28th annual INCOSE international symposium
Washington, DC, USA
July 7 - 12, 2018

INCOSE Working Group Addresses System and Software Interfaces

Sarah Sheard, Ph.D.
CMU Software Engineering Institute
(412) 268-7612
sheard@sei.cmu.edu

Rita Creel
CMU Software Engineering Institute
(703) 247-1378
rc@cert.org

John Cadigan
Prime Solutions Group, Inc.
(623) 853-0829
johncadigan@psg-inc.net

Joseph Marvin
Prime Solutions Group, Inc.
(623) 853-0829
joemarvin@psg-inc.net

Leung Chim
Defence Science & Technology Group
+61 (0) 8 7389 7908
Leung.chim@dst.defence.gov.au

Michael E. Pafford
Johns Hopkins University
(301) 935-5280
mepafford@verizon.net

Abstract. In the 21st century, when any sophisticated system has significant software content, it is increasingly critical to articulate and improve the interface between systems engineering and software engineering, i.e., the relationships between systems and software engineering technical and management processes, products, tools, and outcomes. Although systems engineers and software engineers perform similar activities and use similar processes, their primary responsibilities and concerns differ. Systems engineers focus on the global aspects of a system. Their responsibilities span the lifecycle and involve ensuring the various elements of a system—e.g., hardware, software, firmware, engineering environments, and operational environments—work together to deliver capability. Software engineers also have responsibilities that span the lifecycle, but their focus is on activities to ensure the software satisfies software-relevant system requirements and constraints. Software engineers must maintain sufficient knowledge of the non-software elements of the systems that will execute their software, as well as the systems their software must interface with. Similarly, systems engineers must maintain sufficient awareness of the software to enable early identification and resolution of software risks and issues driven by other system elements. Thus, to enable continued progress in creating and sustaining capability in complex, interconnected systems, systems and software engineers must commit to improving the interfaces between their disciplines, to aligning and integrating their terminology, processes, methods, and tools.

Recognizing the need to improve the system engineering-software engineering interface, INCOSE approved the charter of the System and Software Interface Working Group (SaSIWG) in 2017. At its initial meeting at the INCOSE International Symposium 2017 (IS 2017) in Adelaide, Australia, the SaSIWG derived working group objectives from lists of brainstormed systems and software issues. This paper documents the interface issues elicited, grouped into seven categories, along with system-software interface use cases identified by SaSIWG members. The interface issues and use cases expose questions for the SaSIWG to prioritize and respond to. The paper concludes with a summary of the SaSIWG's plan to respond to these questions and strengthen the interface between the systems engineering and software engineering disciplines.

Introduction

> *Software is fundamental to the performance, features, and value of most modern engineering systems. It is not merely part of the system, but often shapes the system architecture; drives much of its complexity and emergent behavior; strains its verification; and drives much of the cost and schedule of its development. Given how significant an impact software has on system development and given how complex modern systems are, one would expect the relationship between the disciplines of systems engineering (SE) and software engineering (SWE) to be well defined. However, the relationship is, in fact, not well understood or articulated (Pyster et al. 2015, p. 708).*

Not long ago, the critical capabilities we rely on—for example, power production and delivery, transportation, defense, and medical procedures—were performed by mechanically controlled systems or by humans. Increasingly, these capabilities are implemented in and controlled by software. Further, with the emergence of the digital thread, the role of software is expanding in the design and manufacture of physical system components and execution of systems engineering analyses (Hedberg et al. 2016; Kraft 2013). Advances in computing and communication technology have enabled the rapid growth of software-driven capabilities that continue to change the way we live and work in countless ways. Our ability to understand the software that drives so many systems and processes, and to effectively manage its development and evolution, has not kept pace with software's expanding role. Late, over-budget delivery of systems and software that fall short in performance is commonplace.

> *The software did exactly what it was told to do. The reason it failed is that it was told to do the wrong thing (Somers 2017).*

Telling the software to do the *right* thing throughout the system and software lifecycle is what the system-software interface is all about, and systems engineers have a significant role to play. This role, informed by a global perspective that includes stakeholder coordination and cross-domain engineering, is essential to the successful design, implementation and sustainment of complex software that may span system elements and drive communication with external systems (Sweeny, Hamman, & Biemer, 2011). Systems engineering activities, combined with methods of Agile development that leverage software's flexibility, provide a solid foundation that enables rapid deployment of new capabilities and features (Boehm & Turner, 2003). But success depends on the ability to effectively integrate the software and systems domains. Software engineering must participate in systems engineering activities that generate the information needed to *tell the software to do the right thing;* systems engineers and software engineers must maintain a common picture of this *right thing* as requirements and technology evolve over the lifecycle.

What is systems engineering? What is software engineering? And what, exactly, is their interface? Systems engineering is an "interdisciplinary approach governing the total technical and managerial effort required to transform a set of stakeholder needs, expectations, and constraints into a solution and to support that solution throughout its life" (ISO/IEC/IEEE 2015, p. 10; ISO/IEC/IEEE 2017, p. 10). Software engineering is "application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software" (ISO/IEC/IEEE 2017, p.8). Although software engineering spans the lifecycle just like systems engineering, the perception lingers that software engineering is equivalent to coding or programming. The interface between the two disciplines has many facets, from generalities to details, and in data, processes, tools, standards, even professional organizations.

In Figure 1, below, systems and software engineering are shown as separate but interacting disciplines. They perform many of the same activities—requirements analysis, cost and schedule estimation, architecture development, planning, design, implementation, test, integration, verification and validation—but with different areas of focus. As shown in the figure, software construction is the formal term for software engineering's design, code/assembly (assembly from COTS software or existing code bases), and unit test activity (ISO/IEC/IEEE 2017; Bourque & Fairley, 2014). We use the term *software systems engineering* to refer to the intersection of the disciplines, the activities, products, and environments that comprise the system engineering-software engineering interface.

