Development of an Exploratory Computer Model for Acquisition Management of System-of-systems

Shayani Ghose and Dr. Daniel DeLaurentis
School of Aeronautics and Astronautics, Purdue University, West Lafayette, IN
Email: ghose@purdue.edu, ddelaure@purdue.edu

Abstract. The Department of Defense (DoD) has placed a growing emphasis in recent years on the pursuit of agile capabilities via net-centric operations. In this setting, systems are increasingly required to interoperate along several dimensions. Yet, the manner in which components of these "system-of-systems" are acquired (designed, developed, tested and fielded) has not kept pace with the shifts in operational doctrine. Acquisition programs have struggled with complexities in both program management and engineering design. This paper presents a classification of underlying complexities in the acquisition of system-of-systems and describes a conceptual model that exposes the connectivity between systems, requirements, and externalities throughout the acquisition lifecycle. Implementing this model in an exploratory computer simulation highlights the relationship between requirement dependencies, risk profiles of the project and management parameters like span-of-control of SoS engineers and managers. The objective of the simulation (which remains under development) is to allow acquisition professionals to develop intuition for procuring and deploying system-of-systems, providing a venue for experimentation to develop insights that will underpin successful acquisition of SoS-oriented defense capabilities.

Introduction

A system-of-systems (SoS) consists of multiple, heterogeneous, distributed systems that can (and do) operate independently but can also assemble in networks and collaborate to achieve a goal. According to Maier (1998), the SoS typically demonstrate traits of operational and managerial independence, emergent behavior, evolutionary development and geographic distribution. Networks of component systems often form among a hierarchy of levels and evolve over time as systems are added to or removed from the SoS. However, these component systems are often developed outside of the context of their interactions with the future SoS. As a result, the systems may be unable to fully interact with the future SoS, adapt to any emergent behavior or be robust in the face of external disturbances.

The Future Combat System (FCS) program exemplifies a Department of Defense (DoD) acquisition process for an SoS. FCS seeks to modernize the US Army and provide soldiers with leading-edge technologies and capabilities allowing them to dominate in asymmetric ground warfare and to sustain themselves in remote places (U.S. Army). FCS has faced technical and management challenges that have come to typify acquisitions in SoS environments.

In 2003, the FCS program was comprised of an information network and 18 primary systems (categorized as manned ground systems, unmanned ground systems, and unmanned air vehicles). The Army's initial schedule allotted a 56-month system development and demonstration (SDD) phase [2003-2008] with the goal of achieving full operational capability by 2013. The Army's initial cost estimate was $108 billion (U.S. GAO 2003). Over the past four years, the FCS has been restructured twice in an effort to reduce the high risk attributed to the presence of immature technologies in critical paths and the challenges of concurrently developing these technologies with product development. The Government Accountability Office (GAO) criticized the Army's

acquisition strategy and concluded that the total cost for the FCS program had increased by 76 percent ($160.7 billion) from the Army's first estimate of $108 billion. However, independent estimates predicted an increase to $234 billion (116%).

In addition to the technical challenges, the FCS program also faced managerial challenges stemming from the Army's partnership with an industry Lead System Integrator (LSI). The role of the LSI is to reach across Army organizations to manage development of the SoS (U.S.GAO June 2007). Given the high risk involved in implementing a complex SoS, the GAO specifically underlined the importance of oversight challenges faced by the LSI in this area (U.S.GAO March 2007). The challenges of the FCS Program have pushed the Army to decrease the scope of the program to 14 systems and extend the time estimate for achieving full capability to 2030 instead of 2013.

Other non-DoD organizations are also struggling with systems integration of a collection of complex system. The US Coast Guard's (USCG) Integrated Deepwater System (IDS) is an example of a Department of Homeland Security (DHS) acquisition process for an SoS, that has also faced challenges stemming from the lack of collaboration between contractors and the marginal influence wielded by system integrators to compel decisions between them (U.S.GAO 2006). The NextGen Air Transportation System and the NASA Constellation program are also facing similar challenges in attempting to apply generic system engineering processes for acquisition in an SoS environment. Integration challenges faced by the Constellation Program are documented in a recent NRC report (Committee on System Integration for Project Constellation 2004). These examples possess the key drivers motivating the research described in this paper.

The overarching goal of our research is to understand the types of complexities present in acquisition management for SoS, and then to develop approaches that can increase the success of an acquisition process in the SoS setting. The three research questions derived from this goal are:

1. Is there a taxonomy by which one can *detect* classes of complexities in particular SoS applications?
2. What are the underlying systems engineering (SE) and program management functions that are affected?
3. How can exploratory modeling generate SE and acquisition management modifications to improve the probability of success?

In order to answer some of the questions posed, we aim to:

1. Identify the complexities in the acquisition of SoS based on historical trends of 'failures' especially in the context of the DoD
2. Develop a conceptual model of a generic acquisition process that is customizable to different SoS applications.
3. Develop a computational model based on the conceptual model and, through simulation, provide insight on and answer questions about process modifications.

Complexities

Simon (1996) and Bar-Yam (2003) define complexity as the amount of information necessary to describe a system effectively. In the context of a system-of-systems, the necessary information encompasses both the systems that comprise the SoS and their time-varying interactions with each other and the 'externalities.' Rouse (2007) summarized that the complexity of a system (or model of a system) is related to: the intentions with which one addresses the systems, the characteristics of the representation that appropriately accounts for the system's boundaries, architecture,

interconnections and information flows, the multiple representations of a system, all of which are simplifications; hence, complexity is inevitably underestimated, the context, multiple stakeholders, and objectives associated with the system's development, deployment and operation [Polzer et al.(2007) explored the issue of multiplicity of perspectives, where perspective is a system's version of operational context], the learning/ adaptation exhibited during the system's evolution.

Historical data from previous unsuccessful defense acquisition programs show a distinct correlation with the causes for complexity identified by Fowler (1994) points out some of the causes for the failure of the Defense Acquisition Process to be "over specification and an overly rigid approach on development," unreasonably detailed cost estimates of development and production, impractical schedules and extremely large bureaucratic overhead. Dr Pedro Rustan, director of advanced systems and technology at the National Reconnaissance Office, identified four specific shortcomings in the acquisition process for defense space systems: initial weapons performance requirements that are too detailed and lacking flexibility, insufficient flexibility in the budget process, a propensity to increase performance requirements in the middle of the acquisition cycle and demands to field entirely new spacecraft to meet new requirement (Spring 2005).

