Software Engineering Institute

U.S. Army Workshop on Exploring Enterprise, System of Systems, System, and Software Architectures

John Bergey, Stephen Blanchette, Jr., Paul Clements
Mike Gagliardi, John Klein, Rob Wojcik, Bill Wood

March 2009

Research, Technology, and System Solutions

http://www.sei.cmu.edu

Carnegie Mellon

Table of Contents

List of Figures

List of Tables

Acknowledgments

Abstract

This report summarizes a U.S. Army workshop on architecture that was held at the Carnegie Mellon® Software Engineering Institute (SEI) in September 2008, under the auspices of the Army Strategic Software Improvement Program (ASSIP). The workshop organizers invited accomplished practitioners from government, academia, and industry to discuss the various "genres" of architecture: enterprise architecture, system of systems architecture, system architecture, and software architecture. The goal of the workshop was to clarify the relationships among the different genres, explore and identify areas of commonality and difference, and to discuss the role of the Department of Defense Architecture Framework (DoDAF) in helping to capture these architectures.

After a selection of opening talks by individuals that provide overviews of each subject area, the workshop dissolved into working groups. Each group was tasked with working on a specific set of issues from the perspective of one of the abovementioned architecture genres, and then summarizing their conclusions for the whole workshop. The issues discussed by each group include these:

1. What are the major activities involved in each genre?

2. What is the boundary (e.g., information flow) between architecture in one genre and architectures in the other genres?

3. What do architectures in each genre need to address in order to be considered successful?

4. How do we document an architecture in each genre? What notations and approaches are available? What are the minimum views and information necessary to ensure the architecture documentation will be adequate to support development and to conduct an evaluation as part of an acquisition?

5. How can the DoDAF be used to represent an architecture in each genre? What are its strengths and weaknesses with respect to each genre? How could it be improved? What is the current state of the practice with respect to using the DoDAF with each genre?

This report summarizes the workshop and its findings.

---

1 A Workshop on Architecture Genres for the U.S. Army

1.1 Background

The U.S. Army Strategic Software Improvement Program (ASSIP), which is sponsored by the Office of the Assistant Secretary of the Army for Acquisition, Logistics, and Technology (ASA(ALT)), is a multiyear effort targeted at dramatically improving the way in which the Army acquires software-intensive systems. The ASSIP is predicated on the idea that better acquisition practices will lead to better systems and overall results.

As part of its involvement with the ASSIP, the Carnegie Mellon® Software Engineering Institute (SEI) was asked to "conduct an Army-wide workshop to explore and clarify the relationships between different kinds of architecture, (especially the link between the U. S. Department of Defense Architecture Framework [DoDAF] and system and software architecture)." This workshop was held on September 23-24, 2008 in the SEI's Arlington, Virginia, offices. This report summarizes the proceedings and findings of the workshop.

1.2 Workshop Purpose

The purpose of this workshop was to examine the four major "genres" of architecture:

- enterprise architecture
- system of systems (SoS) architecture
- system architecture
- software architecture

The goal was to clarify the relationships among these different genres, explore and identify areas of commonality and difference, and discuss the role of the DoDAF in helping to capture these architectures.

1.3 Workshop Participants

This workshop assembled a number of experts from the major fields of architecture, as well as Army representatives of acquisition efforts in each of the covered fields. The participants of this workshop are listed (in alphabetical order) in Table 1. The participants were invited based on their expertise and interest in exploring the different genres of architecture and the interrelationships of these architectures with one another and the DoDAF. A total of 23 individuals participated in the workshop; five were from the Army, eight were from contractor organizations, two were from MITRE, and eight were from the SEI.

®   Carnegie Mellon is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

Table 1:   Workshop Participants

Name	Organization
John Bergey	SEI
Stephen Blanchette	SEI
Paul Clements	SEI
Judith Dahmann	MITRE
Fatma Dandashi	MITRE
Bradley Drake	U.S. Army ARDEC
Dave Emery	DSCI
Mike Gagliardi	SEI
John Grove	Booz Allen Hamilton
Steve Kishok	U.S. Army PEO Soldier
John Klein	SEI
John Andrew Landmesser	U.S. Army PEO C3T PM BC
Mark Maier	The Aerospace Corporation
Linda Northrop	SEI
Don O'Connell	Boeing
Shawn Rahmani	Boeing
Rolf Siegers	Raytheon
John Tieso	Booz Allen Hamilton
Ron Vandiver	U.S. Army, Army Architecture Integration Management Directorate, ARCIC, TRADOC
Jeff Vermette	Lockheed Martin Aeronautics
Rob Wojcik	SEI
William Wood	SEI
Carol Wortman	U.S. Army CIO/G-6

1.4 About This Workshop

After a short welcome and workshop overview that established the purpose and expected outcomes of the workshop, a number of invited presentations were given by invited participants in accordance with the workshop agenda shown in Table 2.

Table 2:   Workshop Agenda

Day 1		
Time	**Topic**	**Presenter**
0800-0830	Continental breakfast	
0830-0845	Welcome and background, including summary of SEI/Army architecture work	John Bergey, SEI
0845-0900	Workshop goals, purpose, and methodology	Paul Clements, SEI
0900-0945	Overview of Enterprise Architecture: definition, IT alignment, governance, notations and languages, Zachman, TOGAF,	Carol Wortman, U.S. Army CIO/G-6

Day 1		
Time	**Topic**	**Presenter**
	FEAF, major activities and state of the art/practice	
0945-1030	Overview of System of Systems Architecture: major activities and state of the art/practice	Judith Dahmann, Senior Principal Systems Engineer, Center for Acquisition and Systems Analysis, MITRE Corporation
1030-1045	Break	
1045-1130	Overview of System Architecture: major activities and state of the art/practice	Mark Maier, System Architect / Engineer, The Aerospace Corporation
1130-1215	Overview of Software Architecture: major activities and state of the art/practice	Mark Klein, head, SEI Software Architecture Technology initiative
1215-1315	Lunch	
1315-1345	Overview of DoDAF 1.5	Fatma Dandashi, Simulation Modeling Engineer, MITRE Corporation
1345-1415	Critique of DoDAF 1.5 for use in system architecture	David Emery, DSCI
1415-1445	Overview of DoDAF 2.0	John Tieso, Associate, Booz Allen Hamilton
1445-1645	Formation of working groups (EA, SoS, Sys, SW); mission statements given to working groups; working groups meet[1]	All

Day 2		
Time	**Topic**	**Presenter**
0800-0830	Continental breakfast	
0830-1200	Working groups meet.	All
1200-1330	Working groups report (~00:20 each). Working lunch.	All
1330-1400	Workshop wrap-up; assignment of report sections; planning of production of final report	All

The architecture genres that were actively explored and discussed by the participants included enterprise architectures, SoS architectures, system architectures, and software architectures. Presentations in each of these architecture fields of interest provided helpful background. Following the presentations, working groups were formed to tackle some or all of the following issues with respect to each of the genres:

1. What are the major activities involved in each genre? Some examples include understanding architecturally significant requirements, designing, documenting, evaluating, checking for conformance, governance, sustainment, and infrastructure alignment. Do the genres share an over-arching set of activities?

2. What is the boundary (e.g., information flow) between architecture in one genre and architectures in the other genres?

---

[1] The abbreviations used in this table for the working groups are described as follows: EA (enterprise architecture), SoS (system of systems), SYS (system architecture), and SW (software architecture).

3. What do architectures in each genre need to address in order to be considered successful? For software architectures, the "usual" quality attributes (QAs) include software performance, security, modifiability, etc. For system architectures, concerns include the above plus system size and weight, power consumptions, etc. What qualities should be considered for the other types of architectures? What is the relationship to business and mission goals? How can these architectures be evaluated or analyzed?

4. How do we document an architecture in each genre? What notations and approaches are available? What are the minimum views and information necessary to ensure the architecture documentation will be adequate to support development and to conduct an evaluation as part of an acquisition?

5. How can the DoDAF be used to represent an architecture in each genre? What are its strengths and weaknesses with respect to each genre? How could it be improved? What is the current state of the practice with respect to using the DoDAF with each genre?

1.5 Organization of This Report

This document summarizes the presentations and discussions from the architecture workshop. This report is laid out as follows.

- Section 2 summarizes the invited presentations made by SEI, Army, and external presenters, respectively.

- Section 3 summarizes the discussions and findings of each of our working groups.

- Section 4 presents a set of workshop findings synthesized from the working group results.

- Section 5 summarizes important findings of the workshop that substantiate how software architecture practices are beneficial and relevant to Army programs. It also contains ideas for future work.

2 Presentation Summaries

During the first day, a number of background presentations were given by experts in the various architectural genres, as well as experts in the current and future versions of the DoDAF. This section summarizes each of those presentations:

- Enterprise Architecture—Carol Wortman, U.S. Army CIO/G-6

- System of Systems Architecture—Judith Dahmann, Center for Acquisition and Systems Analysis, MITRE Corporation

- System Architecture—Mark Maier, System Architect / Engineer, The Aerospace Corporation

- Software Architecture—Mark Klein, head, Software Architecture Technology Initiative, Software Engineering Institute

- DoDAF 1.5—Fatma Dandashi, Simulation Modeling Engineer, MITRE Corporation

- Critique of DoDAF 1.5—David Emery, DSCI

- DoDAF 2.0—John Tieso, Associate, Booz Allen Hamilton

2.1 Enterprise Architecture—Carol Wortman, U.S. Army CIO/G-6

Carol Wortman presented the U.S. Army CIO/G-6 perspective on enterprise architecture, which is seen as the way to translate business vision and strategy into effective enterprise change.

2.1.1 Defining Enterprise Architecture

The CIO/G-6 working definition of enterprise architecture is provided by Gartner Group:

> *Enterprise architecture is the process of translating business vision and strategy into effective enterprise change by creating, communicating and improving the key requirements, principles and models that describe the enterprise's future state and enable its evolution. The scope of the enterprise architecture includes the people, processes, information and technology of the enterprise, and their relationships to one another and to the external environment. Enterprise architects compose holistic solutions that address the business challenges of the enterprise and support the governance needed to implement them* [Lapkin 08].

This definition makes several important points about enterprise architecture. First, enterprise architecture is an ongoing process. Next, the goal of enterprise architecture is to translate business strategy into enterprise change, with a focus on future state and evolution. It is distinguished from other types of architecture because the scope includes people and processes. Finally, the enterprise architecture must provide holistic solutions.

Carol shared her experiences that show why enterprise architecture has become important to the U.S. Army. In Afghanistan in 2001, systems were generally not networked, and changes to any individual system did not ripple out to other systems. By the time the Army entered Iraq in 2003, systems were networked, and changes to any system rippled through the network. Today, Army systems are connected to inter-agency networks, and change management is even more critical.

Enterprise architects must understand the seams, pieces, and boundaries, and take a holistic view of how design decisions in one system affect other systems.

2.1.2 Army Enterprise Architecture Focus Areas

End users want to connect anywhere and at any time, which leads to concerns about security and scalable service delivery. The Army is addressing this with an enterprise architecture that can be viewed as having three elements:

- Network Service Center—This is an Army effort to reduce the number of networks and the number of access points. While the goal is to get to a single network, this is not realistic. We do think we can reduce the number of access points from 400 to 5, which supports integration with DoD security infrastructure.
- DoD Security and Identity Management—This is a joint service effort necessary to support sharing data across services within DoD.
- Data Strategy—This element comprises many programs, inside the services, across DoD, and includes industry. Where do I get what I need, what is the format, who has access to it? This includes the unsolved problem of how to enforce need-to-know. Current architectures based only on clearance level do not provide the granularity needed for authorization decisions.

It was noted that these are *actions*, and architecture usually focuses on *qualities*. Carol shared the "soldier story" motivation behind this plan. When a soldier deploys from CONUS (Contiguous United States) to theater, everything changes. Online identity, quality of service, and the customary data sources don't move with the soldier, and there is a huge impact on productivity and effectiveness. The near-term priority for Army enterprise architecture is to ameliorate this situation by defining the solution at the enterprise level and focusing on remediation at the system level.

2.1.3 Questioning the Definition of Enterprise Architecture

The discussion then shifted to questioning the Gartner definition of enterprise architecture.

Is enterprise architecture a process or a solution? The Gartner definition says that enterprise architecture scope is elements (people, processes, information, and technology) and the relationships to one another and to their environment. This implies that enterprise architecture is a product/solution. However, enterprise architecture is also concerned with the "when and how" of change, implying that it is a process.

There was a question of whether the focus on distinguishing between enterprise, SoS, system, and software architectures is misplaced, and should we simply focus on elements and qualities. Carol pointed out that the Gartner definition could apply to any level of architecture by substituting different element types and external environments to adjust the scope.

It was also pointed out that there are "enterprises within enterprises." For example, the Army contains a business enterprise and a warfighting enterprise. A logistics enterprise bridges between them. There are also inter-agency enterprises, which complicate the topology; the representation is not simply concentric circles.

2.1.4 The Role of the Enterprise Architect

Carol concluded by discussing challenges unique to enterprise architecture. The scope of enterprise architecture includes people, processes, information, and technology. However, of these, the enterprise architect can control only the technology element, and has less (or at best, indirect) influence over the other elements. This makes the role of the enterprise architect especially challenging.

2.2 System of Systems Architecture—Judith Dahmann, Senior Principal Systems Engineer, Center for Acquisition and Systems Analysis, MITRE Corporation

Dr. Judith Dahmann presented an overview of the Department of Defense (DoD) SoS systems engineering perspective as the context in which to understand SoS architecture issues. Her presentation was based on research conducted to support development of the DoD Systems Engineering Guide for SoS, Version 1.0 [DUSD 08]. This section is a synopsis of her presentation.

2.2.1 SoS Systems Engineering

In order to understand SoS architecture, it is useful to think about SoS system engineering first. An SoS is a set or arrangement of systems that results when independent and useful systems are integrated into a larger system that delivers unique capabilities. Further, there are different types of SoS, and these different types pose different issues. Table 3 shows these variations [DUSD 08, p. 5].

Table 3: Taxonomy of Systems of Systems[2]

Directed	SoS objectives, management, funding and authority in place; systems are subordinated to the SoS
Acknowledged	SoS objectives, management, funding and authority in place; systems retain their own management, funding and authority in parallel with the SoS
Collaborative	No objectives, management, authority, responsibility, or funding at the SoS level; systems voluntarily work together to address shared or common interest
Virtual	Like collaborative, but systems don't know about each other

In *Directed* and *Acknowledged* systems of systems, there is a deliberate attempt to create an SoS. The key difference is that in the former, the SoS exercises control over the constituent systems while in the latter, the constituent systems retain a high degree of autonomy in their own evolution. *Collaborative* and *Virtual* systems of systems are more ad hoc, absent an overarching authority or source of funding and, in the Virtual case, even absent the knowledge about the scope and membership of the SoS.