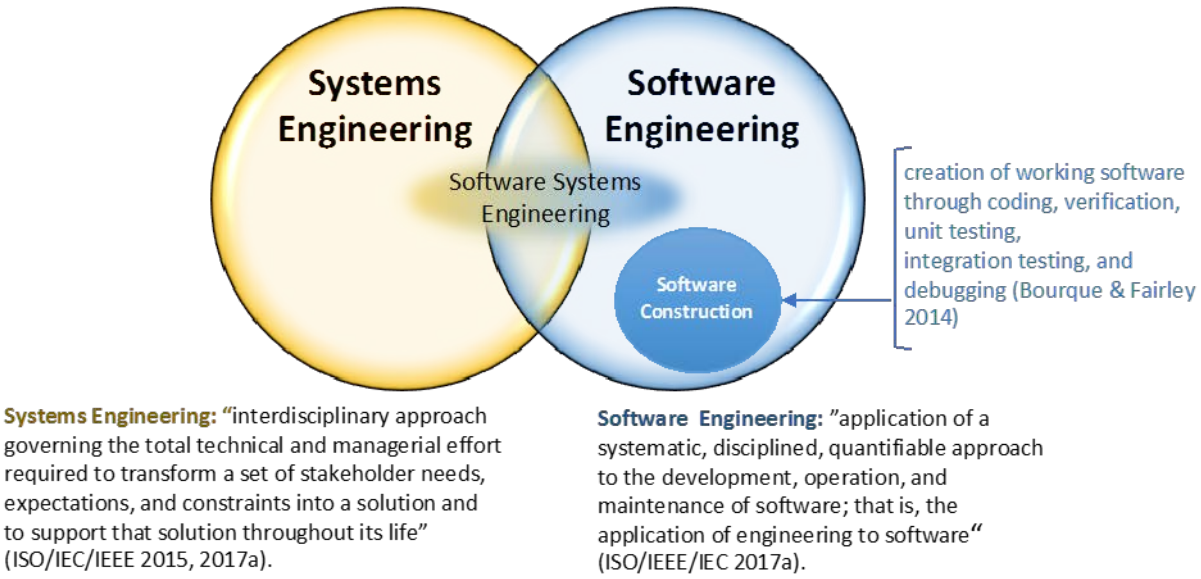

Figure 1. Systems Engineering and Software Engineering

At the INCOSE International Symposium 2017 (IS 2017), INCOSE Past President Heinz Stoewer warned that if INCOSE "doesn't address the digital thread" (Kraft 2013) quickly to understand software and start leading in software-intensive systems, INCOSE is at risk of becoming irrelevant. As a result of the INCOSE Corporate Advisory Board voting "software" as one of its top concerns in 2016, the INCOSE Systems and Software Interface Working Group (SaSIWG) was formally established in 2017. The SaSIWG recognizes the challenges facing both systems and software disciplines, and is dedicated to exploring, describing, and improving tools and techniques needed for these domains to achieve better results collectively.

The information presented in this paper is the result of initial SaSIWG collaborative, structured brainstorming activities. The paper organizes and provides this information as a foundation for further work, rather than a report of exhaustive research, conclusions, or recommendations.

The remainder of this paper is organized into five sections. The first section reviews the literature about the interaction between systems engineering and software engineering. The second section discusses systems and software interface issues identified by the SaSIWG during its initial meeting at IS 2017 in Adelaide, Australia. The meeting participants were INCOSE members from career backgrounds in software and systems engineering. The third section contains system and software engineering use cases provided by SaSIWG members as examples. The interface issues and use cases described in the first two sections are meant to provide foundational information to inform future SaSIWG work and products. The fourth section, Potential Interfaces: INCOSE and External Working Groups, identifies other groups that are identifying and resolving systems-software interface issues in the course of their work. The fifth section briefly reviews the working group's activities to date

and identifies next steps. Future activities will include prioritizing and analyzing issues and articulating recommendations that will feed directly back into the SaSIWG efforts and execution plans for development of technical products.

Literature Review

Today's systems build on yesterday's systems, connecting them together to provide more capability. Tomorrow's systems will continue the trend, creating ultra-large-scale systems. (Northrop, 2006 and Sillitto, 2010). In most of these systems, what binds the systems together, and causes the emergence of the desired capability, is software (Fairley, 2011b). Software allows more functions to be performed than hardware and those tend to be much more complex (Sheard, 2004). Because of the resulting complexity, "our large software systems can no longer be constructed as monoliths…and tested within known performance limits," (Brownsword, 2006) so that a field of systems of systems engineering has arisen (Jamshidi, 2017).

Affect each other. Both systems engineering and software engineering have been blamed for problems in the other discipline.

> **Systems engineering causes software problems**. The US Air Force's Weapon Systems Software Management Guidebook (USAF, 2008) describes software problems arising in part because of "ineffective systems engineering interface to the software development process," and Dr. Barry Boehm concurs: "…the root causes of failed software projects…tend to be failures in doing the systems engineering" (Lane, 2009). As early as 1991, Unisys Defense Systems noted that good systems engineering was important to prevent software problems (Snyder, 1991).
>
> **Software causes system problems**. Kasser and Shoshany (2000) blame "massive failures" in complex projects on software engineering. As early as 1986, Dr. Nancy Leveson identified system failures caused by unsafe software, even then resident in more than 80% of weapons systems. (Leveson, 1986). Similarly, Knight (2002) sees software as a cause of safety and security failures.

Must evolve. Over time, software engineering practices changed a great deal—they did in the last century and they will need to change more in the next. This is necessary to cope with evolving needs and increasing complexity (Boehm, 2006b). Vierhauser et al. (2014) contend that both the traditional systems engineering approach and software engineering approaches need to evolve to address these new systems. Of course, how to evolve is a question that has gotten much attention.

Early comparisons. Before 2000, many papers compared and contrasted systems engineering and software development, either the disciplines or the practitioners. These include Andriole in the U.K. (1993; in a software journal), Wray (1993), Armstrong and Pyster (1997, asking who should lead?), Sheard (1998, on the issues and gaps between the two fields) and Kasser and Shoshany (2000). Boehm (2000) urged unification through the use of the Capability Maturity Model, Integrated® (2000) to stimulate the needed culture change.

Later comparisons, in many cases, began to suggest how the two disciplines could reconcile. Maier's seminal paper on reconciling systems and software architecture (2006) is still cited frequently. Fairley and Willshire (2011a and 2011b) wrote how to educate each discipline in the other's knowledgebase. Sheard (2014), mapping the two disciplines in a Venn diagram, called for increased collaboration. Wang (2009) compared cost models for systems engineering and software engineering. Giese (2005) and Sheard (2004) looked at how software engineering practices and systems engineering practices, respectively, would need to change for more software-intensive systems in the future. Interestingly, the systems engineering body of knowledge (or SEBOK) (2017), which is a wiki kept current by INCOSE, has a fairly long section on software, including software engineering in the SysE lifecycle,

the nature of software, an overview of the SWEBOK, key points a systems engineer needs to know about software, and software engineering features. In contrast, the software engineering body of knowledge (SWEBOK, 2014), a book most recently published in 2014, has only two paragraphs about systems engineering, a definition that is tied to nowhere else in the book.

Learning the other discipline. To address the problem between systems and software, the call has come both to apply system concepts better to software (Boehm 1991, Oliver 1995, Boehm 2008) and to apply software concepts to systems (Steiner 2004, Oliver 1995). In 1997, Rose modeled the interactions between systems engineering and software engineering, as well as with project management, using the IDEF0 formalism. Several groups developed courses to teach software concepts to systems engineers (Fairley 2011b, Pafford 2017, Sheard 2017) or to teach systems engineering to software engineers (Harbaugh 1993, Giese2005, Lane 2009, Fairley and Willshire 2011a).

"Software systems engineering." This term appeared apparently independently in several different places: Sage and Palmer (1990) published a book and Andriole (1993) a paper suggesting a new discipline of that name; 20 years later Kossiakoff et al. (2011) published a book and Pyster et al. (2015) a paper also using the term. In between, Nelson (2007) used the term "system software engineering" for a similar concept.

Interfaces. The only known previous attempt to define the interfaces between systems and software in a broad manner is from an effort in 1996 in which one of this paper's authors participated. (Frerking 1996). The report from that workshop was never made public and was not used to advance the field. This is a basis on which the INCOSE SaSIWG intends to build, 20 years later.