Using the above examples, we summarize the common causes of failure (Rouse 2007) within SoS acquisition processes as: a) *misalignment* of objectives among the systems, b) limited *span of control* of the SoS engineer on the component systems of the SoS, c) *evolution* of the SoS, d) *inflexibility* of the component system designs, e) *emergent behavior* revealing hidden dependencies within systems, f) *perceived complexity* of systems and g) the challenges in *system representation*.

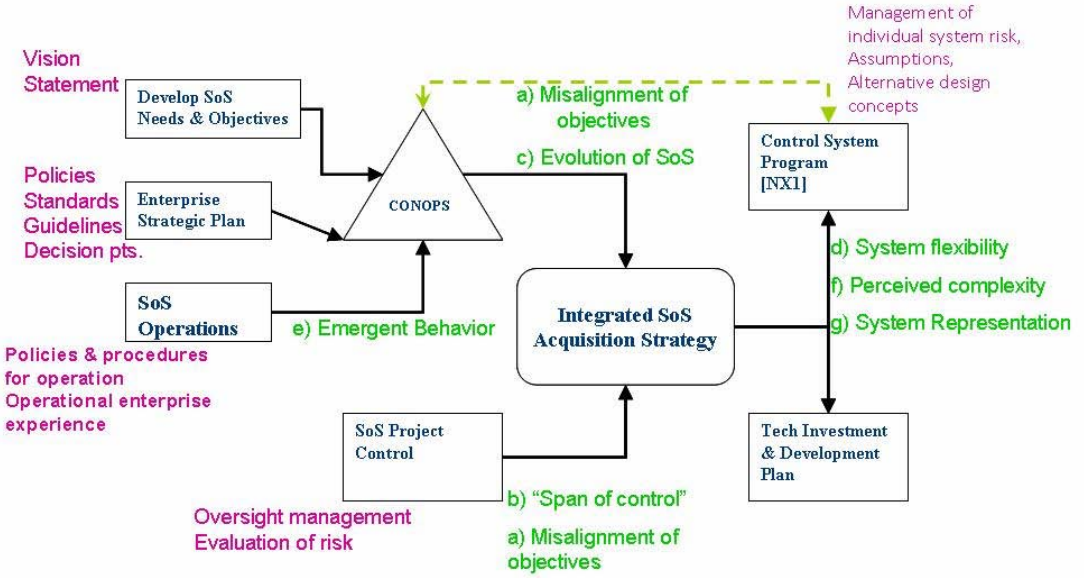


Figure 1: Complexities mapped to a section of the SoSE Process Model (Sage & Biemer 2007)

To provide context, we mapped these complexities to a System-of-systems Engineering (SoSE) Process Model designed specifically for SoS applications by Sage and Biemer (2007) (Fig 1). This mapping represents where complexities (green) might arise and how they may affect the acquisition process. Keywords to describe the functionalities of the various components in the model are indicated next to the component boxes (purple). For example: SoS operations could demonstrate emergent behavior and result in a change in the CONOPs for the SoS. Evolution of

the SoS changes the CONOPS of the SoS, resulting in a subsequent change in the Acquisition Strategy. Misalignment of objectives of the component systems in an SoS can arise from both the CONOPs as well as the SoS Project Control. System inflexibility, perceived complexities and challenges in representing systems occur mostly between or within systems. Accurate representation of component systems is complicated by the presence of both hidden and visible dependencies between systems, fuzzy boundaries, unknown architectures, etc.

Development of a Conceptual Model

Pre-Acquisition Model

We developed a pre-acquisition model to understand the impact of external stakeholders on the acquisition process. The model is based loosely on the Sage and Biemer (2007) SoSE Process Model and categorizes the external inputs to the SoS acquisition strategy model into 'Capabilities & Possibilities' (CAP), 'Technology Assessment, Development, Investment and Affordability Plan' (ADIA) and the funding received.(Ghose & DeLaurentis 2008) The CAP and the Technology ADIA Plan translate into technical requirements for the SoS. Provision of a computational model of the pre-acquisition activities is outside the scope of this paper. Instead, we focus on realizing a model for the acquisition strategy, described next.

Acquisition Strategy Model

Development of a 'brand new' SoS has been and will remain a rare occurrence. In their 2005 study on SoS, the United States Air Force (USAF) Scientific Advisory Board (Saunders 2005) stated that one of the challenges in building an SoS is accounting for contributions and constraints of legacy systems. These legacy systems may be used 'as-is' or may need re-engineering to fulfill needs of the new SoS. New systems are also incorporated to develop the capabilities of the SoS. Again, the new systems may range from off-the-shelf, plug-and-play products to custom-built systems dependent of the working of a legacy system.

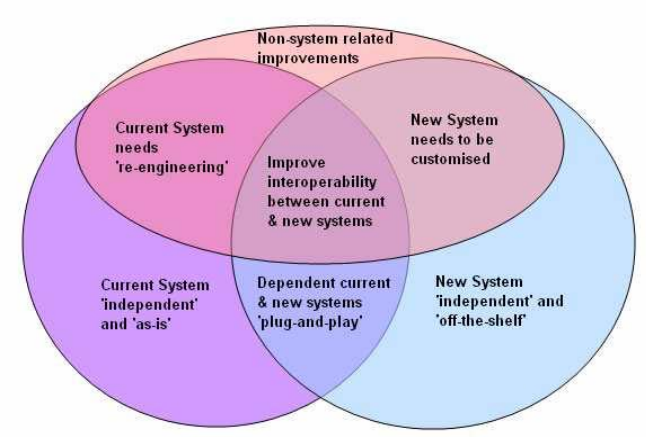


Figure 2 : Heterogeneity of component systems in an SoS

Sub-categories arise when the two or more categories overlap (Fig 2). For example: Improvements can be non-system related such as improvements in business practices for the SoS, or system-related such as re-engineering legacy systems or customizing/developing new systems to meet the needs of the SoS.

Implementing and integrating these different kinds of systems is made more complex by the evolutionary nature of an SoS. Though many systems may be dependent on others during the implementation or integration phases, the process to achieve this is often not centrally controlled. This requires that the individual systems' developers have an incentive to collaborate with each other. These issues are merely a sub-set of the challenges faced by an acquisition process in an SoS environment.

