Sea Level Requirements as Systems Engineering Size Metrics

Ricardo Valerdi University of Southern California	Jatin Raj San Diego State University

Abstract. The Constructive Systems Engineering Cost Model (COSYSMO) represents a collaborative effort between industry, government, and academia to develop a general model to estimate systems engineering effort. The model development process has benefited from a diverse group of stakeholders that have contributed their domain expertise and historical project data for the purpose of developing an industry calibration. But the use of multiple stakeholders having diverse perspectives has introduced challenges for the developers of COSYSMO.

Among these challenges is ensuring that people have a consistent interpretation of the model's inputs. A consistent understanding of the inputs enables maximum benefits for its users and contributes to the model's predictive accuracy. The main premise of this paper is that the reliability of these inputs can be significantly improved with the aide of a sizing framework similar to one developed for writing software use cases. The focus of this paper is the first of four COSYSMO size drivers, # of Systems Requirements, for which counting rules are provided. In addition, two different experiments that used requirements as metrics are compared to illustrate the benefits introduced by counting rules.

Introduction

COSYSMO is part of a trend to improve cost estimating accuracy and increase domain understanding in systems engineering that can potentially lead to increased productivity [Boehm, et al 2000]. The model estimates the effort associated with systems engineering of projects in the aerospace domain based on a variety of parametric drivers that have been shown to have an influence on cost. Based on stakeholder inputs, it uses EIA/ANSI 632 *Processes for Engineering a System*[1] as a basis for defining the activities being estimated by the model and ISO/IEC 15288 *Standard for System Life Cycle Processes*[2] as a basis for defining the Systems Engineering life cycle phases.

Throughout the development of the model a significant amount of feedback has been received from industrial partners such as Raytheon, Lockheed Martin, SAIC, BAE Systems, and General Dynamics; government partners such as the US Army and the Aerospace Corporation; and professional societies such as INCOSE, Practical Software and Systems Measurement, Space Systems Cost Analysis Group, and the International Society of Parametric Analysts. The crucial step in reaching consensus among these organizations has been defining consistent counting rules that provide guidance on how to interpret the model parameters. Specifically, these counting rules help define *what* to count and *how* to count it. This paper provides the counting rules for what is believed to be the common language for communicating user needs: #

[1] Electronic Industries Alliance/American National Standards Institute 1999
[2] International Organization for Standardization 2002

of System Requirements. Pertinent background is provided on the model development and the relationship to the other size drivers in the model. An example system specification is used to illustrate the counting rules and a use case framework is tailored to help define the appropriate level for requirements identification. Finally, two studies involving requirements counting are compared to quantitatively demonstrate the benefits of counting rules.

Model Development

In response to the need for System Engineering cost estimation, USC formed a working group to develop the model using volunteer experts from industry, government, and professional organizations. Initial investigations revealed that the modeling of system engineering activities relied on heuristics and rules of thumb. This led to a lack of confidence in the estimates from both contractors and acquisition organizations. Moreover, system engineering contractors were struggling to justify their estimates during their bid and proposal process because of the absence an equivalent model used by software engineering such as COCOMO II[3] or Cost Xpert[4] and hardware engineering such as SEER-H[5] or PRICE-H[6]. Program managers found it easy to slash the systems engineering budget which generally was inconsistently defined across organizations. It was also a challenge to estimate the right amount of systems engineering for certain programs for which some rules of thumb had been developed [Honour 2004].

An initial framework of the model was developed using behavioral analyses performed with the help of the aforementioned organizations. These analyses included the cost and size drivers as well as their range of variation. Members of the team collaborated to solidify the definitions of the parameters and focused on the most difficult drivers: the size parameters. While experts agreed that the number of requirements was an indicator of the functional size of the systems engineering effort, most agreed that it was not a sufficient predictor. Additional parameters were needed to capture the effort associated with specifying the interface requirements, algorithm development, and operational concepts of the system. As the team dug deeper it found that determining how to count these parameters was not a straight-forward task. More difficult was the fact that requirements were not consistently defined across organizations or contracts. Moreover, it was discovered that the domain in which systems engineering was performed influenced the selection of these parameters as well as how these parameters were counted.

