X	X				
	33. End Products Validation	X	X	X	X	X	X	X	X			X	X	X		X	X				

Reuse Categories

Fig. 1. Systems engineering activity versus life cycle phase mapping by reuse categories.

off-nominal extensions, developed as part of the operational architecture.

V. Weight Definition for Reuse Categories

We present in this section the approach used to define the weights for the reuse categories in the COSYSMO (2). It is important to note that the approach outlined below is designed to capture the statistical behavior of a group of the projects, rather than individual behavior of a particular project. In fact, on an individual basis, a project may exhibit a vastly different pattern of labor distribution relatively to reuse. An *adopted* or *modified* element could prove to be more costly than a brand-new element in terms of life cycle systems engineering effort.

The approach taken is bottoms-up activity-based, by which we define the reuse weights by evaluating life cycle systems engineering activities. In particular, we examined the 33 systems engineering activities in five activity groups defined by the ANSI/EIA 632 standard [1] relative to four life cycle phases derived from (but not exactly the same as) the stages defined in ISO/IEC 15288—Systems Life Cycle Processes.

The result of this analysis is presented in the matrix in Fig. 1, where we identify the applicable activities by life cycle for each defined reuse category. Along the x-axis, the four life cycle phases are repeated for each defined reuse category, namely, *Conceptualize*, *Develop*, *Operational Test & Evaluation*, and *Transition to Operation*. Along the y-axis are the 33 systems engineering activities in the five activity groups.

The analysis determines the applicability of an activity across the life cycle for a particular reuse category in this framework. As an example, realizing a new requirement into a product would in general incur all of the activities as specified by EIA-632. A reused requirement, on the other hand, would likely exclude some of the activities. The underlying assumption is that a reused element generally saves systems engineering effort compared to a new element. This matrix allows qualitative distinction between relative scales for *reused* and *new*. The exercise was conducted through round-tables. During an exercise, for example, a particular relationship is proposed, discussion among participants would ensue, and a consensus would eventually be reached. If an activity relationship existed, an "X" was placed in a matrix where the activity, category, and life cycle phase intersect. If an activity relationship did not exist, the cell was left blank. After completing the matrix, the weight of each reuse category was obtained by effectively summing the number of "X's" in each category.

The next step is to turn the qualitative relationship to a quantitative one. This is done with an effort distribution table derived from an industry wide-band Delphi survey [26], as shown in Fig. 2. Similarly, the four life cycle phases from ISO/IEC 15288 and the five systems engineering activities from ANSI/EIA 632 are presented. The value in each cell of the matrix represents the

	Phases				
EIA 632 Fundamental Process	Conceptualize	Develop	Operational Test & Eval.	Transition To Operation	Fundamental Process Total
Acquisition & Supply	1.96%	3.57%	0.91%	0.56%	7.00%
Technical Management	3.74%	6.46%	4.25%	2.55%	17.00%
System Design	10.20%	12.00%	5.10%	2.70%	30.00%
Product Realization	1.95%	4.50%	4.80%	3.75%	15.00%
Technical Evaluation	5.58%	8.37%	12.40%	4.65%	31.00%
Percentage of Total Systems Engineering Effort Per Phase	23.43%	34.90%	27.46%	14.21%	**100.00%**

Fig. 2. Life cycle systems engineering effort distribution [15].

Num. Requirements

	New	Modified	Deleted	Adopted	Managed
	100.00%	64.65%	50.70%	43.37%	15.36%
Easy	0.5	0.32325	0.2535	0.21685	0.0768
Nominal	1	0.6465	0.507	0.4337	0.1536
Difficult	5	3.2325	2.535	2.1685	0.768

Num. Interfaces

	New	Modified	Deleted	Adopted	Managed
	100.00%	64.65%	50.70%	43.37%	15.36%
Easy	1.1	0.71115	0.5577	0.47707	0.16896
Nominal	2.8	1.8102	1.4196	1.21436	0.43008
Difficult	6.3	4.07295	3.1941	2.73231	0.96768

Num. Algorithms

	New	Modified	Deleted	Adopted	Managed
	100.00%	64.65%	50.70%	43.37%	15.36%
Easy	2.2	1.4223	1.1154	0.95414	0.33792
Nominal	4.1	2.65065	2.0787	1.77817	0.62976
Difficult	11.5	7.43475	5.8305	4.98755	1.7664

Num. Scenarios

	New	Modified	Deleted	Adopted	Managed
	100.00%	64.65%	50.70%	43.37%	15.36%
Easy	6.2	4.0083	3.1434	2.68894	0.95232
Nominal	14.4	9.3096	7.3008	6.24528	2.21184
Difficult	30	19.395	15.21	13.011	4.608

Fig. 3. Activity-based weight derivation for reuse categories.