Looking at the many SoS projects within the DoD, the most prevalent form of SoS is the Acknowledged type[3]; hence, the DoD SoS Systems Engineering guide focuses there. The guide, drawing from pilot efforts with current practitioners, identifies the core elements of SoS systems

---

[2]   The taxonomy shown is an extension of work done by Mark Maier in 1998.

[3]   Most military systems are part of an SoS operationally, but it is only in rare exceptions that they are engineered and managed at an SoS level.

engineering and discusses the application of systems engineering processes to those core elements.

Within the DoD, Acknowledged SoS programs share some common characteristics. First, the systems are typically an ensemble of individual existing systems brought together to satisfy user capability needs. Second, they are typically not new acquisition efforts.[4] Third, the SoS manager does not control requirements or funding of the individual systems and therefore likely must rely on influencing skills rather than directing skills. In fact, the SoS manager and the SoS systems engineer may not even be aware of all the systems that may impact their objectives (and both the systems and the objectives may change over time). Fourth, the focus of the SoS is on evolution of capability over time, which is a necessary consequence of not being able to engineer the SoS from the ground up. Fifth, the top-down direction for an SoS capability is concurrent with the independent directions and autonomy in the operation and development of constituent systems, meaning that there are multiple levels of objectives, multiple management authorities with independent priorities, funding, and development plans, and multiple technical authorities. Finally, much of the SoS functionality is in the extant capabilities of the individual systems.

As one might imagine, the independent and concurrent management and funding authorities present in Acknowledged SoS cause significant management issues. In the DoD, a solid governance and management approach is seen as key for successful SoS programs. Yet, independent authorities (such as program managers and systems engineers) at the system level are unlikely to accept direction from an SoS systems engineer with no direct authority over the constituent systems. One argument often offered as a solution is to make an Acknowledged SoS into a Directed SoS, thereby vesting all authority and funding in the SoS manager. However, such an approach is made difficult by multi-mission systems, which are important to multiple systems of systems.

One can contrast these issues with Acknowledged SoS outside of the defense sector. These systems of systems are able to exist and evolve without top-down management. The individual systems or services are designed to be broadly useful and have as their business objective the support of numerous user applications. They naturally retain authority over decisions regarding their development and are not likely to agree to limit themselves to one specific customer. Given the differences between defense and commercial business models, this does not suggest that commercial approaches can directly apply to defense. However, it does suggest that there may be alternatives to top-down hierarchical control.

Table 4 summarizes the differences in the technical and management challenges posed by systems and Acknowledged systems of systems in the defense sector. From management and oversight, to operational environment, to implementation, and engineering and design considerations, the Acknowledged SoS presents significant challenges when compared to non-SoS systems. Consequently, systems engineering approaches at the SoS level must recognize and specifically address those challenges.

---

[4]   Interestingly, even a nominally Directed SoS like FCS, which is a new acquisition effort, still must integrate with legacy systems, making it, at best, a hybrid between Directed and Acknowledged.

Table 4:   Technical and Management Challenges, Systems vs. Acknowledged SoS [DUSD 08, p. 11]

MANAGEMENT & OVERSIGHT		
	System	**Acknowledged SoS**
Stakeholder Involvement	Clearer set of stakeholders	Stakeholders at both system level and SoS levels (including the system owners), with competing interests and priorities; in some cases, the system stakeholder has no vested interest in the SoS; all stakeholders may not be recognized.
Governance	Aligned Program Management (PM) and funding	Added levels of complexity due to management and funding for both the SoS and individual systems; SoS does not have authority over all the systems.
OPERATIONAL ENVIRONMENT		
	System	**Acknowledged SoS**
Operational Focus	Designed and developed to meet operational objectives	Called upon to meet a set of operational objectives using systems whose objectives may or may not align with the SoS objectives
IMPLEMENTATION		
	System	**Acknowledged SoS**
Acquisition	Aligned to ACAT Milestones, documented requirements, Software engineering (with a Systems Engineering (SE) Plan	Added complexity due to multiple system life cycles across acquisition programs, involving legacy systems, developmental systems, new developments, and technology insertion; typically have stated capability objectives upfront which may need to be translated
Test & Evaluation	Test and evaluation of the system is generally possible	Testing is more challenging due to the difficulty of synchronizing across multiple systems' life cycles; given the complexity of all the moving parts and potential for unintended consequences.
ENGINEERING & DESIGN CONSIDERATIONS		
	System	**Acknowledged SoS**
Boundaries and Interfaces	Focuses on boundaries and interfaces for the single system	Focus on identifying the systems that contribute to the SoS objectives and enabling the flow of data, control and functionality across the SoS while balancing needs of the systems
Performance & Behavior	Performance of the system to meet specified objectives	Performance across the SoS that satisfies SoS user capability needs while balancing needs of the systems

As illustrated in Figure 1, there are seven interrelated *core elements* of SoS systems engineering. The SoS systems engineer must be concerned with translating capability objectives into high-level requirements, which is a somewhat different activity than in traditional systems engineering in the sense that systems engineers typically begin with requirements rather than broad capabilities. For an Acknowledged SoS, objectives are often modulated by the practical considerations of the functions available in existing systems. Consequently, important to the SoS systems engineering role is the need to understand the underlying systems of the SoS and their relationships both to each other and to the overall SoS. Further, since the constituent systems in an Acknowledged SoS will evolve independently of the SoS, monitoring and assessing change becomes another core element of SoS systems engineering that is not prevalent in traditional systems engineering.

Other core elements that address implementation and evolution become important. These elements include addressing requirements and solution options within the SoS trade space, orches-

trating updates to the SoS as the constituent systems evolve, and continually assessing performance against SoS capability objectives.
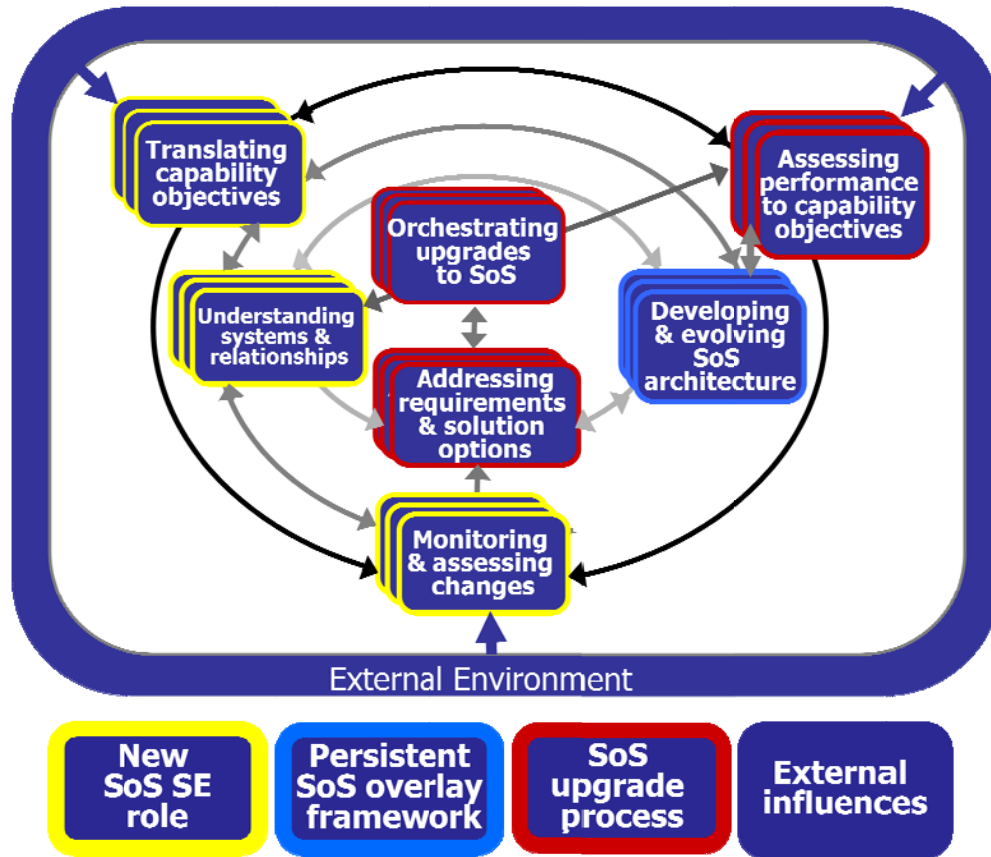
Figure 1:   Core Elements of SoS Systems Engineering [DUSD 08, p. 30]

Central to SoS systems engineering, though, is the notion of developing and evolving the SoS architecture. The SoS architecture depends heavily upon the translation of capability objectives into requirements as well as a solid understanding of the existing systems and their relationships, but it also informs those activities over time as the SoS and its constituent systems evolve. Likewise, the SoS architecture is essential to being able to assess the SoS performance against capability objectives and to addressing solution options into the future.

2.2.2   SoS Architecture

Not only is the SoS architecture a cornerstone of SoS systems engineering, it is also foundational to the SoS development, particularly for an Acknowledged SoS where the constituent systems are able to evolve independently of the SoS. The architecture of an SoS represents a persistent technical framework that guides the evolution of an SoS over time.

Developing and evolving an SoS architecture includes creating and maintaining the SoS concept of operations, determining the systems, their functions, and their relationships and dependencies (both internal and external to the SoS), and establishing an understanding of the end-to-end functionality, data flows, and communications within the SoS. A successful SoS architecture will pro-

vide the technical framework for assessing the options and the implications for meeting SoS requirements over time. It will have persistence and a tolerance for change.

That said, developing the SoS architecture is a non-trivial task. As with any architecture development effort, the task requires analysis and assessments of trades among different options. Such analysis may be supported by different approaches including modeling, simulation, and experimentation. However, the architecture of an SoS is somewhat constrained by the structure and content of the individual systems within it, and particularly by the extent to which changes in those systems are affordable and feasible, since the systems typically will need to continue to function in other settings in parallel with participation in the SoS.

The notion of persistence also presents challenges. Ideally, the architecture of an SoS will persist over multiple increments of SoS development (including increments of development of systems populating the SoS), allowing for change in some areas while also providing stability in others. This duality is a difficult balancing act to get right. Nevertheless, the ability to persist and provide a useful framework in light of changes is a core characteristic of a good architecture for an SoS. Over time, the SoS will face changes from a number of sources (e.g., capability objectives, actual user experience, changing concepts of operation [CONOPS] and technology, and unanticipated changes in constituent systems) that may all affect the viability of the SoS architecture and may call for changes. Consequently, the SoS systems engineer and architect must regularly make assessments of the architecture to ensure that it supports the SoS evolution.

In most cases, because the nature of an Acknowledged SoS is an overlay on multiple existing systems, the migration to an SoS architecture will be incremental. One example is the technical evolution of the information management architecture of the Air Force's Distributed Common Ground System (DCGS). The Air Force envisions a transition from the current "as is" state of a "net-enabled" DCGS, through a net-centric DCGS capability, and ultimately to the desired "to be" state of a net-centric intelligence/surveillance/reconnaissance (ISR) capability featuring DCGS by the middle of the next decade.[5]

In some situations, the evolution of an SoS begins by improving the way the SoS functions without making any explicit architecture changes. In an Acknowledged SoS, there is typically a *de facto* SoS architecture in place, often based on a Collaborative SoS. Most Acknowledged systems of systems build on or adapt this architecture to support SoS capability objectives. The *de facto* SoS architecture may not be sufficient for true evolutionary development; rather, from the SoS architecture as a starting point, the individual systems can be evolved in directions more conducive to the objective SoS, resulting in a second-generation SoS architecture that will be evolvable. As an example, the Air Force Air and Space Operations Center (AOC) Weapon System program is taking just such an approach, improving current systems by replacing their existing stove-piped, message-based processing with net-centric, service-oriented processing and then integrating those improved systems in a follow-up increment of the SoS architecture.[6]

---

[5]   More information is available at http://www.dtic.mil/ndia/2004interop/Tues/meiners.ppt.

[6]   More information is available at
      https://www.aocws.com/aoc/info_sharing_doc/aocws/AOCandWSI_Overview_prelogin.ppt

Either way, as the architecture of an SoS is employed to increase the capability of the SoS, it will evolve and mature over time through the result of technical reviews at the SoS level and due to its linkage to specific systems comprising the SoS.

Given the importance of the SoS architecture to the overall development of an SoS, it seems appropriate to ask how the DoDAF supports SoS architecture development. The current version, 1.5,[7] is "a three-volume set that inclusively covers the concept of the architecture framework, development of architecture descriptions, and management of architecture data" [DoD 07]. DoDAF v1.5, addresses the move toward net-centric warfare by applying essential net-centric concepts and recognizing advances in enabling technologies. The DoD has invested in the DoDAF and required that it be used in a variety of ways—for instance, as a mechanism to document relationships and design decisions—by DoD systems of record. As such, the DoDAF is a resource for SoS engineers.

2.2.3 Concluding Thoughts

In assessing what is working so far from the DoD perspective, some Acknowledged SoS systems engineering principles emerge:

- Organizational as well as technical perspectives must be addressed: broader set of considerations for trade space and technical planning.
- Only areas that are critical to the SoS should be in the domain of SoS systems engineers: Leave the rest to the systems engineers and architects of the individual systems.
- The technical management approach should reflect the need for transparency and trust, with focused active participation.
- SoS designs/architecture are best when they are open and loosely coupled:
    - They impinge on the existing systems as little as possible.
    - SoS designs/architecture should be extensible, flexible, and persistent over time.
- Continuous analysis that anticipates change is a key activity:
    - Design strategy and trades should be performed upfront and throughout the life of the SoS and should be based on robust understanding of internal and external sources of change.

Finally, there are some differences in the different types of SoS architecture that are worth mentioning. Table 5 summarizes these observations.

---

[7] At this writing, DoDAF v2.0 has not yet been officially released.

Table 5: Observations about Different Types of SoS Architecture

SoS Type	Architectural Considerations
Virtual	*Virtual SoS* architecture is limited to a small set of common agreements, *a la* the Internet.
Collaborative	*Collaborative SoS* may or may not have an explicit architecture depending on the nature of the collaboration. *Collaborative SoS* looks like *Acknowledged SoS* if there is a high degree of shared interests and productive collaboration.
Acknowledged	(has qualities of other forms)
Directed	Directed SoS shares characteristics of Acknowledged SoS to the degree that the Directed SoS incorporates legacy systems. Directed SoS shares characteristics of system-level architecture if Directed SoS is a new development (rare).

Because of their broad scope, Virtual SoS architectures are typically characterized by a small set of interfaces and principles. A classic example is the Internet, where the core of the architecture specifies TCP/IP as the communication protocol but little else. In some cases, Collaborative SoS may not even have explicit architectures. It is interesting to note that they can begin to look like Acknowledged SoS if the levels of shared interest and collaboration are high (the discriminating factor would still be the presence or absence of an overarching authority for the SoS). Directed SoS also can look much like Acknowledged SoS if it must also account for legacy systems in their operation. The Army's FCS (Future Combat Systems) program is a prime example. FCS is a new acquisition of 14 interdependent systems being developed explicitly for the SoS. However, the fielded FCS SoS must also work together with other systems in the Army's Current Force. In the rare cases where a Directed SoS is purely a new development, its architecture can have many of the characteristics of a system-level architecture.