An environment surrounded by inconsistent systems engineering domains and dynamic definitions of requirements offered two distinctively difficult challenges. To address the first challenge the group decided to leverage off the existing EIA/ANSI 632 *Standard for Engineering a System* as a common work breakdown structure. This helped participating companies adapt their WBS to emulate EIA/ANSI 632 as much as possible. Asking organizations to emulate similar WBS activities created significant work since the mapping had to be done retroactively [Valerdi & Wheaton 2005]. Nevertheless, most domains were compatible with the standard to some degree. The second challenge, the focus of this paper, was more difficult to address because the accuracy of COSYSMO was largely dependent on a consistent definition of requirements.

---

[3] COCOMO II (Constructive Cost Model) is a product of the Center for Software Engineering at USC
[4] Cost Xpert is a product of Cost Xpert Group, Inc.
[5] SEER-H (System Evaluation and Estimation of Resources -Hardware) is a product of Galorath, Inc.
[6] PRICE-H (Parametric Review of Information for Costing and Evaluation-Hardware) is a product of PRICE Systems, LLC.

Size Driver Attributes and Counting Rules

After achieving consensus about the model cost drivers, the biggest outstanding challenge for COSYSMO was to ensure that the sizing inputs involved in the system engineering effort were specified consistently. In response, a two phased approach was developed: (1) ensure that the definitions have little room for misinterpretation, and (2) provide a framework that users can use to understand the scope associated with each driver.

Phase One. The first phase has been an ongoing effort since the inception of the model in 2001. Working group meetings present an opportunity to review and refine the definitions of each driver to prevent confusion. The four size drivers that the working group converged on are *# of Requirements, # of Interfaces, # of Critical Algorithms,* and *# of Operational Scenarios* and are listed in Figure 1. Added together, these four drivers represent the functional size of a system which is believed to be a significant predictor of systems engineering effort.

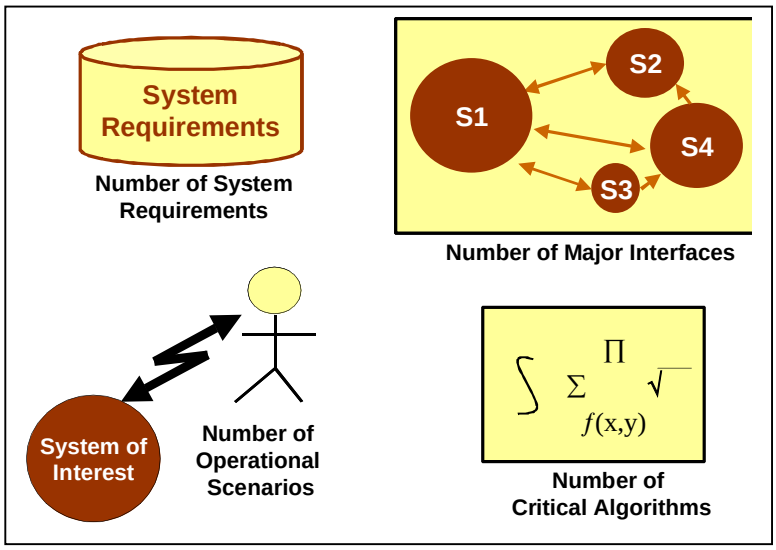

Figure 1. COSYSMO Size Drivers

The scope of this paper is limited to the *# of System Requirements* driver; the most fundamental of the four. A similar process will be used to define and improve the other three drivers. The definition for the requirements driver is shown in Table 1.

Table 1. Definition for the *# of System Requirements* driver

This driver represents the number of requirements for the system-of-interest at a specific level of design. 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 may have effort associated with it such as verification & validation, functional decomposition, functional allocation, etc. System requirements can typically be quantified by counting the number of applicable shalls/wills/shoulds/mays in the system or marketing specification.

Each of the drivers in Figure 1 is adjusted with three factors: volatility, complexity, and reuse. System requirements are frequently volatile and dynamic nature; often known to increase as the project progresses. This phenomenon, known as scope creep, is commonly quantified by expansion and stability patterns [Hammer, et al 1998]. Although new requirements are created, deleted, and modified throughout the life cycle of the project, empirical studies suggest that there tends to be an average number of low level requirements that need to be written in order to satisfy the requirements at the previous i.e. high level. These studies show that the expansion of requirement shows an expected bell curve. Intuitively, it makes sense to implement stable requirements first and hold off on the implementation of the most volatile requirements until late in the development cycle [Firesmith 2004].

