NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

EXECUTION OF SYSTEMS INTEGRATION PRINCIPLES DURING SYSTEMS ENGINEERING DESIGN

by

John K. Logan Jr.

September 2016

Thesis Advisor: Eugene Paulo
Second Reader: Gary Parker

Approved for public release. Distribution is unlimited.

THIS PAGE INTENTIONALLY LEFT BLANK

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 instruction, 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** September 2016	**3. REPORT TYPE AND DATES COVERED** Master's thesis	
4. TITLE AND SUBTITLE EXECUTION OF SYSTEMS INTEGRATION PRINCIPLES DURING SYSTEMS ENGINEERING DESIGN			**5. FUNDING NUMBERS**
6. AUTHOR(S) John K. Logan Jr.			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol number ____N/A____.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.			**12b. DISTRIBUTION CODE**

13. ABSTRACT (maximum 200 words)

Systems integration (SI) is an extensive task conducted as part of the bottoms-up systems engineering (SE) lifecycle approach. Implementation of a newly developed system depends on successful accomplishment of systems integration. Complexities of system design solutions are making SI success more difficult to achieve; integration failures have become more common and tend to drive costly redesign efforts. This research explores some of the integration failures and causes and proposes SE developmental phase considerations regarding requirements, stakeholders, testing, and system boundaries. Additionally, this thesis discusses use of systems architecture frameworks and models and the consistent use of model-based systems engineering throughout development. Lastly, it proposes formal methods language for improving models. This research describes how all of these solutions can facilitate identifying and resolving common SI failures prior to the completion of system development. By doing so, the success of the integration effort and the system as a whole is ensured.

14. SUBJECT TERMS systems integration, integration failures, formal methods			**15. NUMBER OF PAGES** 81
			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** UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

EXECUTION OF SYSTEMS INTEGRATION PRINCIPLES DURING SYSTEMS ENGINEERING DESIGN

John K. Logan Jr.
Civilian, Department of the Navy
B.S., Southern Illinois University, 2001
M.S., Colorado Technical University, 2005

Submitted in partial fulfillment of the
Requirements for the degree of

MASTER OF SCIENCE IN SYSTEMS ENGINEERING MANAGEMENT

from the

NAVAL POSTGRADUATE SCHOOL
September 2016

Approved by: Eugene Paulo, PhD
Thesis Advisor

Gary Parker
Second Reader

Ronald Giachetti, PhD
Chair, Department of Systems Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Systems integration (SI) is an extensive task conducted as part of the bottoms-up systems engineering (SE) lifecycle approach. Implementation of a newly developed system depends on successful accomplishment of systems integration. Complexities of system design solutions are making SI success more difficult to achieve; integration failures have become more common and tend to drive costly redesign efforts. This research explores some of the integration failures and causes and proposes SE developmental phase considerations regarding requirements, stakeholders, testing, and system boundaries. Additionally, this thesis discusses use of systems architecture frameworks and models and the consistent use of model-based systems engineering throughout development. Lastly, it proposes formal methods language for improving models. This research describes how all of these solutions can facilitate identifying and resolving common SI failures prior to the completion of system development. By doing so, the success of the integration effort and the system as a whole is ensured.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I. INTRODUCTION....1

A. BACKGROUND AND OVERVIEW....1

B. SYSTEMS INTEGRATION OVERVIEW....1

1. Systems Integration Defined....1

2. An Example of Systems integration....2

C. THE SYSTEMS INTEGRATION PROBLEM....3

D. SYSTEMS ENGINEERING OVERVIEW....4

1. Systems Engineering Defined....4

2. Systems Engineering Development Lifecycle Explained....4

3. Typical Phases within a Lifecycle....5

E. SYSTEMS INTEGRATION EFFORTS WITHIN THE LIFECYCLE....5

1. SI Functions that Occur on the Right Side of the VEE Model....6

2. Need for Systems Integration on the Left Side of the VEE Model....6

F. PROPOSED METHODOLOGY....7

G. RESEARCH QUESTIONS....8

H. SUMMARY OF SYSTEMS INTEGRATION....8

II. SYSTEM DEVELOPMENT AND EMPHASIS ON SYSTEMS INTEGRATION....9

A. INTRODUCTION....9

B. STAKEHOLDER/CUSTOMER/USER NEEDS ANALYSIS....9

1. Who Are the Stakeholders and What are Needs?....9

2. Importance of Conducting a Needs Analysis....10

3. Impacts to Systems Integration....11

4. Summary of Stakeholders and Needs Analysis....11

C. SYSTEM BOUNDARIES AND INTERFACES....12

1. Introduction....12

2. System Interfaces....12

3. Impacts to Systems Integration....13

4. Summary of System Boundaries and Interfaces....13

D. REQUIREMENTS DEVELOPMENT....13

1. Introduction....13

2. Importance of Traceability....15

3. Importance of Using a Requirements Management Tool....16

4. Full Coverage of Requirements Testing for Risk Mitigation....16
5. Impacts to Systems Integration....17
6. Summary of Requirements Development....18
E. SYSTEMS INTEGRATION TEST DEVELOPMENT....19
1. Introduction to Testing Development....19
2. The Connection between Requirements Traceability and Testing Development....19
3. Summary of Systems Integration Test Development....20

III. IMPROVING SYSTEMS INTEGRATION THROUGH ADVANCED SOLUTIONS....21
A. EMPLOYING A SYSTEMS ARCHITECTURE FRAMEWORK....21
1. Systems Architecture Defined....21
2. Systems Architecture Ties to Systems Engineering....21
3. Contributions to Systems Integration....23
4. Summary of Systems Architecture....25
B. UTILIZING MODEL-BASED SYSTEMS ENGINEERING....26
1. Model-Based Systems Engineering Explained....26
2. Contributions to Systems Integration....27
3. Summary of Model-Based Systems Engineering....28

IV. SOLUTION FOR ADDRESSING COMPLEX SYSTEMS INTEGRATION PROJECTS....31
A. UNIQUE SYSTEMS INTEGRATION CIRCUMSTANCES THAT CAN BENEFIT FROM THE UTILIZATION OF ADVANCED SOLUTIONS:....31
1. Integration of New and Complex System Solutions with Legacy Systems....31
2. External Factors that Impact System Requirements and Design....32
3. Summary of Circumstances Requiring Advanced Solutions....32
B. USING FORMAL METHODS TO ANALYZE AND DESIGN SYSTEM INTERFACES....33
1. What are Formal Methods?....33
2. What is Lifecycle Modeling Language?....33
3. Phased Approach for Implementation of Formal Methods....35
4. Contributions to Systems Integration....36
C. SUMMARY OF FORMAL METHODS....37

V. SYSTEMS INTEGRATION CASE STUDIES 39
A. SYSTEMS INTEGRATION CASE STUDIES FOR DOD SYSTEMS 39
1. System #1 – Key Stakeholder Left Out of Development 39
2. System #2 – Reduced Stakeholder Involvement and Lack of Legacy Systems Requirements Analysis 40
3. System #3 – Failure to Analyze Software Interfaces and Behaviors 42
4. Other Integration Issues 43
B. NON-DOD INTEGRATION FAILURE CASE STUDIES 44
1. FBI Virtual Case File Project 44
2. Ariane 5 45
C. SUMMARY OF CASE STUDIES AND SYSTEMS INTEGRATION ISSUES 45

VI. RECOMMENDATIONS AND CONCLUSION 47
A. RECOMMENDATIONS 47
1. Thorough Stakeholder Analysis Reduces Design Rework During Systems Integration 47
2. Requirements Traceability Improves Translation of Stakeholder Needs to System Requirements to System Design 47
3. Utilization of a Systems Architecture Framework Improves Systems Integration for Complex Systems 48
4. Implementation of Model-Based Systems Engineering Improves System Requirements, Design and Integration 49
5. Incorporation of Formal Methods Patterns Enforces Systems Integration in Design Specifications 49
B. CONCLUSION 50
C. OPPORTUNITIES FOR ADDITIONAL RESEARCH 50

LIST OF REFERENCES 53

INITIAL DISTRIBUTION LIST 57

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

Figure 1. SI events. Source: SEBOK (2016). 2

Figure 2. Example of Systems Integration for Aircraft Cockpit Electronics. Source: Pickar (2015). 3

Figure 3. VEE Process Model. Source: SEBOK (2016). 5

Figure 4. System Lifecycle Requirements Traceability. Source: ITABoK (2016). 15

Figure 5. MBSE Requirements Traceability. Source: Giachetti (2015)..... 17

Figure 6. Systems Architecture Views. Source DOD (2015b)..... 22

Figure 7. CORE's DODAF Version 2 Schema. Source: Vitech (2016). 25

Figure 8. Requirements Inputs to Model-Based Systems Engineering. Source: OMG (2016)..... 26

Figure 9. Class/Relationship Diagram. Source: Rodano and Giammarco (2013). 35

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

Table 1. Benefits of Well-Written Requirements. Source: NASA SEH (2007). 18

Table 2. Benefits of Using Model-Based Systems Engineering. Source: INCOSE UK (2016). 28

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS.

AOA	analysis of alternatives
CI	configuration item
CM	configuration management
DOD	Department of Defense
DODAF	Department of Defense Architecture Framework
DOORS	Dynamic Object Oriented Requirements System
EMMI	energy, matter, material wealth, information
EOL	end of life
FM	formal methods
GFI	government furnished information
IOC	initial operational capabilities
IOE	intended operating environment
IPT	integrated product team
LML	lifecycle modeling language
MBSE	model-based systems engineering
PM	program manager
RFP	request for proposal
SA	systems architecture
SDLC	systems engineering development life cycle
SDS	Shipboard Data System
SE	systems engineering
SEBOK	Systems Engineering Book of Knowledge
SI	systems integration
SLOC	software lines of code
SOI	system of interest
SOS	system of systems
STIG	security technical information guideline

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

A design agent conducts Systems Integration (SI) efforts as a part of an overall systems engineering process. SI execution occurs after all other system design efforts are complete. During SI, component assembly occurs and functional, and interface, testing is accomplished to build assemblies, subsystems and systems. Eventually the integration of the new system into its intended operating environment occurs. For systems within a system of systems (SOS) architecture, testing of functions and interfaces occurs once more. Failures observed during SI are on the rise and threaten the success of implementing the new system. In many cases, and due to its criticality, the new system must be implemented and therefore endure costly redesigning and subsequent regression testing. These failures influence both Department of Defense (DOD) and non-DOD systems. Solutions exist to reduce the likelihood of these failures occurring by discovering them early in development. This research explores the issues encountered during the execution of SI, explores problematic systems engineering tasks executed during development, and proposes solutions that can ensure SI success.

Systems integration is most concerned about testing functionality of objects and interactions via interfaces between objects whether those objects reside within the same assembly, subsystem, or system. Interfaces also connect objects that reside in different systems.

> Clean interfaces make a big difference in the error rate of the design. Some have estimated that errors and rework, though affecting only a small fraction of a design, may account for half the design cost. Worse yet, errors due to vague or sloppy interfaces usually surface late, during systems integration. Nastier to find, costlier to fix, impact the whole system schedule. (Brooks 2010, 94)

This research makes several recommendations to design agents to improve the likelihood of accomplishing systems integration successfully. First, this research proposes a more detailed approach to the development of the system. Specifically, this means the manner in which stakeholders are identified and engaged during development of the system, considerations that must be made regarding system boundaries and interfaces, the

importance of requirements development and traceability throughout development and into testing activities, and the need for thorough testing development in support of SI.

Secondly, this research proposes the utilization of advanced solutions in conjunction with existing systems engineering processes to include the creation of an integrated system architecture (SA) via a framework and its associated models, utilization of model-based systems engineering (MBSE) for analyzing potential requirements changes, and the application of formal methods to enforce desirable patterns for solidifying models.

Analysis of DOD and industry case studies in addition to errors encountered by this researcher during DOD systems integration, revealed failures observed during the conduct of systems integration. The root cause or causes to these failures are traceable to inadequate systems engineering development efforts conducted prior to conducting SI. The recommendations made in this thesis could have prevented the failures these systems observed during SI. This thesis discusses integration failures observed by DOD and non-DOD systems as, inadequate stakeholder analysis, incomplete problem space and design solution, inadequate requirements traceability, lack of requirements traceability between system and test requirements, and a lack of system boundaries awareness and external interfaces.

In addition to implementing the systems engineering developmental recommendations for improving SI success, this thesis recommends execution of advanced solutions concurrently within the SE process phases. This thesis explores the benefits of the initial execution of a systems architectural framework at system conceptualization, the establishment of MBSE tools and its continued use throughout development, and SI and the implementation of formal methods to further enforce desirable patterns or requirements within the modeling language. This thesis documents the benefits of using each solution and it is useful to achieving systems integration success.