Thus, the lines of demarcation between different types of SoS are somewhat blurry. Nevertheless, it is important to understand the subtle differences, as it is these differences that will influence the architectural decisions made and, ultimately, the SoS systems engineering approach.

2.3 System Architecture—Mark Maier, System Architect/Engineer, The Aerospace Corporation

Mark Maier, Ph.D., Distinguished Engineer, The Aerospace Corporation, made the presentation for system architecture, titled "Systems Architecture: Key Concepts for Placement."

Mark began the presentation by providing his bottom-line thoughts on architecture, prior to providing some definitions. Architecture is a complex property of a thing, *any* thing. Whether that thing be a system, program, SoS, enterprise, or organization, architecture is the fundamental structure and is typically embodied in physical, logical, and programmatic structures. An architecture description is different from an architecture; there are different standards and purposes for each. An architecture framework serves as a standard for architecture descriptions. An architecture framework (as defined by TOGAF[8]) is a tool which can be used for developing a broad range of

---

[8]   TOGAF is The Open Group Architecture Framework.

different architectures. It should describe a method for designing an information system in terms of a set of building blocks and for showing how the building blocks fit together. It should contain a set of tools and provide a common vocabulary. It should also include a list of recommended standards and compliant products that can be used to implement the building blocks. Starting with the right concepts of architecture is key; confusing the basic concepts has bad effects.

Mark then introduced some terminology to support the remainder of the presentation:

- *System*—A collection of components organized to accomplish a specific function or set of functions (IEEE 610.12) [IEEE Standards Board 90]; a collection of components exhibiting emergent properties; big things and small things are systems; things with humans inside the boundary and all-machine things are systems

- *Architecture of a system*—A fundamental or unifying structure of a thing (Dictionary); the fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution (ANSI/IEEE Std 1471-2000) [IEEE 09]; a set of information that defines a system's value, cost, and risk sufficiently for the purposes of the system's sponsor (Maier's rule of thumb)

- *Architecture Description*—A collection of products to document an architecture (ANSI/IEEE Std 1471-2000) [IEEE 09]

- *Systems Architecting*—The activities of defining, maintaining, improving, and certifying proper implementation of the system architecture

Mark provided an emphasis that architecture is a structure, or a set of decisions. A discussion ensued about what is in scope for a system architecture. A system architecture includes hardware, firmware, software, people, and data components, as well as architecturally significant aspects of the range of engineering disciplines (e.g., mechanical, electrical, or chemical engineering) of the system.

Mark described the domain of architecting as being on a continuum and not having a hard boundary with engineering, as depicted in Table 6.

Table 6:   Characteristics of Architecture and Engineering

Characteristic	Architecting	Architecting & Engineering	Engineering
Situation/Goals	Ill-structured	Constrained	Understood
	Satisfaction	Compliance	Optimization
Methods	Heuristics	←——→	Equations
	Synthesis	←——→	Analysis
	Art and Science	Art **and** Science	**Science** and Art
Interfaces	Focus on "Mis-Fits"	Critical	Completeness
System Integrity Maintained Through	"Single Mind"	Clear Objectives	Disciplined Methodology and Process
Management Issues	Working for Client	Working with Client	Working for Builder
	Conceptualization and Certification	Whole Waterfall	Meeting Project Requirements
	Confidentiality	Conflict of Interest	Profit versus Cost

Some takeaways from the discussion on the domain of architecting continuum are that architects work for the client and with the builder; architects generate requirements as much as receive requirements; architects develop information in all of the views needed to make the client's decision; and architects write an architecture description as a consequence of the information developed to support the decision. Based on this discussion, Mark asked the question whether this is any different if the system is an SoS, a software, or an enterprise.

Mark then presented a slide on the classification of the problem-system space (depicted in Table 7 and Table 8). This turned out to be a very useful classification and the table was used in each genre break-out session. The shaded area represents the textbook (in Table 7) or classic (in Table 8) view.

Table 7:   Textbook Engineering Problem-Systems

	Simple	←——————————→		Complex
Sponsors	One, w/ $	Several, w/ $	One, w/o $	Many, w/o $
Users	Same as sponsors	Aligned with sponsor	Distinct from sponsor	Unknown
Technology	Low	Medium	High	Super-high
Feasibility	Easy	Barely		No
Control	Centralized	Distributed		Virtual
Situation-Objectives	Tame	Discoverable	Ill-structured	Wicked
Quality	Measureable	Semi-measureable		One-shot and unstable

Table 8: Real World "Classic" Systems Architecting Characteristics

	Simple	← →		Complex
Sponsors	One, w/ $	Several, w/ $	One, w/o $	Many, w/o $
Users	Same as sponsors	Aligned with sponsor	Distinct from sponsor	Unknown
Technology	Low	Medium	High	Super-high
Feasibility	Easy	Barely		No
Control	Centralized	Distributed		Virtual
Situation-Objectives	Tame	Discoverable	Ill-structured	Wicked
Quality	Measureable	Semi-measureable		One-shot and unstable

Using the case study of the DC-3 versus Boeing 247, Mark made the point that the architecture is a set of decisions and that value, cost, and risk are in these decisions, while the documents come later. Referring back to the classification tables, Mark stated that problem-systems that are completely in the "right column" (i.e., all complex) usually do not work.

Mark described the four-way tension that exists, which the architect must balance:

- **Organization**—Who's doing what? What are they good/bad at? What is their strategic identity?
- **System**—What are we building? What are the components? What are the key technical decisions? How is it tested?
- **Problem**—What are we doing? What delivers value? What is the environment? What is success?
- **Program**—How do we build/operate? (separation of responsibilities)

Mark finished the presentation with his concluding thoughts on system architecture:

- **We engineer (or architect) things (buildings, spacecraft, organizations)**—What thing(s) are we working on? If we don't know what the "thing" is, then we are unlikely to be effective at architecting it.
- **Architecture is the small set of attributes (structures, rules, protocols) that defines most of the value/cost/risk**—Architecture is about discerning the most important from the less important; Architecture is smaller than design.
- **Architecture descriptions should flow from the attributes needed to make a decision**—All other definitions (e.g., frameworks) can at best be just generic good practices; they can't define what an architecture is.
- **Architecture and architecture description standards are different and satisfy different objectives**—Don't confuse them.

2.4 Software Architecture—Mark Klein, Head, SEI Software Architecture Technology Initiative

Mark Klein, the head of the SEI's Software Architecture Technology Initiative, made the presentation for software architecture.

2.4.1 What is Software Architecture?

Pointing out that the Army depends on software to enable the warfighter to carry out his or her mission, Mark reminded us that the quality and longevity of a software-intensive system is largely determined by its architecture. Many large system and software failures point to inadequate software architecture education and practices and/or the lack of any real software architecture evaluation early in the life cycle.

Risk mitigation early in the life cycle has been shown to be a key to averting project failures. The software architecture is an early life-cycle artifact and perfectly poised to serve as an early life-cycle risk mitigation vehicle. In this way, mid-course correction is possible before great investment; risks don't become problems that have to be addressed during (for example) integration and test.

Mark presented the SEI's definition of software architecture:

> *The software architecture of a program or computing system is the structure or structures of the system, which comprise the software elements, the externally visible properties of those elements, and the relationships among them* [Bass 03].

The implications of this definition include

- Software architecture is an abstraction of a system.
- Software architecture defines the properties of elements.
- Systems can and do have many structures.
- Every software-intensive system has an architecture.
- Just having an architecture is different from having an architecture that is known to everyone.

These implications together suggest that if you don't develop an architecture, you will get one anyway—and you might not like what you get!

2.4.2 The Role of Software Architecture

If achievement of functionality were all that mattered in modern software-intensive systems, we would not need to worry about software architecture; a single mass of jumbled source code would be completely adequate. But functionality is often the least of our worries. We are usually at least as concerned with making the software easy to understand and change, letting it be developed by separate teams (perhaps in separate organizations), or making sure the user can operate it effectively. These and other quality attributes, such as performance, security, availability, and a host of others, pose the most challenging part of software design.

As software architecture has emerged as a field of study, it has become clear that one of its biggest contributions to software system development is its role as the primary carrier of quality attributes. Quality attributes such as performance or modifiability cannot be achieved without

making system-wide design decisions up front; qualities cannot be "stapled on" to an already existing software system any more than safety can be added to a car after it rolls off the assembly line.[9]

Quality attributes often reflect user needs, such as

- required capability
- low learning threshold
- ease of use
- predictable behavior
- dependable service
- timely response
- timely throughput
- protection from unintended intruders and viruses

But quality attributes also represent the needs of other stakeholders as well. For example, the customer organization may care deeply about how easy the system is to modify or evolve over its life cycle, how easy it is to integrate and test, or how different (even competing) organizations may cooperate to build different pieces that will go together seamlessly. Software architecture addresses all of these concerns, and more.

Software architectures "show up" in the architectures of the other genres discussed in this workshop. enterprise architecture and system architecture provide an environment in which software lives. Both provide requirements and constraints to which software architecture must adhere. Elements of both are likely to contain software architecture, but neither substitutes for or obviates a software architecture. In a large, complex, software-intensive system both software and system architectures are critical for ensuring that the system meets its business and mission goals. As for an SoS, each software-intensive system in an SoS has system and software architectures. The SoS has an architecture where the elements are themselves the software architectures of the individual systems. Thus, software architecture is even more important in an SoS context, not less.

2.4.3   Axioms

Mark presented three axioms about software architecture.
1. Software architecture is the bridge between business and mission goals and a software-intensive system.
2. Quality attribute requirements drive the software architecture design.
    – Quality attribute requirements stem from business and mission goals.
    – Key quality attributes need to be characterized in a system-specific way.
3. Software architecture drives software development throughout the life cycle.
    – Software architecture must be central to software development activities.
    – These activities must have an explicit focus on quality attributes.

---

9   Cars modified for world-class racing competition are an exception, but at considerable cost beyond the reach of normal consumers.

- These activities must directly involve stakeholders, not just the architecture team.
- The architecture must be descriptive and prescriptive.

Interestingly, these axioms generated a spirited discussion in the context of the other architecture genres. Participants suggested that software architecture is not the only bridge between business and mission goals and a system that meets those goals. For instance, a system architecture is just as important in the system realm, as is an SoS architecture in the SoS realm.

2.4.4   Architecture-Centric Activities

If architecture plays such a central role in software system development, what other activities does it inform? Architecture-centric activities include the following:

- creating the business case for the system

- understanding the requirements

- creating and/or selecting the architecture

- documenting and communicating the architecture

- analyzing or evaluating the architecture

- setting up the appropriate tests and measures against the architecture

- implementing the system based on the architecture

- ensuring that the implementation conforms to the architecture

- evolving the architecture so that it continues to meet business and mission goals

Mark concluded his presentation by discussing what DoD acquirers can do to achieve successful architectures. The steps included ensuring that

- Mission and business goals are used to explicitly identify and characterize key quality attributes.

- There is a chief architect and a competent architecture team.

- A software architecture is designed that satisfies constraints, meets functional requirements, and satisfies the key quality attributes.

- The software architecture is documented in multiple views.

- The software architecture is proactively evaluated and needed actions are taken.

- There are processes and tools in place to ensure development according to the architecture.

- The architecture and its documentation are evolved.

- The appropriate stakeholders are involved throughout.

- The impact of technology decisions is understood.

- Quality attributes are consistently addressed by the system and software architectures.

2.5 Summary of DoDAF Presentations

There were three presentations directly associated with the DoDAF, and these are described in the following sections. Some comments on the DoDAF based on sources external to the presentations are also included in the session descriptions for completeness.

2.5.1 DoDAF 1.5 – Fatma Dandashi, MITRE Corporation

The purpose of the DoDAF is to provide guidance for describing architectures to ensure a common denominator for understanding, comparing, and relating subject architectures across organizational boundaries.

The DoDAF has a relatively long history. It started as a Command, Control, Communications, Computers, Surveillance and Intelligence Architecture Framework (C4ISRAF), which defines four views (All, Operational, System, and Technical). Each view contains many products (a total of 22). The DoDAF v1.0 kept the same products as the C4ISRAF, but specifically considered how these products should be used by a variety of DoD stakeholders and contained a more uniform approach to product interrelationships.

The DoDAF v1.5 was released on April 23, 2007. It includes three volumes, namely, "Volume I: Definitions and Guidelines," "Volume II: Product Descriptions," and "Volume III: Architecture Data Descriptions" [DoD 07]. The DoDAF v1.5 aims at providing better support for architects in representing net-centric architectural constructs within various views and work products and at providing better support for decision making by program and portfolio managers. Presumably, v1.5 makes it easier to review and compare architecture descriptions to facilitate requirements and capability-based analysis and verification of systems and services interoperability. Architectural views are stored in an integrated architecture model repository that includes:

- operational nodes and operational activities
- information exchanges
- systems functions and services
- systems and service interfaces
- systems and service data exchanges

Deliverables for v1.5 consist of an integrated architecture visual model which is made up of a set of views that are mandated by various joint standards and DoD instructions using industry modeling standards such as Integration Definition for Function Modeling (IDEF0) and Unified Modeling Language (UML). The model is organized by views and products.

The intent is to use the information contained in the architecture model to document technical standards, interoperability requirements, and Net-Ready Key Performance Parameters for an existing or planned system (or a family of systems).

During the workshop presentation, a number of shortcomings were noted regarding the DoDAF v1.5 along with recommendations that were provided to the DoDAF v2.0 development committee, including

- The DoDAF should be expanded to include SoS architecture modeling and beyond to include the wide range of Doctrine, Organization, Training, Materiel, Leadership and Education, Personnel and Facilities (DOTMLPF) systems.
- The DoDAF should allow for modeling business/mission goals and program information and relating that information to system solutions.
- The DoDAF should provide support for using an architecture model during capability integration, acquisition, portfolio management analysis, budget planning, securing funding, iden-

tifying capability needs, eliminating redundant systems, and identifying measures for attaining net-centricity.

- More guidance and support are needed regarding tailoring architecture views and work products for specific purposes and levels of abstraction.

- More user-friendly views are needed for specific types of decision makers.

- More support is needed at the enterprise and domain levels (à la Zachman).

The DoDAF v2.0 release was officially scheduled for November 1, 2008; it is described in more detail later in this document.

2.5.1.1 MODAF, UML, and SysML Descriptions

- The Ministry of Defence Architectural Framework (MODAF) extends the DoDAF by defining two further architectural views (each with a number of products) covering the strategic goals of the enterprise, and the people, processes and systems that deliver those goals. It also includes Capability Management and programmatic aspects such as project dependencies.

- The Unified Modeling Language (UML) is an Object Management Group (OMG) standard used to build a number of views of software architecture, and to define relationships between them. UML is used extensively throughout industry and is supported by a number of commercial toolsets.