How to work together, or achieving integrated systems and software engineering. The evolving state of the art in integrating systems engineering is shown by a partially chronological view. Snyder (1991) suggested how to improve the handoff of specifications from systems engineering to software. Alford (1992) proposed to strengthen the systems-software engineering interface by specifying how requirements for software end up being allocated to processes of the computer hardware architecture. Oliver (1995) proposed that to improve the success rate of system acquisition, "the development of rigorous specifications that match user needs is critical." While all three of these ideas seem dated in today's Agile world, it should be noted that Oliver proposed an "object meta-model for a system hierarchy" and two "process meta-models" in this same paper. Subsequently, Rose (1997) modeled the interactions between systems engineering, software engineering, and project management using IDEF0 formalism. This was done on a general basis, not on an interface-by-interface basis.

White (2005) hoped to provide to the Engineering of Computer Based Systems technical committee of IEEE a beginning of a guideline on Integrated Systems and Software Engineering, to be developed. In 2006, Barry Boehm addressed trends in the interaction of systems and software engineering (2006a), all of which seem true, even obvious today, and the difficulty of staying current with a number of software trends and anti-trends over the decades (2006b). Also in 2006, Maier published a paper still being cited today that suggests ways to reconcile systems and software architectures.

The following year, Boehm and Lane (2007) published the Incremental Commitment model to integrate systems and software engineering and system acquisition. Turner et al. (2009) proposed a "touchpoint" framework for integrating systems and software engineering, which notes process faults such as gaps, clashes, and waste, particularly for "*interdependent* systems" where hardware and software cannot be separated and must be designed in an integrated manner. Boehm et al. (2009) took this further with "Architected agile solutions for software-reliant systems," which sounds very timely even today. Rosser et al. (2014) further described how to do systems engineering using Agile methods in cross-functional teams. Wrubel et al. (2014) described the engagement of Agile software teams with systems engineering on DoD programs.

In 2011 Boehm spoke of “future software engineering opportunities and challenges” that require significant changes in, and integration of, both software and systems engineering processes, with a focus on generating value and a dynamic balancing of agility, discipline, and scalability. This same year, Fairley and Willshire (2011b) described education in software engineering that systems engineers need, stating, “Smooth integration of the development processes used in systems engineering and software engineering is a continuing and ongoing challenge.”

Systems and Software Interface Issues and Gaps

SaSIWG members have a variety of perspectives, based on their professional and educational backgrounds and the experiences they have had on projects at the intersection of systems and software engineering. The issues identified at IS 2017 and IW 2017 reflect this variety and illustrate the many challenges that both types of engineers must meet as we work to integrate the knowledge and practice of our respective disciplines.

In developing, operating, and maintaining today’s complex, software-intensive systems, a variety of interfaces between the systems and software domains become important. In addition to technical interfaces among hardware and software technical products, engineers need to be concerned with interfaces between system and software processes, requirements, and designs. They need to be concerned with the terms they use to communicate across their domains. Frequently, when defining detailed technical interfaces between systems and software components and subsystems is the first time a program becomes aware of issues and gaps between systems and software engineering. By then, it may be too late to mitigate the issues without increasing cost or time to delivery, or reducing performance expectations. More robust system-software communication can speed detection and evolution of such issues.

In brainstorming sessions, members of the SaSIWG discussed interactions between systems and software engineering practices and identified several issues (overlaps or conflicts) and gaps. To help the working group address these effectively, the groups steps include gathering and organizing questions surrounding the issues and gaps, defining criteria for prioritizing them according to needs, prioritizing them, and creating roadmaps for additional analysis and identification of proposed solutions. This paper reports on the initial gathering step. The remaining steps are part of the plan forward. We expect to use lessons learned from a variety of sources, including INCOSE, to develop the solutions.

Moving forward, the working group will identify additional software engineers to supplement the group’s existing software engineering expertise. Currently, the group has a preponderance of systems engineers and only a few participants with direct, practical software engineering experience.

Figure 2 is a collage of example outputs from the SaSIWG affinity mapping and structured brainstorming sessions.

Through the use of collaborative structured brainstorming and affinity mapping exercises, the SaSIWG at IS17 informally identified categories of systems and software engineering issues and gaps. The identified issues mostly fit into the following categories:

- disciplines
- processes
- standards
- organization
- component specifications
- management
- data
- tools

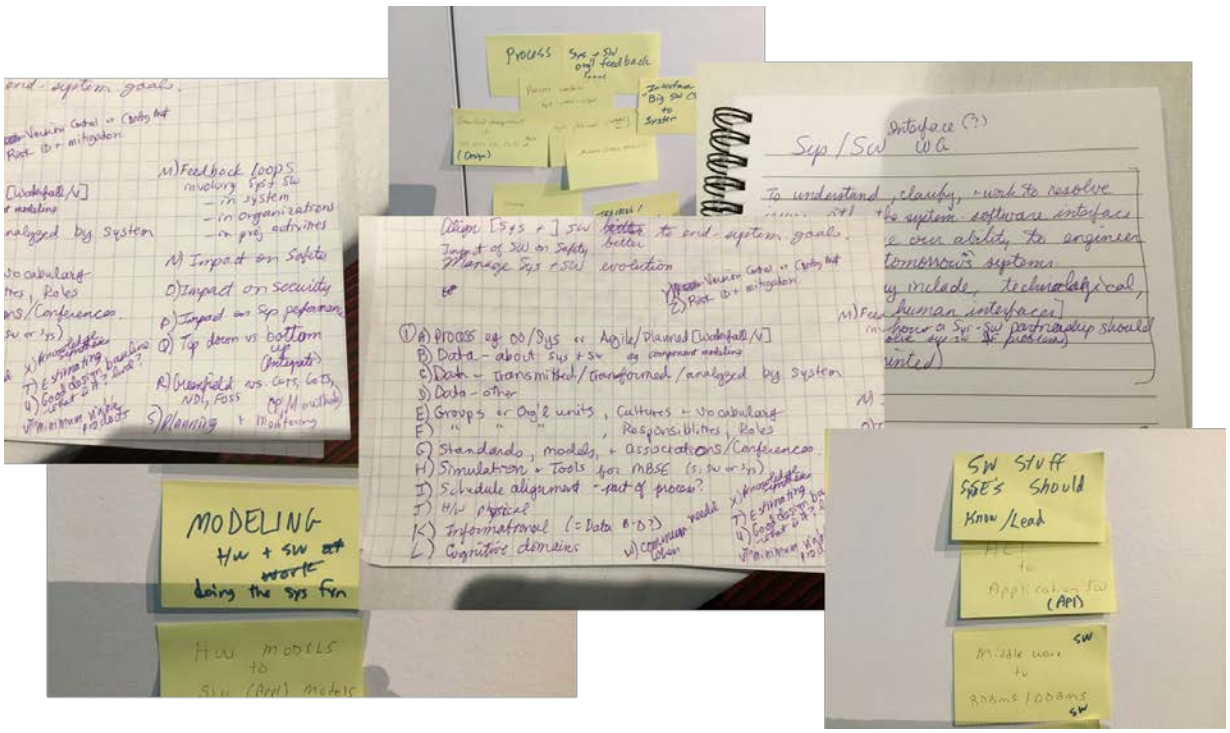

Figure 2. Examples of Brainstorming Outputs from SaSIWG at IS17

Although the tools category is very important, the group did not focus on it because of time constraints and because it is being addressed by others in INCOSE. Thus tools are not included in the discussion below, which elaborates on the breadth of interface issues between systems and software. However, the group does plan to ensure the tools are well integrated as part of future work.