New	Modified	Deleted	Adopted	Managed
100.0%	64.7%	50.7%	43.4%	15.4%

Fig. 4. Aggregated weights for the reuse categories.

percentage of the total effort applied to a particular activity in a particular life cycle phase. Each systems engineering process yields a unique effort profile. For example, the Acquisition and Supply activity typically represents 7% of the total systems engineering effort across four phases of the life cycle. The total sums to 100%, which corresponds to the life cycle effort of developing a *new* system or system element from concept to delivery.

The weights of the reuse categories are derived by combining the results in Fig. 1 with the data in Fig. 2 and by prorating the effort percentage for a given activity. For example, in the *adopted* category, the effort for System Design process is not significant in the Development phase. Hence, we will assign the value of 0% or remove the original effort value (12%) for that cell. On the other hand, the Technical Evaluation effort is significant and comparable to that in the *new* category for the Operational Test and Evaluation phase. We retain the original percent effort value (12.4%) for that cell. These exercises result in the series of weight tables for each reuse category, as shown in Fig. 3. Aggregated weight values for a size driver of *nominal* difficulty are shown in Fig. 4.

The reuse weights are summarized along a continuum in Fig. 5 to illustrate two additional points. First, it should be noted

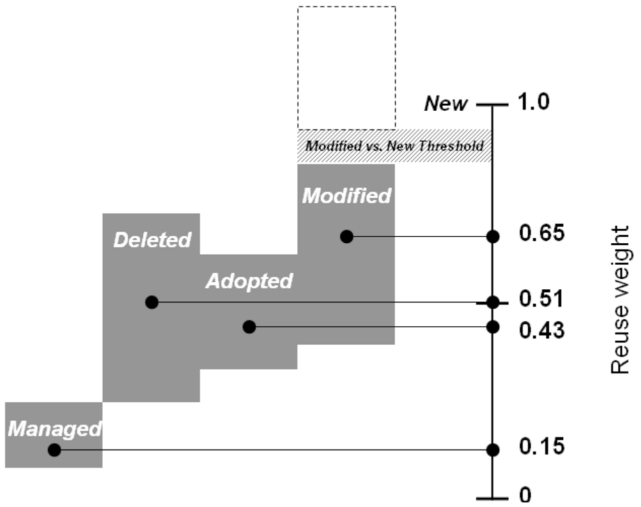


Fig. 5. Reuse continuum.

that the weight values in Fig. 4 represent the nominal values, or the mode, for the respectively categories. The exact weights may fall within a range of possible values that may be greater than or less than the suggested values or a set of distributions whose mode is represented by the values in Fig. 4. This presents an opportunity for further tailoring by each organization that wishes to incorporate reuse into their COSYSMO implementation and to more accurately capture organizational productivity. For example, in the *modified* case the corresponding weight may be lower than 0.65 in situations where there is very little modification taking place. Such a situation may arise when the color of an airplane is changed from a Forest Green to Sea Grey. This is a simple modification of a requirement that does not demand critical changes in systems engineering effort. On the other hand, significant modifications may emerge which can result in a higher weight for the *modified* parameter. This may arise when the previous requirement is modified to work in a new environment that was previously considered. Such a scenario frequently arises when companies attempt to modify system components from commercial helicopters to military helicopters. Different operational and performance criteria apply when such components are incorporated into the military domain.

The second point illustrated by the continuum in Fig. 5 is the existence of the *Modified vs. New Threshold*. This is relevant in cases where extreme modification of requirements causes the original reused requirement to be more complex than a new requirement. In this situation, the systems engineer must make a tradeoff decision to determine whether it is better to "throw away" the old requirement and start with a new one or keep the

	Tech. Management	Requirement Definition	System Analysis & Design	Architecture Change & Implementation	Tailoring / Interface Change	V&V Testing
New	√	√	√	√	√	√
Modified	√	√	√		√	√
Deleted	√	√	√		√	√
Adopted	√	√				√
Managed	√					

Fig. 6. Activities-based classification wizard for reuse classification.

old requirement in spite of its extra expense. The range of possible weights for *modified* requirements may theoretically exceed the weight for *new*, but the exploration of such values was beyond the scope of this analysis.