The second factor used to adjust the size drivers of COSYSMO model is the complexity level of the requirements. A typical systems project may have hundreds or potentially thousands of requirements which are decomposed further into requirements pertaining to the next subsystem. Naturally, not all requirements have the same level of complexity. Some may be more complex that others based on how well they are specified, how easily they are traceable to their source, and how much they overlap with other requirements. It has been determined that a simple sum of the total number of requirements is not a reliable indicator of functional size. Instead, the sum of the requirements requires a complexity weight to reflect the corresponding complexity of each requirement. Logically, the more complex a requirement the greater the weight that is assigned to it and vice versa.

Reuse is the third important factor used to adjust the number of requirements. As reuse facilitates the usage of certain components in the system it tends to bring down the efforts involved in the system development. The sum of requirements is adjusted downwards when there are a significant number of reused requirements. This is meant to capture an organization's familiarity with the development, management, and testing of requirements.

The definition and three adjustment factors alone do not capture all the impact introduced by requirements. Additional work is involved in decomposing requirements so that they may be counted at the appropriate system-of-interest. These rules help clarify the definition and adjustment factors while providing consistent interpretations of the size drivers for use in cost estimation.

Phase Two. The second, more recent, phase involves the development of a framework that can aide in improving the reliability of the size driver interpretation across multiple stakeholders. For example, one such issue faced with regards to sizing requirements is to decide what level is "too high", "too low", and more importantly "just right." The issue of requirements granularity is very important to the accuracy of the COSYSMO estimate. Software cost models have rules on how to count lines of code based on the software language being used. Unfortunately, no such rules exist in systems engineering for counting requirements. This creates an opportunity to define such rules with the goal of improving them as feedback is received from practitioners and researchers. Since *# of System Requirements* is one of the four size drivers in the model, an initial set of rules has been developed to help systems engineers produce consistent requirements counts that can be used in COSYSMO.

The challenge with requirements is that they can be specified by either the customer or the contractor. In addition, these organizations often specify system requirements at different levels of decomposition and with different levels of sophistication. Customers may provide high level requirements in the form of system capabilities, objectives, or measures of effectiveness; these are translated into requirements by the contractor and decomposed into different levels depending on the role of the system integrator. The prime contractor could decompose the initial set of requirements and expand them to subcontractors below it as illustrated in Figure 2.

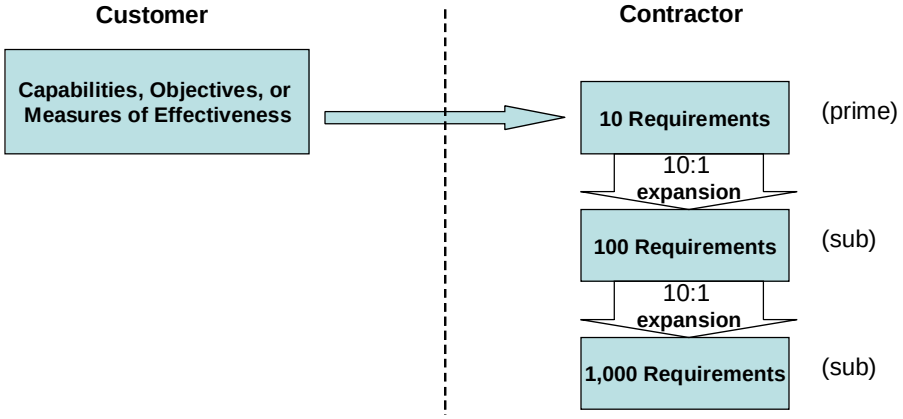

Figure 2. Example of Requirements Translation from Customer to Contractor

For purposes of this example, the expansion ratio from one level of requirement decomposition to the other is assumed to be 10:1. Different systems will exhibit different levels of requirements decomposition depending on the application domain, customer's ability to write good system requirements, and the functional size of the system. The requirements flow framework in Figure 2 provides a starting point for the development of rules to count requirements. These rules were designed to increase the reliability of requirements counting by different organizations on different systems regardless of their application domain. The five rules are as follows:

1. Determine the system of interest. For an airplane, the system of interest may be avionics subsystem or the entire airplane depending on the perspective of the organization interested in estimating systems engineering. This key decision needs to be made early on to determine the scope of the COSYSMO estimate and identify the requirements that are applicable for the chosen system.