Disciplines

It is recognized that both systems and software engineering have degree programs and curricula. The INCOSE members participating in the first working group did not know of a distinction that is widely agreed on, although they believe software engineers are generally expected to be able to write computer code and systems engineers may not have to; rather, systems engineers may be able to do all their important analyses in tools such as Microsoft Excel or commercial tools more specific to systems engineering.

In general, the working group believes that both disciplines could benefit from a more widely accepted description of the relationship between the two. This would involve specifying what is in each Figure 1, to some degree of specificity and maybe in different cases (such as commercial vs. defense, IT vs. embedded systems, greenfield development vs. maintenance, small vs. large systems, etc.). It may turn out that for software-intensive systems, there is no difference between system and software activities when speaking generally, and that only when tailoring the general tasks for a specific project would the levels and specifics of the roles be different.

Some differences between the disciplines are in their areas of focus:

- The systems discipline focuses on program lifecycle concepts such as system requirements, system architectures, concept of operations, and test.
- The software discipline focuses determining requirements for software, software architecture, determining algorithms, ensuring messages are correct and complete, and proving that the software is correctly written and meets necessary quality attributes.

In the overlapped area of the Venn diagram (Figure 1) both are engaged in similar activities but at different levels; these activities have many interfaces which must be resolved in a timely fashion.

(Sheard 2014) describes joint activities as identification of needs and requirements, architecting of the systems and software, identification of stakeholders, views, interfaces, and making tradeoffs, among others. This is where many software architects belong: those who essentially perform systems engineering for the software part of the system, or for the whole system which has software as its main implementation material.

The working group believes that in the long run, some engineers will need to continue to focus on breath and the client, whereas other will need to continue to focus on logical consistency, digital reliability, and the rapid growth and change of the software engineering field. Whether these two groups of engineers should be grouped together or kept separated as “systems” and ”software” is a question that cannot be answered with any authority today, but both breadth and details, consistency and speed of change will always need to be addressed.

Processes

Today, many organizations building systems with significant software content already have well-documented processes for systems engineering, for software engineering, and for their interaction. In many cases such processes were documented as part of compliance with a Capability Maturity Model, like the integrated model, CMMI®.[1] Still, some differences remain, and organizations that are less well integrated, or teams consisting of people from different organizations, may experience conflicts as a result. Some examples of conflicts include

- Systems engineers use the classic systems engineering "Vee" or emerging Model Based Systems Engineering (MBSE) processes, but software engineers use incremental build, Agile and Development Operations (DevOps) processes.
- Systems processes focus on controlling risk, but software processes emphasize flexibility, innovation and creativity, and software engineers may feel overly constrained.

There is a wide range of implementations in these processes that influence systems and software interfaces. The SaSIWG hopes to understand them and possibly recommend some standardized implementations for certain conditions. It is clear that both systems and software talent should be involved in determining the most effective and efficient processes to follow.

During its general discussion of systems and software engineering interfaces, the SaSIWG identified a number of system-software process issues. In some cases the issues may involve potential conflicts to be resolved. In others, there may not be a current well-understood way to integrate differences that are fundamental.

Potential process conflicts to be confirmed and, if true, resolved include the following:

- systems engineering functional decomposition of narrative needs and problem statements vs. software engineering object-oriented decomposition
- systems engineering functional modeling vs. software engineering object-oriented modeling
- systems engineering parsing of requirements to functions vs. software engineering parsing of requirements into features
- software engineering speed of adoption of new development methodologies (e.g., Development Operations (DevOps), Agile) outpacing systems engineering
- separate and possibly conflicting costing and estimation of work items
- systems engineering still predominantly plan-driven vs. software engineering increasingly Agile
- systems engineering WBS-based earned value analysis (EVA) vs. software engineering stories and task-based effort estimation

[1] CMMI is a registered mark owned by CMMI Institute LLC.

- different ways that communications occur, whether internal to one team or both, or organizational communications, or external stakeholder communications
- different terminology: should identify systems engineering and software engineering "languages" (e.g., vocabulary, terminology, taxonomy, ontology, etc.) (Sheard 2017)
- systems engineering use cases vs. software engineering user stories (Segue 2015)
- upgrading or refreshing hardware capability/function vs. software capability/functions (e.g., at the same time, at different times, etc.)
- technical feedback loops: Which systems and software modules/components participate in feedback loops with each other? For example, feedback that takes place when software triggers hardware attenuators.
- systems engineering prototypes and production vs. software minimum viable products.
- systems engineering development toolset vs. software engineering integrated development environment (IDE).
- systems engineering physical development environments vs. software engineering logical development environment

The group determined that the following factors may cause a variation in the way systems and software processes may interact:

- how information is managed (e.g., within an organization, across agencies, etc.)
- for any effort (e.g., development, acquisition, maintenance, etc.): How the software and system design of contributes to or harms safety and security (including cybersecurity) attributes
- how autonomous systems are developed, operated, and maintained
- potentially new types of technical interfaces for newer systems (e.g., autonomous systems)
- use of data analytics in systems or software development, acquisition, or maintenance, as well as decision making
- implementation of risk identification and analysis practice for physical risk, technical risk, cybersecurity risk, and economic risk
- processes used in greenfield or new system development, and variations to those processes to be used for alternative kinds of system and software development (e.g., reusing software or using COTS, GOTS, NDI, FOSS, as well as adding to and modifying existing systems)

A key issue identified by the SaSIWG at IS 2017 is the lack of methods and approaches to fully incorporate software engineering considerations in traditional systems engineering activities (e.g., systems analysis, requirements analysis, requirements engineering, architecting, etc.). This is especially true of the systems engineering specialty disciplines. For example, some systems engineering organizations assume the reliability of software is 1, despite published work on software reliability (Musa 1996, Schneidewind 2009).

Each of the systems engineering "specialities," however those are defined, may have its own system-software interface issues. For example, considering the "requirements" focus of systems engineering, there is disagreement and often there are conflicts or gaps when handing off requirements (how mature are the requirements, and are they in document or model form, for example). When organizations do this well, there is a give and take in the whole requirements process: text requirements don't suddenly appear to software engineers, who then have to decipher, decompose and react to them. The working group would like to understand what communication and clarification needs to occur and at what points.

Standards

A number of standards exist, for software, for systems engineering, and in some cases for both. Historically, systems engineering standards started with the U.S. Department of Defense (DoD)'s MIL-STD-499 and became through IEEE (12207) or EIA (632) in the 1990s.The ISO/IEC 15288 standard "System Lifecycle Processes" was architected by INCOSE members Stuart Arnold and Harold Lawson (released 2002), in order to move the knowledge in systems engineering standard to the international community. Software standards also began in the U.S. DoD (MIL-STD-498) but stayed as military standards longer (DOD-STD-2167, DOD-STD-2168, DOD-STD-7935) then jumped to a joint IEEE/EIA release (J-STD-016), before becoming an international standard in 1995 (ISO/IEC 12207) (Sheard 2001).