Complexities of system design solutions are making SI success more difficult to achieve; integration failures have become more common and tend to drive costly redesign efforts. To achieve success, strategies for addressing systems integration must change or

the observation of past failures. These failures can prevent a system from succeeding. Maier and Rechtin (2009, 10) state, “If a system is to succeed, it must satisfy a useful purpose, an affordable cost, for an acceptable period of time.”

References

Brooks, Frederick P. 2010. *The Design of the Design, Essays from a Computer Scientist*. Boston, MA: Pearson Education.

Maier, Mark W., and Eberhardt Rechtin. 2009. *The Art of Systems Architecting*. Boca Raton, FL: CRC Press.

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

There are many individuals who I am thankful for assisting, mentoring and or supporting me throughout this effort to research and author this thesis.

First, I thank my God for giving me the perseverance to endure this two-year endeavor to write this thesis. Every “ah ha” moment originated from Him and sometimes came from speaking with other individuals.

Second, I would like to thank my wife and two sons for understanding and supporting me during the long nights and weekends spent working on this thesis rather than spending time with them. The end has come and it feels great!

Third, I would like to thank Gene Paulo and Gary Parker for mentoring me with the research, editing and defending of this thesis. A special thanks goes to Kristin Giammarco for instructing and consulting me in formal methods and MBSE. I would like to thank Barbara Berlitz and Noel Yucuis for assisting me with improving my writing and grammar skills. I would also like to thank Wally Owen and Heather Hahn for their logistical support and guidance in meeting the thesis deadlines.

Lastly, I would like to thank my NPS professors for instructing and preparing me for writing this thesis. In many cases, some of you invested additional time to assist me in understanding advanced principles. Thank you very much!

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

The purpose of this research is to explore the principles, considerations, decisions and tasks related to systems integration (SI) that must occur during the early phases of system development This thesis lists and describes some realized DOD program issues associated with SI deficiencies and provide recommendations for implementing proven principles, effective software application tools and emerging methods to utilize within the SE process for ensuring physical SI success.

A. BACKGROUND AND OVERVIEW

This chapter explores what SI is, its relationship to systems engineering (SE), systems architecture (SA) and as an integral part of the overall systems engineering development lifecycle. Merriam-Webster Dictionary (2016) states, a "system" refers to, "a group of related parts that move or work together" and the term "integration", from the action to "integrate" refers, "to combine (two or more things) to form or create something" (2016). The next chapter goes into detail of what constitutes systems integration and how it applies in the creation of a system.

B. SYSTEMS INTEGRATION OVERVIEW

1. Systems Integration Defined

Experts in their respective fields state definitions for SI. Jeff Grady (2010, 6) defines SI as, "The art and science of facilitating the marketplace of ideas that connects the many separate solutions into a system solution … a process that unifies the product components and the process components into a whole." Gary Langford (2012, 2) defines SI as, "A method that facilitates outcomes that are beyond what an individual object can do either individually or by a number of objects acting independently, that is, makes things happen that would otherwise not happen." In the words of this researcher, SI is the process of combining objects together to accomplish a common goal or mission. Figure 1 provides a depiction of SI events. The events are hierarchical, in which a subsequent event builds on the previous event. The combining of hardware and/or software-

configured items (CI) creates an assembly which is tested at the assembly level; assemblies are combined into a system and tested at the system level and finally the system is installed and tested into its intended operating environment and testing the completed system. Most designed systems involve to certain degree systems integration.

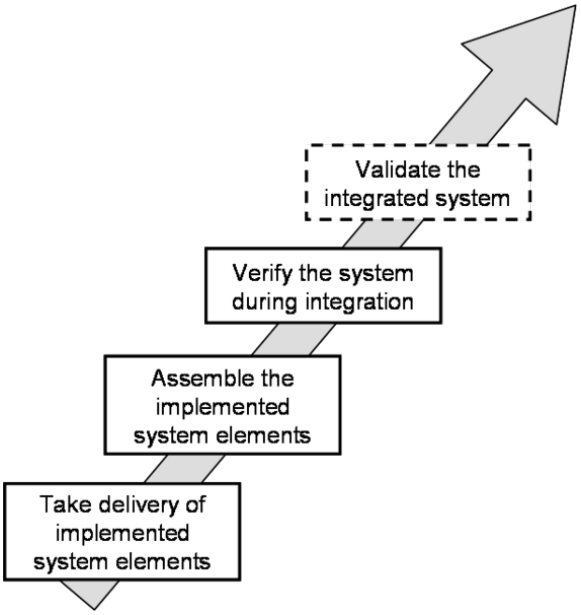

Figure 1. SI events. Source: SEBOK (2016).

2. An Example of Systems integration

Figure 2 is a visual depiction of a proposed systems integration flow for an aircraft cockpit and electronics system. Observe how hardware components or objects are integrated together to create a larger hardware assembly, possibly a subsystem. Testing of software coding happens concurrently with the hardware integration. The combining of hardware and software creates an assembly or subsystem. Additional testing occurs at the subsystem and/or follow-on system level. The execution of final testing or user acceptance testing occurs with the designed assemblies/subsystems/system is installed or integrated into the intended operating environment, an aircraft cockpit. The intended operating environment (IOE) is a location in which the user or for a weapons system the warfighter will operate the system. This testing ensures all acceptable performance of the

integrated system's external interfaces, with the other systems in the operating environment; all energy, matter, material wealth, and information (EMMI) exchanges are occurring as designed. All aforementioned testing answers the question, "can the newly integrated system effectively exchange EMMI with other systems hosted in the operating environment?"

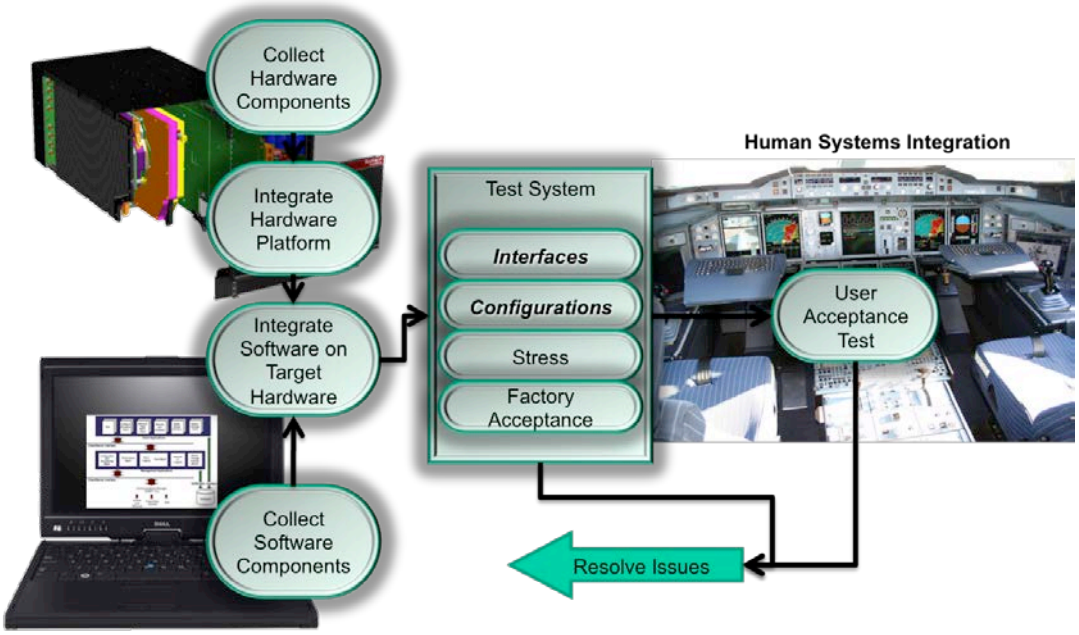

Figure 2. Example of Systems Integration for Aircraft Cockpit Electronics. Source: Pickar (2015).

C. THE SYSTEMS INTEGRATION PROBLEM

DOD program managers must maintain weapon system operational readiness for longer than planned lifecycles and with less funding. As these systems are being operationally sustained for a longer periods, issues emerge that challenge a systems engineering design team's ability to successfully integrate new technology solutions with existing legacy systems components utilizing integration principles that may have been adequate for first-time integration of the legacy as a whole, but alone will not suffice without significant risk to future system development. Those issues are hardware

functions replaced by software functions, legacy hardware functions replaced by advances in technology solutions, new software solutions containing more software lines of code and emerging cyber security threats that drive changes to system requirements. This thesis describes real life systems integration challenges and setbacks experienced as a result of one or more of these issues listed. Additionally, this thesis prescribes proven solutions that empower a design agent to address integration risks prior to obtaining a mature system design. Observing any integration risks subsequent to achieving a mature design will cause an integration failure and possibly system redesign.

D. SYSTEMS ENGINEERING OVERVIEW

1. Systems Engineering Defined

According to Systems Engineering Body of Knowledge (SEBOK), an online professional wiki, systems engineering (SE) is defined as, "an interdisciplinary approach and means to enable the full life cycle of successful systems, including problem formulation, solution development and operational sustainment and use" (SEBOK 2016). Another definition lectured by Langford (2015), "Systems Engineering is a discipline for solving problems by analyzing risk and value propositions, through a structured process that facilitates actions that account for available resources, lifecycle of the solution, and the lifecycle of the need."

2. Systems Engineering Development Lifecycle Explained

A full lifecycle is comprised of phases, each with its own overall specific purpose and tasks performed by an integrated product team (IPT) of experts with a common goal of successful design, development, test and deployment of a system. A lifecycle structure consists of phases within a methodology or process model. There are many different methodologies, each with its respective benefits and shortcomings. This thesis references the VEE process model. Figure 3 depicts this process model.

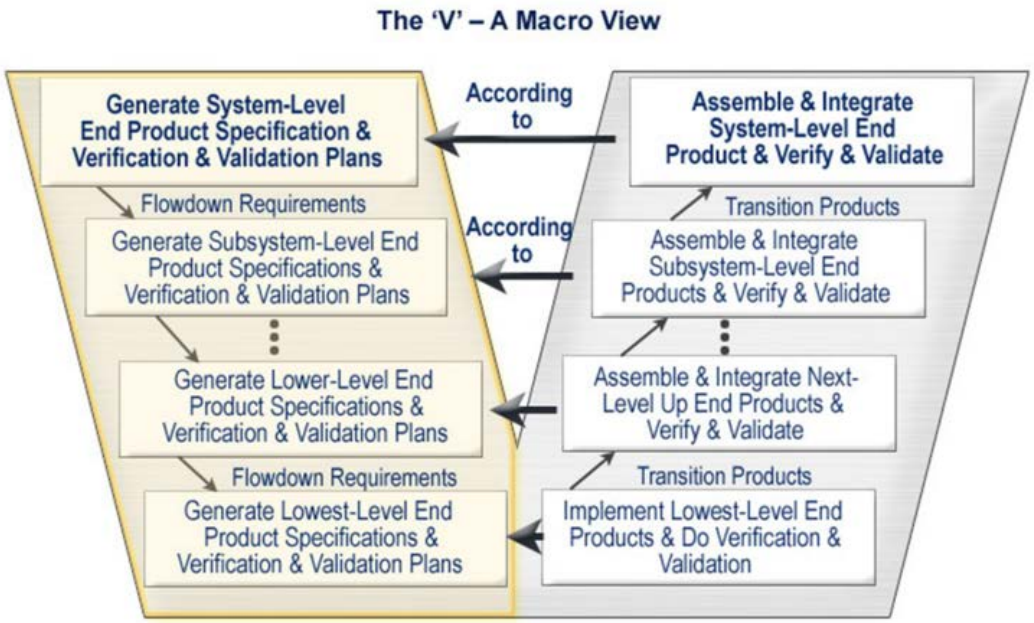

Figure 3. VEE Process Model. Source: SEBOK (2016).

3. Typical Phases within a Lifecycle

The VEE Process Model is only one of many different lifecycle process models; each model has benefits and shortcomings respectively. Each process model is comprised of phases, a logical separation of events within the process model usually separated by customer reviews. The purpose of these reviews are for the design team to demonstrate recent progress made on the development efforts for that system to the customer, to obtain concurrence from the customer to commence the next phase within the process model, and to discuss any design changes that had already been previously implemented.

E. SYSTEMS INTEGRATION EFFORTS WITHIN THE LIFECYCLE

> Systems integration is a component of the systems engineering process that unifies the product components and the process components into a whole. It ensures that the hardware, software, and human system components will interact to achieve the system purpose or satisfy the customer's need. (Grady 2010, 6)

Figure 3 depicts the execution of systems integration events on the right side of the model during the bottoms-up phases. Those events include verification of components, subsystems, and system; system validation, and finally commissioning of the system or implementation into the IOE.