2. Decompose system objectives, capabilities, or measures of effectiveness into requirements that can be verified or designed. The decomposition of requirements may be performed by the customer or the contractor. The level of decomposition of interest for COSYSMO is the level in which the system will be designed to; which is equivalent to the "A Spec". This is often also the level at which the system will be tested or verified. Several levels of decomposition may take place before the requirements are at the right level of design for use in COSYSMO depending on the level in which they are initially available.

3. Provide a graphical or narrative representation of the system of interest and how it relates to the rest of the system. This step focuses on the hierarchical relationship between the system elements. This information can help describe the size of the system and its levels of design. It serves as a sanity check for the previous two steps.

4. Count the number of requirements in the system or marketing specification for the level of design in which systems engineering is taking place in the desired system of interest. The focus of the counted requirements needs to be for systems engineering. Lower level requirements may not be applicable if they have no effect on systems engineering.

OR

Count the number of requirements in the verification test matrix for the desired system of interest. As an alternative, requirements may be counted from the Requirements Verification Trace Matrix (RVTM) that is used for testing system requirements. The same rules apply as before: all counted requirements must be at the same design or bid level and lower level requirements must be disregarded if they do not influence systems engineering effort. The RVTM has traditionally been used in systems engineering as an indicator for the number of systems engineering requirements and is widely used by the stakeholders of COSYSMO.

5. Determine the volatility, complexity, and reuse of requirements. Once the quantity of requirements has been determined, the three adjustment factors must be applied. Currently three complexity factors have been determined: easy, nominal, and difficult. These weights for these factors were determined using expert opinion through the use of a Delphi survey [Valerdi, et al 2003].

These five steps lead users down a consistent path of similar logic when determining the number of system requirements for the purposes of estimating systems engineering effort in COSYSMO. It has been found that the level of decomposition described in step #2 may be the most volatile step as indicated by the data collected thus far. To alleviate this, an example of requirements decomposition for a large system is provided.

Requirements Counting Example: FAA's En Route Automation Modernization

The Federal Aviation Administration (FAA) has undergone a large acquisition effort for a new En Route Automation Modernization (ERAM) system that will coordinate the in flight resources in the U.S. [FAA 2005]. As the customer, the FAA provided a specification with two types of requirements: core and extensible [FAA 2003]. The core requirements are the minimum functional capabilities and performance required for acceptable operational suitability and effectiveness. The extensible requirements are the goals in functional capabilities and performance that the FAA desires to achieve in the future.

A core requirement in this system specification can be found at the fourth level down in the document hierarchy. This requirement can be tracked by the word "shall" but cannot be counted in COSYSMO since there is still one more level of requirements decomposition in existence. The fourth level of decomposition would be considered too high in the system hierarchy to use for counting requirements. Therefore, the requirements count should be done at the sixth level of the document hierarchy, or X.X.X.X-Y.Y-Z.

For example, the ERAM System Specification[7] uses the following hierarchical categorization which complies with IEEE and NASA standards [Wilson, et al 1997]:

3	Requirements
3.2	Functional Capabilities
3.2.1	En Route – General
3.2.1.0-16	The system **shall** check input messages to ensure message completeness and coherency
3.2.1.0-16.0-1	The system **shall** provide format, logic, and validity checks for incoming data.

As the customer, the FAA decomposed their requirements one level. It is at this level that they can communicate their needs for the ERAM system and it is left up to the developer who is awarded the contract to implement the requirement. As an exercise, the lowest level requirement that was provided by the FAA has been decomposed into the following three additional requirements:

3.2.1.0-16.0-1 The system **shall** provide format, logic, and validity checks for message type 1.
3.2.1.0-16.0-2 The system **shall** provide format, logic, and validity checks for message type 2.
3.2.1.0-16.0-3 The system **shall** provide format, logic, and validity checks for message type 3.

[7] The ERAM requirements have been sanitized due to the proprietary nature of the system

The need to decompose a single FAA requirement into three requirements is an example of a typical elaboration exercise performed by contractors. In this example, three different types of messages were assumed to exist. Each message has unique characteristics that will require it to be verified and validated independently. For COSYSMO purposes it is assumed that the contractor will have to implement three requirements. It is up to the systems engineer to determine the volatility, complexity, and reuse associated with the requirements.