As early as 1995 there were papers written about harmonizing systems and software standards, (Singh 1995) but actual harmonization has been a process, with the systems lifecycle standard 15288:2015 version finally calling for the software lifecycle processes standard 12207:2008 version (Syntell 2014). However, there are many other IEEE and other standards that apply more or less to one or the other of systems and software. Examples include ISO/IEC 15939:2017 on the systems and software engineering measurement process, architecture description ISO/IEC/IEEE 42010:2011 and (draft) architecture processes ISO/IEC/IEEE 42020, and quality assurance standards such as ISO/IEC 15504 and the ISO 9000 series.

In addition there are process maturity models such as capability maturity models of software, systems, and integrated varieties. These are generally written by people not also on standards committees (Sheard 1998, p. 596).

The net effect is a large web of standards all citing other standards—and some use different terminology than others. The INCOSE Standards Working Group is INCOSE's torch bearer for this issue, but the SaSIWG may feel inspired to help with respect to specific system/software issues.

Organization

Any organization intending to build a large software-intensive system product will be divided up in some way, with smaller organizations responsible for smaller tasks. At some point, most of these organizations split system and software developers into different groups. Whether these exist as teams but work together in a project, or exist as companies and only work together through contracts, there are interface issues to be addressed. We show examples later in this paper that give a couple of examples about system-software interface issues that are characterized as organizational.

The SaSIWG noted the following aspects of interfaces between systems engineering and software engineering organizational groups or roles:

- synthesizing knowledge for one or both groups (e.g. literature, knowledgebases)
- boundary issues: Where does software scope end and systems or hardware scope begin?
- means of collaboration to create coherent organizational processes, developing today's systems … processes, tools, contracts, what?
- What communication between systems and software is needed and when?
- defining roles: systems engineering, system architecting, software architecting, software engineering, others;—and what are their parts of the process? Who takes the lead in project management?
- related: Does the U.S. DoD requirement for systems engineers to lead certain projects make sense, or should this be open to software engineers as well?
- should any aspects or parts of the software development process be led by systems engineers? What are the requisite skills?

- Who can say what is “not my problem?” If they do, does that mean they aren’t systems engineers?
- How does reduction of uncertainty occur from vague customer needs, through systems engineering and software engineering, to very-precisely defined software product occur?
- How should a good cost estimate be developed from systems and software estimates?
- What associations and conferences exist for systems and software and how do they relate?
- How do perceived authority, organizational heritage, and other cultural aspects lead to interface issues that need to be addressed?
- How do interfaces and information flow become part of a greater System of Systems entity?

Component Specifications

Today’s complex socio-technical systems (Sommerville 2016) are made up of an increasing number of physical and logical components (e.g., sub-systems, elements, etc.). Specifications for these physical and logical system components continue to be written in multiple system domains by various engineers (e.g., systems, hardware, software, network, etc.). Issues can arise during the specification of these integrated socio-technical system components, and their interfaces. Issues may occur when systems and software engineers separately but concurrently apply component-based development (CBD)[2] tools and techniques to the specification of physical and logical components, including interfaces, without the benefit of cross-domain knowledge or coordination. Relevant scenarios include systems engineers writing CBD specifications for physical components without knowing about possible software (i.e., logical) components, or interfaces with these components, and software engineers writing CBD specifications for logical system components without the requisite systems engineering domain knowledge or coordination.

Management

Enterprise management, engineering or systems engineering management, and software management may all occur within an organization. Good project management practices have been defined that help any project meet targets and provide required output. If that required output is necessarily a joint effort among different organizations (such as systems and software engineering organizations), then collaboration may need to be intense to make things work well, consistently. The SaSIWG identified the following ways where management-related systems and software interfaces need to be addressed:

- How can governance of systems engineering, software engineering, and data be made compatible?
- Policies (e.g., security) need to accommodate both disciplines participating.
- Planning: If the systems organization does the overall program plans and the software organization plans the software—which is the major piece of the program—and especially if the software piece is in many ways different from other pieces, then there can be problems. Sample question: Is it the responsibility of systems engineering to ensure the software part is organized and engineered for total lifecycle value?
- Lifecycles: This is about, primarily, the relationship of Agile (software engineering) to the Vee or waterfall lifecycle (systems engineering). Agile has defined timescales and variable scope, whereas the Vee has contracted scope/cost/schedule. There are reasons for both. How to make them work together?
- Engineering approach: How to make top-down (systems engineering) and bottom-up (software engineering) work together most effectively?

---

[2] The Gartner IT Glossary defines CBD as a set of reuse-enabling tools and techniques, that allow development organizations to go through the entire development process via the use of pre-determined component-enabling technologies (e.g., patterns, frameworks, design templates, etc.)

- There isn't always schedule alignment of systems and software activities. The overall schedule must at the least ensure correct timing of dependencies and interfaces.
- Tracking and monitoring of both systems and software should be transparent and made available to both as appropriate.
- Version control vs. hardware configuration management (CM). Software CM, well versed in version control, has somewhat different roots than systems engineering CM, based in part on hardware serial-number tracking. How can these be better interfaced than they are today?
- Risk mitigation and strategy. How are software-related risks combined with other risks? Are they comparable in terms of technical risk, schedule risk, cost risk, economic risk, or other kinds of risk?
- How is data provenance (records of the history of the data) managed across systems and software?
- How to manage the human-software interface. Is this strictly a software task or should systems engineers be involved, and if so, how?
- Managing contact with users. Both systems and software engineers have users to satisfy; how is this best done? Are they at different levels (systems: more like business managers, software: more like end users) and on different time scales? (systems: long-term, six months-two years between contacts; software: daily to weekly contact?)
- Similarly, both systems and software engineers interface with operators: what guidelines should apply?
- Regarding developers and administrators, both systems and software engineers should be supporting these in a non-conflicting and hopefully synergistic manner

Data

"Data" arose as a category for a number of somewhat disjoint interfaces. The SaSIWG needs to understand this category better.

- What are the modes of interaction? (e.g., real time, awaiting, etc.)
- Is an interface uni- or bi-directional?
- What protocols, naming conventions, etc. apply?
- Interfaces needed for data analytics: what data will be used in "big data" analysis and where does it come from?
- Interfaces needed regarding data transmitted or transformed or analyzed by the system (product)
- Interfaces needed regarding data about the system, the software, or their components (meta-data)
- Interface of software development environment to software applications used by the customer
- Interfaces among software, firmware, and computer hardware
- What specific software interfaces do the project's lead systems engineers have to be able to talk to?
    - HCI software to Application SW (API)
    - middleware to RDBMS/ ODBMS
    - middleware to application (API)
    - operating systems to application software
    - HCI software to RDBMS/ ODBMS
    - application SW to RDBMS/ ODBMS
    - OS to middleware
    - database to data source interfaces
    - HW models to SW (Application) models to Database models
    - HW–SW interfaces in general, also firmware

Examples

Example 1. Prime Solutions Group

Prime Solutions Group (PSG) is a small business involved with systems and software development. The company is a Very Small Entity (VSE) with fewer than 25 employees. Even though it is small, the business is focused on big challenges in ground processing systems and innovative research in complex system modeling and simulation. This research requires innovative software development which must simultaneously be flexible enough to accommodate continuous experimentation and apply systems theory.