The process of physically combining or integrating and testing objects together occurs on the right side of the VEE process model. This thesis does not refute this fact but emphasizes the importance of SI execution and planning to facilitate successful integration and testing that occurs on the right side of the VEE. Otherwise, a design agent finds him or herself re-designing components and/or interfaces while concurrently testing the system. Figure 3 shows systems integration as a concurrent effort along with verification and validation. These two events, within the lifecycle, occur prior to production. This thesis presents evidence to support the stance that execution of SI related tasks must occur early enough to ensure successful SI in the right side of the VEE model.

1. SI Functions that Occur on the Right Side of the VEE Model

The right side of the VEE Process Model lists the systems integration activities or physical systems integration that occurs after all hardware and software developed is complete. Integration of CIs creates assemblies, subsystems and eventually the functionality of the system verifies that the sum of the CIs operates as a whole system. Physical systems integration of a system occurs on the right side of the VEE process model (Grady 2010, 11). This thesis does not go into further description of what happens during physical SI.

2. Need for Systems Integration on the Left Side of the VEE Model

SI events shown on the right side of the VEE model are of great importance to ensuring successful SI. This process is a bottom up approach as indicated on the right side of the VEE Model depicted in Figure 3. When issues occur during the conduct of bottom up SI, these issues jeopardize successful implementation of the designed system. This research focuses on the implementation of solutions on the left side or top down

tasks of the VEE model prior to the execution of any physical SI. These solutions improve bottom up SI success.

Two experts agree the scope of integration is not limited to the physical integration process. Grady (2010, 11) states, "It appears we will have to do integration work throughout the development period. The author believes this to be true." He goes on to share that physical integration efforts do not solely comprise the entire integration effort. Intellectual integration activities occur during development and prior to the physical integration activity (Grady 2010, 11). In Langford's (2012, 19) book titled *Principles of Integration* he states in Principle 5, "Integration is a primary, key activity, not an afterthought considered as the result of development." Both Grady and Langford agree that the process starts during development, top-down phases approach and then proceeds with physical SI during bottom up phases of the lifecycle. This thesis proposes, in detail some of the SI top-down tasks that need to be conducted, by doing so will increase the likelihood of systems integration success.

F. PROPOSED METHODOLOGY

The preceding paragraphs have introduced to the reader the concept of systems engineering, its application via the use of a process model and the utilization of systems integration in the latter phases of a process model. The purpose of this thesis is to outline the need for systems integration efforts during the early phases of systems engineering development. This thesis focuses on specific task accomplishment during top down development. Later, it introduces implementation of solutions for these tasks. Lastly, this thesis details SI failures via case studies and traces proposed solutions to the observed failures. From a general standpoint, the proposed solutions for early SI are early design via systems architectural modeling, programmatic considerations, development considerations, and new practices.

G. RESEARCH QUESTIONS

The proceeding chapters address the following research questions in detail:

Research Question #1

What are the tasks executed during system development that assist in revealing issues prior to commencing SI?

Research Question #2

What are the systems engineering principles and tools that improve the likelihood of completing SI successfully?

H. SUMMARY OF SYSTEMS INTEGRATION

This chapter introduced the concept of systems integration, its common application during bottoms-up systems engineering phases, the problems with not executing systems integration efforts early in system development in support of the bottom up physical systems integration work and the challenges of integrating advanced technology system solutions with legacy systems objects.

The next chapter will begin to detail systems integration and the importance of executing early in the developmental phases within a systems engineering process model. Lastly, Pickar (2015, 3) states the importance of SI, “SI interprets the overall performance needs of a sponsor into technical performance specifications and ensures that system requirements are met.” Systems integration depends on successful testing of all system requirements.

II. SYSTEM DEVELOPMENT AND EMPHASIS ON SYSTEMS INTEGRATION

A. INTRODUCTION

The purpose of this chapter is to list and detail various aspects of systems integration (SI) considerations and work scope that need to occur during the systems engineering top-down development phases. That is, the developmental aspects of SI that must be analyzed and actions taken to ensure the designed solution can be successfully physically integrated the first time without the need for any systems redesign after development has been completed. That redesign involves analysis of potential impacts to other system components, and regression testing. Langford (2012, 4) identifies integration failure as, "attempting to integrate two objects where one or a combination of both requires an amount of rework that is more constrained by cost or time than starting anew, the result is failure to integrate." This thesis explores and proposes solutions for avoiding these types of failures.

B. STAKEHOLDER/CUSTOMER/USER NEEDS ANALYSIS

1. Who Are the Stakeholders and What are Needs?

A stakeholder is any person, group of persons or an organization with an interest in the system. A stakeholder is also any entity that influences the systems engineering: development, design, test, production, implementation and sustainment efforts and any associated business or policy decisions made by the integrated product team. Each stakeholder has "wants" and "needs." Each system requirement decomposes into multiple lower level design requirements. Blanchard and Fabrycky (2011, 48) state, "It is essential that one start off with a good understanding of the customer need and a definition of system requirements." Some examples of stakeholders are, project sponsor(s) or customer, users, developers, testers, and policy makers. Identifying all applicable stakeholders is the first step toward successful physical systems integration. Inadvertently missing a stakeholder and his or her associated needs will result in an incomplete set of system level requirements and any associated lower level design requirements. Missing

requirements can lead to an inadequate system design. Unfortunately, discovery of these inadequacies occur when attempting to accomplish integration.

A key stakeholder is an individual, group, or organization with the most influence or impact to the success or failure of the system. Langford (2012, 231) states, "Key stakeholders are those who represent the totality of the people who have various needs associated with the product or service that is to be built by systems engineers." Revisiting the cockpit electronics, systems integration depiction shown and discussed in Chapter I an example of a key stakeholder for that avionics electronics package is the group of users that will operate the sustained system. If the users were not included as stakeholders for designing the system then there is a greater chance the system will experience an integration failure. Addressing any failure of this kind affects the program schedule. If the impact is great, say in years then it affects the users operating the legacy system. This system will eventually experience an upward trend in equipment failures.

2. Importance of Conducting a Needs Analysis

Capturing the stakeholder and especially the user needs is of critical importance to understanding the entire problem space and to deriving a complete set of system level user requirements for consideration into the system design. Inadvertently overlooking users as key stakeholders or programmatically excluding them creates considerable risk that the designed and sustained system does not meet a complete set of user needs/ requirements and system redesign is imminent. Failure to redesign the system to meet user needs can eventually lead to the user changing the system to meet his or her mission needs.

> One of the biggest challenges … is the identification of the set of stakeholders from whom requirements should be elicited. Customers and eventual end-users are relatively easy to identify, but regulatory agencies and other interested parties that may reap the consequences of the system-of-interest should also be sought out and heard. Stakeholders can include the interoperating systems and enabling systems themselves, as these will usually impose constraints that need to be identified and considered. (INCOSE 2010, 59)

Stakeholder needs are the most important inputs into formulating a design solution to a problem and addressing all needs. Each need translates into a system level requirements that will trace downward into lower level subsystem and component requirements: “Identifying the problem and accomplishing a needs analysis in a satisfactory manner can best be realized through a team approach involving the customer, the ultimate user, the prime contractor or producer and major suppliers” (Blanchard and Fabrycky 2011, 58).

3. Impacts to Systems Integration

Langford (2012, 261) states, “A possible defect is missing a stakeholder of consequence.” Failing to identify a key stakeholder or consciously deciding to exclude a key stakeholder may cause dire consequences to the system design and create issues for cost and schedule. Overlooking a stakeholder or even missing a single stakeholder need affects the solution and creates deficiencies in the requirements.

Langford (2012, 260) suggests conducting a stakeholder analysis followed by the creation of scenarios that require potential stakeholder interactions in an effort to identify additional stakeholders that may have been overlooked during the initial analysis. A real world example of the consequences of not conducting adequate stakeholder analysis is included in the FBI Virtual Case File case studies.

4. Summary of Stakeholders and Needs Analysis

A decision by the customer (usually an individual or organization within the DOD with the authority to award contracts) not to conduct a user needs analysis can ultimately result in increased SI costs. An undiscovered or unknown set of user needs will result in a “failure to integrate” (Langford 2010). This thesis later discusses systems that experience SI failures due to an inadequate stakeholder analysis. Every stakeholder “need” eventually traces to one or more system requirements (Blanchard and Fabrycky 2011, 58). Other stakeholders may include regulatory agencies and owners of interoperating systems (INCOSE 2010). Finally, “integration is only as good as architecture captures stakeholder requirements” (Langford 2012, 15). Architecture derives from lower level requirements that support the system level requirements. Definitions of those system

level requirements originate from all stakeholder requirements or needs obtained from conducting a needs analysis.

C. SYSTEM BOUNDARIES AND INTERFACES

1. Introduction

Understanding and establishing all system boundaries during early development ensures coordination of all system-level interfaces. This activity involves integrating the new designed and tested system into its intended operating environment. That environment can be one or more of the following engineering test bed (ETB), a training facility or a tactical environment onboard a naval vessel, or aircraft. This is not an exhaustive list of operating environments in which installation, testing, and sustainment of the system occurs. Each operating environment comes with its own respective and possibly unique set of considerations for: constraints, EMMI needs, boundaries, and installed systems. These considerations affect the accomplishment of systems integration throughout all the systems engineering phases.

2. System Interfaces

Interfaces cross boundaries to connect components or subsystems together. They are essential for the conducting exchanges of EMMI between objects. Maier and Rechtin (2009, 10) state, “The architect’s greatest concern and leverage are still, and should be, with the systems’ connections and interfaces.”

As part of the development phases, requirements analysis traceability of all designed and legacy interfaces contributes to accurate and well-defined systems architecture models. Any legacy requirements considered for reuse or pull through should require analysis to determine applicability to the newly proposed IOE. It is risky to assume any legacy requirements are applicable as written to the new IOE. Unfortunately, the design agent does not expend enough resources to fully understand all interfaces; issues are discovered after the design solution is complete, which results in costly redesign efforts. New objects that interface with legacy objects must support integration.

Langford (2012, 33) states that, “objects that interact via one or more interfaces and create a binding relation with other objects are integrated.”

3. Impacts to Systems Integration

During physical systems integration efforts, the following events occur; combining objects into assemblies, subsystems and systems; individual object form, fit, and functional testing and, verification of interfaces between objects. This is an inconvenient time to discover physical incompatibilities between two or more objects. These incompatibilities are present during development, but usually not discovered until the conduct of SI. As a result, an SI failure occurs and the system requires redesign to address the defects. The design team must re-formulate the solution space to account for this issue, redesign objects to address the incompatibilities, and attempt physical systems integration a second time. Cost and schedule impacts are expected.

4. Summary of System Boundaries and Interfaces

For a newly designed system of interest (SOI) within a system of systems (SOS) architecture, boundaries and interfaces are a great leverage and a great concern (Maier and Rechtin 2009). Other systems within the SOS that exchange EMMI with the SOI will impose constraints on that system. It is crucial to the design team to not only understand the SOI’s customer needs and constraints but also that of any and all interfacing systems within the SOS (INCOSE 2010). Failure to consider and analyze the SOI boundaries and interfaces to other systems can result in integration failures and post development redesigning to address broken interfaces.

D. REQUIREMENTS DEVELOPMENT

1. Introduction

Requirements are the building blocks for designing a system. Completion of a customer/stakeholder/user needs analysis yields needs in the form a Request for Proposal (RFP). This document gives a general view of customer needs to the prospective design agent or contractor. Subsequent to awarding the contract, meetings are held between the design agent and the customer to further understand the stated needs from the RFP and to

elicit any other needs, being careful to eliminate any “wants” stated by the customer as being just that or potentially as a “need” that was mistakenly communicated as a “want” (Langford 2015).

Requirements will also come from the needs of the users or warfighter, obtained from conducting a user needs analysis. These users will operate and maintain the system or equipment in the field or the intended operational environment. Analysis of all customer and user needs will generate the highest set of requirements, the system requirements capture, and approval by the stakeholders prior to starting any design or lower level requirements generation. Traditional systems engineering accomplishes this via pen, paper and sometimes by using COTS software applications not necessarily ideal for requirements management. However, there are solutions to managing requirements; model-based systems engineering accomplishes this via models. Grady (2010, 276) suggests, “All requirements should be derived using models so all requirements should be traceable to a modeling artifact from which was derived.” It is important to ensure proper configuration management of all requirements documents. Any inadvertent change to even just one requirement has as cascading effect to all lower level requirements that trace to the modified system requirement. After completion of system design, any proposed system level requirements changes require analysis by all stakeholders. This analysis must be objective and any decision supported by accurate data.

From the system requirements, all lower level requirements will trace back to one or more system level requirements and down to individual configured items that meet the requirement. Application of traceability occurs in either a forward (or top-down) approach or a backward (or bottoms-up) approach (Figure 4).