The conceptual model for acquisition strategy proposed in this section is based on the 16 basic technical management and technical system-engineering processes outlined in the Defense Acquisition Guidebook (U.S. DoD 2003), often referred to as the 5000-series guide. However, an SoS environment changes the way these processes are applied. The Systems Engineering Guide for System-of-Systems (SoS-SE) (U.S. DoD) addresses these considerations by modifying (in some cases revamping) some of the 16 processes in accord with an SoS environment. These new processes and their functions are described in Table 1. Our conceptual model for acquisition in an SoS environment (illustrated in Fig 3) is centered on these revised processes depicted in a hierarchy to show the flow of control between the processes throughout the acquisition life-cycle.

Table 1: Modified Technical Management and Technical Processes as described in the SoS-SE Guide (U.S. DoD 2003)

Requirements Development	Takes all inputs from relevant stakeholders and translates the inputs into technical requirements
Logical Analysis	Is the process of obtaining sets of logical solutions to improve the understanding of the defined requirements and the relationships among the requirements (e.g., functional, behavioral, temporal)
Design Solution	Process that translates the outputs of the Requirements Development and Logical Analysis processes into alternative design solutions and selects a final design solution.
Decision Analysis	Provide the basis for evaluating and selecting alternatives when decisions need to be made.
Implementation	The process that actually yields the lowest level system elements in the system hierarchy. The system element is made, bought or reused.
Integration	The process of incorporating the lower-level system elements into a high-level system element in the physical architecture.
Verification	Confirms that the system element meets the design-to or build-to specifications. It answers the question “Did you build it right?”
Validation	Answers the question of “Did you build the right thing?”
Transition	The process applied to move the end-item system to the user.
Technical Planning	Ensure that the systems engineering processes are applied properly throughout a system’s life cycle.
Technical Assessment	Activities measure technical progress and the effectiveness of plans and requirements.
Requirements Management	Provides traceability back to user-defined capabilities
Risk Management	To help ensure program cost, schedule and performance objectives are achieved at every stage in the life cycle and to communicate to all stakeholders the process for uncovering, determining the scope of, and managing program uncertainties.
Configuration Management	The application of sound business practices to establish and maintain consistency of a product’s attributes with its requirements and product configuration information.
Data Management	Address the handling of information necessary for or associated with product development and sustainment.
Interface Management	Ensures interface definition and compliance among the elements that compose the system, as well as with other systems with which the system or systems elements must interoperate.

As indicated in Fig 3, *Requirements Development* provides the technical requirements of the SoS, based on the relevant external inputs. The technical requirements are then sent to *Logical Analysis* to check for relationships amongst the requirements. This also helps check for inconsistencies amongst requirements and how that might affect the functioning and behavior of the future SoS.

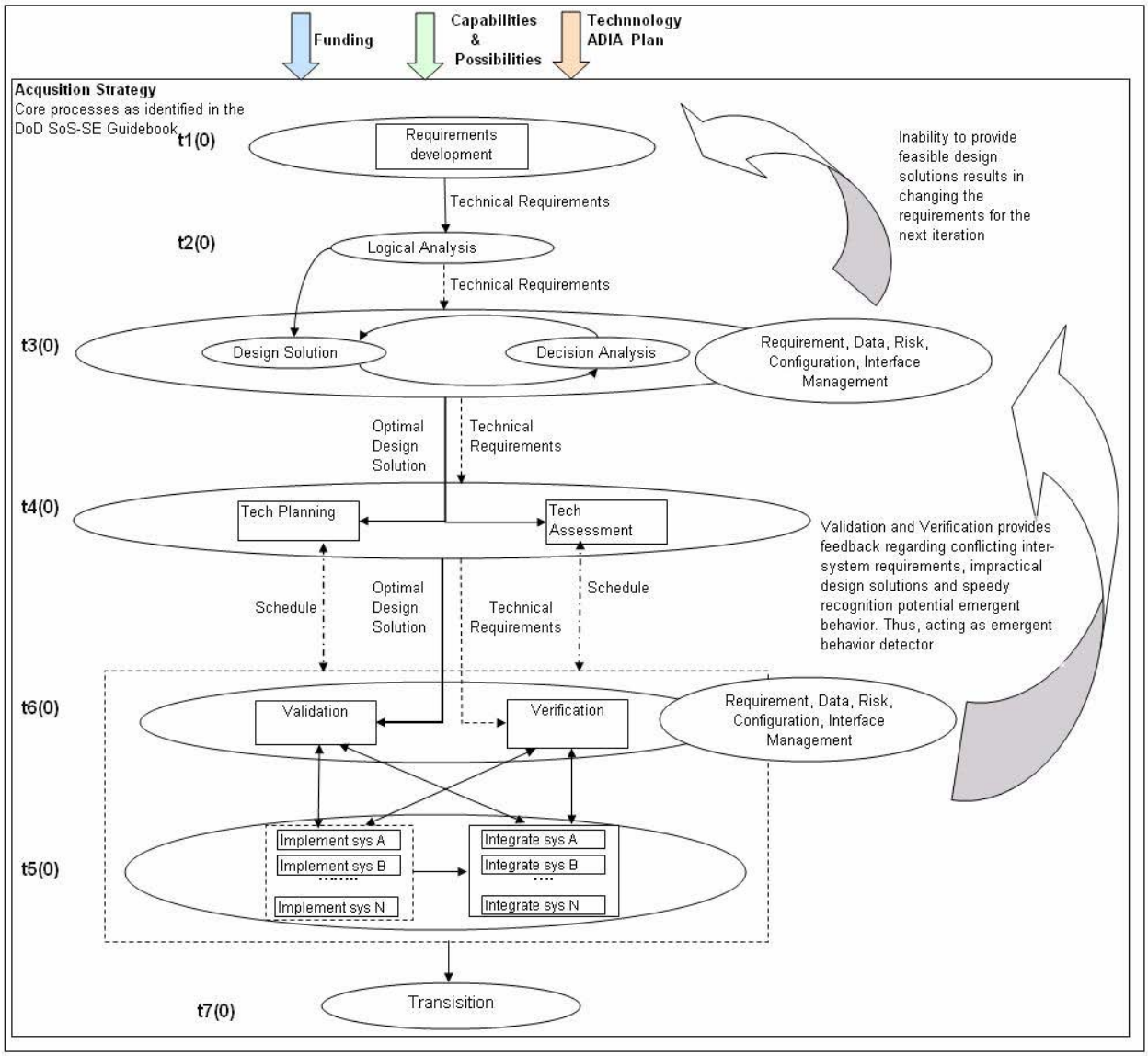


Figure 3: Conceptual model of Acquisition Strategy based on SoSE Process described in Table 1