What PSG lacked initially was a system architecture to communicate internally as well as externally to research advisors and potential customers. Also, it was difficult to integrate advanced systems concepts with the Agile methodology that the PSG software developers use. Through association with INCOSE, PSG stumbled on the Agile Systems Engineering Life Cycle Management (ASELCM) and Patterns-Based Systems Engineering (PBSE) concepts.

ASELCM and PBSE emphasize features, a key concept that PSG thinks helps align systems and software development. Agile software developers focus on software features to drive development during Agile sprints. With PBSE, systems engineers focus on features. This conveniently offers a common basis for communication between Agile systems and software developers. Systems engineers can indicate the focus of a software increment or sprint in the context of the system solution.

PSG sought solutions for harmonizing the systems and software engineering interface regarding the effective communication and representation of a project. PSG is currently experimenting with PBSE techniques to ingest them into the organizational systems engineering processes. PBSE gives an idea for structure and a process oriented mechanism for creating artifacts to help address the need for overall systems and software harmony within the organization.

Example 2. Comments about Systems Engineering

The statements in Table 1 arose in conversations with software engineers. The comments and responses show the need for systems and software engineers to understand each other better.

Table 1. Software Engineering Comments about Systems Engineering

Comment	Implication	Systems Engineering Response
“We shouldn’t suggest systems engineers don’t do details! That’s incredibly insulting to them!”	People who don’t do details are useless or stupid, because details are the only things that matter.	Breadth is equally challenging as depth. Big systems are systems engineered precisely because sub-optimization happens when all the details conflict all the time. Someone has to do their best possible job at understanding the tradeoffs and just make a decision. Here the systems engineer often functions as a technical manager, when making a reasonable, pretty-good decision is better than making none.
“What is the ‘secret sauce’ that systems engineers do and we don’t?”	We software engineers already do more complex things than you systems engineers	Systems engineering specializes in mixing, blending, balancing, using a wide variety of tools (many not software-based, rather conceptual) to share language, enable communication, create based on vague goals, identify precision in a measured way, prioritize, etc. These

Comment	Implication	Systems Engineering Response
	do, and if you just tell us what those special things are that we don't do, we'll figure out how to do them too, and you wouldn't be needed.	needs are not just software topics and they require knowledge of not just software tools. Systems engineering doesn't have a "special sauce" that is added. Rather it ensures that everyone bakes in the necessary ingredients to make a good product.
"Systems people don't realize software can keep their systems from being fielded."	Systems engineers think of software as an unimportant "part" when it's really a huge, complex system of its own.	True, some systems engineers, project managers, and program managers don't understand the challenges software engineers have to overcome to work as well as it needs to. This is partly a communication problem because software engineers don't excel at boiling major issues down to the basics for management. This issue has to be addressed on both sides.
"Software is driving the bottom line and programs (and organizations) are too old-fashioned to realize that, they still think in terms of physical systems."	Programs think hardware is important and costs money, and we software engineers know none of that really counts.	True, program managers often have had long careers and thus missed the era where software was a driving force. However, they aren't stupid. Software engineers underestimate the cost of ill-fitting hardware or making changes in airframes or performing maintenance on engines. Airplanes cannot fly on software alone. There has to be knowledge all around.
"Systems engineers often use ad hoc tools that don't work together and can't be used as input to software people. They're too spoiled to try to use tools that it takes a few weeks to learn. They just want their Excel, or worse, drawing tools."	Systems engineers are undisciplined and don't take enough time to think about things and make them consistent across the system and across the industry.	Systems engineers, as a rule, don't get to sit in an office for weeks at a time, working on a few things. They are often on call to "just fix it" for many things, ranging from next week's issue that is threatening to derail a design review, to the $100K-a-day test that is on hold due to an unresolved anomaly. Their job involves people using tens to dozens of different tools, of different versions, and they know that many of the inputs they get today will be obsolete tomorrow. Reality doesn't stand still. So, many cope with general tools that they can use "well enough" and they and their managers turn down the option of taking a week to a month to learn a complex tool that does everything consistently, because there are 99 other tools that they still won't know, and before they learn five or six of them, the first will have new versions and new capabilities.
"SysML was made for systems engineers to do model-based systems engineering (MBSE). Why don't they use it then? Why do they use this hodgepodge of home-grown analyses?"	Systems engineering is a design discipline that can be performed by sitting at a desk making models in a highly capable tool.	Most of systems engineers' most valuable duties cannot be performed alone in an office. They walk around talking to people, looking for information, gathering perspectives, calling meetings, ensuring assumptions different people are making are compatible, following through on action items, reviewing work products to be sure "ilities" (system quality attributes) are being addressed by all decisions, estimating costs and other impacts of impending decisions, etc. SysML was in fact created for a particular type of systems engineers, the designers of mostly-software-based systems. It makes much less sense to use in other contexts. Regarding why don't they use a certain tool, why don't

Comment	Implication	Systems Engineering Response
		software engineers use any other language than the one they're using?

Potential Interfaces: INCOSE Working Groups and External

INCOSE has recently signed a memorandum of understanding (MOU) with the Object Management Group (OMG); likely the System and Software Interface Group will have a role to play in realizing the intent of the memorandum.

The following INCOSE working groups have been identified as potentially holding keys to the solution to some of the issues identified above. These groups include the following. It is recognized that this is not a complete list:

- Agile Systems and Systems Engineering
- Architecture
- Competency
- Decision Analysis
- Enterprise Systems
- Life Cycle Management
- Model Based Systems Engineering Initiative and a number of working groups within
- NAFEMS-INCOSE Systems Modeling and Simulation
- OOSEM
- Patterns
- PM-SE Integration
- Process Improvement
- Requirements
- Risk Management
- Systems Security Engineering
- Tools Integration and Model Lifecycle Management
- Training

The plan is for several SaSIWG members to select working groups and serve as liaison. Questions include: Has your group created any information that could help the SaSIWG in its task? Might SaSIWG members be able to help you in your task? Do you have an overview of your working group that you want to give to the SaSIWG? What information might you like from the SaSIWG?

Other useful interfaces will involve software engineering individuals and groups. ACM and IEEE have been suggested to start with, but more groups will be identified that are interested in the system-software interface. Collaboration efforts will begin with dual members of SaSIWG and these other groups, and expand as volunteer interest, skills, and time allows.

Summary and Plan Forward

In summary, the new INCOSE Systems and Software Interface Working Group (SaSIWG) has started investigating how systems engineering and software engineering do interface and how they should interface in the future to ensure that the large, software-based systems of the future still are created with good, balanced systems engineering. Collecting and organizing issues is a first step that is necessary to define the problem that the working group will address. The plan going forward will collect information to prioritize solution of the problems, and then begin to address the most important needs by working in small groups that have the skills and energy to determine the best way forward. This way forward is intended to change systems and software engineering practices in order to

improve effectiveness of the systems that are created, to reduce overall lifecycle cost, and to prevent problems as much as possible.