- The Systems Modeling Language (SysML) is also an OMG standard, which includes a number of system concepts and representations, is being used in industry, and is supported by a number of commercial toolsets. It allows easy flow-down and traceability from architecture frameworks to system models.

2.5.1.2 Unified Profile for DoDAF and MODAF

The presentation also provided an overview of the Unified Profile for DoDAF and MODAF (UPDM). UPDM defines an industry standard representation for DoDAF- and MODAF-compliant enterprise architectures based on UML and SySML. UDPM results from U.S. DoD and UK MOD interests in leveraging commercial standards for their Military Architecture Framework Profiles (MAFP). The MAFP is based on a formal metamodel to promote architectural framework (e.g., DoDAF, MODAF, TOGAF, and NATO Architecture Framework [NAF]) and MAFP tool interoperability and understanding of profile requirements.

UPDM provides guidance on the products that make up DoDAF and MODAF deliverables. The UPDM approach treats work products as queries against a model or models that produce the needed view. Under this approach, the team derives work products from the model. Work products do not determine the model. Within the UML community, there are a variety of modeling approaches and styles each well suited to a particular set of applications, including SysML, UML with action language, or UML following a Unified Process workflow. UPDM remains neutral in these implementation questions and has been constructed to work with current UML profiles as well as any of those produced in the future.

The benefits of UPDM as an industry standard include:

- providing a common structure for comparison, reuse, and evolution
- enhancing the quality, productivity, and effectiveness associated with architecture and SoS modeling
- promoting architecture model reuse and maintainability
- improved tool interoperability and communications between stakeholders
- reduced training impacts due to different tool implementations and semantics

The request for proposal (RFP) for UPDM was issued by OMG in September 2005. The scope of the RFP included:

- using the DoDAF v1.5 as a baseline
- incorporating MODAF acquisition and strategic views
- supporting modeling of systems that include hardware, software, data, personnel, procedures, and facilities (i.e., SoS architectures)
- supporting service oriented architectures and net-centricity

Developing the UPDM has involved the DoD, MOD, NATO, many military contractors, and UML tool vendors.[10] A UPDM specification was submitted to the OMG at its September 2008 Technical Committee meeting, using OMG's fast-track Request for Comments adoption process. A vote to adopt it as an OMG specification is scheduled for December 2008. The membership of the UPDM Group comprises development tool vendors and defense industry contractors along with representatives of the key Government agency stakeholders—the DoD in the U.S. and the United Kingdom's MOD. Although the UPDM is still a work in progress, prototype toolsets have already been demonstrated by commercial vendors.

2.5.2   DoDAF 2.0—John Tieso, Associate, Booz Allen Hamilton

The DoDAF v2.0 is expected to provide further guidance on planning, developing, managing, maintaining, and governing architectures through a coherent semantic and structural metamodel. Version 2.0 will place greater emphasis on a data-centric approach that will allow the use of architecture by a wider variety of decision makers [Thomson Reuters 08].

Version 2.0 consists of three volumes which were developed following a spiral life-cycle model consisting of four spirals. Each spiral involved drafting, commenting on, and updating the various DoDAF volumes. Although it was not clear in the presentation how the volumes will be titled, the intended audience for each volume has been established as follows:

- Volume I is an overview for managers and leaders.
- Volume II is a technical volume for architects.
- Volume III is a physical exchange specification for data engineers and toolset vendors.

---

[10]   See http://updmgroup.org/index.htm.

Key features of the DoDAF v2.0 are as follows (with comparison to the DoDAF v1.5 noted where appropriate):

- a data-centric (v2.0) vs. product-centric (v1.5) approach to architecture
- architecture data kept in a metadata registry
- a wider range of example models
- work products and views clearly presented as examples, rather than requirements
- fit-for purpose development guidance provided to ensure that views will be developed only for stakeholders who actually need them
- better linkages to major departmental programs (e.g., JCIDS, DAS PM, SE)
- integrated data model for conceptual, logical, and physical exchange specifications
- backward compatibility with the DoDAF v1.0 and v1.5
- a DoDAF journal provided as an online reference in wiki format
- linkages to the Defense Architecture Registry (DARS) to register architectures for information sharing
- linkages to the Defense Metadata Registry for registration/discovery of reusable data
- no requirement to register data anywhere but at the DoD level
- MODAF and NAF updated, based on the DoDAF v2.0
- additional views (including some borrowed from MODAF and NAF). The DoDAF v2.0 views are
    − All—consistent with the DoDAF v1.5
    − Operational—consistent with the DoDAF v1.5
    − Services—almost equivalent to the DoDAF v1.5 Systems View
    − Systems—almost equivalent to the DoDAF v1.5 Systems View
    − Standards—equivalent to the DoDAF v1.5 Technical View
    − Data and Information—new
    − Capability—new
    − System Engineering—new
- products now called models
- cross-cutting quality attributes placed/found in the Capabilities section of the DoDAF
- conceptual model describes data in 13 metamodel groups:
    − Performer
    − Resource Flow
    − Data & Information
    − Doctrine
    − Training/Skills/Education
    − Capability
    − Services
    − Project
    − Goals
    − Rules
    − Measures

- – Location
- – Activity

- mapping between each metamodel group and the views, models, and core processes associated with the group

- four tiers of responsibilities
  - – Enterprise—enterprise architecture
  - – Segments—joint capabilities—segment architectures
  - – Components—component architectures
  - – Solution—solution architectures

- user defined names allowed (as opposed to having to go to the XML registry to find the official name for something)

2.5.3 Examining the DoDAF from the Perspective of ISO/IEC 42010:2007—David Emery, DSCI

Dave Emery framed his talk by showing some overlapping and inconsistent statements about architecture, which demonstrated that there are conflicting opinions about: what architecture is and how it should be derived, represented, and used. He also described some "tough nuts" to crack, and they are listed below:

- building instances of a product family versus building a single system

- hazards/Faults/Errors

- fault tolerance/reconfigurations

- performance (including quality of service [QoS])

- data behavior (e.g. quality, timeliness, ownership, create-read-update-delete [CRUD])

- modeling interface boundaries (e.g., layers, strict versus loose)

- deciding, and deferring, allocation (e.g., software/hardware/wetware)

Dave then went through each of the tough nuts, outlining what made them difficult, including the reasons listed below:

- the types of architectural decisions that have to be made, the rationale for making a decision, and the implications of the decision, and the patterns available to the architect

- the lack of support of the products in the DoDAF v1.5 to document the basis of the decisions, and the resulting architecture, to sufficiently understand the impact of the decisions

- the fact that the DoDAF products are not integrated into the "systems architecture development processes," but are merely an after-the-fact way of documenting the results of these processes. In fact, they may even lead some systems engineers to focus on the DoDAF v1.5 products, and lure her away from a more coherent system engineering process.

Dave expressed concern that the DoDAF provides a limiting set of conventions for expressing architecturally significant concepts. Dave cited the Sapir-Whorf Hypothesis, which states that language constrains thought. Dave's application of this was that our inability to capture the tough nuts in current DoDAF modeling practices tended to limit, or even prevent, the architect's ability to understand and capture/render/describe many concepts that are significant architectural problems.

The DoDAF v1.5 does not comply with the ISO 42010/ANSI/IEEE 1471-2000 standard[11]; the DoDAF v2.0 should bring itself into alignment with this standard, by doing the following:

- align the DoDAF 2.x with the standardized terms in ISO 42010
- add *generic stakeholders* and explicit *concerns* for each DoDAF viewpoint
- identify a specific product to carry rationale (perhaps SV-3?)
- explicitly identify correspondences between models

---

[11]   ISO/IEC 42010 is the same as ANSI/IEEE Std 1471-2000 [IEEE 09].

3 Working Group Summaries

3.1 Overview

Each working group was assigned one of the architecture genres and was asked to discuss the following questions from the perspective of that genre:

1. What are the major activities involved in each genre? Some examples include understanding the architecturally significant requirements, designing, documenting, evaluating, checking for conformance, governance, sustainment, and infrastructure alignment. Do the genres share an overarching set of activities?

2. What is the boundary (e.g., information flow) between architecture in one genre and architectures in the other genres?

3. What do architectures in each genre need to address in order to be considered successful? For software architectures, the "usual" quality attributes include performance, security, modifiability, etc. For system architectures, concerns include the above plus performance, size and weight, power consumptions, etc. What qualities should be considered for the other types of architectures? What is the relationship to business and mission goals? How can these architectures be evaluated or analyzed?

4. How do we document an architecture in each genre? What notations and approaches are available? What are the minimum views and information necessary to ensure the architecture documentation will be adequate to support development and to conduct an evaluation as part of an acquisition?

5. How can the DoDAF be used to represent an architecture in each genre? What are its strengths and weaknesses with respect to each genre? How could it be improved? What is the current state of the practice with respect to using DoDAF with each genre?

In addition, each working group was invited to discuss three additional topics that arose during the overview presentations:

1. Mark Maier's characterizing tables for architecture and engineering (see Section 2.2)

2. Mark Klein's three axioms about architecture (see Section 2.4.3)

3. Dave Emery's tough issues for architecture (see Section 2.5.3)

3.2 Enterprise Architecture Working Group

This section is a synopsis of the discussions and presentation of the enterprise architecture working group. Participants in this working group were

- John Klein
- Mark Klein
- Rolf Seigers
- Ron Vandiver
- Robert Wojcik

3.2.1 Question #1: What are the major activities involved in the enterprise architecture genre?

The group began by addressing the major activities involved in enterprise architecture. These included

- determining architecturally significant requirements
- designing, documenting, and evaluating the architecture
- checking implementation for conformance to the architecture
- governance of the evolution of the architecture
- sustainment of the systems built using the architecture
- aligning the architecture with other infrastructure decisions within the enterprise

At first glance, these activities could be interpreted as those essential to architecture at any level. However, the group decided that there are some distinguishing characteristics of these activities for the enterprise architecture scope.

The first distinguishing characteristic of enterprise architecture is a focus on governing the evolution of the architecture, which usually begins by modeling the "as-is" current state architecture. There is value in doing this: it provides support to portfolio management and change management. The portfolio management activities, such as license consolidation and platform standardization, can produce significant cost savings for the enterprise. This "low hanging fruit" is frequently the starting point for enterprise architecture adoption. Understanding the current state is also critical for the change management of any architecture evolution.

After understanding the current state, the enterprise architecture must then define a future state that is aligned with the enterprise business/mission goals. Enterprise architecture then can be described as the activities and work products that define the current state, a desired future state, and the plan for evolving from the current to future state.

The Army wants to understand the necessary future capabilities and how to embed those capabilities into the enterprise over time. There is currently no end-state model, just incremental modifications to the existing enterprise architecture. The Army's enterprise architecture roadmap extends out to 2024.

A second distinguishing characteristic of enterprise architecture is scale, which cuts across all of the architecture activities and leads to the need to consider the tooling used to describe the architecture. This topic is discussed in Section 3.2.5, in which the group considered the appropriateness of the DoDAF for describing enterprise architecture.

3.2.2 Question #2: What is the boundary (e.g., information flow) between architecture in the enterprise architecture genre and architectures in the other genres?

The group spent much of its discussion time on this question.

In order to characterize enterprise architecture, the group first needed to define *enterprise*. The DoDAF v2.0 is expected to define it as a collection of organizations (e.g., military organizations,

departments, or business lines) that has a common set of goals and/or a single bottom line. The organizations in an enterprise are linked by a common managed purpose.

A simpler definition of enterprise is a collection of people brought together to try to achieve some purpose. The group found that recurring elements in definitions of enterprise were *people* and *common goal or purpose*.

It is important to separate the tasks performed by the enterprise from the enterprise goals. In many cases, the tasks performed by an enterprise do not change over time. For example, since the time of George Washington, field commanders in the Army have needed to perform tasks such as: collect and analyze intelligence information; maneuver the force; target and provide fire support; conduct mobility and counter-mobility operations; and address logistics to supply and resupply stocks and maintain equipment. The technology to support these tasks changes, but the tasks themselves have not really changed.

In contrast to enterprise tasks, enterprise goals define the quality attributes for task performance and are heavily influenced by current technology capabilities. These goals then drive the enterprise architecture. software architectures, system architectures, and SoS architectures are constrained by the enterprise architecture—and at the same time are enablers of it.

The working group tested some of these assertions and hypotheses by looking at a specific enterprise architecture that was developed for the National Oceanic and Atmospheric Administration (NOAA). This enterprise architecture includes a business strategy that contains a collection of key questions that the architecture must be able to answer, a collection of success factors for the enterprise flowed down to the architecture, and goals and objectives related to the success factors. The NOAA enterprise architecture was organized around "Observing Systems," an abstraction which includes environmental phenomena, environmental parameters, measurements, sensing elements, physical systems, and higher level systems.

The discussion of the NOAA enterprise architecture highlighted that dimensions of scope, scale, and focus distinguish enterprise architecture from SoS architecture, system architecture, and software architecture. In particular, the questions that enterprise architects answer using the enterprise architecture come from executives and business strategists and are framed in terms of enterprise goals. The answers that enterprise architects provide are often expressed using the same units that the organization uses to measure its bottom line (i.e., in dollars). Typical questions will involve decisions on investment, change, cost reduction, or pricing to customers. These questions also seem to highlight the enterprise architecture concern of evolution, breaking down into questions about the enterprise as it exists today and how to achieve goals that the enterprise is not meeting today.

Finally, these questions drive the documentation of the architecture. The models and artifacts that make up the enterprise architecture are selected and designed to facilitate answering these questions.

3.2.3   Question #3: What do architectures in the enterprise architecture genre need to address in order to be considered successful?

As noted previously, enterprise architecture is the genre that is most closely related to the enterprise's business/mission goals. The architecture must describe the current processes and workflows that the enterprise uses to achieve business/mission goals and must support quantifying the extent to which the enterprise is achieving those goals.

Next, the architecture must have evolution options that allow the enterprise to change processes and workflows to meet new business/mission goals. This implies that the enterprise architect must understand the range of strategic options available to the enterprise and accommodate likely changes in the architecture.

Finally, enterprise architecture evolution must be *resource informed*—that is, the evolution plan must fit within the funding, people, expertise, and capital constraints of the enterprise.

3.2.4   Question #4: How do we document an architecture in the enterprise architecture genre?

There is no "one size fits all" framework for developing and documenting an enterprise architecture. The enterprise architect must understand internal and external stakeholders and develop the artifacts necessary to answer the stakeholder's questions. Also, there is no *de facto* standard for notation.

Frameworks like the Zachman Framework offer a starting point for enterprise architecture development, and the Federal Enterprise Architecture (FEA) initiative offers guidelines for development, adoption, and institutionalization. Finally, the scale of enterprise architectures points to the need for tool support, which will in turn have an influence on the development and documentation of the architecture.

3.2.5   Question #5: How can the DoDAF be used to represent an architecture in the enterprise architecture genre?

The DoDAF v2.0 may be appropriate for documenting enterprise architectures. Its "fit for purpose" philosophy seems to help. There may be an opportunity to standardize lists of views (profiles) and representations to document each view, but the working group felt that more study was needed.