This example provides a uniform approach for counting the *# of System Requirements* size driver. Since COSYSMO is being developed from the contractor point of view, the main focus is on determining the best level at which to count *performed* systems engineering requirements rather than *managed*. The FAA example is oversimplified as it contains a somewhat sophisticated specification from the customer. Often times this initial specification is not available and the authorship of the requirements is left to the developer or prime contractor. The important lesson here is that even when a customer provides detailed requirements the contractor needs to further decompose them to determine the actual systems engineering effort associated with the development, implementation, testing, and maintenance of the system. The next section provides a framework that is helpful for describing the best level to count requirements.

Cockburn's Use Case Hierarchy

The field of software engineering provides a rich environment that has influenced many systems engineering principles. In his latest book, *Writing Effective Use Cases*, well known author and researcher Alistair Cockburn describes a framework for capturing software use cases. A use case is "a collection of possible sequences of interactions between the system and its users, relating to a particular goal. The collection of use cases should define all system behavior relevant to the actors to assure them that their goals will be carried out properly. Any system behavior that is irrelevant to the actors should not be included in the use cases." [Cockburn 2001]

While the focus of the book is on software use cases, the framework is applicable to systems engineering as well. Cockburn describes a *sea level* metaphor for named goal levels of use cases. Like in use cases, the driving force behind systems engineering involves meeting user goals. According to Cockburn, there are three named goal levels involved with the requirement use cases: summary goals, user goals, and subfunctions. The basic idea behind the *sea level* metaphor is that goals exist above and below *sea level* each serving a specific function. User goals, however, are the most important since they often determine the success of a system. The existence of the system is justified by user goals and its support for the various primary actors. Moreover, these goals are the basis for prioritization, delivery, team division, estimation and development. In the *sea level* metaphor, *sea level* corresponds to the user goals. A cloud or a kite indicate a higher level where a fish or a clam indicate lower levels as shown in Figure 3.

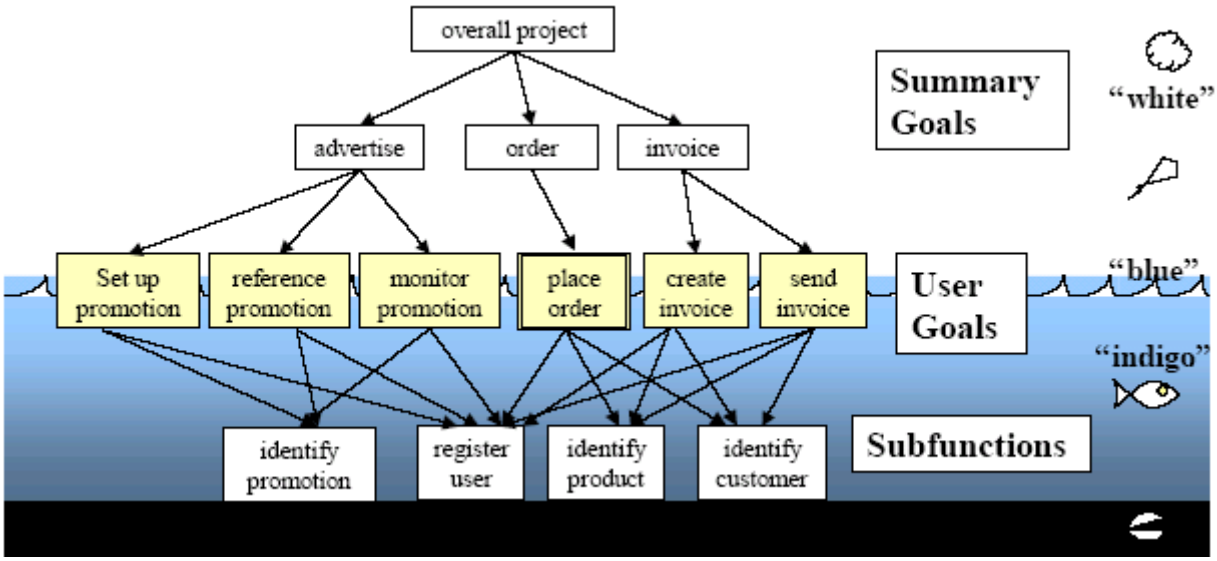

Figure 3. Three named goal levels framework for Use Case
(Used with permission. Copyright 2001 Alistair Cockburn)

