# AGILE AND COLLABORATIVE SYSTEMS ENGINEERING

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF INFORMATICS
OF
THE MIDDLE EAST TECHNICAL UNIVERSITY

BY

EMRAH AŞAN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
IN
THE DEPARTMENT OF INFORMATION SYSTEMS

JANUARY 2014

**AGILE AND COLLABORATIVE SYSTEMS ENGINEERING**

Submitted by **EMRAH AŞAN** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy in Information Systems, Middle East Technical University** by,

Prof. Dr. Nazife BAYKAL _____
Director, **Informatics Institute**

Prof. Dr. Yasemin YARDIMCI ÇETİN _____
Head of Department, **Information Systems**

Prof. Dr. Semih BİLGEN _____
Supervisor, **Electrical and Electronics Engineering, METU**

**Examining Committee Members:**

Prof. Dr. Onur DEMİRÖRS _____
Information Systems, METU

Prof. Dr. Semih BİLGEN _____
Electrical and Electronics Engineering, METU

Assist. Prof. Pekin Erhan EREN _____
Information Systems, METU

Assoc. Prof. Dr. Altan KOÇYİĞİT _____
Information Systems, METU

Assist. Prof. Dr. Ömer Özgür TANRIÖVER _____
Computer Engineering, Ankara University

**Date:** 10 January 2014

**I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.**

**Name, Last name: Emrah Aşan**

**Signature :**

## ABSTRACT

AGILE AND COLLABORATIVE SYSTEMS ENGINEERING

Aşan, Emrah
Ph.D, Department of Information Systems
Supervisor: Prof. Dr. Semih Bilgen

January 2014, 116 pages

In this study, an agile and collaborative systems engineering approach is developed with specific focus on system of systems challenges. First, agility related problems are identified by an interview-based exploratory survey. Afterwards the novel approach is formulated to address agility problems in system of systems environments. The approach is based on eliciting existing knowledge, visualizing it by immediate modeling, verifying consistency of the knowledge by continuous review, and identifying and planning for missing knowledge. Agility is mainly targeted with 1) performing these activities concurrently in modeling workshops, where operational, functional and physical aspects of the solution are developed simultaneously; and 2) with the collaboration of cross domain/organization knowledge sources. Continuous customer training is another tool adopted for enhancing collaboration effectiveness with the customer. The proposed approach was implemented and assessed in a case study in which its potential to increase agility of the systems engineering process in system of systems projects was validated.



Keywords: System of Systems Engineering, Agile Systems Engineering, Model-based Systems Engineering.

# ÖZ

ÇEVİK VE İŞBİRLİKÇİ SİSTEM MÜHENDİSLİĞİ

Aşan, Emrah
Doktora, Bilişim Sistemleri Bölümü
Tez Yöneticisi: Prof. Dr. Semih Bilgen

Ocak 2014, 116 sayfa

Bu çalışmada, özellikle sistemlerden meydana gelen sistem projelerinin sorunlarına odaklanan çevik ve işbirlikçi bir sistem mühendisliği yaklaşımı geliştirilmiştir. İlk aşamada mülakata dayalı keşifçi anket yöntemi kullanılarak çeviklik ile ilgili sistem mühendisliği problemleri tespit edildi. Daha sonra, sistemlerden meydana gelen sistemlerin geliştirilmeye çalışıldığı ortamlardaki çeviklik problemlerini hedefleyen yeni bir yaklaşım tasarlandı. Yeni yaklaşım mevcut bilginin ortaya çıkartılması, modelleme ile görselleştirilmesi, bilginin tutarlılığının devamlı olarak gözden geçirilmesi ve eksik bilginin tespit edilerek elde edilmesinin planlanması adımlarına dayanmaktadır. Çeviklik 1) bütün bu adımların, çözümün operasyonel, fonksiyonel ve fiziksel boyutlarının aynı anda geliştirildiği modelleme çalıştaylarında eşzamanlı olarak uygulanması ve 2) farklı alan ve organizasyonlardan bilgi kaynaklarının işbirliği ile hedeflenmektedir. Yeni yaklaşım müşteri ile işbirliğinin etkinliğinin artırılması için devam eden müşteri eğitimlerini de kullanmaktadır. Önerilen yaklaşım bir örnek olay çalışmasında uygulanarak değerlendirilmiştir. Sonuçlar önerilen yöntem ile sistem mühendisliğinin çevikliğinin arttırılabilecegini teyit etmektedir.

Anahtar Kelimeler: Sistemlerden meydana gelensistemler için sistem mühendisliği, Çevik sistem mühendisliği, Model-tabanlı sistem mühendisliği.

*To Semih and Suna*

## ACKNOWLEDGEMENTS

I would like to thank;

My supervisor Prof. Dr. Semih Bilgen for his valuable supervision and insightful comments throughout this thesis. I am grateful to him for his guidance and positive attitude which encouraged me to do my best.

My wife Nuran Aşan for her love, understanding and support. A simple thank you will never be enough.

My son Semih and my daughter Suna for their patience that they have shown, particularly in times when I was not with them.

Prof. Dr. Onur Demirörs and Assoc. Prof. Dr. Cüneyt F. Bazlamaçcı for their critical remarks and guidance during the whole period of my research.

This thesis could not have been completed without the support of my company Airbus Defense and Space.