The SaSIWG will also find out what other INCOSE working groups are doing in this space and coordinate to reduce duplication of effort and to add additional knowledge and perspectives. A similar step with software engineering institutions and groups is also likely, but the details are not as clear since these are non-INCOSE organizations.

The working group is interested in attracting workers to meet at the symposium and at telecons inbetween international meetings to accelerate progress on more fronts.

In this paper we have published our initial answer to the question "in what ways should the interfaces between systems and software be improved?" We welcome feedback as to additions to the lists and relative priorities of the issues listed. Interested INCOSE members are encouraged to contact the group chair and co-chairs listed on the SaSIWG web site and volunteer to participate.

References

Alford, Mack. "Strengthening the Systems/Software Engineering Interface." In Proceedings of 2nd NCOSE Conference. 1992.

Andriole, Stephen J., and Peter A. Freeman. "Software systems engineering: The case for a new discipline." Software Engineering Journal 8, no. 3 (1993): 165-179.

Armstrong, James R., and Arthur Pyster 1997. "Resolved: Software Should Lead in Systems Engineering." In INCOSE International Symposium, vol. 7, no. 1, pp. 317-324.

BKCASE (Body of Knowledge and Curriculum to Advance Systems Engineering Project) 2017. Guide to the Systems Engineering Body of Knowledge (SEBoK), sebokwiki.org/wiki/Guide _to_the_Systems_Engineering_Body_of_Knowledge_(SEBoK), Retrieved April 19, 2017.

Pyster, A. and D.H. Olwell, eds., 2013. The Guide to the Systems Engineering Body of Knowledge (SEBoK), v. 1.2, Stevens Institute of Technology; http://sebokwiki.org.

Boehm, 2000 Unifying SWE and SysE. IEEE Computer, March 2000, pp 114-116

Boehm, Barry, and Richard Turner. *Balancing agility and discipline: A guide for the perplexed*. Addison-Wesley Professional, 2003.

Boehm, Barry 2006a. "Some future trends and implications for systems and software engineering processes." Systems Engineering 9, no. 1: 1-19.

Boehm, Barry 2006b. "A view of 20th and 21st century software engineering." In Proceedings of the 28th international conference on software engineering, pp. 12-29. ACM. 2006.

Boehm, Barry, and Jo Ann Lane, 2007. "Using the incremental commitment model to integrate system acquisition, systems engineering, and software engineering." CrossTalk 19, no. 10: 4-9.

Boehm, Barry, Jo Ann Lane, Supannika Koolmanojwong, and Richard Turner 2010. "Architected agile solutions for software-reliant systems." In Agile software development, pp. 165-184. Springer, Berlin, Heidelberg.

Boehm, Barry. 2011. "Some future software engineering opportunities and challenges." In The future of software engineering, pp. 1-32. Springer, Berlin, Heidelberg.

Bourque, P. and R.E. Fairley, eds. 2014. Guide to the Software Engineering Body of Knowledge (SWEBOK), Version 3.0, IEEE Computer Society Press; www.swebok.org.

Brownsword, L., Fisher, D., Morris, E., Smith, J., and Kirwan, P., 2006, "System-of-Systems Navigator: An Approach for Managing System-of-Systems Interoperability."

Demir, Kadir Alpaslan. 2015. "Multi-view software architecture design: case study of a mission-critical defense system." Computer and Information Science 8, no. 4: 12.

Fairley, Richard E., and Mary Jane Willshire. 2011a. "Teaching Systems Engineering to Software Engineering Students." In Software Engineering Education and Training (CSEE&T), 24th IEEE-CS Conference, pp. 219-226. IEEE.

Fairley, Richard E. and Mary Jane Willshire. 2011b. "Teaching software engineering to undergraduate systems engineering students." *Proceedings of the 2011 American Society for Engineering Education (ASEE) Annual Conference and Exposition*. 26-29 June 2011. Vancouver, BC, Canada.

Fairley, Richard E. 2014. "A Software Engineering Competency Model (SWECOM)." IEEE Computer Society. Ch. 17: Software Systems Engineering Skill Area.

Frerking, Roland, et al. 1996. “Systems and Software Engineering Interfaces,” outputs of an EIA workshop, Baltimore MD.

Giese, Holder. 2005-2006 “Software Engineering for Software-Intensive Systems: I. Introduction”. Winter semester, University of Paderborn Software Engineering Group.

Government Accountability Office (GAO), 2016, *Weapon System Requirements: Detailed Systems Engineering Prior to Product Development Positions Programs for Success* (GAO-17-77)

Hagan, C., Hurt, S., & Sorenson, J., 2013, “Effective approaches for delivering affordable military software,” *Crosstalk*, pp.26–32

Harbaugh, Sam. “Experiences in Training Software Engineers to Perform Systems Engineering.” In INCOSE International Symposium, vol. 3, no. 1, pp. 783-786. 1993.

ISO/IEC/IEEE 2017. 12207-2017, IEEE Standard for Systems and Software Engineering—Software Life Cycle Processes, IEEE.

ISO/IEC/IEEE 2015. 15288-2015, IEEE Standard for Systems and Software Engineering—System Life Cycle Processes, IEEE.

Kasser, Joseph, and Sharon Shoshany, 2000. “Systems Engineers are from Mars, Software Engineers are from Venus,” Proceedings of the Thirteenth International Conference on Software & Systems Engineering and Their Applications. Paris, December.

Kossiakoff, Alexander, William N. Sweet, Samuel J. Seymour, and Steven M. Biemer 2011. Systems engineering principles and practice. Vol. 83. John Wiley & Sons.

Kraft, Ed, 2013. “Expanding the Digital Thread to Impact Total Ownership Cost.” AEDC PA 2013-198. NIST MBE Summit, Gaithersburg, MD, Dec. 18-19.

Lane, Jo Ann, Doncho Petkov, and Manuel Mora 2009. “Software engineering and the systems approach: A conversation with Barry Boehm.” Strategic Information Systems: Concepts, Methodologies, Tools, and Applications: Concepts, Methodologies, Tools, and Applications: 333.

Maier, Mark W. 2006. “System and Software Architecture Reconciliation.” *Systems Engineering* 9, no. 2: 146-159.

Musa, John D. 1996. *Software-Reliability. Handbook of Software Engineering*, Van Nostrand Reinhold Company.

Naegle, B., 2015a, ‘DoD Software-intensive Systems Development: A Hit and Miss Process,’*12th Annual Acquisition Research Symposium.* (NPS-GSBPP-SYM-AM-15-089) Monterey, CA: Naval Postgraduate School, May 13-14.

Naegle, B., 2015b, *Gaining Control and Predictability of Software-Intensive Systems Development and Sustainment*, Acquisition Research Program Sponsored Report Series [NPS-AM-14-194]. Monterey, CA: Naval Postgraduate School

Nelson, Georgia E. 2007. “Applying Object-Oriented Concepts to the Engineering of Complex Software Systems.” University of Maryland University College, MSWE 603 Systems Engineering.