The approach as presented above can be followed to derive organization-specific reuse weights. Operationally, it is important to note that these weights, once defined or derived in an organization, should be applied to all data points consistently, between the calibration data and new estimates. It is not to be redefined for each data point or new estimate.

VI. A Practical Application Example

For the past three years, a diversified aerospace industry organization has been developing an estimating tool based on COSYSMO by calibrating the model to its product lines. During the course of this project, a significant amount of historical data was collected to calibrate the model to the major product lines and platforms. This development effort provided the first organizational implementation and validation of the COSYSMO reuse model. At the same time, the organization has been part of the core stakeholder group and led the industry's effort in defining the reuse extension. In order to achieve a practical and deployable estimation tool, the reuse definition was elaborated and additional specifications were added so that it is better adapt to organizational process and development models. To guide the local implementation, a set of guidelines were followed, as shown below.

- Be consistent with industry definition. Define such reuse categories by adding refining details to the industry definitions so as to avoid any potential conflicts and inconsistencies.
- Provide clear and consistent *operational guidelines* for driver counting and classification, by using unambiguous verbiage in the reuse definition.
- Establish clear boundaries between categories to ensure easy separation and consistency.

Therefore, this organizational implementation further instantiated the reuse model and provided more specific definitions for the five categories, as follows (differences are *italicized*).

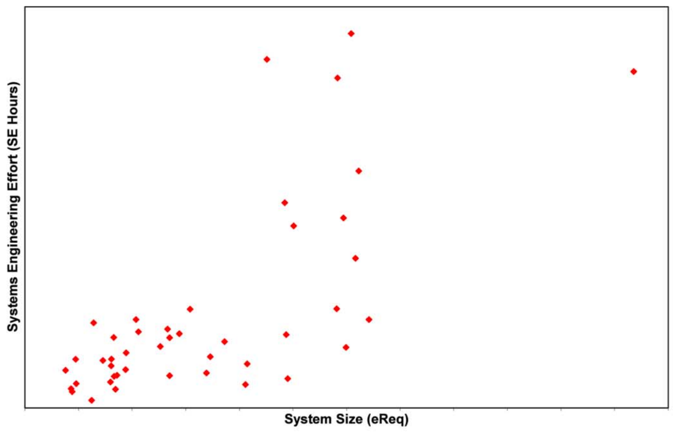
1) *New*: Items that are completely new.
2) *Modified*: Items that are incorporated *but require tailoring or interface changes, and verification and validation testing*.
3) *Deleted*: Items that are removed from a *legacy* system, *which require design analysis, tailoring or interface changes, and verification and validation testing*.
4) *Adopted*: Items that are incorporated unmodified *but require verification and validation testing*. Also known as "black box" reuse.
5) *Managed*: Items that are incorporated unmodified and untested, *and require no additional SE effort other than technical management*.

Several points are worth noting for the above definitions. First, these definitions are directly inherited from the industry definition. However, additional clarification of the base definitions has been given with added qualifiers.

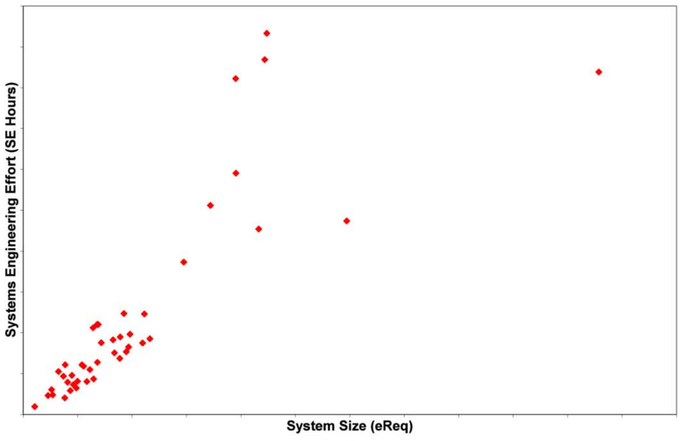
Secondly, one of the challenges to overcome is the subjectivity of COSYSMO, which is prone to individual interpretation and, consequently, inconsistent sizing of systems. Our approach is to define a classification framework for counting size drivers with two orthogonal dimensions—*reuse* by systems engineering activities and *levels of difficulties* by relative effort—to enable finer grain estimation of these drivers. The activity-based framework involves six high-level systems engineering activities for the development life cycle: 1) Technical Management; 2) Requirement Definition; 3) System Analysis and Design; 4) Architecture Changes & Implementation; 5) Tailoring and Interface Changes; 6) Verification & Validation Testing. A reference table was created, as shown in Fig. 6, to serve as a Rosetta Stone between the industry definitions and the organizational implementation. Easy to apply in practice and can be related by most systems engineers, they were used by end-users as the discriminators in delineating the reuse categories. The determination of a reuse category depends upon the required activities to realize a size drive (e.g., requirement) in an end-to-end development life cycle.