Design Solution development and *Decision Analysis* are the next processes. They produce the optimal design solution from the set of feasible solutions to meet the given requirements. The optimal design solution is based not only on the current set of requirements and solution alternatives but also takes into account all previous information and data available through requirements, risk, configuration, interface and data management processes. Since most SoS acquisitions are multi-year projects involving many different parties, the overlap between the management processes, *Design Solution* and *Decision Analysis* processes, allows for greater traceability for decisions made. The optimal design solution obtained from this phase is then sent to the next stage: *Technology Planning* and *Technology Assessment*. In the event that there isn't an optimal or sub-optimal design solution to successfully implement the given requirements, the feedback loop to *Requirement Development* translates into a change in the technical requirements for the SoS.

Technology Planning and *Technology Assessment* are essentially scheduling processes that oversee the implementation, integration, validation and verification for all the component systems in the SoS. Systems in the SoS are often dependent on other systems for either implementation, integration or both. These dependencies correspond to time-lags in the acquisition process. For example: If system A is a legacy system and system B is being built, the integration of A with B will not occur until B has been completely implemented. This generates a time lag, especially if another system C is waiting to be implemented based on the integration of A with B. As more systems are added to the SoS, it becomes necessary to generate a schedule that can help co-ordinate the process. This schedule also needs to be continually updated to reflect unexpected delays and identify bottle-necks.

Due to the heterogeneity of component systems that comprise the SoS and the interactions between then, *Validation* and *Verification* processes need to not only check for suitable implementation of the 'optimal design solution' on a system-level but also be on the lookout for any misaligned objectives between systems, hidden dependencies amongst the systems and any emergent behavior that may affect the functioning and/or behavior of the future SoS. In most situations, early detection of an emergent behavior will prevent the re-designing of major system components and ensure that the SoS functions satisfactorily. Even though *Validation* and *Verification* processes oversee *Implementation* and *Integration*, they occur after *Implementation* and *Integration* have begun.

While *Implementation* and *Integration* are the lowest levels of the acquisition model shown, much of the feedback from this level translates into developing different design solutions and sometimes changing the technical requirements. This level deals with acquiring the systems in the SoS and integrating them based on their dependencies with other systems. These processes consume the bulk of the financial resources as well as consume the most time. Therefore, it is understandable why system engineers are often reluctant to re-design functional systems and want to make sure that once the system has been developed, integrated and tested, it doesn't go back into the *Implementation* phase.

Developing an Exploratory Computational Model

Overview

Our purpose in constructing a computational exploratory model is to help acquisition professionals develop intuition for procuring and deploying system-of-systems. Thus, the objective is not to provide a model validated and ready for deployment in real acquisition programs, but to expose the complexities in SoS acquisition. The specific complexities targeted are related to evolutionary development of the SoS and the span-of-control possessed by the SoS managers and engineers.

Several challenges arise in transforming the acquisition model to a computational one for purposes of simulation and learning. One challenge lies in converting all the qualitative concepts into quantitative measures to support the computational model for SoS acquisition. Disruptions occur at various stages in the model and are governed by the risk associated with the project. A high risk project, for example, will be more vulnerable to disruptions than a low risk project. A second challenge is building a model that can accommodate the dynamic addition and removal of components in the SoS. In addition, these component systems need to reflect the heterogeneity of the systems in a real acquisition process. We included parameters such as *level of completeness* to demonstrate the difference between legacy systems, new systems and partially implemented/

integrated systems. A third challenge arises from the numerous methodologies that can be applied to reflect the integration and implementation processes. In a simplified model, it is much easier to begin integration once all the systems have been implemented. However, this method is neither cost nor time efficient, especially in multi-year projects involving numerous systems. On the other hand, dynamically implementing and integrating systems is time-efficient but often not possible when dependent systems are outside the span of control of the systems engineers.

A model that captures all the complexity of the acquisition process for SoS in a modest span of time is impossible. Therefore, our coarse-scale engineering model will specifically target challenges related to the evolution of the SoS and the span-of-control of the SoS engineer(s).

Simple SoS Example

A simple SoS acquisition strategy with 2 requirements and 5 component systems (Fig 4) is presented first to illustrate the model workings. Requirement 1 is to improve rescue operations performed by a certain fleet, and Requirement 2 is to improve communication and coordination between air and ground units. The 3 types of component systems fulfilling Requirement 1 are helicopter (A), ship (B) and communication system (C). Similarly, the 3 component systems fulfilling Requirement 2 are ground units (A*), airborne units (B*) and a communication system (C*). Since Requirement 1 needs to use the communication system (C*) built by Requirement 2, Requirement 1 is dependent on Requirement 2. The directional dependencies within the component systems fulfilling each requirement are shown in Fig 4 using dashed yellow (bidirectional) and red (unidirectional) lines. The requirement level dependency matrix and the system-level dependency matrices for each requirement are shown in Table 2.

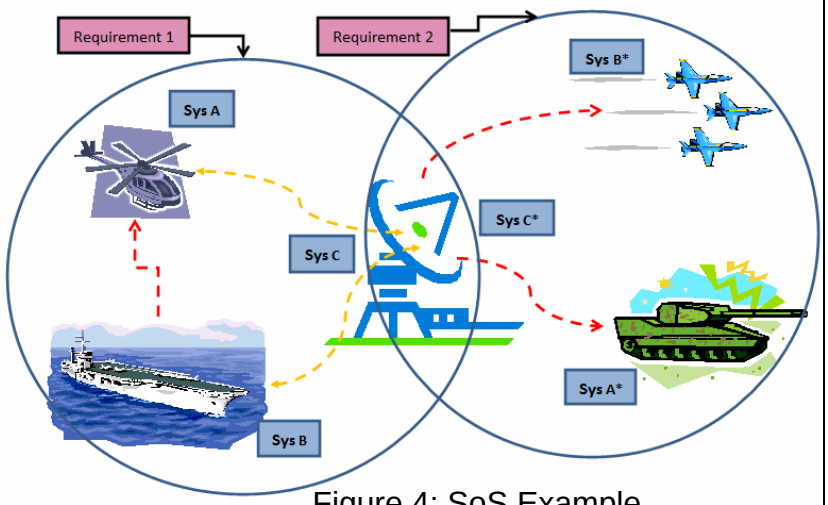


Figure 4: SoS Example

Table 2: Dependency matrices

Requirement Dependency Matrix	$\begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$