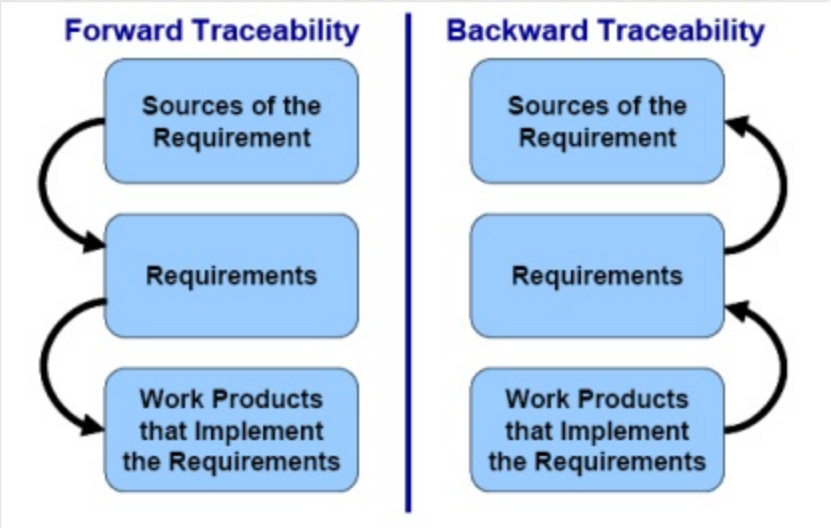

Figure 4. System Lifecycle Requirements Traceability. Source: ITABoK (2016).

2. Importance of Traceability

Failure to implement traceability between the hierarchies of requirements can create holes in the system design. In other words, the design solution will not address the entire problem space. Any holes in the requirements will later create holes in the execution of systems integration testing. SI test derive from the previously written and decomposed design requirements. Inadequate traceability of any design requirement to the lowest design documentation, affects the respective test procedure will not account for that untraced requirement(s). Manually tracing requirements between multiple levels of design documentation to multiple levels of testing documentation is very difficult to execute to a high degree of accuracy. Shchupak (2015, 42) states, “Full traceability is another key feature that is critical for successful systems engineering. The goal is to ensure that there is clear traceability from stakeholders’ needs to requirements to the design and to verification and validation.” There are software application tools that make requirements management executable and an integrated systems architectural framework can ensure traceability.

3. Importance of Using a Requirements Management Tool

Requirements Management tools such as Dynamic Object-Oriented Requirements System (DOORS) provides functionality for requirements documentation, generation for development and testing, revision control, traceability support, configuration management, and customer approval.

> In the requirements-centric approach, child requirements derive directly from parent requirements, thereby eliminating the possibility that a lower level document has parentless requirements (orphans). This approach would provide better visibility into whether a parent requirement has all of the child requirements that are needed to support eventual satisfaction of the parent requirement. (Perz 2006, 83)

4. Full Coverage of Requirements Testing for Risk Mitigation

All well written requirements are capable of being tested. Eliminating holes in requirements traceability reduces integration test failures and ensures collectively that all requirements documents provide 100% testing coverage.

SI testing must execute each design requirement at least once, and at the appropriate level. Testing of software requirements happens as part of the software configured item lower level testing, unless that software requirements traces to a higher-level systems requirement and allocated as part of the system level testing. There are, however, risks involved with delaying and allocating any requirements testing of a lower level requirement to a higher test. Any problems discovered later will leave little time for rework, retest, and result in schedule delays. Mitigating test risk by testing each requirement at the earliest possible opportunity leaves more time to recover, but since it might involve duplicate testing could cost more. Some overlap of requirements testing is expected, and the stakeholders should identify any high-risk requirements that require testing early and more than once. Model-based Systems Engineering ensures traceability between requirements and verification steps during test. This relies on models that capture requirements traceability as shown in Figure 5. Additionally, applying formal methods language to MBSE ensures specified requirements adhere to predefined constraints tested during verification and validation testing events.

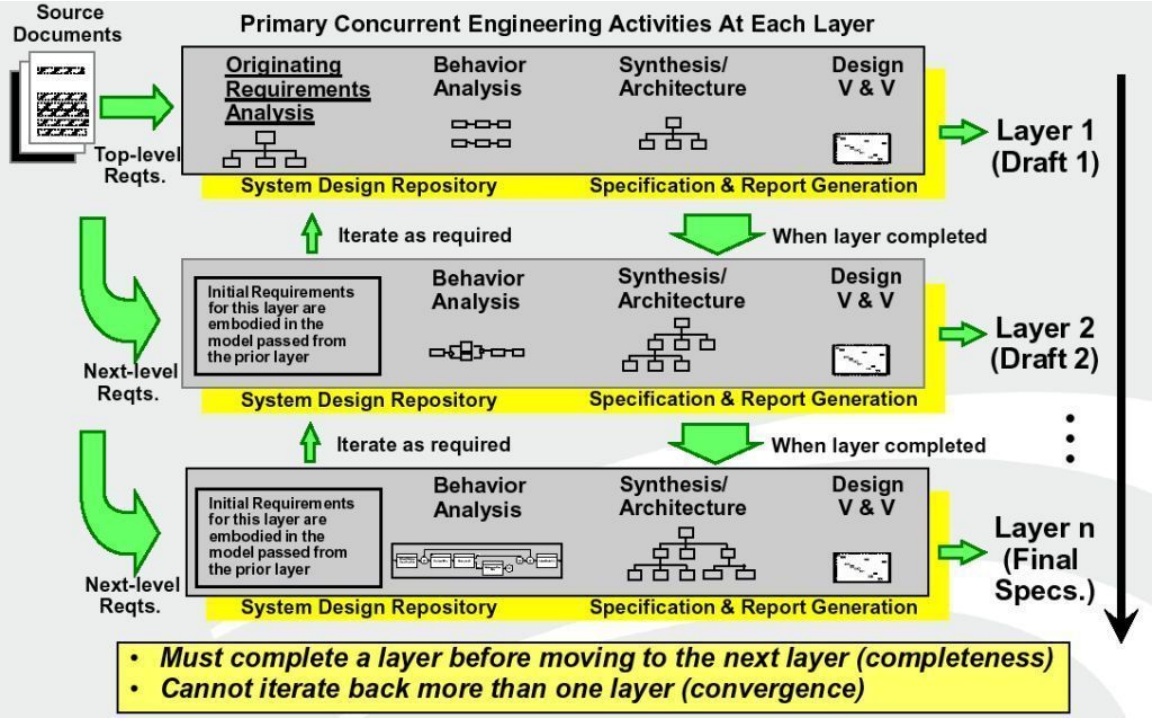

Figure 5. MBSE Requirements Traceability. Source: Giachetti (2015).

5. Impacts to Systems Integration

From the perspective of interfaces and interactions between objects, requirements specify the design of interfaces between objects. From the perspective of objects, hardware and software configured items, requirements documentation specify how CIs interact via interfaces logically and physically. Requirements documentation also needs to specify interactions or interfaces among integrated CIs. Having a concise set of system and lower level requirements will provide a solid baseline set of requirements needed for test engineer to author the test procedures. Any poorly written or undocumented requirement will create holes in the test procedures and contribute to unexpected test failures or assembly issues during integration. Additionally, Giammarco (2016) states, "A typical requirements statement defines what a system must do, but stops short of defining how it will be done." Systems architecture answers the "how" question. Table 1 lists some benefits to formulating well-written requirements.

Table 1. Benefits of Well-Written Requirements. Source: NASA SEH (2007).

Benefit	Rationale
Establish the basis for agreement between the stakeholders and the developers on what the product is to do	The complete description of the functions to be performed by the product specified in the requirements will assist the potential users in determining if the product specified meets their needs or how the product must be modified to meet their needs. During system design, requirements are allocated to subsystems (e.g., hardware, software, and other major components of the system), people, or processes.
Reduce the development effort because less rework is required to address poorly written, missing, and misunderstood requirements	The Technical Requirements Definition Process activities force the relevant stakeholders to consider rigorously all of the requirements before design begins. Careful review of the requirements can reveal omissions, misunderstandings, and inconsistencies early in the development cycle when these problems are easier to correct thereby reducing costly redesign, remanufacture, recoding, and retesting in later life-cycle phases.
Provide a basis for estimating costs and schedules	The description of the product to be developed as given in the requirements is a realistic basis for estimating project costs and can be used to evaluate bids or price estimates.
Provide a baseline for validation and verification	Organizations can develop their validation and verification plans much more productively from a good requirements document. Both system and subsystem test plans and procedures are generated from the requirements. As part of the development, the requirements document provides a baseline against which compliance can be measured. The requirements are also used to provide the stakeholders with a basis for acceptance of the system.
Facilitate transfer	The requirements make it easier to transfer the product to new users or new machines. Stakeholders thus find it easier to transfer the product to other parts of their organization, and developers find it easier to transfer it to new stakeholders or reuse it.
Serve as a basis for enhancement	The requirements serve as a basis for later enhancement or alteration of the finished product.

6. Summary of Requirements Development

Requirements traceability, when implemented correctly, can ensure all stakeholders needs trace through the system hierarchy to each respective: system, subsystem, assembly, and object solutions.

> Requirements definition challenge is compounded by the fact that development programs predominantly involve upgrades of existing systems. Even truly new systems have to interoperate with existing or "legacy" systems. Legacy requirements may be incomplete, ambiguous, out-of-date, in conflict with other requirements, or un-testable. Similarly, legacy architectures may not be sufficiently developed to support requirements or interface analysis. (Hoff 2009, 2)

> Creating a hierarchy of requirements with clear parent-child relationships would support efforts to verify a designed solution satisfies its requirements and eventually support design verification activities. (Perz 2006, 76)

E. SYSTEMS INTEGRATION TEST DEVELOPMENT

1. Introduction to Testing Development

The testing of design requirements happen during verification, validation and systems integration testing procedures. In a traditional systems engineering (SE) process, these procedures are conducted during physical systems integration, a bottoms-up approach. Ideally, the conduct of these procedures can be started during the SE development phases via virtual or simulation testing or through an iterative process model such as Agile. Requirements development, traceability and overall change management must be transparent to the testing engineers. This testing process will validate each design requirement; each requirement is also validated (Grady 2010, 277).

2. The Connection between Requirements Traceability and Testing Development

In addition to implementing requirements traceability, configuration control ensures all designers and testers are working from the same version of a requirements document. Ineffective configuration control of requirements documents can create disparities between design requirements documentation and the requirements testing procedures. It is crucial that designers and testers are using the appropriate versions of his or her respective documentation. Involvement of both the designers and testers are necessary when making any requirements changes after SI testing has commenced.

Test procedure generation and modifications should possess adequate agility to respond to rapid changes throughout early process phases. Hoff's statement alludes to

this need, systems requirements development should be an iterative process (2009, 23) and effective iterative communication between user and developers is required to successfully evolve a complete set of system requirements (2009, 64).

> In most cases, systems being acquired through the government's acquisition process are not complete, stand-alone entities. The newly acquired system will almost always need to fit into a larger operational architecture of existing systems and/or operate with systems that are being separately acquired. To be completely effective and suitable for operational use, the newly acquired system must interface correctly with the other systems that are a part of the final operational architecture. Integration testing, or SOS testing, verifies that the building blocks of a system will effectively interact and that the system as a whole will effectively and suitably accomplish its mission. (MITRE 2016)

In addition to the scenario described in the fore mentioned quote, Chapter IV lists other scenarios that require careful attention to system integration, some of which will drive requirements changes throughout development and beyond. External imposed on the stakeholders can force the designers to revisit and change existing requirements or write additional requirements in response to these factors. Encountering these scenarios requires the design team to communicate any requirements changes to the integration testing team as changes will influence what is tested and how.

3. Summary of Systems Integration Test Development

It is important for the design team to have a seamless traceability of requirements from the system level requirements; translated from stakeholder needs, to the lowest level of component or object level design. Testing conducted subsequent to design and executed in accordance with the allocated requirements for each object and groups of objects that create assemblies, subsystems and systems. Perz (2006, 81) explains the connection between traceability and testing, “clear traceability of lower-level requirements up to system-level requirements supports final validation, verification, and testing.”

III. IMPROVING SYSTEMS INTEGRATION THROUGH ADVANCED SOLUTIONS

A. EMPLOYING A SYSTEMS ARCHITECTURE FRAMEWORK

1. Systems Architecture Defined

Giammarco (2015, 23) states, systems architecture (SA) is the, "art and science of creating and building systems too complex to be treated by engineering analysis alone. That part of system development most concerned with scoping, structuring, and certification. It is a combination of the principles of both systems and of architecting." Inadequate systems architecture has caused systems integration failures as observed in the architecture used to design the DDG-1000 and Hubble Space Telescope. Langford states (2012, 276), "Architecture describes what the system does and generally how it does it." The act of designing a system, "brings order to misleading, ill-fitting and confounding data; at-odds opinions; differing values; and problematic convergence; architecture is a tool that allows us to tame complexity" (Langford 2012, 277).