3.2.6   Conclusions

In summary, enterprise architectures add value by being

- resource informed: The enterprise architecture fits within the scope and constraints (funding and people) of the enterprise.
- outcome-based: There is clear linkage between the enterprise architecture and the business/mission goals of the enterprise.
- integration focused: The enterprise architecture integrates the processes and activities that make up the enterprise.

3.3 SoS Architecture Working Group

This section is a synopsis of discussions by the SoS architecture working group. Participants in the focus group included

- Stephen Blanchette Jr.
- Judith Dahmann
- Fatma Dandashi
- Steve Kishok
- Jeff Vermette
- Bill Wood

In addition to the original five questions posed by workshop organizers, the group addressed three questions that arose during the workshop presentations:

1. How does Mark Maier's chart apply?
2. How do Mark Klein's axioms apply?
3. How do David Emery's difficult problems apply?

Prior to tackling the assigned questions, the group engaged in some open-ended discussion about SoS and architecture in general, from the participants various viewpoints. The Army is implementing a new SoS engineering activity within the Office of the Assistant Secretary for Acquisition, Logistics, and Technology. This activity explicitly separates SoS systems engineering from architecture.

The group considered tackling the assigned questions from the perspective of a Directed SoS. The Army's FCS program is an example of a Directed SoS, and some of the group members have experience with that program and what it is doing regarding SoS architecture.

The group noted that architecture is often done after an initial decomposition of the problem into areas to be addressed by different suppliers. Unfortunately, such decomposition usually involves architecturally significant decisions, and those decisions are typically made by people who are not architects and who may not grasp the implications of their decisions to the ultimate technical solution. In effect, the *architecture of program* ends up being different from the *architecture of the product*.

Members observed that there are a lot of issues with respect to the use of the DoDAF for architecture development in any genre. The good news is that architecture work is being done. In many cases (but certainly not in all), programs are performing architecture tasks as part of their normal systems engineering efforts, but they are not using the DoDAF for architecture development. Rather, it seems a common practice is to develop DoDAF views to meet DoD requirements after the initial architecture work has been largely completed. Instead of an architecture development tool, the DoDAF often is used as an "after-the-fact" documentation tool.

SoS notions frequently affect the development of individual platforms. There is an increasing focus on mission-level articulation of SoS requirements to platforms; impacts on a system (and on platform acquisition/development) are driven by the fact that the system must participate in an SoS. The difficulty of SoS systems engineering is felt not just by the SoS developer but by the

developers of individual systems as well. Systems need to consider the impacts of their changes on other platforms/systems within the SoS, including other instances of the same system (different versions of a type of aircraft within the same air wing, for example).

In considering the various issues, it became clear that the group's discussions needed to be bounded. In addition, responses to the workshop questions might well vary depending upon the type of SoS under consideration. In order to maximally leverage the collective experience of the members, the group elected to answer the questions from the perspective of an Acknowledged SoS architecture. As a reminder from Section 2.2, an Acknowledged SoS has SoS objectives, management, funding, and authority, but it also must cope with the fact that its constituent systems retain their own management, funding, and authority in parallel with the SoS.

3.3.1 Question #1: What are the major activities involved in the SoS architecture genre?

There are several major activities involved in SoS architecture development; the ideas presented here are far from an all-encompassing list. The identified activities are in a roughly sequential order, but the focus group recognized that there will be overlap in practice.

One of the largest tasks is determining the capability objectives of the SoS. Further, once those objectives are known, the next challenge is decomposing them into a set of high-level SoS and system requirements. Some members of the group noted that decomposing capabilities is not a well-understood practice and should not be trivialized. Closely associated with determining capability objectives is determining the applicable measures of performance and measures of effectiveness (MOPs and MOEs[12]) for them. In some circles, MOPs and MOEs might be cast as quality attributes; terminology varies between architecture genres and even within one genre. In any case, these measures or attributes represent the non-functional qualities of the completed SoS that will ultimately determine its acceptability to users.

Another significant task in SoS architecture development is understanding the vignettes and associated mission threads that describe the dynamics of the SoS. Included in this understanding is the context or environment in which the SoS will operate as well as the CONOPS for the SoS. For example, a warfare or strike force vignette would provide insights into the breadth of SoS participation (e.g., single service, multi-service, or multi-national), the required robustness of communications and operations, and other types of architecturally significant aspects of the SoS. For an Acknowledged SoS, the context and CONOPS may be determined, in part, by the roles of the existing systems that will compose the SoS.

Armed with an understanding of the external influences, the SoS architect must make decisions about which functional elements within the SoS will meet the capability objectives. In an Acknowledged SoS, part of that answer will be derived from determining the existing or in-development systems that may contribute to these objectives and the functions that those systems provide. A given system may satisfy a capability objective outright while other objectives may be achievable only through some combination of functionality from two or more systems. The SoS

---

[12]   MOP is measure of performance; MOE is measure of effectiveness.

architect must judge how well system functionalities align with SoS elements, especially the end-to-end flow that will be employed to meet the objectives.

As important as these technical considerations are, the architect must also understand to whom the systems belong as well as their positions in their relative development cycles. These non-technical considerations may have profound impacts on the SoS architecture, in some cases forcing alternate strategies (if, for example, a particular system is too immature to be relied upon to participate in the SoS in the near term).

Mundane, but no less significant, tasks for the SoS architect include determining the high-risk areas and how to analyze them. For example, the architect might employ prototyping or engineering development models to identify potential risk reductions for high-risk technical areas. Such models can also be used to hypothesize the architecture alternatives, refine architecture objectives, or bound the technical trade space.

Lastly, once all of the above considerations have been duly pondered, the SoS architect, like all architects, must develop and document the architecture.

3.3.2 Question #2: What is the boundary (e.g., information flow) between architecture in the SoS architecture genre and architectures in the other genres?

The boundaries of SoS architecture tend to be at the system architecture level at the low end and at the enterprise architecture level at the high end. At the SoS/system architecture boundary, the working group observed that the architecture for an Acknowledged SoS is an overlay on existing systems that have their own architectures. Trying to force SoS architecture elements down into the systems' architectures is unlikely to lead to success; system architectures are best left to the systems engineering and architecture professionals at that level.

Organizational relationships between the SoS and system programs[13] are particularly important for an Acknowledged SoS, because of the need to address conflicting goals over the life of the SoS. There needs to be a mechanism in place to address situations where the SoS needs are counter to the needs of the system(s), and vice versa. Such problems are compounded when a given system is part of multiple systems of systems.

At the higher level, an SoS exists within one or more enterprises. Consequently, an SoS must address the tenets or constraints of the respective enterprises' architectures. An SoS that assists in planning a joint strike force with coordinated capabilities (an enterprise activity) provides a context for the SoS within the DoD enterprise. An example of the multiple enterprise case might be an SoS that crosses the Intelligence and DoD spheres of influence, each of which has a different mindset towards data sharing.

Regardless of the boundary being considered, it is important to explicitly address the interface points for contributions to and from (as appropriate) the SoS.

---

[13] Here, we are differentiating between systems, which are materiel solutions, and programs, which are the organizations that produce those solutions.

3.3.3 Question #3: What do architectures in the SoS architecture genre need to address in order to be considered successful?

Given the heavy dependence of SoS on their constituent systems, the characteristics of successful Acknowledged SoS architectures are significantly influenced by the relationships between the systems and the SoS. Recognizing the independence of the systems and allowing them as much autonomy as possible is a vital success characteristic. Said another way, the SoS architecture must not impose itself on the systems unduly; only elements essential to the SoS should be flowed down.

Resilience, especially to asynchronous development and deployment of features, is another key characteristic of a successful Acknowledged SoS architecture. Changes (in objectives, threats, technology, etc.) are inevitable for all useful systems; for an Acknowledged SoS, change may be unplanned and uncoordinated as the SoS and its systems evolve independently. A successful SoS architecture must be able to accommodate changes in its systems, while also limiting the impacts of change on other parts of the SoS. In addition, the SoS architecture must be able to cope with lack of change. There may be constraints on systems that are core to SoS functionality, so the architecture may need to accommodate those systems which cannot change.

Finally, a successful SoS architecture may need to establish a degree of understanding (akin to a service level agreement or SLA) between the SoS and its systems. Such agreements are most easily established in a Directed SoS with new development; the SoS architecture would provide guidance on key areas early in development, including an understanding of what the systems are responsible for and what the SoS will provide to the systems (e.g., data, service, power, refueling/re-supply, computing or communication infrastructure). For an Acknowledged SoS, the need is greater, but so is the difficulty of establishing agreements with systems that already exist independently.

3.3.4 Question #4: How do we document an architecture in the SoS architecture genre?

When it comes to documenting SoS architectures, the group members noted that in the DoD each SoS program takes a different approach. While some SoS programs produce architecture documents, many forego specific architecture documentation, opting instead to develop white papers with rationale on important aspects of the SoS architecture (such as performance, fault tolerance, or security). Still other programs have attempted to use integrated databases containing requirements, schedules, allocation responsibilities, budgets, and the like. Further, different methods and tools are used to analyze different aspects of an SoS architecture, complicating any effort to develop a consistent representation of an SoS architecture.

In general, the technical community is still learning about how to engineer an SoS, and in particular what the architectural concerns in an SoS context are. Hence, the approaches to architecture analysis and documentation in this genre are evolving. Commercial tools, while usually not specifically developed for use at an SoS level, are often applied and adapted to the needs of SoS architects. In addition, the DoDAF provides a useful way to depict some views of an SoS, but, as discussed in the next section, there are limitations to the DoDAF's usefulness.

3.3.5 Question #5: How can the DoDAF be used to represent an architecture in the SoS architecture genre?

The group noted that while the DoDAF provides a useful way to depict some views of an SoS architecture, it is not without some shortcomings. For instance, the extensions being developed for the DoDAF v2.0 do not appear to add much value for SoS use in describing combat management systems. Trying to expand the DoDAF to meet *all* uses dilutes its potential for effectiveness. Further, if some organizations want to use DoDAF views as reporting formats to demonstrate compliance with DoD guidance on architecture development, they should be allowed to do so. However, there is no need to make these views universal to all programs. It is not a good idea to interrupt the natural engineering and architecting process by focusing on specific views, regardless of their technical necessity for solving the problems at hand.

The DoDAF is limited in terms of its ability to support the analytic functions of architecture development, and much of this support is manual rather than automated. There are a number of commercial tools already available for doing different types of architecture analysis, and the group felt it was appropriate to use those tools rather than trying to force-fit the DoDAF into that role.

There is an issue with the composability of DoDAF products from an SoS perspective. One would like to be able to compose portions of the SoS architecture from portions of the individual systems' architectures. However, collecting architecture products from the individual systems in an SoS and fitting them together to support SoS architecture development does not work. The individual products will each have been produced using different languages and notations or (at best) different dialects or profiles of the same language and will not be amenable to whole-system understanding or analysis.

Lastly, given that the basic challenges of developing SoS and architectures are independent of the DoDAF, the tendency for architects and program managers to focus on developing DoDAF products to meet real or perceived mandates can be a distraction from the issues (and, more importantly, from the work).

3.3.6 Characterizing SoS Architecture

During the plenary session of the workshop, Mark Maier described his classification of system development problems along a continuum from simple systems to complex ones. Using this continuum, the focus group characterized the Acknowledged SoS architecture space.

Table 9 depicts the characterization, with the lightly shaded area (in the two middle columns) showing the most likely scenarios.

Table 9: A Classification of System Problems—Acknowledged SoS

	Simple	← →		Complex
Sponsors	One, with $	Several, with $	One, without $	Many, without $
Users	Same as sponsors	Aligned with sponsor	Distinct from sponsor	Unknown
Technology	Low	Medium	High	Super-high
Feasibility	Easy	Barely		No
Control	Centralized	Distributed		Virtual
Situation-Objectives	Tame	Discoverable	Ill-structured	Wicked
Quality	Measurable	Semi-measurable		One-shot and unstable

Key: Never | Perhaps | Usually

For the Acknowledged SoS, the most common occurrence is to have many sponsors, each with money and influence over the ultimate outcome. By definition, the Acknowledged SoS never has only one sponsor with sole responsibility or several sponsors without the money to influence outcomes. It certain rare cases, it may be possible for the Acknowledged SoS to have a single sponsor without money or authority.

It is never the case that the Acknowledged SoS has only one user or that there is an unknown number of users. Users typically represent a separate group from the sponsors, although the two groups may be closely related in some circumstances.

From a technology perspective, the Acknowledged SoS is usually of medium to high complexity, with the potential for few to be of either low or super-high complexity. Accordingly, the feasibility of developing such systems of systems typically is right on the edge of the possible. The nature of the Acknowledged SoS is such that it is never the case that it is trivial, nor is it ever the case that it is infeasible (at least not at the outset—otherwise, it could never be funded in the first place).

The Acknowledged SoS obviously exhibits distributed control; if it were subject to centralized control it would be the Directed SoS type. Similarly, virtual control would be a characteristic of a Virtual SoS.

The situation-objectives for a majority of Acknowledged systems of systems tend to be discoverable or ill-structured. It is possible for them to be tame in some select instances, although this observation does not suggest a "simple" problem space; Acknowledged systems of systems always have significant management challenges even if the technology aspects are relatively straightfor-

ward. It is possible, too, that the situation-objectives are *wicked*—that is, they are so complex and dynamic that change occurs at a rate that is faster than the SoS can be developed.[14]

Finally, in terms of quality, Acknowledged systems of systems tend to be semi-measurable (quality in this context includes testing). That is, they often have one or more aspects that cannot be effectively measured or tested, although they will also have others aspects that are measurable. The group also conceded that it is possible for Acknowledged systems of systems to be simple enough to be completely measurable or so complex as to be completely unmeasurable.

3.3.7   Applicability of SEI Software Architecture Axioms to SoS Architecture

Mark Klein presented the SEI's "software architecture axioms" during the opening plenary session and challenged attendees to determine how well, or poorly, they applied to the different architecture genres.

The first axiom states that *architecture is the bridge between business and mission goals and a software-intensive system*. The focus group felt that this axiom did not apply well in the SoS case. Rather than a bridge, the architecture for an Acknowledged SoS is more of a roadmap; it is essentially an overlay onto existing individual systems, one that must collaborate with the architectures of those systems.

The second axiom states that *quality attribute requirements drive architecture design*. The group agreed that this axiom applied to SoS architecture, although members felt that the language was not quite correct. Rather than quality attributes, the group suggested that the terms MOPs and MOEs would be more recognizable to architects working at the SoS level.

The third axiom states that *architecture drives software development through the life cycle*. In the case of the Acknowledged SoS, the architecture cannot drive the development of the SoS, because the SoS is dependent upon its constituent systems to deliver capability, and each of those systems evolves independently of the others and independently of the SoS. Instead, the group felt it would be more appropriate to say that the SoS architecture provides a framework for the incremental development of the SoS.