A closer look at the categorization of these use case levels brings out their relevance to the five step rule framework described to count the *# of requirements*. At step 1 the system of interest is determined. It involves outlining the overall project and its summary goals. The system of interest directly corresponds to the summary goals of the project. Step 2 is where the breakdown of the high level requirements into the actual user goals takes place. The decomposition occurs at multiple levels with the objective of describing high level requirements in terms of actual user goals by customer or the contractor. Initial decomposition is more likely to be performed by the customer and is likely to provide direction to the contractors to further decompose the overall goals to specific sections of user goals. At step 3, the hierarchical relationship between the different levels of the decomposition is targeted at defining the relationship between the system elements at the design level. The level against which system testing and acceptance takes place is the level of user goals i.e. sea level. At step 4, the number of requirements in the system specification corresponds to the summary goal level. As the decomposition of these requirements arrives at *sea level*, the contractor or subcontractor can aggregate the number of requirements at the same level of design. In doing so, lower level requirements should be ignored since they have no influence on systems engineering effort. At step 5, adjustment of the requirements size according to their volatility, complexity, and reuse takes place.

The development of COSYSMO can be used to further illustrate the *sea level* metaphor. The summary level, or *sky level*, use case is written for either a strategic or system scope. For example, a *sky level* goal for COSYSMO is to "build a systems engineering cost model." The stakeholders of the model stated this as their basic need that in turn drives a collection of user level goals. A *kite level* goal provides more detailed information as to "how" the *sky level* goal will be satisfied. In the case of COSYSMO, it includes the standards that will drive the definition of systems engineering and system life cycle phases. The *sea level* goals represent a user level task that is the target level for counting requirements in COSYSMO. Continuing the example, it involves "utilizing size and cost drivers, definitions, and counting rules" that will

enable the accurate estimation of systems engineering effort, also providing more information on "how" the higher goals at the *kite level* will be satisfied. The *sea level* is also important because it describes the environment in which the model developers interact with the users and stakeholders. A step below is the *underwater level* which is of more concern to the developer. In this example it involves the selection of implementation and analysis tools required to meet the user goals. The examples are mapped to Cockburn's hierarchy in Figure 4.

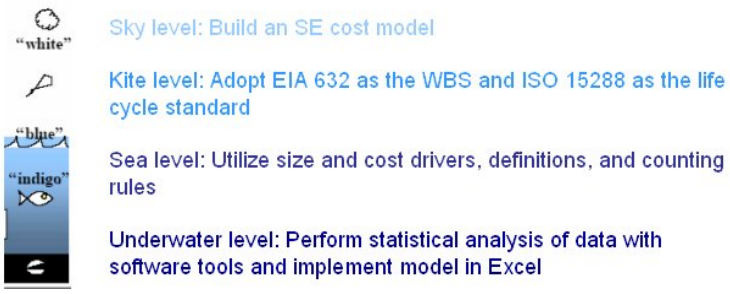

Figure 4. Cockubrn's Hierarchy as Related to COSYSMO Use Case Levels (adapted from Cockburn)

Going down the hierarchy from *sky* to *underwater* provides information on "how" a particular requirement will be satisfied by the system while going up the hierarchy provides information on "why" a particular requirement exists.

Comparison of Two Experiments

In order to quantitatively illustrate the benefits of using counting rules for the *# of System Requirements* driver, two experiments that used this metric are compared. The first experiment is an empirical study performed on fourteen student eService projects over a three year period at USC [Chen, et al 2004]. These projects were part of a two-semester software engineering course which followed the MBASE[8] development framework [Boehm, et al 2000]. The results are organized in the CeBASE[9] archives; an NSF-sponsored center. Among other things, the results of the experiment show that the software size in terms of SLOC is moderately well correlated with the both the number of external use cases in use case diagrams and the number of classes in class diagrams. The experiment also demonstrates that the number of sequence diagram steps per external use case is a possible complexity indicator of software size.

However, for this 14-project eServices sample, the UML-based requirements metrics were insufficiently defined to serve as precise sizing metrics. The teams that participated in this experiment did not operate under any specific set of instructions on how to count requirements. As a result, each team adopted the best method that suited their needs. This inconsistency in requirements decomposition, while not deliberate, allowed for different implementation philosophies. It was no surprise that the teams described their software systems using requirements at different levels of design. Intuitively there is an expectation that the more software requirements associated with a system the more software effort will be involved to meet those requirements. To no surprise the requirements associated with the eServices projects did

[8] Mode-Based System Architecting and Software Engineering
[9] Center for Empirically Based Software Engineering

not correlate well with software development effort[10]. The relationship, yielding an R-squared[11] of 0.02, is shown in Figure 5.