2. Systems Architecture Ties to Systems Engineering

Systems engineering formulates a solution to a problem space; systems architecture improves the clarity of a problem space.

> In the process of accomplishing the problem space modeling work we will have developed insight into three things of interest: (1) knowledge of the entities of which the system should consist, (2) knowledge of the relationships (interfaces) between these entities, (3) knowledge of the requirements that apply to the entities and the relationships that should flow into specifications for the former and interface documents for the latter. (Grady 2010, 222)

The result should be a systems design that satisfies all aspects of the problem space. For some system problems, obtaining a feasible solution via systems engineering is adequate. For complex systems with complex problems, there is a necessary pairing of systems architecture with systems engineering. This pairing provides the stakeholders the ability to model the system via frameworks views. These framework views assist the architect and the engineering team to ensure proper and complete form of the system

from the preceding functions. SA viewpoints, as defined by the DOD Architecture Framework (DODAF) and depicted in Figure 6, are vital to establishing requirements that define, structure, function and relationships or interactions between objects (SEBOK 2016). "Architects select the viewpoints and models to develop based on the purpose of their architecture" (Pilcher 2015, 15). While the current version of the DODAF includes 8 Viewpoints, this thesis discusses the four primary viewpoints and provides an example for each below.

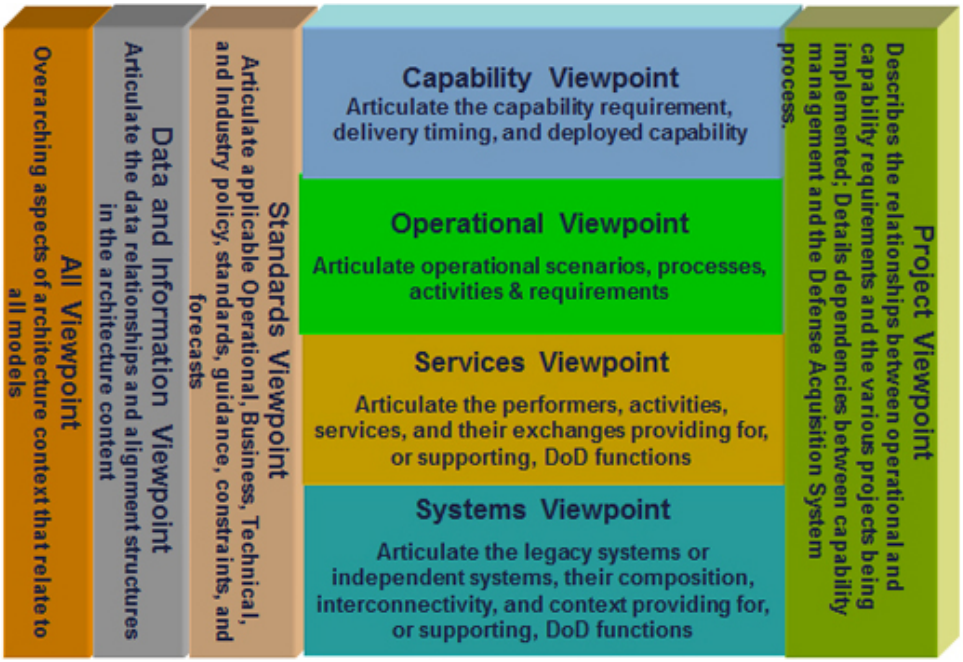

Figure 6. Systems Architecture Views. Source DOD (2015b).

a. Capability Viewpoints

These views depict the capability requirements; specifically they answer the questions, "Who or what receives it and when it is received by what." An example of one of these views is CV-2: Capability Taxonomy, which depicts a system's capabilities in a hierarchical timeline.

b. Operational Viewpoints

These views capture the operational scenario requirements and activities that support the capabilities and answers the questions of, "how, when and where?" An example of one of these views is the OV-2, Operational Resource Flow Description. This view depicts resources exchanges that occur between operational activities.

c. Systems Viewpoints

The systems views depict the interconnections within a system and between two or more systems. These views support the operational and capability requirements. An example of one these views is the SV-1, Systems Interface Description which depicts a system, its objects, and the interfaces those objects share.

d. Services Viewpoints

These views capture the exchanges between performers, activities and services that support the operational and capabilities functions. An example of one these views are the SvcV-2, Services Resource Flow Description, which depicts resource exchanges that occur between services. There are 51 models organized into eight categories of viewpoints. "The meaning of the different views, simply stated, is the operational views describes what a system does, the systems view describes how a system performs, and the technical view comprises applicable technical standards that constrain the solution" (Hoff 2009, 30). Tables 2–5 contain four of the eight for mentioned viewpoints and associated models with descriptions.

3. Contributions to Systems Integration

Systems are more complex than ever, and it is essential to implement systems architecture within the systems engineering process. System architecture facilitates a full understanding of all objects, the manner in which they interact and behaviors performed. Blanchard and Fabrycky (2012, 92) state the importance of architecture and interactions, "Architecture describes how various requirements for the system interact." SI is concerned with how objects interact with objects via interfaces. Figure 6 depicts DODAF architectural viewpoints. Collectively, these viewpoints "facilitate planning for

integration…predicts how each object will interoperate with the system (as a whole)" (Langford 2014, 173). No one model depicts the entire system but a suite of models is used to depict various aspects of a system and is used together to depict the entire system. Giammarco (2010, 523) states, "Architecture frameworks are employed to create, communicate consistent architecture descriptions."

> These views are vital in establishing requirements and are inputs to those responsible for defining the functions, structures, and relationships needed to achieve the desired product or service" (SEBOK 2016). When a design agent employs a set of views and associated models, it creates an integrated architecture one that should depict the system design as a whole. "Consequently, system design and architecture are profoundly important to integration. (Langford 2012, 174).

These architectural views, when modeled correctly, gives the design team a low fidelity overall systems model or system of systems model that depicts all objects that will go through physical systems integration. These architectural views serve as a roadmap for the design team during system development. When an architectural view depicts EMMI exchanges between, for example, objects A and B, the proposed design interface between these two objects, both objects must support that interface and its requirements. When the design team understands this interface from an architectural standpoint, it guides the documentation authors to ensure traceable and interoperable requirements that support the EMMI needs of objects A and B. The Vitech CORE Systems Architecture schema in Figure 7 displays the interface and relationships among design elements.

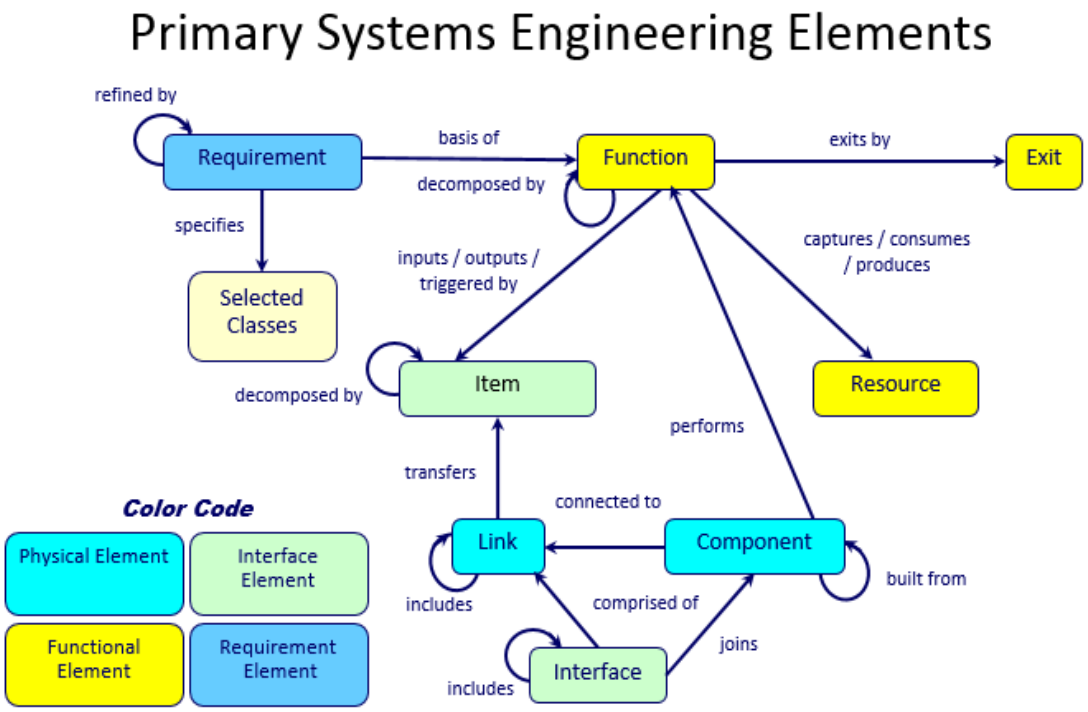

Figure 7. CORE's DODAF Version 2 Schema. Source: Vitech (2016).

4. Summary of Systems Architecture

Systems architecture (SA) describes what a system does and how it will do it and it also addresses systems complexity (Langford, 2012). SA gives the design team insight into entities, relationships and requirements that contributes toward specification and interface documents (Grady 2010). Requirements specify and document the system's design; design and architecture are very important to integration (Langford 2012).

> Systems architecting is part of the systems engineering design process that results in the partitioning of a system into components, the defining of interfaces among those components, and the processes that govern their change over time. This is a critical step in the acquisition of a system since it sets a framework and provides a roadmap for all the work that follows. More important is that systems architecting supports the holistic perspective of systems engineering and combines the art of balancing stakeholder concerns with the rigorous use of engineering analysis to handle complex problems that require a system solution. (Robinson 2013, 28)

B. UTILIZING MODEL-BASED SYSTEMS ENGINEERING

1. Model-Based Systems Engineering Explained

Complex systems with complex interfaces, functions and behaviors are difficult if not almost impossible to capture, trace, and analyze via document or paper centric SE manner. MBSE provides a better alternative to managing complex systems designs. Shchupak (2015, 18) clarifies, "MBSE does this by providing clear traceability between the products associated with each process." This traceability is captured in Figure 8; starting on the left side requirements trace to the operational, functional, and constructional or physical visions or views. Traditional SE modeling does not trace to these views as MBSE "enhances specification and design quality, reuse of system specification and design artifacts, and communications among the development team." Shchupak then quotes Friedenthal, Moore, and Steiner (2012, 15), "This focus on higher quality, reduction of rework, and improved communications, as well as the process driven approach, makes MBSE a powerful tool to support systems engineering management."

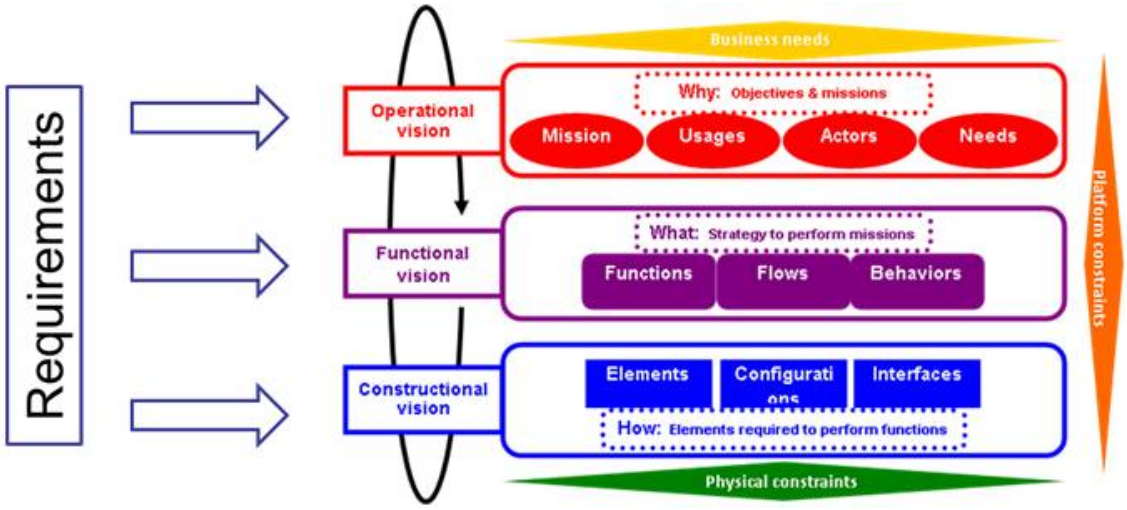

Figure 8. Requirements Inputs to Model-Based Systems Engineering. Source: OMG (2016).