Requirement	System Dependency Matrix
1	$\begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$
2	$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$

Model Inputs

Three levels of inputs are used in the model: project-level, requirement-level and system-level. The three user-defined project-level inputs are project-risk, span-of-control of SoS managers and engineers and estimated amount of time needed to complete the project. A project can have low, medium or high project-risk profile and this profile determines: a) the probability of the project being affected by disruptions at *Design Solution* (Level t3(0), Fig 3) and *Implementation & Integration* (Level t5(0), Fig 3) stage, and b) the probability of a new requirement being added

during the project life-cycle. The span-of-control of an SoS engineer or manager indicates whether component systems are directly or indirectly accountable to the SoS manager or engineer. A project's span-of-control is either '0' or '1', where '0' represents low span-of-control. A project with low span-of-control implements dependent systems sequentially instead of in parallel.

The requirement-level inputs to the exploratory computational model are initial number of requirements, dependencies between requirements, component systems fulfilling each requirement and the dependencies between the component systems. The top-down flow of the computational model begins from *Requirement Development* (Level t0(0), Fig 3) to *Design Solution* (Level t3(0), Fig 3) through *Logical Analysis*(Level t2(0), Fig 3). This flow of control linkage using the inputs from Table 2 is shown in the Fig 5. The dependencies between the requirements determine the schedule by which the requirements will be implemented.

For the simple example problem, as shown in Table 2, there are 2 requirements (1, 2) and each has a dependency vector associated with it. The vectors are concatenated to form the dependency matrix for requirements ('0' is placed for all diagonal elements since a requirement cannot be dependent on itself). The vector for Requirement 1 ([0 1]) shows that Requirement '1' is dependent on Requirement '2' and '1' cannot be realized until '2' is implemented. In real world applications, communication upgrade to the North-Atlantic fleet may be independent of the weaponry upgrade for the same group of systems. In such a case, both the requirements on the same group of systems may be implemented simultaneously. Each requirement affects a subset of the systems present in the SoS, and the systems in each subset share a unique dependency matrix with other systems in that subset.

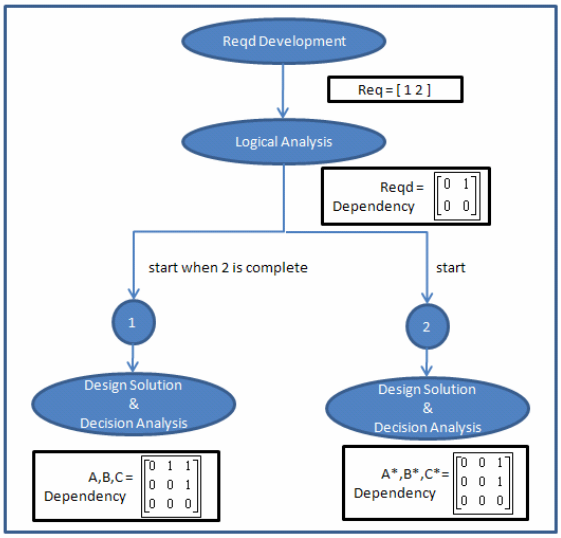


Figure 5 : Flow of control and parallel processing of requirements

All component systems of the SoS have user-defined and calculated system-level parameters that expose their heterogeneity and help track their progress through the acquisition process. Some of the parameters used to describe each system in the SoS are described in Table 3. While most of the parameters are user-defined, Imp.completeness and Int.completeness, are only initialized by the user and ID is assigned by the model.

Table 3: System-level Parameters used to describe component system of the SoS

Parameter	Description
ID	Unique ID assigned to the system
Imp.completeness[]	An array that tracks the progress of the system in the implementation phase
Imp.dependencies[]	Dependency vector that shows if system implementation is dependent on information from any other system
Imp.time	Maximum time needed to complete implementation
Int.completeness[]	An array that tracks the progress of the system in the integration phase
Int.dependencies[]	Dependency vector that shows if system integration is dependent on information from any other system

Int.time	Maximum time needed to complete integration

Implementation or Integration of a system[A] is either dependent on information from other systems satisfying the requirement or independent of any such information. Thus, all the tasks necessary to successfully implement or integrate system[A] can be divided into smaller subsets depending upon which systems they need information from. At time-step t, the *level of completeness* of system[A] with regard to system[X] (denoted by t^c_{AX}) is defined as the percent of tasks needed to successfully implement/integrate system[A] that are dependent on information from system[X] and have been completed. *Level of completeness* for both integration and implementation processes can vary between 0 and 100% ($0 \le t^c_{AX} \le 1$). The *level of completeness* of system A with regard to *N* individual systems is summed to calculate the total *level of completeness* of system A, as shown in Equation 1.

$$T^c_A = t^c_{AA} + t^c_{AB} + \cdots\cdots + t^c_{AN} \qquad \textbf{(1)}$$

Note that though the tasks are dependent on information from system[A], the *level of completeness* says nothing about the status of system[A]. Note also that the model works in discrete time.

Similar to requirements, each system has a pre-defined dependency vector for implementation and integration processes. These vectors are concatenated to form a dependency matrix for the systems fulfilling each requirement. The system-level dependency matrices for the example in Fig 4 are shown in Table 2. As previously mentioned, ID is assigned by the model. When the system is added to the SoS, it is assigned an ID to uniquely identify it throughout the life-cycle of the SoS.

Model Dynamics

The model starts at the *Requirement Development* (Level t0(0), Fig 3) stage which initializes all processes by supplying requirements to be implemented, project span-of-control and project risk. Disruptors here signify a change in requirements or addition of new requirements. When a requirement is changed after the acquisition process has begun, it affects all subsequent processes.