Finally, we strongly advocated and recommended the category called "*managed*" to the industry definition, which we believe is important in capturing the intricacies of today's evolutionary and spiral development, as well as prevalent teaming arrangement between industry partners. This category is intended for two main circumstances. The first is when a new system incorporates legacy elements that have already been developed and verified and validated from a prior system, the systems engineering activities now are mostly limited to technical management. The second situation is when a part of the system under development is subcontracted out or uses COTS/GOTS-based components that are "turn-key" or "plug-and-play." The requirements and other drivers related to these subtracted parts have already been verified and validated by the providers. To the prime contractor, the majority of the activities required are technical (subcontract) management in nature.

The instantiated model was calibrated with a set of historical program data from several lines of business and major sites



(a)



(b)

Fig. 7. Distribution of the same data set, (a) before and (b) after applying the extended COSYSMO reuse model. (a) Before applying the reuse model, all drivers are counted as new. (b) After apply the reuse model.

across the country in at a diversified aerospace engineering company. The set included data from over 50 systems development projects. The data was analyzed with and without applying the reuse model. The result is significantly improved data correlation and calibrations with a higher-degree of estimation accuracy and confidence level when reuse is applied. Fig. 7(a) shows the data before applying the reuse model and Fig. 7(b) shows the same data points after applying the reuse model, with everything else in the COSYSMO model held constant.

It is evident from the heteroscedasticity of the data that the reuse model significantly improved the predictive accuracy of the COSYSMO model. This improvement is a result of adjusting the model closer to reality, as it has been proved by practical experience that reuse is more the rule rather than the exception.

VII. Conclusion and Recommendations

In this paper, we have defined a reuse extension for COSYSMO. We presented an industry definition, as well as an organizational implementation as the practical validation and implementation example of the reuse model. We discussed the approach applied to this development and the method used for deriving the reuse categories and weights. We also presented the updated COSYSMO size driver definitions to be consistent and compatible with the reuse extension.

To implement this extension for the operational use, an organization can directly apply the method presented in this paper. It may consider further instantiating the definitions to establish refined boundaries that are tailored to its business model, product lines, and engineering process of the respective organizations. Organizations may also find it necessary to add additional reuse categories. When doing so, it is recommended that the original reuse categories be preserved rather than changing the established categories.

The weights for reuse, once defined, should be consistently applied across all data points and over time, between calibration data and new estimates. They should not be changed for a single estimate and calibration point to avoid comparing "apples and oranges". Any change to these definitions may require recollection of all the calibration data points, and change to the derived weights may require recalculation of all the system sizes. This can be costly. This is required for the necessary level of consistency between programs and between calibrations and new estimates. In other words, this is to ensure that consistency of requirements is realized across programs and system size is measured with the same scale and counting rules.

As a community of systems engineers interested in cost estimation, we cannot dictate each individual organization's extension of the reuse model, but we should, however, agree on a set of values for reuse weights. This is desirable to ensure consistent understanding of estimate system size and to better communicate basis of estimates. This will be a continuing effort in the refinement of the reuse approach which will involve feedback from key stakeholders from leading organizations.

The initial version of COSYSMO has established a frontier for systems engineering cost estimation. However, as with any other methodology in its early stage, it requires continuous improvement so that it can gain the level of maturity required by operational use and potentially as a new industry standard. The reuse model is still evolving. At the completion of this paper manuscript, the authors have collaborated with the industry stakeholder groups and further extended the reuse model by adding one additional category. "*Design for Reuse*" is envisioned as the sixth reuse category and defined as the system artifacts that require additional upfront investment in order to improve potential reusability in later system life cycle. This category is generally assigned a weight of greater than 1.0, which in represents greater amount of systems engineering effort than that required for the *New* category. The validation of this category is in progress.

The authors are also engaged in other enhancement efforts to further improve the fidelity of the model. One of these areas is the cost drivers or the effort multipliers used in the model to scale the estimate effort based on the system size [30]. We are following a similar strategy in combining expert opinion and historical data to develop the most realistic and accurate model possible. We will report the progress of these activities in the near future.