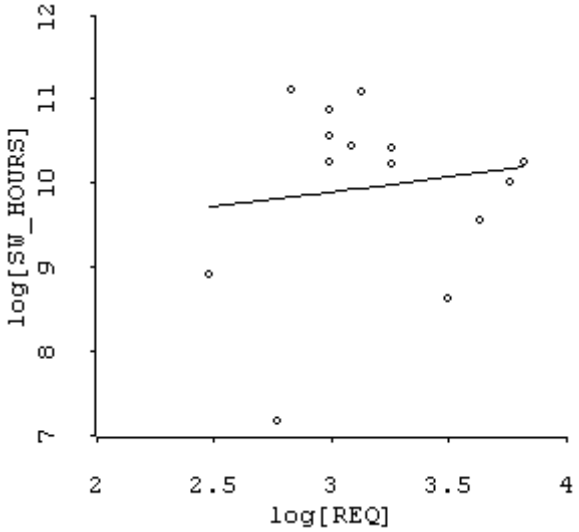

Figure 5. Requirements vs. SW Hours (adapted from Chen, et al 2004)

Numerous factors could have contributed to this poor relationship including the difference in the complexity of the requirements. More importantly the absence of uniform counting rules in the experiment could have negatively influenced the results. Since the study was performed in retrospect there were no opportunities to control for these inconsistencies in the experiment. Additional threats to validity exist that could have contaminated this longitudinal experiment including the fact that students were not being graded on the quality of the requirements they produced; contributing even more to the variability in which requirements were reported.

The second experiment performed was under a significantly different environment. The aforementioned 5-step process for counting the *# of System Requirements* was used on a set of thirty-five systems engineering projects. Rather than using students, this experiment involved six aerospace companies[12] representing a significant portion of the US military's portfolio. There was an anticipated positive relationship between the number of system requirements, as defined earlier in this paper, and the amount of system engineering hours involved with the development, implementation, testing, and maintenance of those requirements. Results confirm that the relationship between requirements and systems engineering effort are indeed positively correlated. Figure 6 shows a relationship with an R-squared of 0.55.

[10] Chen's study provided requirements vs. software lines of code (SLOC). The SLOC axis was normalized to effort in hours using the Constructive Cost Model (COCOMO II)

[11] R-squared is also known as the coefficient of determination. Values range from 0 to 1, where 0 indicates no correlation and 1 indicates perfect correlation

[12] BAE Systems, General Dynamics, Lockheed Martin, Northrop Grumman, Raytheon, and SAIC

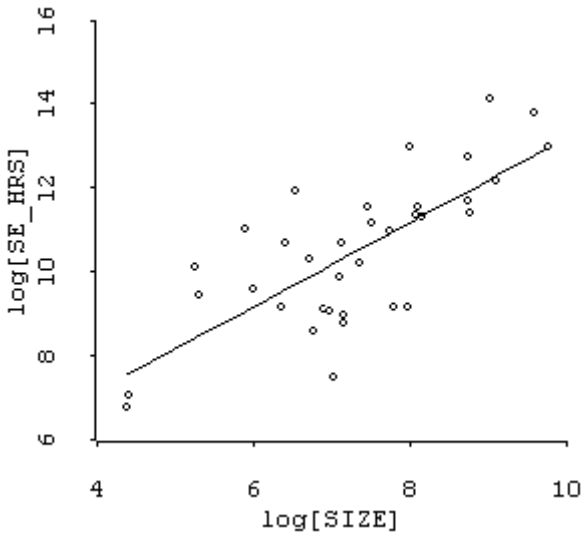

Figure 6. Requirements vs. SE Hours

While this experiment shows a much stronger relationship between requirements and effort there is still plenty of room for improvement. As discussed at the beginning of this paper the number of requirements is not a sufficient predictor of systems engineering effort. It is evident that other metrics are needed to arrive at a more reliable prediction of systems engineering size and, consequently, systems engineering effort. The refinement of these metrics and their counting rules will involve our near-term goals as part of the development of COSYSMO.