Using the user-defined inputs from *Requirement Development*, *Logical Analysis* (Level t2(0), Fig 3*)* generates a schedule to realize the given requirements. The requirements get implemented in series or in parallel (per the dependencies). As shown in Fig 5, every requirement being implemented gets fed into its own *Design Solution* and *Decision Analysis* (Level t3(0), Fig 3*)* process. The *Design Solution* and *Decision Analysis* processes feed into each-other and any disruptions would indicate that the design solution provided wasn't feasible. If the solution fails in multiple consecutive time-steps, then the requirement is sent back to *Requirement Development* stage, otherwise the set of component systems and their user-defined parameters are sent to the *Technology Planning* and *Technology Assessment* (Level t4(0), Fig 3*)* processes.

Technology Planning generates a schedule to realize the implementation and integration of component systems. The systems are divided into smaller batches based on the priority of systems. By default, systems in the critical path of most other systems are assigned a higher priority. These smaller batches are reminiscent of the technological 'spin-outs' introduced during the FCS program. For each batch, a synchronization matrix is generated to keep track of the number of systems in the batch, their expected times of completion and their *iteration-rate*. Iteration rate is defined as the average rate at which a system needs to be implemented /integrated. For example: If system A which is 25% completed needs to be fully implemented in 5 time-steps. Using Equation 2, the iteration rate of system A is calculated to be 0.15.

$$Iteration\_Rate = \frac{1 - completeness(t_0)}{\max\_time} \quad \textbf{(2)}$$

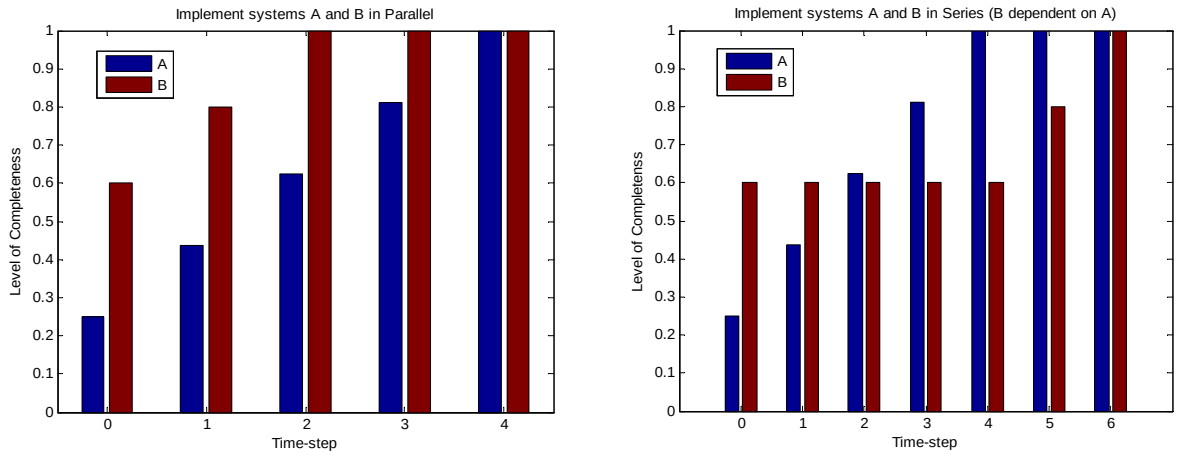
System-level disruptors (determined by the project risk) negatively impact the iteration rate of the systems fulfilling a given requirement, thus decreasing the *level of completeness* of a system and increasing the time needed for implementation/integration. *Technology Assessment* tracks the progress of component systems by comparing their current iteration rates to the expected values. A reduced iteration-rate (due to disruptors) will stall the development of a system mid-process and affect other systems dependent on the stalled system. *Technology Assessment* recognizes the stalled systems and activates enablers to re-adjust their iteration-rate.

Implementation (Level t5(0), Fig 3*)* of systems occur in series or parallel depending on the system dependencies and the span-of-control of the project. The *level of completeness* for implementation increases by the iteration rate at every time-step until it reaches a completeness value of 1. The incremental increase in the level of completeness of two dependent systems in a project with high span-of-control('1') occurs simultaneously, as shown in Fig 6a. In a case of low span-of-control('0'), dependent systems are implemented sequentially, as shown in Fig 6b. If $time_i$ is the time needed to implement system[i], then Equation 3 and 4 are used to calculate the total time to implement all systems with a low and high span-of-control respectively.

For Low span-of-control: $T = \sum_{i=1}^{n} time_i$ **(3)**

For High span-of-control: $T = \max\{time_i\} \; \forall i = 1 \cdots\cdots n$ **(4)**

When a system achieves the implementation completeness = 1, it enters the integration queue.



a) Independent Systems

b) Dependent Systems

Figure 6: Incremental increase in implementation completeness

Two assumptions made for the *Integration* (Level t5(0), Fig 3) process are:

1. Integration tasks for a system are always greater than or equal to its Implementation tasks
2. Integration tasks of a system that are dependent only on itself are assumed to be completed during the implementation process. Therefore, *level of integration completeness* of a system with regard to itself (t^c_{AA}) is '1' at all times.

Similar to *Implementation*, systems can be integrated in series or in parallel depending on the span-of-control. The total time needed to integrate component systems with low or high span-of-control is calculated using Equations 3 or 4 respectively. When both the *Implementation* and *Integration* processes for the given requirement are complete, *Validation* and *Verification* (Level t6(0), Fig 3) checks for a completeness level of 1 for all component systems. If the requirement successfully passes *Validation* and *Verification*, it is said to be ready for *Testing*.

Testing the Exploratory Model

Twelve test scenarios were implemented via simulation using our exploratory model to understand the dynamics underlining acquisition management in an SoS environment. The test-cases specifically study how dependency between requirements, span-of-control of SoS managers and engineers and different project risk profiles affect the time taken to successfully complete the project.