3.3.8   Hard Problems and SoS Architecture

Also during the plenary session of the workshop, David Emery presented his perspective on the hard problems found at every level of architecture:

- instances versus single system
- modeling of hazards/faults/errors
- modeling fault tolerance/ reconfigurations
- modeling performance (including quality of service, or QoS)

---

[14]   An often cited definition of a *wicked* problem is an ill-defined design and planning problem having incomplete, contradictory, and changing requirements. Solutions to wicked problems are often difficult to recognize because of complex interdependencies. This term was suggested by H. Rittel and M. Webber in the paper "Dilemmas in a General Theory of Planning," *Policy Sciences 4*: 155-169. Elsevier Scientific Publishing Company, Inc,, Amsterdam, 1973.

- modeling data behavior (e.g., quality, timeliness, ownership)

- modeling interface boundaries (e.g., layers, strict vs. loose)

- deciding—and deferring—allocation (e.g. software/hardware/wetware)

- coping with the effects of the Sapir-Whorf Hypothesis (and UML)[15]

The focus group agreed that these are, indeed, hard problems at the SoS architecture level and that there are no good solutions at this time.

3.4 System Architecture Working Group

This section is a synopsis of the system architecture focus group. Participants in the focus group included

- John Bergey

- Mike Gagliardi

- John Grove

- Mark Maier

- Shawn Rahmani

The group briefly discussed each person's background in the area of system architecture, reviewed the task for the working session, and then dove into answering each of the questions, starting with the first and progressing through each one to the end. We agreed that there is much experience in the field of system architecture and engineering and in some respects it is very mature in relation to the other genres; however, there are areas where much improvement can be made as well.

3.4.1 Question #1: What are the major activities involved in the system architecture genre?

The group addressed this question by listing out the major activities in the system architecting process and noting any issues, questions, or concerns relating to each activity.

The group first discussed what activities the government needed to support, before articulating the architect's activities. The government needs to support the architecting process by developing appropriate acquisition and program planning/management strategies, having the appropriate personnel on staff with the right tools, and putting the right contract in place to obtain the necessary data rights.

The architecting activities were then discussed by the group in the following order:

Determining and structuring the problem the system is to address. The group agreed that scoping was an important aspect of this activity and it was important to understand where the system scope encroached upon the scope of an enterprise architecture or SoS architecture. Requirements selection, decomposition, and negotiation were also an important part of this

---

[15]   In this context, the Sapir-Whorf hypothesis was explained as architects being limited in their thinking about problems by the languages and tools available to express their ideas.

activity. Non-functional requirements (quality attributes) captured as stakeholder-elicited scenarios seemed the best way to articulate some of the quality attribute expectations.

Forming system concept. Developing a CONOPS and a mission concept for the system is the responsibility of the government. Developing a system concept is a responsibility shared between the architect and the government. The architect must come up with an architecture development approach, which includes the appropriate metrics and associated costs and schedule estimation methods.

Development of system architecture. The group agreed that this is an area where the system architect has a tremendous amount of work to do, in relation to the other genres. The architect must

- account for component (e.g., hardware, software, data), make-buy decisions, commercial off-the-shelf (COTS), legacy, government-furnished equipment (GFE), etc.
- develop an appropriate reuse strategy to reap the benefits of component reuse while balancing the potential mismatch of reused component requirements (especially with quality attributes) and the system requirements
- account for expected system variant considerations
- support the flow-down and flow-up to a diverse set of disciplinary architectures and engineering considerations (e.g., mechanical, electrical, chemical)
- account for modeling, simulation, and analysis data in the process
- manage the development based on a sound risk reduction approach which includes prototyping, simulation, modeling, etc.

The risk reduction approach must first identify the high-risk areas of the system address; and the system architect must work closely with the software architect to ensure that the system and software architectures are consistent in the way they deal with functional requirements and quality attributes.

Documenting and communicating the system architecture. Currently, the state of the practice for system architecture documentation typically includes block diagrams, use cases, context diagrams and views from the DoDAF (sequence and event traces); value and objective models (text and graphs – value curves); and prototyping, simulation and analysis reports. The ability to capture and document the quality attribute requirements and show how the architecture supports them is an area that needs additional work. Also, the transition between system and software architecture views could use some attention.

Architecture evaluation. The group agreed that a scenario-based, quality attribute focused method with stakeholder participation (e.g., System & Software ATAM®) would be effective in identifying system architectural risks early in the development cycle.

® ATAM is registered in the U. S. Patent and Trademark Office by Carnegie Mellon University.

Incremental integration strategy. The group agreed that a strategy for incremental integration is needed for successful system integration. The question that was posed is, "How does the architecture accommodate this?" We could not come up with a satisfactory answer.

System integrity maintenance. It is too easy to document some architectural views early and then not update the architecture documentation when things change, which results in a mismatch between architecture and implementation/design. Equally frustrating is the practice of documenting the architecture post-implementation, which runs the risk of developing a system without documented guidance from the architect. There needs to be a way to check the conformance of a design/implementation to the architecture; this is an area that needs further development.

Assistance in validation for use. The group discussed the need for the system architecture to be validated for its use and the extent to which the architecture can assist in this validation. The main questions posed were "Is the system architecture fit for use and fit for purpose?" and "Can the system architecture be easily adapted to another use/purpose?" This area needs further development.

What is shared with the other genres? Enterprise architecture and SoS architecture typically have more long-term evolution and focus on integration, communications and interoperability, and standards adherence than systems. System and software architectures typically share some set of functional and quality attribute requirements.

3.4.2   Question #2: What is the boundary (e.g., information flow) between architecture in the system architecture genre and architectures in the other genres?

The group discussed this question at length. We grappled with the observations that if an SoS is directive about components, then it drives system architecture; however, in the case of using legacy system components, the legacy system architecture will drive the SoS architecture. Also, for a system that is in a product line environment, the software architecture will, in a sense, ride above the system architecture; at other times, the software architecture will be subsumed within the system architecture.

We agreed that unfortunately there is not a clean hierarchy between the genres although people want one. The community needs to define and manage the "real" relationships that exist. There is a basic element of commonality between the genres: purpose, mission, QAs, and so on. However, for legacy components, higher level architectures need to ensure that these are met. This area should be investigated and clearly defined.

3.4.3   Question #3: What do architectures in the system architecture genre need to consider in order to be considered successful?

This focus group agreed, first and foremost, that the architecture of that system doesn't matter if the system is not being used. This conclusion implies that, in any genre, it is critical to determine whether the system is delivering value with respect to the its mission. The architecture needs to support the validation of the mission goals and requirements and the assessment of options. There needs to be a way to capture mission goals and requirements and evaluate the architecture against

them early in the development cycle. A key measure of architectural success is mission fulfillment.

The group then discussed the architectural metrics needed to measure success. Perhaps metrics can be developed for the individual quality attributes. This area needs further development. When an architect is developing in an "uncertain requirements" environment, then an incremental architecture development approach is necessary. The group wondered why an architect wouldn't always want to develop the architecture incrementally. Cost is a major prohibiting factor in some cases.

We also discussed whether an architecture can be used to *advocate* for the system. Does advocacy have anything to do with success? We were not sure about this line of reasoning; we would want to examine some instances where an architecture was used to advocate for a system and look into its effectiveness in doing so.

We discussed the practice of competitive prototyping, which is typically capability driven. We discussed the extent to which a prototype should/must address "architectural" risks, perhaps in combination with capability prototyping. "Architectural prototyping" may be needed in some cases to address high-risk areas in the architecture, which would also require some investigation/analysis into the relationship between development costs and production costs.

3.4.4   Question #4: How do we document an architecture in the system architecture genre?

Currently, the state of system architecture documentation typically includes the following: block diagrams, use cases, context diagrams and views from the DoDAF (sequence and event traces); value and objective models (text and graphs—value curves); and prototyping, simulation and analysis reports. The ability to capture and document the quality attribute requirements and how the system architecture supports them is an area that needs additional work. Also, the transition between system and software architecture views also could use some attention.

The group discussed domain specific architecture techniques and patterns. Could they be developed and used in the system architecture process (e.g., command and control [C2], vehicle management, mission management, etc.)? We discussed the need for a high-level "market-tecture" (an architectural overview created to convey broad understanding quickly) to convey the appropriate messages at the executive level.

3.4.5   Question #5: How can the DoDAF be used to represent an architecture in the system architecture genre?

The group kicked off this discussion by wondering whether (1) it is too late to impact v2.0 and (2) the DoD has incorporated lessons learned from the use of v1.0 and v1.5. The DoDAF should reflect the architecture's ability to meet its requirements, especially the quality attributes. At a minimum, some placeholder to annotate needs should be linked to the view.

The group agreed that the DoDAF is a good basis for dialog about architecture. However, in practice it seems that the DoDAF is used as yet another post-design document tool, and there is very little support for analysis on the back end.

The group considered whether the DoDAF can be used to represent system architecture; the highlights of the discussion are captured in the following points:

- For all acquisition programs, you need additional materials and representations beyond those the DoDAF provides (e.g., cost models, discipline-specific models, quality attribute specifications, and schedule). The DoDAF does appear to not lend itself to back-end analysis.

- Using the DoDAF seems to add little to no value to the system development effort. There is not much advantage over other current practices.

- Supplying DoDAF views of an architecture can give a false impression of "architecting" being complete.

- The DoDAF does not help with software interfaces, representing user interfaces, or interface design (i.e., who controls what); layering abstractions are not easily done in DoDAF views.

- Cross-cutting quality attributes need representation within DoDAF views. System quality attributes are very important drivers in system architecture development.

- There is some concern that a lack of DoDAF expertise within the acquisition community is contributing to the problem.

3.4.6   Characterize System Architecture using Mark Maier's Table

The group discussed the characterization of system architecture using Mark Maier's table from his presentation, and we agreed upon the depiction in Table 8 on page 16, which is reprinted here.

	Simple	⟵————————————⟶		Complex
Sponsors	One, w/ $	Several, w/ $	One, w/o $	Many, w/o $
Users	Same as sponsors	Aligned with sponsor	Distinct from sponsor	Unknown
Technology	Low	Medium	High	Super-high
Feasibility	Easy	Barely		No
Control	Centralized	Distributed		Virtual
Situation-Objectives	Tame	Discoverable	Ill-structured	Wicked
Quality	Measureable	Semi-measureable		One-shot and unstable

We also considered some additional rows for system architecture, as follows:

- additional engineering disciplines that must be considered
- budget versus return on investment (ROI) versus schedule constrained
- size

3.4.7   Apply Mark Klein's Axioms to System Architecture

We discussed Mark Klein's axioms and their applicability to system architecture. We agreed that some simple modifications to the axioms are needed for applicability to system architecture. The modified axioms are as follows (Mark's original axioms are on page 18):

1.   Architecture is a bridge between mission/business goals and the system.

2. Quality attribute requirements drive architecture design.

3. Architecture drives system development throughout the life cycle.

3.4.8 Address Dave Emery's "Tough Nuts"

We decided to address Dave Emery's challenges by discussing them one at a time, starting with faults and data behavior. The discussion also touched on software and system engineering issues.

Faults

Understanding what can go wrong (fault model and recoveries unspecified at various levels) is critical. Typically, you cannot "shoe-horn" a comprehensive fault tolerance approach into an existing implementation. Rigor and completeness are needed, accompanied with supporting analysis.

Instrumentation and fault detection mechanisms are needed at various levels, as well as fault containment. A System Integration Lab architecture/implementation is needed to drive/test fault tolerance approaches.

Data Behavior

The data architecture is typically pushed down to the software architecture, yet the software architecture is not the right level to address system data architecture. Attention to data architecture at the appropriate levels is necessary. The association between data architecture and its quality attributes also needs to be developed and documented.

Some other observations were made during the discussion:

- Software architecture is not always addressed early enough in system architecture development.

- System architecture is not up to speed in a *layered abstraction* approach.

- System engineering sometimes does a poor job defining the right information to support software engineering activities.

3.4.9 Major Conclusions

The group spent some time developing major takeaways from the discussions and included some recommendations.

- A nice clean hierarchy does not exist between the genres; the community needs to recognize this and move on.
  - Clear, concise strategies are needed to delineate relationships and interfaces (e.g., cross-cutting parts and QAs, shared parts, etc.).
  - An iterative process to define the architectures is the best way to go (e.g., address software architecture at the enterprise architecture level, then iterate).
  - Recommendation: Develop a process to coordinate architecture development activities across Army programs and software engineering centers.

- The integrated architecting approach is missing and is needed. We need a way to bring the enterprise architecture/SoS/system/software architectures together (e.g., may be a combination of processes and frameworks).

- The fundamental test of architecting success is about mission and users. We need metrics to measure architecture success.

- The DoDAF is not sufficient to address many system architecture concerns. You can request to be a v2.0 reviewer, but it is not open to public review until later (which may be too late to influence v2.0). There are still some fundamental issues with respect to DoDAF use for system architecture.
  - Recommendation: ASA(ALT) forms a DoDAF v2.0 review committee to provide inputs to DoDAF v2.0 activities as soon as possible.

- The system architecting process does not usually account for software architecture concerns, sometimes pushing system architectural requirements down to software.
  - Recommendation: Revise the OSD's SoS System's Engineering Guide[16] (draft 2007) system engineering process to include software engineering considerations early in the life cycle. Can/does the CMMI® framework help?

- The acquisition cycle does not allow adequate time and resources for due diligence for the architecting process. The role that the DoDAF (and the analysis of the DoDAF) plays or should play in the acquisition cycle is unclear.
  - There is a need for DoD-level architecture standards, guidelines, education, definition of roles, authority, etc.

- There is a concern that we have stove-piped ourselves in this workshop. Perhaps a follow-on workshop can address this.

3.5 Software Architecture Working Group

The participants in this working group were

- Paul Clements
- Brad Drake
- David Emery
- Don O'Connell

The group began by formulating a hypothesis that attempted to find common ground among all of the genres. The hypothesis is this:

Architects (of any genre) need to understand the system, its environment, its stakeholders, their concerns, and the solution approach.

The solution approach varies across genres. In software architecture, the solution approach is software-oriented. For system architecture, the solution approach is hardware-oriented. For enterprise architecture, the solution approach is people- and business-oriented.