2. Contributions to Systems Integration

Grady (2010, 55–56) writes, “There is much we do not understand about integration as it occurs inside the human mind.” Later, he discusses “Unexposed integration” issues. Grady (2010, 57) states, “Models can help us identify unexposed integration issues, and conscious thought about how the system will be used from both an operational and maintenance perspective within the context of these models will be helpful.” Pilcher (2015, 22) explains further emphasizes the importance of models, “Architects use models as tools to communicate the system requirements to the stakeholders for approval, verification, and validation of the system prior to its implementation. Iterative reviews of the models with the appropriate stakeholders provide for early discovery and correction of design issues.” Previously, this thesis introduced what systems architecture was and what contributions it made to systems engineering and to systems integration. Development and completion of the SA view models precedes any selection of forms, which is an object that satisfies a function and physical architecture. This is the essence of MBSE and one of the reasons why its application during SE developmental is garnering more attention by SE professionals. Another benefit to MBSE is its use to investigate design decisions without the need to commit to physical forms; “experimental investigation using a model yields design or operational decisions in less time and at less cost than direct manipulation of the system itself” (Blanchard and Fabrycky 2012, 172). Traditional models do not provide the same insight or benefits that MBSE offers. Table 2 provides a list of other MBSE benefits.

Table 2. Benefits of Using Model-Based Systems Engineering. Source: INCOSE UK (2016).

Benefit	Explanation
Reduced risk	• Improved cost estimates • Early and on-going requirements validation through inspection, and design verification through the use of simulation and automatic verification • Improved systems assurance • Fewer errors during integration and testing
Improved communications	• With project stakeholders • Between engineering disciplines • Across spoken language barriers
Improved quality	• Improved requirements specification and allocation to subsystems • Early identification of requirements issues • More rigorous requirements traceability • Enhanced system design integrity • Consistent documentation, both within and across projects
Increased productivity	• Improved impact analysis of requirements / design changes • Improved interaction across a multi-discipline team • Reuse of existing models to support design and technology evolution • Automated generation of documentation • Common definitions means changes are made in fewer places

Expounding on some of the benefits to utilizing MBSE, reduced risk means fewer errors during integration and testing. By utilizing MBSE, analysis of interactions between objects provide early detection of design errors that affect integration. Improved quality provides for, early identification of requirements issues. MBSE software tools are useful in testing requirements implementation within various architectural models such as functional flow block diagrams.

3. Summary of Model-Based Systems Engineering

> Models can help us identify unexposed integration issues, and conscious thought about how the system will be used from both an operational and maintenance perspective within the context of these models will be helpful. (Grady 2010, 57).

Grady refers to models generated by MBSE software application tools. Lastly, the modeling efforts are continuous throughout systems engineering process. Maier and Rechtin (2009, 12) assert, “From a modeling perspective, there is no stopping. Rather modeling is seen to progress and evolve, continually solving problems from beginning of a system’s acquisition to its final retirement.” Continuous modeling throughout development, equips the design team with a way to implement, test, and observe system responses to a proposed change usually driven by external factors, without the need to implement the change to any physical form.

MBSE provides early and detailed insight into object functionality and interactions with other objects; even for objects located in different systems. Traditional SE cannot replicate this level of insight and traceability using a paper-based method.

> In the process of accomplishing the problem space modeling work we will have developed insight into three things of interest: (1) knowledge of the entities of which the system should consist, (2) knowledge of the relationships (interfaces) between these entities, (3) knowledge of the requirements that apply to the entities and the relationships that should flow into specifications for the former and interface documents for the latter. (Grady 2010, 222)

Again, the main concern of systems integration is at the interfaces; object interfaces within the same system and object interfaces between different systems.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. SOLUTION FOR ADDRESSING COMPLEX SYSTEMS INTEGRATION PROJECTS

A. UNIQUE SYSTEMS INTEGRATION CIRCUMSTANCES THAT CAN BENEFIT FROM THE UTILIZATION OF ADVANCED SOLUTIONS:

There are certain systems engineering projects and scenarios that will warrant additional considerations by the stakeholders to ensure successful systems integration. The scenarios present challenges, when not addressed will produce integration risks that can lead to failures.

1. Integration of New and Complex System Solutions with Legacy Systems

Due to funding constraints, DOD systems are expected to remain in service for a longer than the initially planned life span. The life span of a system is referred to as the end of life (EOL); it is the duration between the times in which the first system is implemented, expressed as the initial operational capability (IOC) until the last system is removed from service, expressed as final operating capability (FOC). When extending a DOD systems' EOL, it creates obsolescence issues with the equipment used in the system.

Part of the development efforts for the system, logistics calculations determine reliability based on availability, mean time between failures (MTBF), and the customer's need for overall sustainment support period for the equipment. The sustainment period should be the same duration as the EOL. These calculations determine the procurement quantities of the equipment to keep the system running throughout the established EOL duration. These quantities include equipment actively sustained in the IOE and any spares to keep the system running should any sustained equipment fail.

The systems integration challenge occurs when the DOD program manager (PM) either proactively or reactively refresh system, within a SOS or system components within a system, in response to emergent obsolescence issues. Additional budget constraints prevent the PM from replacing an entire system but only parts of the system

or possibly some systems from a SOS. The integration challenges are the need to ensure requirements traceability, component interoperability and compatibility, component and interface function, and form between the legacy components and new components. It is imperative for advanced systems design solutions be utilized to properly architect the new and modified interfaces between the newly designed objects and the unchanged legacy objects are interoperable. This issue affects DOD systems for the DDG-1000 Legacy Missile System and Shipboard Data System.

2. External Factors that Impact System Requirements and Design

Cyber security threats are real and constantly evolving. As a result, DOD agencies charged with maintaining the security policies for DOD systems either reactively or proactively change security policies to protect the sustained systems. When the need arises to design a new system or refresh a sustained system, cyber security policies and associated requirements affect the system's design solution. Some of those security requirements will conflict with one or more system/subsystem requirements.

Sometimes the augmentation of new or existing yet modified security requirements or policies are ill timed and happens late in development. Thus, there is a need for ongoing integration and testing throughout development and into physical systems integration. Software is constantly subject to pressures of change (Brooks 1987, 3). Today, most of those pressures are from policy changes made external to the design agent and program manager's organizations. There also needs to implement and analyze policy driven requirements changes into the system via modeling prior to implementing the changes into the physical system. This issue is with the Shipboard Data System program driven by emergent changes to cyber security technical information guidelines (STIG) and overarching policies.

3. Summary of Circumstances Requiring Advanced Solutions

There are circumstances that when realized, increases the system's design complexity and associated integration efforts. When this happens, risks involving systems integration become more likely and with far worse consequences to the system. The next section will explore a solution for addressing integration challenges such as these.

B. USING FORMAL METHODS TO ANALYZE AND DESIGN SYSTEM INTERFACES

1. What are Formal Methods?

One proposed way to mitigate complex systems integration risks is the application of formal methods (FM) for solidifying systems architecture models. As defined in a lecture by Giammarco, a formal method is, "the use of formal notation to represent system models during program development with the goal of establishing system correctness via mathematical rigor" (2016). Use of these formal methods facilitates the need by stakeholders for a quick and more detailed analysis of system design changes affecting interfaces that can later compromise systems integration. As explained by Giammarco (2010, 522) formal modeling of systems architecture is necessary, "Using formal methods, stakeholders can decompose and express architecture data quality expectations unambiguously, and in a way that is abstract and independent of tool." Formal methods utilization is superior to traditional systems engineering paper-based methods used to evaluate emergent system design changes.

> Architecture can be modeled informally using such tools as viewgraphs, word processing documents, drawing tool diagrams, and unlinked spreadsheet tables. Because there is no programmed logic linking the data in and among these tools, opportunities to develop inconsistencies in such informally modeled architectures exist. Users of the architecture data are continuously engaged in the manually intensive effort of carefully coordinating the inevitable changes to the data. Capability to perform analyses (especially quick ones) is extremely restricted because it takes time to describe the data for different scenarios and keep the data in multiple views synchronized. (Giammarco 2010, 523)

2. What is Lifecycle Modeling Language?

Lifecycle Model Language (LML) is a modeling language that is useful to designers through the systems engineering development lifecycle. There are other modeling languages such as SYSML, but for the purpose of this research, the LML is the language of choice. Its simplicity of use derives from use of "everyday language" to define modeling elements depicted in systems architectural views. LML improves the effectiveness of MBSE models.

> LML takes the principles of MBSE beyond the system development and production stages into the conceptual, utilization, support and retirement stages by providing a robust easy to understand ontology that allows one to model the complex interrelationships not only between system components but between those components and programmatic artifacts such as schedules and risk management plans using clear diagrams to express system information. LML was designed to integrate all lifecycle disciplines, including program management, systems and design engineering, as well as test, deployment and maintenance into a single framework. As a result, LML is a language that can be used throughout the lifecycle. (LML 2015, 3)

LML utilizes axioms or statements explicitly written and used to derive an associated predicate logical statement or pattern. There are observable patterns throughout SE development. Some patterns are desirable and require enforcement such via modeling language. A desirable pattern is the need for every object or component to perform at least one function. The associated axiom for this pattern need is, “Every object shall perform at least one function”. Whereas other patterns observed are undesirable and detrimental to system development, like child requirement that traces upward to more than one parent requirement. LML can implement a contra positive axiom to avoid an undesirable pattern. An axiom that represents this need is, “Every child requirement will not trace more than one parent requirement”. This step assures that an axiom statement is unambiguous, and language or tool independent (Giammarco and Rodano 2013, 212). LML contains Ontologies that “provide a set of defined terms and relationships between the terms to capture the information that describes the physical, functional, performance, and programmatic aspects of the system” as shown in Figure 9 (LML 2016). The terms and relationships have similarities to those shown in Figure 7. SA uses terms for an “object” and “function” and formal methods uses the terms “performer” and “activity” respectively.

This researcher believes systems integration can be broken up into the following elements: interoperability, compatibility, functionality, form, and fit to include system boundaries. There are predefined pattern groups for interoperability and functionality; development of other patterns is required.

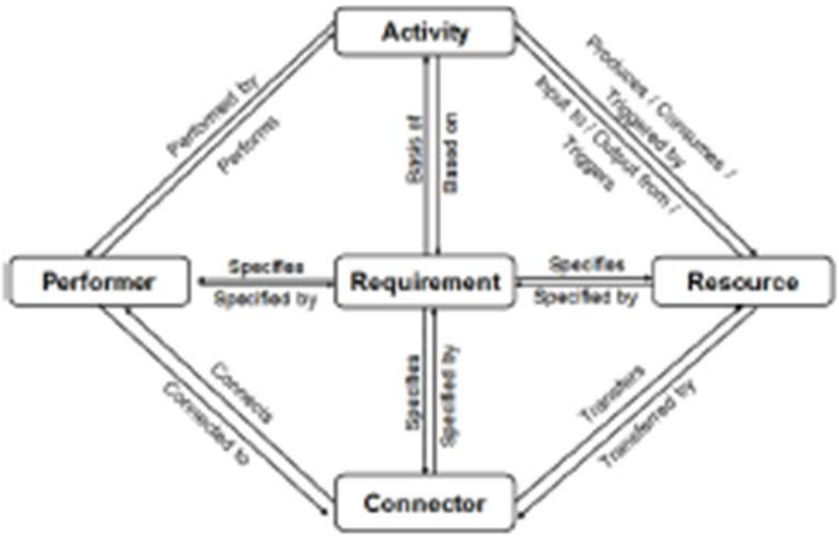

Figure 9. Class/Relationship Diagram. Source: Rodano and Giammarco (2013).

3. Phased Approach for Implementation of Formal Methods

There is cost to designing a system utilizing systems engineering principles and a process model. There is further cost to pairing systems architecture concurrently with systems engineering. There is also cost associated with the start-up or contracting of formal methods (FM). There are benefits to conducting formal methods; the extent in which one chooses to use it will vary. Integration of an SOI into a legacy system of systems architecture benefits from conducting FM, specifically on those interfaces that lie between the SOI and the legacy systems. If one prefers more correctness due to implementing a safety critical SOI, then perhaps all safety critical functions or activities in addition to interfaces or connections are analyzed using FM. The application of FM does not assume an all or nothing approach. The level of FM implementation depends on each SOI and its integration environment and associated challenges. FM establishes correctness of the architectural models and reduces the risk of redesign during or subsequent to SI.

4. Contributions to Systems Integration

Formal methods involve the use of patterns to machine or software testing to verify the correctness or completeness of systems architecture. An architect or engineer identifies the need for a pattern and develops an axiom statement. The axiom statement is similar to a requirements statement without the use of the word “shall.” For example, consider an electrical power system for onboard a submarine that provides power to mission critical systems that must be available 24 hours day. This power system contains redundant motor generators; one is always providing downstream electrical power to the critical systems while the other motor generator is running but not providing downstream power. Both generators share a common, downstream power-seeking transfer switch (PSTS) that controls which generator is primary and which is backup, based on user inputs. Again, these generators supply power to mission critical systems with a requirement of being supplied power 24 hours per day. This requires consideration when writing an accurate axiom statement.