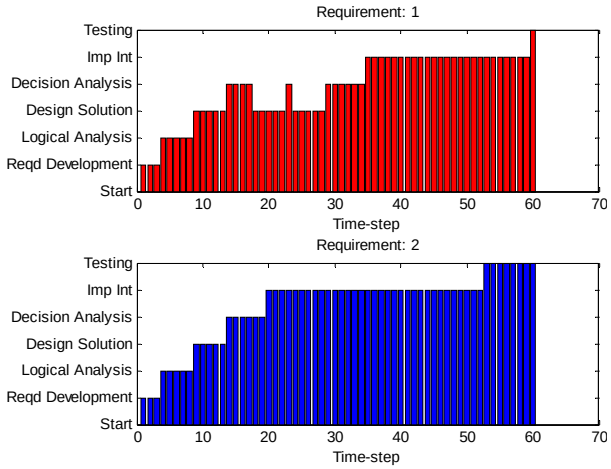


Figure 7 : High span-of-control, independent requirements and low risk profile

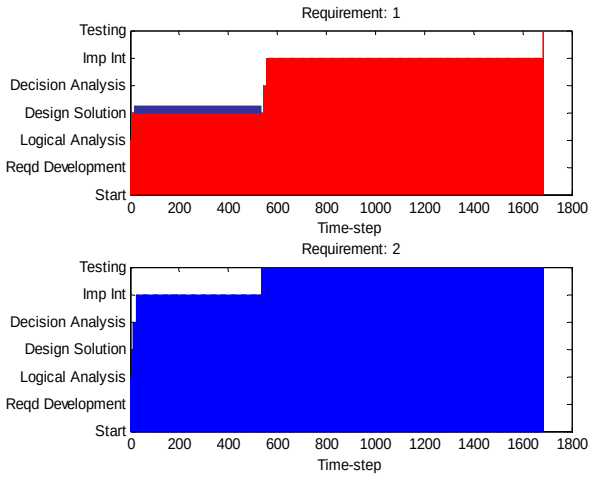


Figure 8 : Low span-of-control, dependent requirements and high risk profile

A project with two independent requirements, a high span-of-control (‘1’) and low risk profile was successfully implemented in 60 time-steps (Fig 7). On the other hand, a project with two dependent requirements (dependency matrices shown in Table 2), low span-of-control (‘0’) and high project risk needs 1682 time-steps to complete (Fig 8).

The time needed to complete projects in all twelve test-cases with independent or dependent requirements, varying span-of-control and different risk profiles is plotted in Fig 9. The abscissa represents span-of-control while the ordinate represents the risk associated with the project (1: Low, 2: Medium, 3: High).
The results from these twelve test-cases were used in a sensitivity analysis to show the relative importance of each of the three parameters on the total time needed to complete the project.

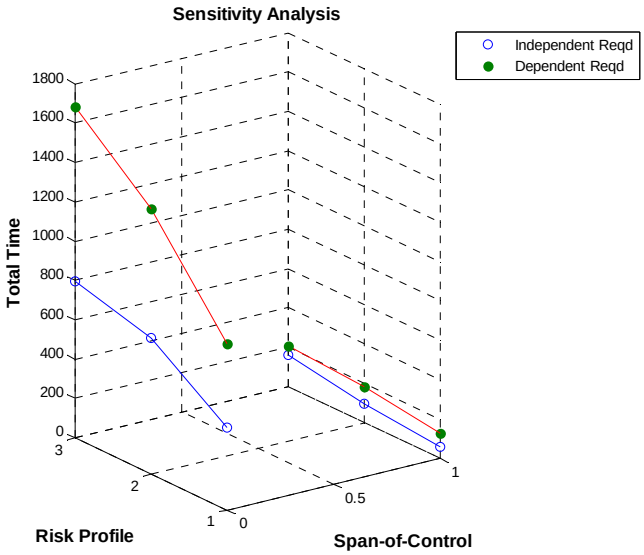


Figure 9 : Effect of requirement dependency, span-of-control and risk profiles on total time to complete

Sensitivity Analysis

Sensitivity analysis further investigates the impact of the three parameters (requirement dependency, span-of-control and risk profile) studied in the 12 test cases.
Requirement Dependency: Compare cases of dependent versus independent requirements while keeping span-of-control and risk profile constant (Table 4). Risk profiles are labeled '1' for Low, '2' for Medium and '3' for High. Projects with dependent requirements take longer (by a factor between 1.28 to 2.13) compared to projects with independent requirements.

Table 4: Effect of Requirement Dependency

Span of control	Risk	Ratio = $\frac{Time(Dependent)}{Time(Indepenent)}$	Span of control	Risk	Ratio = $\frac{Time(Dependent)}{Time(Indepenent)}$
1	1	2.13	0	1	2.004
1	2	1.88	0	2	1.95
1	3	1.28	0	3	2.11

Span-of-Control: Compare cases of low versus high span-of-control while keeping requirement-dependency and risk profile constant (Table 5). Projects with low span-of-control take longer (by a factor between 4.93 to 8.155) compared to projects with high span-of-control.

Table 5: Effect of span-of-control

I/D	Risk	Span-of-control	Ratio	I/D	Risk	Span-of-control	Ratio
I	1	0/1	7	D	1	0/1	6.578
I	2	0/1	7.09	D	2	0/1	7.34
I	3	0/1	4.93	D	3	0/1	8.155

Risk Profile: Compare cases of three risk profiles, while keeping requirement-dependency and span-of-control constant (Table 6).The ratio is an increase from a lower risk profile to a higher risk profile. For example: For a project with independent requirements and high span-of-control, the ratio of time needed for a medium risk (2) profile versus a low risk (1) profile is 1.61.

Table 6: Effect of Increasing Project Risk

I/D	Span-of-control	Risk	Ratio	I/D	Span-of-control	Risk	Ratio
I	1	1	-	I	0	1	-
I	1	2	1.61	I	0	2	1.63
I	1	3	1.65	I	0	3	1.155
D	1	1	-	D	0	1	-
D	1	2	1.43	D	0	2	1.59
D	1	3	1.13	D	0	3	1.25

Results

Some insights gained from testing the exploratory model via a sensitivity analysis are:

1. As expected, time to implement dependent requirements is always greater than the independent case; the amount of increased strongly depends on the span-of-control of the SoS managers and engineers and the project risk.
2. Time needed to implement projects with higher risk profile is always greater than the time needed to implement the project with lower risk profiles.
3. With high span-of-control, time needed to complete a project increases linearly with an increase in project risk for both dependent and independent requirements. However with a low span-of-control, the time needed increases exponentially with an increase in project risk for both dependent and independent requirements.
4. The sensitivity analysis shows that the time needed to complete a project is much more sensitive to the span-of-control of the SoS engineers and managers than to the project risk or the dependencies between the requirements.
5. A project with high span-of-control is better equipped to recover from the debilitating disruptions associated with a high risk, thus making the acquisition process more resilient.