Next, before tackling the assigned questions, the group spent a little time trying to capture the relationships among the genres graphically. Figure 2 shows a tempting but ultimately unsuccessful approach. The group concluded that showing the genres in a nested depiction is not fruitful, because it is not at all clear what it means to show one genre contained (fully or partially) inside another. Nesting generally is used to mean "is part of" or "is a subset of" but neither of these relationships is satisfying in this case. What does it mean to say that software architecture is "part of" an enterprise architecture?[17] Enterprise architects are typically not concerned with software architecture at all (any more than software architects are concerns with source code implementations). They would be surprised to hear that an enterprise architecture "contains" a software architecture and would challenge you to look at an enterprise architecture and show it to them. You might at best show them a place it might occupy later, but that's not the same thing.
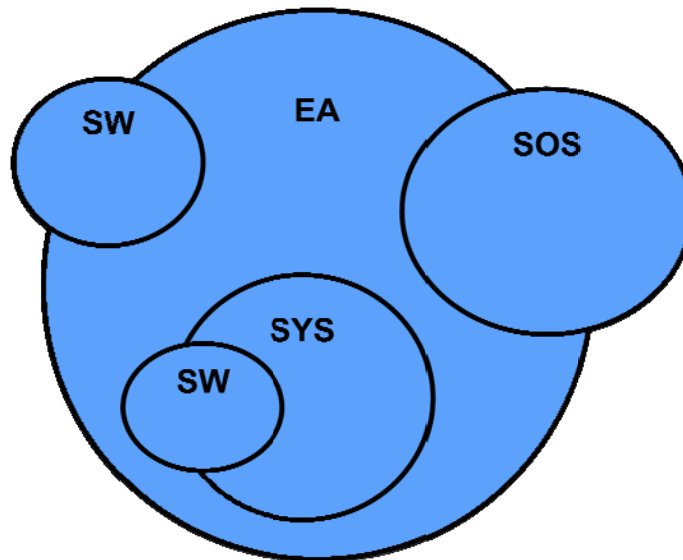
Figure 2:   An Unsuccessful Attempt to Depict the Genres

A more promising approach seems to be to describe the relations among the genres graphically by showing how one genre of architecture constrains or informs or influences another. Thus, a simple entity-relation diagram would seem to be appropriate. For example, the diagram would show that all genres of architecture are influenced by stakeholders' needs and concerns. It would show that

---

[17]   In Figure 2, EA stands for enterprise architecture, SOS for system-of-systems architecture, SYS for system architecture, and SW for software architecture.

system architecture heavily influences software architecture, that system architecture heavily influences SoS architecture in the case of an acknowledged SoS, and so forth.

After delving into these preliminary areas of discussion, the group began to prosecute the assigned questions.

3.5.1 Question #1: What are the major activities involved in the software architecture genre?

David Emery posited that any architecture is motivated by a known set of stakeholders, a known set of concerns, a set of mandatory "architecturally significant requirements," and a "vision" (something to use to make tradeoffs, to say how system might evolve).

These frame what the architect does to produce a set of viewpoints and then instantiate them to show the architecture meets the concerns and complies with the vision. So the activities could be articulated as the following:

- stakeholder and concern analysis
- requirements analysis to produce architecturally significant requirements
- figuring out a representation (viewpoints)
- filling in the views
- evaluating

Quality attributes emerge from the first three activities, which then set the stage for evaluation (by establishing the evaluation criteria).

The group also incorporated the list of architecture-centric activities from Mark Klein's software architecture presentation:

- creating the business case for the system
- understanding the requirements
- creating and/or selecting the architecture
- documenting and communicating the architecture
- analyzing or evaluating the architecture
- setting up the appropriate tests and measures against the architecture
- implementing the system based on the architecture
- ensuring that the implementation conforms to the architecture
- evolving the architecture so that it continues to meet business and mission goals

The group discussed establishing traceability between architecture and requirements as an important activity. The group felt certain that it applied to software, system, and SoS architectures, and could apply theoretically (if not in practice) to enterprise architecture as well. Traceability is often seen in Army programs in which allocation decisions explicitly flow down to and constrain lower-level programs. The new term in the Army for these higher-level concerns is *capabilities*; there are *capability managers* whose job it is to serve as advocates and shepherds for these broad areas.

On the other hand, some programs are just trying to integrate new and/or legacy software in a much more ad hoc fashion, and they cannot point to a trace to requirements that's very neat and clean. The trace, if reproducible at all, would include references to decades-old requirements that long ago stopped evolving.

Checking for conformance of downstream products was also an activity this group added to the list. For software architecture this means "Does the code conform to the architecture?" and "Does the code conform to the requirements?" Conformance of both kinds can be checked by holding a design review or peer review; conformance to requirements can also be checked by testing.

Finally, architecture maintenance made the list of important activities in this genre. When requirements change, resulting in an architecture change, do you have to change the architecture description? Of course, if a project doesn't require conformance to the architectural design to begin with, there will never be any forcing function to change it.

The group concluded its discussion of question #1 by stating the importance of having the software architect involved in requirements stage, RFP stage, etc., to start investigating tradeoffs at the earliest time possible. Often inadvertent architectural decisions are reflected in RFPs and initial system specifications before and after contract award.

3.5.2 Question #2: What is the boundary (e.g., information flow) between architecture in the software architecture genre and architectures in the other genres?

This group concentrated on the system/software boundary. Information flow across the boundary includes the areas of business and technical risk, the scope of the system and its boundaries, and the pedigree of the requirements such as what's a constraint, what's a hard requirement, and what's simply a desirable goal. Specific information needs depend on the system and its context.

The group felt that in most cases, the software architect is on the receiving end of performance and functionality requirements, and little else, from the system engineers. One way to bridge the boundary is to get software architects involved in all of the other stages.

Don O'Connell asserted that software architecture includes computers, networks, and storage—not their procurement but selection and utilization. This seems to be a clear boundary that system and software architecture share. If that's the case, which architect should carry the primary responsibility for these items? Figure 5 shows a graph of the possibilities. At the upper left end of the line, the system architect is given complete authority; at the lower right end, the software architect is in charge. Organizations cannot fail to choose a point along this line, although the group felt that in most organizations the choice is not made as a matter of explicit policy. Instead, organizations find a point on the graph as a matter of legacy history, the strength of the personalities involved, or the implicit leanings of management.

In any case, choosing the *right* point for an organization or system under development depends on the system, the complexity of the hardware to be chosen, and other factors. The right point at a business data processing organization might be quite different from the right point at an avionics company.
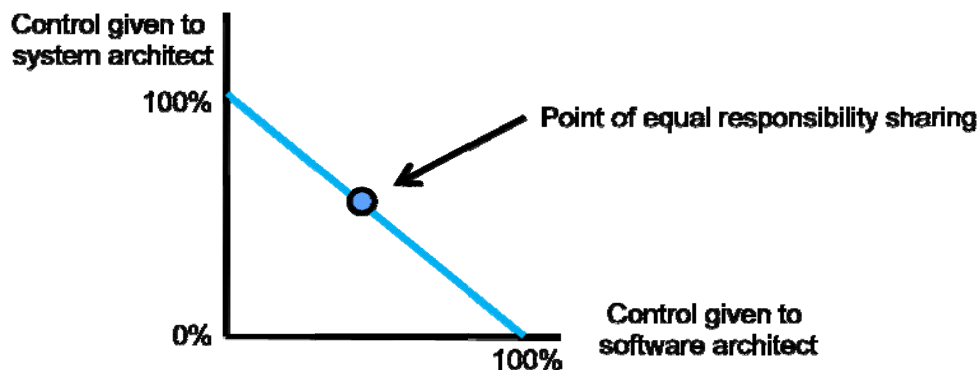
Figure 3:   Sharing Control between System and Software Architects

3.5.3   Question #3: What do architectures in the software architecture genre need to consider in order to be considered successful?

For software architecture, the answer to this question is the usual list of quality attributes. The group felt that "implementability" was a critical quality attribute that doesn't always make the list.

Don O'Connell volunteered that Boeing explicitly assesses for this. If, on a project, architectural guidelines come too late or are so complex that nobody knows how to use them without a new and dedicated training class, then the implementability of that architecture is likely to be low.

Implementability, then, is supported concerns such as understandability and, learnability, under the overall concern of meeting schedule and cost. All in all, successful architectures support the project: They finish on time (contributing to the project finishing on time), they support the schedule, and they don't change.

Don's model is that the software architecture should be complete when the project is 20% complete, and take 2-10% of overall software effort. The group felt that while the numbers (as well as the definition of "project complete") may vary from organization to organization and case to case, it was very helpful to adopt measurable benchmarks like this.

3.5.4   Question #4: How do we document an architecture in the software architecture genre?

For software architecture, the group identified the following approaches:

- Kruchten's (later Rational's) 4+1 Views approach [Kruchten 95]
- SEI Views and Beyond approach [Clements 02] or similar
- ANSI/IEEE Std 1471-2000 [IEEE 09]

In many cases, the documentation approach is specified by contract, sometimes including the DoDAF.

The group briefly discussed the usefulness of tailoring DoDAF v2.0 to come up with standard lists of views ("profiles") for use in particular circumstances.

For system architecture, approaches include documentation based on the "V" model, and generic system specifications with functional and performance requirements.

3.5.5 Question #5: How can the DoDAF be used to represent an architecture in the software architecture genre?

For software architecture, the group felt that the DoDAF v1.5 simply could not serve to represent software architecture. For example, the DoDAF provides no kind of module (i.e., build-time) view for software, and this kind of view is critical for project planning, designing modifiability, allocating work assignments, facilitating incremental development and system extensions, and a host of other critical purposes.

A more charitable thing to say is that the DoDAF is certainly not sufficient. Having said that, there are some DoDAF products[18] that are somewhat useful in representing software architectures. These include

- SV-5, which might be the starting point for a logical view.
- OV-2 and OV-3. Information exchange is covered.
- AV-1 and OV-1 provide contextual views, and those are useful for software.
- OV-7 and SV-11 (logical data model and implementation of the data model). These are vital to a software architecture, but it is far from clear how they can be specified before the software architecture is known.

The group felt that improving the DoDAF would involve nothing less than adding necessary architecture views for each genre. The DoDAF is essentially an information exchange framework. People should label their DoDAF-compliant document with the kind of architecture it's supposed to be capturing.

3.5.6 Conclusions

The group concluded by making the point that to bridge the gaps between the various genres, it is essential to get the right architectures involved at the right time throughout the life cycle. Stakeholders and their concerns need to be addressed in all genres.

As Figure 4 shows, software is becoming more vital to (and a larger part of) systems, systems of systems, and enterprises. As a result, software architecture is becoming more important in the satisfaction of requirements of systems, SoS, and enterprises.

---

[18]   In the list shown in Section 3.5.5, some DoDAF views are mentioned. They are as follows: SV-5 (Operational Activity to Systems Function Traceability Matrix), OV-2 (Operational Node Connectivity Description), OV-3 (Operational Information Exchange Matrix), AV-1 (Overview and Summary Information), OV-1 (Operational Concept), OV-7 (Logical Data Model), and SV-11 (Physical Schema).
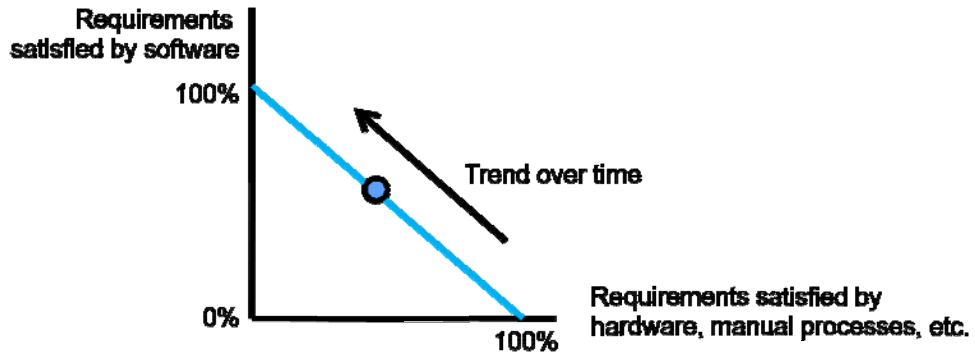
Figure 4:   Growing Importance of Software

4 Synthesis of Workshop Findings

This section summarizes the findings and positions taken by the working groups on each of the questions the workshop was designed to answer.

4.1 What are the Major Activities Involved in Each Genre?

Here we summarize the activities identified by each working group. A cursory examination suggests that the activities fall into four major categories:

1. understanding goals, context, and requirements
2. creating, evaluating, and documenting architecture
3. managing the architecture post-creation
4. assisting in post-architecture activities

The following summary is formed by taking the activities from each working group summary, usually verbatim, assigning them to one of the four categories, adding a prefix indicating its working of origin, and alphabetizing the result.[19] (Of course, if an activity is missing from a genre, it doesn't mean that the corresponding working group felt that it wasn't appropriate to that genre. The group simply may not have discussed it.)

1. Understanding goals, context, and requirements

- (EA) Aligning the architecture with other infrastructure decisions within the enterprise
- (EA) Defining architecturally significant requirements
- (EA) Modeling the "as-is" current state architecture
- (SoS) Decomposing objectives into a set of high-level SoS and system requirements
- (SoS) Determining the applicable measures of performance and measures of effectiveness (MOPs and MOEs) (e.g., quality attributes)
- (SoS) Determining the capability objectives of the SoS
- (SoS) Understanding the architecturally significant aspects of the SoS
- (SoS) Understanding the CONOPS for the SoS
- (SoS) Understanding the context or environment in which the SoS will operate
- (SoS) Understanding the vignettes and associated mission threads that describe the dynamics of the SoS
- (SoS) Understanding to whom the systems belong as well as their positions in their relative development cycles
- (SYS) Forming system concept; developing a CONOPS and a mission concept
- (SW) Creating the business case for the system
- (SW) Establish quality attribute requirements
- (SW) Requirements analysis to produce architecturally significant requirements (ASRs)
- (SW) Stakeholder and concern analysis
- (SW) Understanding the requirements

---

2.   **Creating, evaluating, and documenting architecture**

−   (EA) Defining a future state that is aligned with the enterprise business/mission goals
−   (EA) Designing, documenting, and evaluating the architecture
−   (SoS) Deciding which functional elements within the SoS will meet the capability objectives
−   (SoS) Developing and document the architecture
−   (SYS) Determining and structuring the problem the system is to address
−   (SYS) Developing system architecture
−   (SYS) Documenting and communicating the system architecture
−   (SYS) Evaluating architecture
−   (SW) Analyzing or evaluating the architecture
−   (SW) Creating and/or selecting the architecture
−   (SW) Documenting and communicating the architecture
−   (SW) Establishing traceability between architecture and requirements
−   (SW) Evaluating
−   (SW) Figuring out a representation (viewpoints)
−   (SW) Filling in the views

3.   **Managing the architecture post-creation**

−   (EA) Governance of the evolution of the architecture
−   (SYS) Assistance in validation for use
−   (SW) Architecture maintenance
−   (SW) Evolving the architecture so that it continues to meet business and mission goals

4.   **Assisting in post-architecture activities**

−   (EA) Checking implementation for conformance to the architecture
−   (EA) Sustainment of the systems built using the architecture
−   (SoS) Determining the high-risk activities and how to analyze them
−   (SYS) Incremental integration strategy
−   (SYS) System integrity maintenance
−   (SW) Checking for conformance of downstream artifacts
−   (SW) Ensuring that the implementation conforms to the architecture
−   (SW) Implementing the system based on the architecture
−   (SW) Setting up the appropriate tests and measures against the architecture

Interestingly, if not surprisingly, there is clearly a great deal of overlap in the activities associated with each of the genres. Creation, analysis or evaluation, and documentation are universal. Understanding the goals and requirements for the system being developed is universal, as is shepherding the architecture through its downstream usage and evolution.