The following axioms represent the aforementioned system and enforce a needed pattern or expected behavior of the PSTS:

If a user provides command input to the PSTS, then the PSTS will open the primary circuit if and only if power is available from the backup generator and if the backup circuit is closed.

And

If no power is available from the backup generator then the primary circuit remains closed and the backup circuit remains open.

The PSTS will verify that power is available from the current backup generator and the circuit between this generator and the critical system is open. If power is available, then the PSTS will close the circuit between the backup generator and the critical system. The old backup becomes the new primary or online generator. Then, the PSTS will open the new backup circuit. The order of events ensures there are no power interruptions to the critical system. The logic notation represents the mathematical version of the axiom; the final step is to test the logical notation using a software

application such as Alloy Analyzer. Auguston (2012, 5) explains this application as, “a model building tool that helps humans reason about models and construct a more complete set of assertions, bringing any undocumented or implicit assumptions / unexpected states that the system may enter to the attention of the modeler.”

“Formal methods can be used for verification at various stages of the architecture and design process, checking the realization of the entire system against its specification” (Berry 2002). “Formal methods can be use for setting and validating architecture model quality criteria, rather than assuming the criteria will implicitly be met by the systems engineering process or in the tools used in the architecture development and validation effort” (Giammarco 2010, 529). Validation and verification occur as part of the physical systems integration overall efforts.

C. SUMMARY OF FORMAL METHODS

Giammarco and Rodano (2013, 214) further explain the importance of formal methods usage, “By expressing the characteristics of a good system architecture in a formal manner, a modeled system architecture can be automatically analyzed quickly and efficiently to determine whether there are possible issues that would make the system difficult, or even impossible, to realize.”

Most design errors occur but few are identified early in development; prior to creating detailed designs. It is at this point of the systems engineering lifecycle that formal methods can provide significant advantages (Giammarco 2016).

By observing systems integration (SI) testing methodology and areas tested, one can observe patterns inherent to objects and interfaces. During design efforts, most objects need to adhere to these patterns. However, it is the one or few objects that are unique or inadvertently overlooked that will not adhere to the common pattern and will create integration issues.

THIS PAGE INTENTIONALLY LEFT BLANK

V. SYSTEMS INTEGRATION CASE STUDIES

This chapter focuses on specific instances of SI failures. In some instances, it is necessary to obscure the specific DOD program or system name to avoid classification issues.

A. SYSTEMS INTEGRATION CASE STUDIES FOR DOD SYSTEMS

The government and its contractors realized a need to replace the existing legacy deployed system (referred to as "System #1) used onboard the Ohio Class submarines. The legacy system architecture contains a pair of network servers with several remote access laptop clients. This system was facing hardware obsolescence issues and running unsupportable operating systems on the servers. Design, development, test and deployment of that system solution took over seven years to complete in support of first installation or accomplishment of the initial operating capability (IOC) onboard the submarine's operating environment. This systems engineer personally witnessed interface issues discovered during the conduct of physical SI and/or during system implementation of the IOE.

1. System #1 – Key Stakeholder Left Out of Development

a. Description of Problem

In the case of this program, a key stakeholder, a contractor that owned the legacy storage space that would host the new and more complex COTS based SOI was not included as a key stakeholder for the project. The government program manager made the decision to not conduct a stakeholder analysis and in so doing did not award funding for this contractor to contribute to development of the design solution, review of the design disclosure documentation or, at minimum share government-furnished information (GFI) regarding the legacy system to the design agent of the new system. This decision led to several challenges and setbacks involving integration and system implementation. One such challenge involved a lack of insight regarding minor systems architectural differences among the legacy platforms. That is, some locations receiving this new

system had unique configuration differences not adequately captured on the legacy drawings. Some legacy locations had rear door and post assemblies whereas others had rear panel assemblies.

b. Results

During the first instance of implementing the SOI into the IOE, an integration failure caused inadvertent damage to mission essential legacy System #1 equipment. Implementation/integration efforts halted for several days while the installation team conducted a root cause analysis (RCA) to determine the cause of this error. At the RCA out brief, it was determined that one of the causes of this integration failure was a lack of involvement by the owner of the legacy systems that would host System #1. As a result equipment damage, the ship's operational schedule was affected, and the project experienced a cost overrun.

c. Proposed Solutions from this Case Study

It is imperative to understand the criticality of performing a stakeholder analysis as part of SE development, even for legacy systems. This thesis emphasizes the importance of conducting a stakeholder analysis; this should be an iterative process conducted throughout SE development. Generate a list of stakeholders and analyze that list for potential impacts to the problem space and the associated design solution. Streamlining the analysis or exclusion of stakeholders to control cost likely leads to "a failure to integrate" (Langford 2012). Any system redesign caused by an SI failure negates some if not all cost savings expected from streamlining stakeholder analysis to include the exclusion of stakeholders.

2. System #2 – Reduced Stakeholder Involvement and Lack of Legacy Systems Requirements Analysis

a. Description of Problem

Hardware obsolescence drove the need to develop new safety critical system (System #2) to replace an existing legacy system. These are the programmatic decisions:

- Pursue an aggressive project schedule that posed risk to the program. As lectured by Dr. Langford (2015), “If you start a project with your back against the wall then you will fail.”
- Control project costs by involving the fewest number of stakeholders possible. The customer made a poor decision to minimize development costs by streamlining the list of involved stakeholder.
- A selection of COTS hardware for replacing obsolete hardware; lack of traceability between legacy requirements to the new COTS hardware

b. Results

There was a lack of requirements traceability and allocation between legacy hardware designs to the new COTS hardware selection criteria, which included environmental qualification testing requirements. Allocation of environmental test requirements to the COTS components and environmental testing did not occur. This resulted in component redesign and subsequent regression testing, which affected the project schedule and cost. The design agent did not accurately capture legacy problem space and incorporate into the new system requirements that drove design, and the implemented design violated critical safety requirements. Exclusion of the safety range stakeholders resulted in unallocated safety critical requirements into the system design. Due to the criticality of this system, SI failure mitigation was via a system redesign and subsequent SI regression testing.

c. Proposed Solutions from this Research

The results of the case study lead to recommending the conduct of stakeholder and requirements analyses during system development. Pulling through legacy requirements into a new system design involves risks. Risk mitigation includes legacy stakeholder involvement and requirements analysis and verifying traceability and interoperability between legacy and new objects. Eliminating the execution of or minimizing the efforts of these SE development tasks can likely cost the customer more money to redesign the system failing SI. Stakeholders did not verify interoperability between legacy and new requirements and legacy and new objects, which share common

interfaces. MBSE utilizes models for depicting and verifying requirements traceability and formal methods patterns have the capability to enforce interoperability between objects.

3. System #3 – Failure to Analyze Software Interfaces and Behaviors

a. Description of Problem

Direction was given to several government contractors to utilize an existing and sustained COTS-based system (System #3) to host two new contractor-developed software applications and additional COTS software items required to support these developed applications. The prime contractor and the subcontractors for each new application utilized a paper-based approach to systems design. The customer directed utilization of an SE Waterfall process model. These new software applications experienced interoperability issues during early software integration testing of the operating system (OS) image.

b. Results

Testers observed interoperability issues during early software integration testing. Ultimately, this issue drove changes to the operating system OS image and a subsequent image rebuild and regression testing to verify absence of the interoperability issues. This impacted the project schedule; one that was too aggressive to accommodate any delays.

Additionally, emergent security requirements imposed on the system drove requirements changes and a subsequent redesign of the software and system aspects to address the new security requirements.

c. Proposed Solutions from this Research

Utilizing MBSE would empower the design team to model any new or modified system interfaces and software behaviors between the legacy objects and the new objects. One way to accomplish is imposing FM constraint patterns to the models impacted by the requirements changes. In response to those changes, the design agent observes the behavioral changes. This model testing gives the design agent the flexibility to implement

various alternatives to implementing such a requirements change and make a decision on which implantation to execute prior to committing to a redesign physical hardware or software solution.

4. Other Integration Issues

a. Description of Problem

Functional circuit diagrams, interconnection drawings depict internal and external interfaces and the show exchanges of EMMI between systems, subsystems and components. During the development of various DOD systems not previously mentioned, drawing development execution was via traditional systems engineering document-based methods; no modeling application tools utilized to verify and test the functions and interconnections/interfaces depicted in these drawings. As a result, system installers and users discovered errors during implementation and sustainment the new system. Some of the following integration failures observed were due to incorrect spatial constraints.

- missing cables
- incorrect pinning of a cable assembly plug/connector
- missing signals routed by a particular cable or multiple signals on multiple cables
- incorrect terminal board and/or terminal board pins called out for a connector
- incorrect cable length
- incorrect landing points called out for cable lug(s) and/or connector
- hex bolts bottoming out before a specified torque value is reached
- a form that met all spatial fit constraints for some equipment configurations but not for all configurations

b. Results

Most errors encountered are discovered subsequent to production and manufacturing efforts have either started or been completed. Unfortunately, these resulted in corrections to the drawings, redesign, and rework to the affected hardware.

c. Proposed Solutions from this Research

Expose spatial constraint errors during functional and physical modeling development and testing. Once these models successfully complete testing, drafters utilize the model's output data for drawing creation. In addition, formulate and apply formal methods constraint patterns that specifically address spatial shortfalls.

B. NON-DOD INTEGRATION FAILURE CASE STUDIES

Integration failures also occur in non-DOD systems engineering projects. The following case studies document two different integration issues that are detectible during system development.

1. FBI Virtual Case File Project

Hoff (2009, 55) states this case study as, "An example of failure to engage stakeholders … a three year development contract to upgrade the Federal Bureau of Investigation's (FBI's) IT infrastructure and to design what was called the 'Virtual Case File' which would allow the FBI to move from its antiquated paper-based investigation and records to computer-based investigation and records." Hoff (2009, 56) goes onto state, "It appeared that among other issues the FBI used contractors as FBI stakeholders, not agency stakeholders themselves. The system developer in turn may not have exercised due diligence in validating the requirements. The program, nominally a $170M program, was cancelled and begun over with a new development contractor, eventually costing an additional two times the original program cost."

This thesis previously documents that users are stakeholders, stakeholders have needs, each need translates to a system requirement, and each system requirement has many lower level design requirements. Integration fails when the user's needs are not reflective in the system design. Stakeholder analysis, constant developmental involvement increases the likelihood the implemented system design integrates successfully.

2. Ariane 5

The Ariane 5 was a launch vehicle developed for the European Space Agency for the Solar Terrestrial Science Program. Dennis Buede (2006, 368) provides the following facts. On June 4, 1996, Arianne 5 veered off course and disintegrated 37 seconds after launch. The root cause was traced to a concurrent failure of both inertial reference systems specifically; the software caused this concurrent failure when it converted a 64-bit floating-point number to a 16-bit signed integer value. The systems architecture lacked functional redundancy. Specifically, the data conversion failed due to the floating-point value being too large for the 16-bit integer. The SRI processor executed a shutdown, causing the Arianne 5 to lose its inertial reference point and veer off course.

With the implementation of formal methods patterns for interoperability and MBSE functional and behavioral modeling, improves the design agent's ability of detecting integration issues created during SE development tasks.

C. SUMMARY OF CASE STUDIES AND SYSTEMS INTEGRATION ISSUES

The purpose of Chapter V was to provide examples of systems integration issues experienced by this author personally within the last eight and half as a DOD employee stationed at a prime contractor's site. The examples provided collectively infer the need for additional rigor in system integration activities during the system's design efforts to increase the likelihood of physical systems integration and overall systems engineering process success and do so without the need for system redesign.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. RECOMMENDATIONS AND CONCLUSION

A. RECOMMENDATIONS

1. Thorough Stakeholder Analysis Reduces Design Rework During Systems Integration

Specifically this research explored the importance of conducting a thorough stakeholder needs analysis. Stakeholder needs drive system level requirements generation. Each identified stakeholder can possess one or many needs just as each overlooked stakeholder has one or many needs.

Identification of each need and translation to a system level requirement contributes to a complete picture of the problem space and a more complete design solution. Requirements traceability starts with a stakeholder need and ends with one or many design requirements for the same object; many objects comprise systems architecture. Requirements statements are the building blocks to the system design solution that is integrated, implemented, and sustainable.

Do not assume any cost savings associated with attempting to cut funding or shortcut processes from the stakeholder needs analysis. Insufficient stakeholder analysis increases the risk of a partial design solution that becomes evident during systems integration or implementation into the IOE. Missing a need or a stakeholder with needs, affects the overall system design. Integration failures can occur when a system design fails to implement environmental and safety requirements derived from a stakeholder need. Executing a stakeholder analysis improves the overall design solution and physical integration efforts.