Conclusion

It has been shown that the *sea level* analogy developed for use cases provides a useful framework for the decomposition of system requirements from the customer to the contractor's system-of-interest. Leveraging off this framework, summary level requirements can be broken down into user level requirements. These user level requirements are then used to estimate the amount of systems engineering hours associated with the requirements. The guidance provided can help organizations uniformly count the *# of system requirements* for accurate use in COSYSMO. With this framework clearly delineated, it can lead to extensions for the three other size drivers in the model. Finally, counting rules were shown to significantly improve the predictive abilities of requirements in different experiments. While more work needs to be done to refine the counting rules for COSYSMO size drivers, significant progress has been made thanks to the support of INCOSE member companies and USC CSE[13] Affiliates.

[13] University of Southern California Center for Software Engineering

Acknowledgements

The authors appreciate the input provided by Jo Ann Lane, Barry Boehm, and Yue Chen. Comments from anonymous INCOSE reviewers were also very beneficial to this manuscript. Permission from Alistair Cockburn to incorporate the use case analogy is also appreciated.

References

Boehm, B. W., *Software Engineering Economics*, Prentice-Hall, 1981.

Boehm, B. W., "Integrating Software Engineering and Systems Engineering," *The Journal of NCOSE*, Volume I, No. 1, July-September 1994, pp. 147-151.

Boehm, B., Port, D., Al-Said, M., "Avoiding the Software Model-Clash Spiderweb," IEEE Computer, Nov. 2000.

Boehm, B. W., Abts, C., Brown, A. W., Chulani, S., Clark, B., Horowitz, E., Madachy, R., Reifer, D. J. and Steece, B., *Software Cost Estimation With COCOMO II*, Prentice Hall, 2001.

Chen, Y., Boehm, B. W., Madachy, R., Valerdi, R., "An Empirical Study of eServices Product Sizing Metrics," *3rd ACM-IEEE International Symposium on Empirical Software Engineering*, August 2004, Redondo Beach, CA.

Cockburn, A., *Writing Effective Use Cases*, Addison-Wesley, 2001.

EIA/ANSI 632 *Processes for Engineering a System*, January 7, 1999.

Federal Aviation Administration, En Route Automation Modernization (ERAM) website: http://www.faa.gov/asd/ia-or/eram.htm, 2005.

Federal Aviation Administration, En Route Automation Modernization (ERAM) Product Specification, 2003.

Firesmith, D., "Prioritizing Requirements", *Journal of Object Technology*, Vol. 3, No. 8, September-October 2004, pp. 35-47.

Hammer, T., Huffman, L., Rosenberg, L. H., Wilson, W., Hyatt, L. E., "Doing Requirements Right the First Time", *Software Technology Conference*, Salt Lake City, UT, April 1998.

Honour, E., "Toward an Understanding of the Values of Systems Engineering," *2nd Conference on Systems Engineering Research*, Los Angeles, CA, March 2004.

ISO/IEC 15288:2002(E), *Systems Engineering: System Life Cycle Processes*, First Edition, 2002.

Schweiter, G. A. and Stromquist, W. R., "The Effect of Sensor Quality on Tracker/Correlator Performance." *Technical Proceedings of the 1990 Joint Service Data Fusion Symposium,* Laurel, MD, May 15-18, 1990.

Valerdi, R., Boehm, B. W., Reifer, D., "COSYSMO: A Constructive Systems Engineering Cost Model Coming Age," *13th INCOSE Symposium*, July 2003, Crystal City, VA.

Valerdi, R., Wheaton, M., "EIA/ANSI 632 as a Standardized WBS for COSYSMO," *NASA Cost Analysis Symposium*, April 2005.

Wilson, W. M., Rosenberg, L. H., Hyatt, L. E., "Automated Analysis of Requirement Specifications", *International Conference on Software Engineering (IASTED),* Boston, MA - May 1997.

Biographies

Ricardo Valerdi is a PhD candidate at the University of Southern California in the Systems Architecture program. He is a Research Assistant at the Center for Software Engineering and a Member of the Technical Staff at the Aerospace Corporation. He obtained his BS in Electrical Engineering from the University of San Diego and his MS in Systems Architecting from USC. Previously, he worked as a systems engineer at Motorola and General Instrument.

Jatin Raj is a Masters' candidate at San Diego State University in the Computer Science department. He obtained his MBA from San Diego State University and worked as a as Business System Analyst at Unilever, India. His research interests include requirement elicitation, use case development, and systems analysis and design.