4.2 What is the Boundary between the Genres?

Following is a summary of the main discussion points.

•   **EA:** The essential tasks of an enterprise change slowly over time; however, the supporting technology changes rapidly. Current technology capabilities have a significant impact on enterprise goals, which are then expressed as quality attributes that drive the enterprise archi-

tecture. Software architectures, system architectures, and SoS architectures are constrained by the enterprise architecture and at the same time are enablers of it.

- **SoS:** The boundaries of SoS architecture tend to be at the system architecture level at the low end and at the enterprise architecture level at the high end. The architecture for an Acknowledged SoS is an 'overlay' on existing systems that have their own architectures. Organizational relationships between the SoS and system programs are particularly important for Acknowledged SoS, because of the need to address conflicting goals over the life of the SoS. At the higher level, an SoS exists within one or more enterprises. Consequently, the SoS must address the tenets and/or constraints of the respective enterprise architectures.

- **SYS**: If an SoS is directive about its constituent systems, then SoS drives system architecture; however, in the case of using legacy system components, the legacy system architecture will drive the SoS architecture. Also, for a system that is in a product line environment, the software architecture will ride above the system architecture in a sense; at other times the software architecture will be subsumed within the system architecture.

- **SW**: The primary "interface" is the system/software boundary. Information flow here includes the areas of business and technical risk, the scope of the system and its boundaries, and the pedigree of the requirements: what's a constraint, what's a hard requirement, what's simply a desirable goal, and so on. Specific information needs depend on the system and its context. The working group on this genre felt that in most cases, the software architect is on the receiving end of performance and functionality requirements and little else.

Figure 5 shows the "primary" boundaries among the genres, as indicated by the working groups' discussions. An arrow from one genre to another means that the working group associated with the arrow's tail extensively discussed the relationship with the genre associated with the arrow's head.
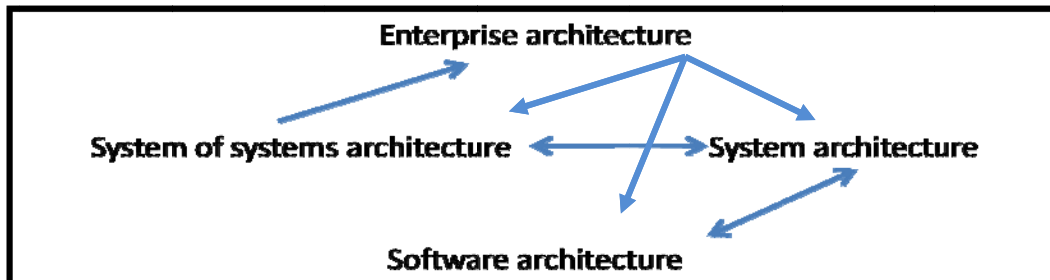

Figure 5:   *Primary Interfaces Across Genres, as Evidenced by Working Group Discussions*

4.3 What Do Architectures in each Genre Need to Consider in Order to be Considered Successful?

Table 10 summarizes the success criteria discussion across the genres.

Table 10: Success Criteria for Architecture Genres

Genre	Criteria for success
EA	Describe the current processes and workflows that the enterprise uses to achieve its business/mission goals and must support quantifying the extent to which the enterprise is achieving those goals The architecture must have evolution options that allow the enterprise to change processes and workflows to meet new business/mission goals. Evolution must be *resource informed*—that is, the evolution plan must fit within the funding, people, expertise, and capital constraints of the enterprise.
SoS	Allowing constituent systems as much autonomy as possible Ability to accommodate changes in systems, while limiting the impacts of change on other parts of the SoS Ability to accommodate those systems that cannot change Ability to establish a degree of understanding (e.g., an SLA) between the SoS and its systems
SYS	Ability to achieve mission success In an "uncertain requirements" environment, then incremental architecture development approach is necessary
SW	The usual list of quality attributes, such as performance, security, modifiability, etc., making sure to include implementability

4.4 How Do We document an Architecture in Each Genre?

Table 11 summarizes the discussion of documentation approaches across the genres.

Table 11: How Architectures are Captured or Documented

Genre	Approaches for capturing architecture
EA	No "one size fits all," no *de facto* standards Frameworks like Zachman offer a starting point; FEA offers guidelines for development, adoption, and institutionalization. Scale is important—tooling must support scale, and choice of tooling may impact documentation approach.
SoS	No standard approach. Some SoS programs produce architecture documents, but many forego specific architecture documentation, opting instead to develop white papers with rationale on important aspects of the SoS architecture (such as performance, fault tolerance, or security). Still other programs have attempted to use integrated databases containing requirements, schedules, allocation responsibilities, budgets, etc. Various commercial tools, while usually not specifically developed for use at an SoS level, are often applied and adapted to the needs of SoS architects.
SYS	Usual approaches include block diagrams, use cases, context diagrams and versions of the DoDAF (sequence and event traces); value and objective models (text and graphs—value curves); and prototyping, simulation and analysis reports. Capturing and documenting the quality attribute requirements and how the system architecture supports them is an area that needs additional work. The transition between system and software architecture views also could use some attention.
SW	Standard approaches include Kruchten's (later Rational's) 4+1 Views approach [Kruchten 95], SEI's Views and Beyond approach [Clements 02], and ANSI/IEEE Std 1471-2000 approach [IEEE 09]

4.5 How Can the DoDAF be Used

Table 12 summarizes the DoDAF discussion across the working groups. The discussion was based on v1.5 and the vision of what v2.0 will offer.

Table 12: Summary of DoDAF Discussion

Genre	DoDAF helps	DoDAF doesn't help
EA	"Fit for purpose" philosophy of 2.0 may help.	Standardization of required views and view representations may be helpful.
SoS	Useful for some views	Composability of views Automated analysis 2.0 doesn't appear to add value
SYS	Good basis for dialog about architecture	Tends to be used as post-design tool only Little support for analysis Cost models Discipline-specific models Quality attribute specifications Schedule Software interfaces User interfaces Interface design Layering abstractions Cross-cutting system quality attributes Poor support for back-end analysis Provides little to no value added compared to other current practices.
SW	SV-5: might be the starting point for a logical view. OV-2 and OV-3: information exchange is covered. AV-1 and OV-1: provide contextual views, and those are useful for software OV-7 and SV-11: logical data model and implementation of the data model	Module (build-time) views DoDAF not sufficient to represent software architecture

Here, there seems to be a clear consensus that the DoDAF is helpful in some areas, but is neither necessary nor sufficient to capture a high-quality rendition of an architecture in any genre.

5 Conclusions and Future Work

This workshop has confirmed what many architecture experts have come to believe: The various architectural genres enjoy more commonalities than differences. Nevertheless, each one has its own important knowledge base; one would not expect a skilled software architect, for example, to take over the job of an enterprise architect without specialized training and (preferably) enterprise architecture experience.

One recurring theme that arose in all of the working groups was the importance of openness among the various architectural tasks within an organization. Enterprise architects, SoS architects, system architects, and software architects should talk to one another more, rather than less, so that each may contribute his or her perspective on the development being undertaken.

There are several clear next steps that are possible for a follow-on workshop to tackle, should one occur. They include

- rearranging the working groups so that each working group tackles an important question across all genres
- using the activities lists summarized in Section 4.1 to create a core set of genre-independent architecture activities
- using the conclusions about the DoDAF summarized in Section 4.5 to inform a white paper of recommendations for the DoDAF v2.0.

A planned next step is to hold an ASSIP-sponsored workshop, open through ASSIP to the Army, to summarize the work and conclusions of this workshop and to invite comment and suggestions.

Appendix   Acronyms and Abbreviations

The following alphabetical list contains the acronyms, abbreviations, and their meanings as used in this report.

ACAT	Acquisition Category
ANSI	American National Standards Institute
AOC	Air and Space Operations Center
ARCIC	Army Capabilities Integration Center
ARDEC	Armament Research Development and Engineering Center
ASA(ALT)	Assistant Secretary of the Army for Acquisition, Logistics, and Technology
ASR	Architecturally significant requirement
ASSIP	Army Strategic Software Improvement Program
ATAM	Architecture Tradeoff Analysis Method
BC	Battle Command
C4ISRAF	Command, Control, Communications, Computers, Surveillance and Intelligence Architecture Framework
CIO/G-6	Chief Information Officer/G-6
CMMI	Capability Maturity Model Integration
CONOPS	Concept of Operations
CONUS	Contiguous United States
COTS	Commercial off-the-shelf
CRUD	Create, read, update, delete
DARS	Defense Architecture Registry
DAS	Defense and Space
DCGS	Distributed Common Ground System
DoD	Department of Defense
DoDAF	Department of Defense Architecture Framework
DOTMLPF	Doctrine, Organization, Training, Materiel, Leadership and Education, Personnel and Facilities
DSCI	D&S Consultants, Inc.
DUSD	Deputy Under Secretary of Defense
EA	Enterprise architecture
FCS	Future Combat Systems
FEA	Federal Enterprise Architecture
FEAF	Federal Enterprise Architecture Framework
GFE	Government-furnished equipment
IDEF0	Integration Definition for Function Modeling
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical & Electronics Engineers

ISO	International Organization for Standardization
ISR	Intelligence, Surveillance, Reconnaissance
JCIDS	Joint Capabilities Integration and Development System
MAFP	Military Architecture Framework Profiles
MOD	Ministry of Defence
MODAF	Ministry of Defence Architectural Framework
MOE	Measure of Effectiveness
MOP	Measure of Performance
NAF	NATO Architecture Framework
NATO	North Atlantic Treaty Organization
NOAA	National Oceanic and Atmospheric Administration
OMG	Object Management Group
OSD	Office of the Secretary of Defense
PEO	Program Executive Office
PM	Program Management
QoS	Quality of Service
RFP	Request for proposal
ROI	Return on investment
SE	Systems Engineering
SEI	Software Engineering Institute
SLA	Service Level Agreement
SoS	System of Systems
SW	Software
SYS	System
SySML	Systems Modeling Language
TCP/IP	Transmission Control Protocol/Internet Protocol
TOGAF	The Open Group Architecture Framework
TRADOC	Training & Doctrine Command
UML	Unified Modeling Language
UPDM	UML Profile for DoDAF and MODAF

References

URLs are valid as of the publication date of this document.

[Bass 03]

Bass, Len, Clements, Paul, & Kazman, Rick. *Software Architecture in Practice, Second Edition*. Addison-Wesley, 2003.

[Clements 02]

Clements, Paul, Bachmann, Felix, Bass, Len, Garlan, David, Ivers, James, Little, Reed, Nord, Robert, & Stafford, Judith. *Documenting Software Architectures: Views and Beyond*. Addison-Wesley, 2002.

[DoD 07]

Department of Defense. *DoD Architecture Framework, Version 1.5, Volume I: Definitions and Guidelines*. Washington, DC: DoD, 2007.
http://www.defenselink.mil/cio-nii/docs/DoDAF_Volume_I.pdf

[DUSD 08]

Office of the Deputy Under Secretary of Defense for Acquisition and Technology. *Systems and Software Engineering. Systems Engineering Guide for Systems of Systems, Version 1.0*. Washington, DC: ODUSD(A&T)SSE, 2008. http://www.acq.osd.mil/sse/docs/SE-Guide-for-SoS.pdf

[IEEE 09]

IEEE 1471. *Recommended Practice for Architectural Description of Software-Intensive Systems, ANSI.IEEE Std 1471::ISO/IEC 42010*. http://www.iso-architecture.org/ieee-1471/ (2009)

[IEEE Standards Board 90]

IEEE Standards Board. *IEEE Standard Glossary of Software Engineering Terminology*. IEEE, 1990.

[Lapkin 08]

Lapkin, Anne, Allega, Phillip, Burke, Brian, et al. *Gartner Clarifies the Definition of the Term Enterprise Architecture* (Gartner ID #G00156559). 12 August 2008.

[Kruchten 95]

Kruchten, Philippe. "Architectural Blueprints—The 4+1 View Model of Software Architecture." *IEEE Software 12*, 6 (November 1995): 42-50.

[Thomson Reuters 08]

"DoDAF 2.0 Saves Taxpayers' Money", Press release, Reuters, August 14, 2008.
http://www.reuters.com/article/pressRelease/idUS218608+14-Aug-2008+BW20080814

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE March 2009	3. REPORT TYPE AND DATES COVERED Final

4. TITLE AND SUBTITLE U.S. Army Workshop on Exploring Enterprise, System of Systems, System, and Software Architectures	5. FUNDING NUMBERS FA8721-05-C-0003

6. AUTHOR(S)

John Bergey, Stephen Blanchette, Jr., Paul Clements, Mike Gagliardi, John Klein, Rob Wojcik, Bill Wood

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213	8. PERFORMING ORGANIZATION REPORT NUMBER CMU/SEI-2009-TR-008

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) HQ ESC/XPK 5 Eglin Street Hanscom AFB, MA 01731-2116	10. SPONSORING/MONITORING AGENCY REPORT NUMBER ESC-TR-2009-008

11. SUPPLEMENTARY NOTES

12A DISTRIBUTION/AVAILABILITY STATEMENT Unclassified/Unlimited, DTIC, NTIS	12B DISTRIBUTION CODE

13. ABSTRACT (MAXIMUM 200 WORDS)

This report summarizes a U.S. Army workshop on architecture that was held at the Carnegie Mellon Software Engineering Institute (SEI) in September of 2008, under the auspices of the Army Strategic Software Improvement Program (ASSIP). The workshop organizers invited accomplished practitioners from government, academia, and industry to discuss the various "genres" of architecture: Enterprise Architecture, system of systems architecture, system architecture, and software architecture. The goal of the workshop was to clarify the relationships among the different genres, explore and identify areas of commonality and difference, and to discuss the role of the Department of Defense Architecture Framework (DoDAF) in helping to capture these architectures.

After a selection of opening talks by individuals that provide overviews of each subject area, the workshop dissolved into working groups. Each group was tasked with working on a specific set of issues and summarize their conclusions for the whole workshop. The issues discussed by each group include these:

What are the major activities involved in each genre?

1. What is the boundary (e.g., information flow) between architecture in one genre and architectures in the other genres?

2. What do architectures in each genre need to consider in order to be considered successful?

3. How do we capture (document) an architecture in each genre? What notations and approaches are available? What are the minimum views and information necessary to ensure the architecture documentation will be adequate to support development and to conduct an evaluation as part of an acquisition?

4. How can the DoDAF be used to represent an architecture in each genre? What are its strengths and weaknesses with respect to each genre? How could it be improved? What is the current state of the practice with respect to using the DoDAF with each genre?

This report summarizes the workshop and its findings.

14. SUBJECT TERMS Architecture, enterprise, system of systems, SoS, software	15. NUMBER OF PAGES 745

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. Z39-18 298-102