2. Requirements Traceability Improves Translation of Stakeholder Needs to System Requirements to System Design

Successful execution of a stakeholder needs analysis and translation into system requirements does not ensure a full design solution makes it to integration. Upward and downward requirements traceability can ensure that each lower requirement accounts for; each eventually takes on one or more forms. These forms take on the aspect of software

and/or hardware objects tested during systems integration. The test engineers will use the requirements that satisfy a form, to write test plans executed during integration. An ill-conceived requirements management and traceability paradigm can yield the same results as overlooking a stakeholder of need and again producing a partial design solution. “In the systems engineering world, poor requirements almost always lead to major schedule, cost, and performance problems downstream” (Eisner 2008, 203).

A disciplined approach to requirements traceability and configuration management improves the overall system design solution and the testing of that design depends on it. Utilizing systems architecture and formal methods can further assist in requirements modeling and testing and potentially provides the design agent with a more complete system for executing simulations prior to committing to any forms. This provides the design agent early detection of interoperability issues among software applications. Early detection of issues improves SI.

3. Utilization of a Systems Architecture Framework Improves Systems Integration for Complex Systems

Systems architecture via model-based systems engineering (MBSE) is required for integrating new system solutions into a legacy platform.

> Any U.S. Navy warfare system being considered for development or improvement must be integrated into existing architectures, whether or not those architectures are well documented. So, to some degree architecture is imposed on a proposed system long before a solution, i.e., design, is conceived. Even for an unprecedented system on an unprecedented platform, the sailors who man the ship, environmental and navigations standards, the weapons, the communications networks, and other interoperating platforms comprise an architecture into which the new system must fit. If the architecture is undocumented, it is incumbent upon the system developer to ensure accurate documentation is produced. If the architecture is documented, the adequacy must be assessed and any shortfalls addressed. (Hoff 2009, 76)

Finally, Robinson (2013, 28) makes this statement in his concluding remarks, “systems architecting supports the holistic perspective of systems engineering and combines the art of balancing stakeholder concerns with the rigorous use of engineering analysis to handle complex problems that require a system solution.” DOD programs

have proven increasingly more complex; this trend continues into the future. Utilizing an architectural framework provides the design agent with a method of modeling legacy system interfaces prior to designing new components to integrate into the legacy platform. Integration failures occur at the interfaces.

4. Implementation of Model-Based Systems Engineering Improves System Requirements, Design and Integration

MBSE provides a toolset to stakeholders for quantifying design options before committing to one. Having this capability will prove invaluable when responding to emergent changes to the system design driven by external influences such as higher authority directives. Regarding the use of MBSE, Tepper (2010, 17) states, "At the heart of MBSE is requirements traceability and enhanced communication. It also has the potential to improve decision making by providing accurate change assessments and by quantifying design options in terms of cost and risk." MBSE provides the design agent with a depiction of the proposed system design and a method for executing simulations and observing interaction among objects. Understanding behaviors or interactions among objects, reduces integration failures.

5. Incorporation of Formal Methods Patterns Enforces Systems Integration in Design Specifications

Formal methods utilization ensures models adhere to its system specification or requirements. Requirements implementation and structure supports the ability to respond to impacts from external factors. Giammarco and Rodano (2013, 211) discuss the use of formal methods to verify systems architecture against its specifications and its use for verifying connections between components or interfaces between objects. Both of these actions support successful systems integration. Formal methods alleviate some burden on the design agent to verify all requirements and constraints. It greatly improves the accuracy of the specification.

Systems engineers and designers need to utilize formal methods patterns prior to committing to any design solution forms. Formal methods patterns analyze proposed form interfaces in detail to facilitate early identification of systems integration risks.

Early risk identification can ensure successful systems integration and testing without the need for redesigning after the fact. Design agents should consider the use of the following formal methods patterns for addressing the challenges with integrating new advanced technology solutions with legacy systems, Activity Performance, Compatibility, Connections, Fit, Form, Input/output, Interoperability, and Requirements Traceability. Formal methods enforce functionality of objects and their interactions; successful integration depends on proper object functionality and interactions.

B. CONCLUSION

Maier and Rechtin (2009, 11) state, "When a system fails to achieve a useful purpose, it is doomed" or "When it achieves some purpose, but at an unfavorable cost, its survival is in doubt, but it may survive." Implementation of all or a subset of the fore mentioned recommendations improve systems integration success by reducing costly redesigns and improving the usefulness and survivability of the designed system.

C. OPPORTUNITIES FOR ADDITIONAL RESEARCH

Create a holistic system of systems level DOD architecture framework that includes considerations for new and legacy systems interfaces for hardware, software and human interactions.

Create a set of formal methods axioms for analyzing interfaces and behaviors between a proposed SOI architecture and the hosting legacy systems architectures within the same system of systems architecture.

Conduct cost estimation for implementing the following solutions such as SA framework, MBSE, and formal methods,. Compared that cost to the continued use of systems engineering utilizing legacy methodologies such as document centric development. Conduct research on the total cost savings of implementing these three solutions for more than one systems engineering project. Compare the total ownership and design costs for each alternative.

Conduct research on the feasibility of utilizing model-based systems engineering and formal methods patterns to test the accuracy of a printed wiring board (PWB)

architecture prior drawing development and production. Look for solutions that minimize the need to redesign a PWB due to an integration defect.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

Auguston, Mikhail. 2012. “A New Approach to System and Software Architecture Specification Based on Behavior Models.” Faculty and researcher publication, Naval Postgraduate School. http://calhoun.nps.edu/handle/10945/14782.

Beam, David F. 2015. “Systems Engineering and Integration As A Foundation For Mission Engineering”. Master’s thesis, Naval Postgraduate School. http://calhoun.nps.edu/handle/10945/47229.

Berry, D. M. 2011. “Formal Methods: The Very Idea Some Thoughts about Why They Work When they Work.” *Science of Computer Programming*, 42.1: 11–27.

Blanchard, Benjamin S., and Wolter J. Fabrycky. 2011. *Systems Engineering and Analysis*. Upper Saddle River, NJ: Prentice Hall.

Brooks, F. P. 1987. *No Silver Bullet: Essence and Accidents of Software Engineering*. IEEE Computer 20.4: 9–10.

Buede, Dennis M. 2006. “The Engineering Design of Systems”. 2nd edition. Hoboken: John Wiley & Sons.

Department of Defense (DOD). 2015. *DOD Architecture Framework Version 2.02, Change 1, Volume 2: Architecture Data and Models*. Washington, D.C.: Department of Defense Chief Information Officer.

Eisner, Howard. 2008. *Essentials of Project and Systems Engineering Management*. Hoboken, NJ: John Wiley & Sons, Inc.

Engineering.com. 2014. “Model-based Systems Engineering – Beyond Spreadsheets”. Wasserman. Accessed April 13, 2016. http://www.engineering.com/DesignSoftware/DesignSoftwareArticles/ArticleID/7352/Model-Based-System-Engineering--Beyond-Spreadsheets.aspx.

Estafan, Jeff A. *Survey of Model-based Systems Engineering (MBSE) Methodologies*. Accessed April 25, 2016. http://www.omgsysml.org/MBSE_Methodology_Survey_RevB.pdf.

Friedenthal, Sanford, Alan Moore, and Rick Steiner. 2012. Chapter 2: *Model-based Systems Engineering.* In A Practical Guide to SysML: The Systems Modeling Language, 15–27. Waltham, MA: Elsevier Inc.

Giachetti, Ronald E. 2015. “Systems Architecture and Design, DODAF.” In lecture, Naval Postgraduate School, Monterey, CA, fall quarter.

Giammarco, Kristin. 2010. “Formal Methods for Architecture Model Assessment in Systems Engineering.” *Conference on Systems Engineering Research*: 522–531.

———. 2012. “Architecture Model-based Interoperability Assessment.” Dissertation, Naval Postgraduate School. http://calhoun.nps.edu/handle/10945/14781.

———. 2015. “Systems Architecture.” In lecture, Naval Postgraduate School, Monterey, CA, fall quarter.

———. 2016a. “Formal Methods for Systems Architecting.” In lecture, Naval Postgraduate School, Monterey, CA, summer quarter.

———. 2016b. In an email message to the author via Mr. Parker on July 20, 2016. Dr. Giammarco revealed “related performers are connected by a common interface.”

Grady, Jeffrey O. 2010. *System Synthesis, Product and Process Design*. Boca Raton, FL: CRC Press.

Hoff, Patrick R. 2009. “Translation of User Needs to System Requirements.” Master’s thesis, Naval Postgraduate School. http://calhoun.nps.edu/handle/10945/4928.

INCOSE. 2010. *Systems Engineering Handbook, a Guide for System Life Cycle Processes and Activities*. Edited by Cecilia Haskins. San Diego, CA: International Council on Systems Engineering.

INCOSE UK. 2016. Z-Guide, “Z9: What is Model-based Systems Engineering”. Accessed May 3, 2016. http://incoseonline.org.uk/Program_Files/Publications/zGuides_9.aspx?CatID=Publications.

IT Architect Body of Knowledge (ITABoK). “Traceability Through the Lifecycle”. Chitchula. Accessed May 5, 2016. http://iasaglobal.org/itabok/capability-descriptions/traceability-throughout-the-lifecycle/.

Langford, Gary O. 2012. *Engineering Systems Integration, Theory, Metrics, and Methods*. Boca Raton, FL: CRC Press.

———. 2015. “Systems Engineering for Product Development.” In lecture, Naval Postgraduate School, Monterey, CA, winter quarter.

LML. 2015. *Lifecycle Modeling Language Specification.* Version 1.1. Accessed May 5, 2016. http://www.lifecyclemodeling.org/specification/.

Maier, Mark W., and Eberhardt Rechtin. 2009. *The Art of Systems Architecting*. Boca Raton, FL: CRC Press.

MITRE. 2014. *Systems Engineering Guide.* The MITRE Corporation.

NASA. 2007. *Systems Engineering Handbook*, Revision 1. Washington, DC, USA: National Aeronautics and Space Administration (NASA). NASA/SP-2007-6105.

Object Management Group (OMG). 2014. Object Management Group. "MBSE Wiki" Object Management Group. Accessed April 27, 2016. http://www.omgwiki.org/MBSE/lib/exe/fetch.php?cache=&media=mbse:alston_asap_high_level_flowb.jpg.

Oravec, Joseph J. 2014. "DDG-1000 Missile Integration: A Case Study" Master's thesis, Naval Postgraduate School. http://calhoun.nps.edu/handle/10945/41425.

Perz, Michael. 2006. "Integrating Stakeholder Requirements Across Generations of Technology" Master's thesis, Naval Postgraduate School. http://calhoun.nps.edu/handle/10945/2614.

Pickar, Charles. 2015. "Systems and Project Management." In lecture, Naval Postgraduate School, Monterey, CA, summer quarter.

Pilcher, Joanne D. 2015. "Generation of Department of Defense Architecture Framework (DODAF) Models Using the Monterey Phoenix Behavior Modeling Approach" Master's thesis, Naval Postgraduate School. http://calhoun.nps.edu/handle/10945/47314.

Robinson, Chris. 2013. "Big 'A' Systems Architecture from Strategy to design: Systems Architecting in DOD". Defense AT&L. March–April 2013.

Rodano, M., and K. Giammarco. 2013. "A Formal Method for Evaluation of a Modeled System Architecture". Proceedings of the 2013 Complex Adaptive Systems (CAS) Conference. Baltimore, MD.

Saunders, Steve. 2011. "Does Model-based Systems Engineering Approach Provide Real Program Savings? – Lessons Learnt". Informal Symposium on Model-Based Systems Engineering DST, Edinburgh, South Australia. http://www.omgsysml.org/Does_a_MBSE_Approach_Provide_Savings-Lessons_Learnt-Saunders-200111.pdf (accessed May 2, 2016).

Systems Engineering Body of Knowledge (SEBOK). "Guide to the Systems Engineering Body of Knowledge (SEBOK)," in BKCASE Editorial Board. 2016. *The Guide to the Systems Engineering Body of Knowledge (SEBOK)*, v. 1.6. R.D. Adcock (EIC). Hoboken, NJ: The Trustees of the Stevens Institute of Technology ©2016, *Released 23 March 2016, http://SEBOKwiki.org/w/index.php?title=Guide_to_the_Systems_Engineering_Body_of_Knowledge_(SEBOK)&oldid=52176 (accessed 02 May 2016 17:32:09 UTC).*

Shchupak, Peter. 2015. "Study of Software Tools to Support Systems Engineering Management" Master's thesis, Naval Postgraduate School. http://calhoun.nps.edu/handle/10945/45942.

Tepper, Nadia A. 2010. “Exploring The Use Of Model-Based Systems Engineering (MBSE) To Develop Systems Architectures in Naval Ship Design” Master’s thesis, Naval Postgraduate School. http://calhoun.nps.edu/handle/10945/24368.

Vitech Corp. “System Definition Language.” http://www.vitechcorp.com/resources/core/onlinehelp/desktop/Key_Concepts.htm (accessed March 23, 2016).

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California