ACKNOWLEDGMENT

The authors wish to thank other collaborators that have provided valuable insight and helpful discussions on this work. G. Roedler and J. Gaffney of Lockheed Martin, J. Rieff of Raytheon, D. Ligett of Softstar Systems, and B. Boehm of USC were influential in the direction and outcome of this work.

REFERENCES

[1] ANSI/EIA-632-1988 Processes for Engineering a System 1999, ANSI/EIA.

[2] S. Biffl and T. Grechenig, "Degrees of consciousness for reuse of software in practice: Maintainability, balance, standardization," in *Proc. Computer Software and Applications Conf.*, 1993, pp. 107–114.

[3] T. Biggerstaff, A. Perlis, T. Biggerstaff, and A. Perlis, *IEEE Trans. Softw. Eng. (Special Issue on Software Reusability)*, vol. SE-10, no. 5, Sep. 1984.

[4] B. W. Boehm, *Software Engineering Economics*. Upper Saddle River, NJ: Prentice-Hall, 1981.

[5] B. Boehm, C. Abts, A. W. Brown, S. Chulani, B. Clark, E. Horowitz, R. Madachy, D. Reifer, and B. Steece, *Software Cost Estimation With COCOMO II*. Upper Saddle River, NJ: Prentice-Hall, 2000.

[6] D. N. Card, "The challenges of productivity measurement," in *Proc. Pacific Northwest Software Quality Conf.*, Portland, OR, Oct. 2006.

[7] J. E. Ettlie and M. Kubarek, "Design reuse in manufacturing and services," *J. Product Innov. Manag.*, vol. 25, no. 5, 2008.

[8] P. Freeman, "Reusable software engineering: Concepts and research directions," in *ITT Proc. Workshop on Reusability in Programming*, New York, 1983, pp. 129–137.

[9] L. Fleming, "Recombinant uncertainty in technological search," *Manag. Sci.*, vol. 47, no. 1, 2001.

[10] J. E. Gaffney and T. A. Durek, "Software reuse—Key to enhanced productivity: Some quantitative models," *Inform. Softw. Technol.*, vol. 31, no. 5, 1989.

[11] D. Garlan, R. Allen, and J. Ockerbloom, "Architectural mismatch or why it's hard to build systems out of existing parts," in *Proc. 17th Int. Conf. Software Eng.*, Seattle, WA, Apr. 1995.

[12] N. Gil and S. Beckman, "Design reuse and buffers in high-tech infrastructure development: A stakeholder perspective," *IEEE Trans. Eng. Manag.*, vol. 54, no. 3, pp. 484–497, Aug. 2007.

[13] B. J. Hicks, S. J. Culley, R. D. Allen, and G. Mullineux, "A framework for the requirements of capturing, storing and reusing information and knowledge in engineering design," *Int. J. Inform. Manag.*, vol. 22, no. 4, 2002.

[14] ISO/IEC. ISO/IEC 15288:2002(E) Systems Engineering—System Life Cycle Processes 2002.

[15] S. Isoda, "Experience report on software reuse project: Its structure, activities, and statistical results," in *Proc. 14th Int. Conf. Software Engineering*, Capri, Italy, Jun. 1992.

[16] Y. Kim and E. A. Stohr, "Software reuse: Survey and research directions," *J. Manag. Inform. Syst.*, vol. 14, no. 4, Spring 1998.

[17] C. W. Krueger, "Software reuse," *ACM Comput. Surv.*, vol. 24, no. 2, pp. 131–183, Jun. 1992.

[18] H. Mili, F. Mili, and A. Mili, "Reusing software: Issues and research directions," *IEEE Trans. Softw. Eng.*, vol. 21, no. 6, pp. 528–562, Jun. 1995.

[19] J. Poulin, J. Caruso, and D. Hancock, "The business case for software reuse," *IBM Syst. J.*, vol. 32, no. 4, pp. 567–594, 1993.

[20] S. T. Redwine and W. E. Riddle, "Software reuse processes," in *Proc. ACM Software Process Workshop*, Kennebunkport, ME, Oct. 1989.

[21] J. Reiff, J. Gaffney, and G. Roedler, "2007: The breakout year for COSYSMO," in *Proc. Practical System and Software Measurement Users Group Conf.*, Golden, CO, 2007.