Northrop, Linda, Peter Feiler, Richard P. Gabriel, John Goodenough, Rick Linger, Tom Longstaff, Rick Kazman et al. 2006. Ultra-Large-Scale Systems: The Software Challenge of the Future. Carnegie-Mellon University, Pittsburgh Pa. Software Engineering Institute.

Oliver, David W. 1995, “Systems Engineering & Software Engineering, Contrasts and Synergism”, Proceedings of the Seventh International Workshop on Computer-Aided Software Engineering (CASE ‘95), IEEE Computer Society Press, August, pp. 186-193.

Pafford, M., 2008-2018. “Software Systems Engineering.” Course Materials for Johns Hopkins University Engineering for Professionals Course EN.645.764.

Pyster, Art, Rick Adcock, Mark Ardis, Rob Cloutier, Devanandham Henry, Linda Laird, Michael Pennotti, Kevin Sullivan, and Jon Wade 2015a. “Exploring the relationship between systems engineering and software engineering.” Procedia Computer Science 44 (2015a): 708-717.

Pyster, A., D.H. Olwell, T.L.J. Ferris, N. Hutchison, S. Enck, J. Anthony, D. Henry and A. Squires (eds.). 2015b. Graduate Reference Curriculum for Systems Engineering (GRCSE™), Body of Knowledge and Curriculum to Advance Systems Engineering (BKCASE) Project. V1.1. Hoboken, NJ, USA

Rechtin, Eberhardt and Mark Maier. 1997. Art of Systems Architecting, Chapter 6, CRC press.

Rose, Susan, 1997. “Engineering Harmony Between Systems and Software.” In *INCOSE International Symposium*, pp. 419-426.

Rosser, Larri, P. Marbach, David Lempia, and Gundars Osvalds 2014. “Systems Engineering for Software Intensive Projects Using Agile Methods.” *INCOSE Annual Symposium*, Las Vegas, NV.

Rozanski, Nick, and Eoin Woods. 2005. *Software Systems Architecture*. Upper Saddle River, NJ: Addison WesleySage, Andrew P., and James D. Palmer. *Software Systems Engineering*. Wiley, 1990.

Sambur, Marvin R. and Peter B. Teets 2004. “Revitalizing the Software Aspects of Systems Engineering,” DoD Memo, Assistant Secretary of the Air Force and Under Secretary of the Air Force, Sept. 20, 2004.
Schneidewind, Norman, 2009. Systems and Software Engineering with Applications. IEEE, 2009.
Seaman, Carolyn B. and Yuepu Guo 2011. “Measuring and Monitoring Technical Debt.” Advances in Computers 82:25-46.
Segue Quality Control Team, 2015. “User Stories vs Use cases: Pros and Cons for Agile Development”. Segue Technologies. https://www.seguetech.com/user-stories-vs-use-cases-pros-cons-agile-development/, accessed 7 November 2017.
Sheard, Sarah A. 1998. “Systems Engineering for Software and Hardware Systems: Point-Counterpoint.” In *INCOSE Annual Symposium*, Vancouver, Canada, pp. 928-936.
Sheard, Sarah A., and Jerome G. Lake, 1998. “Systems engineering standards and models compared.” *INCOSE Annual Symposium*, Vancouver, Canada, pp. 589-605.
Sheard, Sarah A. 2001. “Evolution of the frameworks quagmire.” Computer 34, no. 7: 96-98.
Sheard, Sarah A. 2004. “Adapting Systems Engineering for Software-Intensive Systems.” *INCOSE Annual Symposium*, Toulouse, France, June 20-24.
Sheard, Sarah, 2014. “Needed: Improved Collaboration between Software and Systems Engineering.” Blog post, Software Engineering Institute blog, https://insights.sei.cmu.edu/sei_blog /2014/05/needed-improved-collaboration-between-software-and-systems-engineering.html, May 19, 2014 (accessed Nov. 7, 2017).
Sheard, Sarah, 2017. Best Practice Meets Invisible Force: Systems Engineering for Mostly-Software Systems of Systems. Tutorial, *INCOSE International Symposium*. Adelaide, Australia: July.
Sillitto, H. “Design principles for Ultra-Large-scale Systems (ULS).” In *INCOSE International Symposium* 2010.
Singh, Raghu, 1995. “Harmonization of software engineering and system engineering standards.” In Software Engineering Standards Symposium, 1995 ‘Experience and Practice’, Proceedings, Second IEEE International, pp. 262-267. IEEE.
Snyder, Charles R. 1991. “System engineering impact on software development.” In Proceedings of the conference on TRI-Ada’91: today’s accomplishments; tomorrow’s expectations, pp. 425-431. ACM.
Sommerville, Ian 2010. *Software engineering*. New York: Addison-Wesley, 2010.
Sprunck 2012. Sprunck, Markus. Top 12 things every software engineer should know http://www.sw-engineering-candies.com/blog-1/top10thingseverysoftwareengineershould know
Steiner, Rick. 2004. “Leveraging Software Development Approaches in Systems Engineering.” Naval Postgraduate School Project Seminar, 6 May 2004.
Sweeney, Robert L., Jeffrey P. Hamman, and M. Biemer. "The application of systems engineering to software development: a case study." *Johns Hopkins APL Technical Digest* 29 (2011): 327-337.
Syntell 2014. “15288 Harmonized with 12207”. http://www.15288.com/relationship.php. Accessed 7 November 2017.
Turner, Richard, Arthur Pyster, and Michael Pennotti. “Developing and validating a framework for integrating systems and software engineering.” In Systems Conference, 2009 3rd Annual IEEE, pp. 407-412. IEEE, 2009.
USAF 2018. Weapon Systems Software Management Guidebook. Retrieved March 1, 2018 from http://www.acqnotes.com/Attachments/USAF%20Weapon%20System%20Sofware%20Management%20Guide.pdf
Vierhauser, Michael, Rick Rabiser, and Paul Grünbacher. 2014. “A case study on testing, commis-sioning, and operation of very-large-scale software systems.” In *Companion Proceedings of the 36th International Conference on Software Engineering*, pp. 125-134. ACM.
Wallace 1996. Wallace, Dolores R., Arthur H. Watson, and Thomas J. McCabe. “Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric.” NIST Special Publication 500-235, 1996.
Wang, Gan, Garry J. Roedler, Ricardo Valerdi, Aaron Ankrum, and John E. Gaffney. 2009.”Harmonizing Systems and Software Cost Estimation.” In *INCOSE International Symposium*, vol. 19, no. 1, pp. 232-252.
White, Stephanie M. “Improving the system/software engineering interface for complex system development.” In Engineering of Computer-Based Systems, 2005. ECBS’05. 12th IEEE International Conference and Workshops on the, pp. 281-288. IEEE, 2005.Wray, Richard B. 1993.

"Systems Engineering and Software Engineering: Cooperative or Competitive?." In *INCOSE International Symposium*, vol. 3, no. 1, pp. 833-843.

Wrubel 2014. Wrubel, Eileen, et al. "Agile Software Teams: How They Engage with Systems Engineering on DoD Acquisition Programs." SEI/CMU-2014-TN-013. Software Engineering Institute, July 2014.