Conclusion

From historical data related to past SoS-oriented defense acquisition programs, we summarize the common causes of failure as: a) misalignment of objectives among the systems, b) limited span of control of the SoS engineer on the component systems of the SoS, c) evolution of the SoS, d) inflexibility of the component system designs, e) emergent behavior revealing hidden dependencies within systems, f) perceived complexity of systems and g) the challenges in accurately representing them. These sources of complexity were mapped to a section of the SoSE Process Model recently introduced by Sage and Biemer (2007) to identify where manifestations of these complexities might arise and begin to assess how they may impact the acquisition process.

This mapping in conjunction with the 16 technical and technical management SE processes identified by the SoS-SE Guide (U.S. DoD 2008) was used to develop a conceptual model for pre-acquisition and acquisition strategy activities. The acquisition strategy model takes an incremental approach to the evolutionary development of an SoS and allows processes lower in the

hierarchy to affect change in the processes above them. Thus, the model exposes the interconnections among levels and uses these to implement evolving requirements and design solutions in the component systems of the SoS.

These mappings and conceptual models are all directed toward providing a basis for a computational exploratory model for acquisition strategy in an SoS environment. The purpose of the model was to explore the complexities that arise in SoS acquisition programs due to evolutionary development of the SoS, heterogeneity of the component systems as well as the effect of management parameters such as span-of-control on the acquisition programs. Based on user defined inputs for the requirements and their dependencies on each other, the model uses series and parallel processing to implement and integrate the component systems fulfilling the requirements. Disruptors and enablers are used to affect non-linear behavior in the model. This exploratory model allows evolving requirements and design solutions to trickle through the lower processes and uses disruptors to affect specific component systems, which in-turn affects change in processes higher up in the hierarchy. Results from the test-scenarios underline the importance of span-of-control of SoS managers and engineers on the timely completion of even high-risk projects, by making the acquisition process more resilient and agile in the face of disruptions.

The uniqueness of the models (both conceptual and computational) lie in their ability to provide a better understanding of the acquisition process in an SoS environment along with computational tools for better decision-making for the higher levels of SoS management. We hope that the insights gained from this research will significantly improve the probability of success of future acquisition programs of complex SoS.

Biography

Daniel DeLaurentis is an Assistant Professor in the School of Aeronautics and Astronautics Engineering, Purdue University. He received his Ph.D. from Georgia Institute of Technology in Aerospace Engineering in 1998. His current research interests are in Mathematical modeling and object-oriented frameworks for the design of system-of systems, especially those for which air vehicles are a main element; approaches for robust design, including robust control analogies and uncertainty modeling/management in multidisciplinary design.

Shayani Ghose is a Graduate student in the School of Aeronautics and Astronautics Engineering, Purdue University. She received her BS in Electrical Engineering and Computer Engineering (Dual) from Drexel University, Philadelphia in June 2007. She is currently a part of the System-of-systems Research Group advised by Dr. Daniel DeLaurentis.

References

Bar-Yam, Y. 2003. Complexity of Military Conflict: Multiscale Complex Systems Analysis of Littoral Warfare. *New England Complex Systems Institute*. http://www.necsi.edu/projects/yaneer/SSG_NECSI_3_Litt.pdf

Committee on Systems Integration for Project Constellation. 2004. *Systems Integration for Project Constellation*. The National Academies. http://nap.edu/html/proj-constellation/ltr-rep.pdf

Fowler, C. A. 1994. The Defense Acquisition System Too Late for the Scalpel; Bring Out the Meataxe! *IEEE Aerospace and Electronic Systems Magazine* 9(8):3-6.

Ghose, S. and D.A. DeLaurentis. 14-15 May 2008. Defense Acquisition Management of Systems-of-systems. *5th Annual Acquisition Research Symposium*, Naval Postgraduate School, Monterey, CA.

Maier, M. 1998. Architecting Principles for System-of-Systems. *Systems Engineering* 1:267-284.

Polzer, H., D.A. DeLaurentis, and D.N. Fry. 2007. Multiplicity of Perspectives, Context Scope, and Context Shifting Events. *2007 IEEE International Conference on System of Systems Engineering* :1-6.

Rouse, W. 2007. Complex Engineered, Organizational and Natural Systems. *Systems Engineering* 10(3):260-271

Sage, A. and S. Biemer. 2007. Processes for System Family Architecting, Design, and Integration. *IEEE Systems Journal* 1(1):5-16.

Saunders, T.et al. 2005. Report on System-of-Systems Engineering for Air Force Capability Development. *USAF Scientific Advisory Board.* SAB-TR-05-04.

Simon, H. 1996. *Sciences of the artificial*. Cambridge, MA: MIT Press.

Spring, B. 2005. Congressional Restraint Is Key to Successful Defense Acquisition Reform. *The Heritage Foundation.* http://www.heritage.org/Research/NationalSecurity/bg1885.cfm (March 26, 2008).

U.S. Army. Future Combat Systems. https://www.fcs.army.mil/

U.S. Department of Defense. Defense Acquisition Guidebook. ONLINE. GPO Access. 12 May 2003. Available: https://akss.dau.mil/dag/ (4 April 2008)

U.S. Department of Defense. Systems Engineering Guide for System-of-systems. ONLINE. GPO Access. January 2008. Available: http://www.acq.osd.mil/sse/ssa/initiat_sos-se.html\ (2 March 2008)

U.S. Government Accounting Office. *Issues facing the Army's Future Combat Systems.* ONLINE GPO Access. 13 August 2003. Available: http://www.gao.gov/new.items/d031010r.pdf

U.S. Government Accountability Office. *Coast Guard: Observations on Agency Performance, Operations and Future Challenges*. June 2006. Available: http://www.gao.gov/new.items/d06448t.pdf

U.S. Government Accounting Office. *Future Combat System Risks Underscore the Importance of Oversight*. ONLINE. GPO Access. 27 March 2007. Available: http://www.gao.gov/new.items/d07672t.pdf

U.S. Government Accounting Office. *Role of Lead Systems Integrator on Future Combat Systems Program Poses Oversight Challenge.* ONLINE. GPO Access. June 2007. Available: http://www.gao.gov/new.items/d07380.pdf