[22] A. Sage, "Systems engineering and systems management for reengineering," *J. Syst. and Softw.*, vol. 30, no. 1, pp. 3–25, 1995.

[23] R. W. Selby, "Quantitative Studies of Software Reuse," in *Software Reusability vol. 2, Applications and Experience*, T. J. Biggerstaff and A. J. Perlis, Eds. Reading, MA: Addison-Wesley, 1989.

[24] R. Selby, "Enabling reuse-based software development of large-scale systems," *IEEE Trans. Softw. Eng.*, vol. 31, no. 6, pp. 495–510, Jun. 2005.

[25] R. Valerdi, "The Constructive Systems Engineering Cost Estimation Model (COSYSMO)," Ph.D. dissertation, Univ. Southern California, Los Angeles, May 2005.

[26] R. Valerdi and M. Wheaton, "ANSI/EIA 632 as a standard WBS for COSYSMO," in *Proc. AIAA 1st Infotech@Aerospace Conf.*, Arlington, VA, Sep. 2005.

[27] R. Valerdi, J. Gaffney, G. Roedler, and J. Rieff, "Extensions of COSYSMO to represent reuse," in *Proc. 21st Int. COCOMO Forum*, Los Angeles, CA, Oct. 2006.

[28] R. Valerdi, J. Rieff, G. Roedler, M. Wheaton, and G. Wang, "Lessons learned from industrial validation of COSYSMO," in *Proc. 17th INCOSE Symp.*, San Diego, CA, Jun. 2007.

[29] G. Wang, R. Valerdi, A. Ankrum, C. Millar, and G. Roedler, "COSYSMO reuse extension," in *Proc. 18th INCOSE Int. Symp.*, Utrecht, The Netherlands, Jun. 2008.

[30] G. Wang, R. Valerdi, B. Boehm, and A. Shernoff, "Proposed modification to COSYSMO estimating relationship," in *Proc. 18th INCOSE Int. Symp.*, Utrecht, The Netherlands, Jun. 2008.

[31] G. Wang, L. Saleski, A. Shernoff, and J. C. Deal, "Measuring systems engineering productivity," in *Proc. 20th INCOSE Int. Symp.*, Chicago, IL, Jul. 2010.

Gan Wang (M'83) received the B.S. degree in electrical engineering from Harbin Institute of Technology, China, in 1981, the M.S. degree in electrical engineering from George Mason University, Fairfax, VA, in 1987, the Ph.D. degree in electrical engineering from the University of Virginia, Charlottesville, in 1991, and the M.B.A. degree from the University of Maryland, College Park, in 2002.

He is a Principal Investigator for system-of-systems engineering and integration strategic initiatives at BAE Systems, Reston, VA. He has been engaged in the research and development of decision support methods and life cycle cost modeling and practice for systems engineering and enterprise-level, system-of-systems engineering and management. Prior to joining BAE Systems, he spent many years developing real-time geospatial data visualization applications for mission planning and rehearsal, battlefield command and control (C2), flight simulation, and aircrew training systems. He also developed control systems and aircraft simulation models for various man-in-the-loop flight training systems. He has over 20 years of experience in systems engineering, software development, research and development, and engineering management involving complex, software-intensive systems.

Dr. Wang is a member of INCOSE and PMI.

Ricardo Valerdi (M'95) received the B.S. degree in electrical engineering from the University of San Diego, San Diego, CA, in 1999, and the M.S. and Ph.D. degrees in industrial and systems engineering from the University of Southern California (USC), Los Angeles, in 2002 and 2005, respectively.

He is a Research Associate at the Lean Advancement Initiative, Massachusetts Institute of Technology, Cambridge, and a Founding Member of the Systems Engineering Advancement Research Initiative. He is also a Visiting Associate at the Center for Systems and Software Engineering at USC, and a Senior Member of the Technical Staff at the Aerospace Corporation in the Economic and Market Analysis Center. Formerly, he was a Systems Engineer at Motorola and General Instrument Corporation.

Dr. Valerdi is a member of INCOSE and serves on its Board of Directors.

Jared Fortune (M'09) received the B.S. and M.S. degrees in 2006 and the Ph.D. degree in 2009, all in industrial and systems engineering from the University of Southern California (USC), Los Angeles.

He is a Senior Member of the Technical Staff at The Aerospace Corporation in the Economic and Market Analysis Center. He is also a Visiting Associate at the Center for Systems and Software Engineering, USC.

Dr. Fortune is a member of AIAA and INCOSE.