I would like to acknowledge Oliver Allbrecht (BMW), Aysegul Dersan-Czeslik (EADS), Judith Dahmann (MITRE), Garry Roedler (Lockheed Martin), Ryszard Bil (Cassidian) and Pascal Pezzani (Cassidian) who shared their views and experience with me throughout this research.

## TABLE OF CONTENTS

## LIST OF TABLES

TABLE

## LIST OF FIGURES

FIGURE

## LIST OF ABBREVIATIONS

| | | |
|---|---|---|
| AF | : | Architecture frameworks |
| C2 | : | Command & Control |
| C2ISR | : | Command & Control, Intelligence, Surveillance and Reconnaissance |
| CMMI | : | Capability Maturity Model Integration for Development |
| ConOps | : | Concept of operations |
| COTS | : | Commercial off the Shelf |
| DoD | : | Department of Defense |
| DODAF | : | Department of Defense Architecture Framework |
| HL | : | Highlight |
| INCOSE | : | International Council on Systems Engineering |
| IPPD | : | Integrated product and process development |
| KK | : | Known Known |
| KU | : | Known Unknown |
| LSI | : | Lead Systems Integrator |
| MBSE | : | Model-based Systems Engineering |
| MODAF | : | Ministry of Defense Architecture Framework |
| MoI | | Ministry of Interior |
| NAF | : | NATO Architecture Framework |
| OODA | : | Observe, Orient, Decide, Act |
| PBE | : | Platform-based engineering |
| SDR | : | System design review |
| SE | : | Systems Engineering |
| SKS | : | Solution Knowledge Space |
| SoS | : | System of Systems |
| SoSE | : | System of Systems Engineering |

| | | |
|---|---|---|
| SPL | : | Software Product Lines |
| SysML | : | System Modeling Language |
| TBD | : | To be determined |
| UAV | : | Unmanned Airborne Vehicle |
| UML | : | Unified Modeling Language |
| US | : | United States |
| UU | : | Unknown Unknown |
| WBS | : | Work breakdown structure |
| XP | : | Extreme Programming |

# CHAPTER 1

# INTRODUCTION

In systems engineering (SE), it is very critical to understand the needs of the customer. Some systems engineers prefer using "problem domain" and "solution domain" terms in order to emphasize the importance of the effort on understanding the goals/needs/problems of the customer in separation from the solution design. Monitoring, analyzing and understanding the operating environment of the customer and the threats in that environment are crucial in understanding the need and proposing a proper solution. In the military domain, every change in the mission environment is an early sign to look for threats and/or opportunities. Both the military people and the system companies monitor the environment for changes so that a threat or an opportunity can expose a need for a new system.

For the last two decades, the speed of the change in the business environment is so high that in most of the projects, major changes (changes in the requirements, scope, technology and even the concept) occur within a single project's life cycle (Highsmith & Cockburn, 2001).

From the systems engineering point of view, the changes, therefore the threats, are not limited to the operating environment of the customer. Projects and organizations are the operating environments of the systems engineers. This environment is also changing (Boehm, 2006; Bosch & Bosch-Sijtsema, 2010; Gilb, 2005; Kennedy & Umphress, 2011; Turner, 2007). Size and complexity of the projects/systems, increased use of COTS and subcontractors, global team structure, interoperability issues with legacy systems, re-use considerations and large number of stakeholders with conflicting interests are only a subset of the problems resulting from the changes in the systems engineering environment. Systems engineers are required to deal with the changes and the challenges arising from both environments.

Well-established SE approaches are becoming more inadequate as system projects are becoming increasingly complex, global, software intensive, COTS/re-use based and evolving. Pitfalls of the traditional system development approaches have already been addressed in software engineering and several agile development methodologies have been developed. Agile methodologies such as Extreme Programming (Beck, 2001), Adaptive Software Development (Highsmith, 2000), Crystal (Cockburn, 2006), Scrum (Schwaber & Beedle, 2001) and DSDM (Consortium, 2008) have gained mainstream status in the software development field.

Identifying the need for agile SE is also not new (Batra, Xia, VanderMeer, & Dutta, 2010; Kennedy & Umphress, 2011; Shatil, Hazzan, & Dubinsky, 2010; Stelzmann, Kreiner, Spork, Messnarz, & Koenig, 2010; Turner, 2007). However, studies in this area are strongly affected by the agile software methods and mostly consist of the effort to adopt agile software engineering methodologies in the SE area (Stelzmann, 2012).

Business layer has already identified the need for agility in the development and acquisition of systems. Customers aim to acquire capabilities in a faster way by integrating independently useful systems into a SoS that delivers unique capabilities (ODUSD(A&T)SSE, 2008). Today we are talking about large scale and complex SoS in almost all sectors. Especially in the defense environment, growing number of military capabilities are achieved through SoS approach and this trend is likely to continue. In response to the challenges of engineering complex systems by networking independently designed and independently managed systems, a new discipline of System of Systems Engineering (SoSE) has evolved (Maier, 1998).

SoS integrators or lead systems integrators (LSI) do not have full control over the constituent systems. Constituent systems are constraints on the SoS architecture. Understanding the constituent systems becomes a critical SE activity for the system of systems integrators. Under such a scenario, the level of detail of the information captured in terms of architectural knowledge is much less than that relevant for the detailed development of each individual system in the SoS. As a result, SoSE activities in this environment are more of a knowledge management kind than top-down design activities.

Since many of the constituent systems are already developed and serving different customers in dealing with their operational problems, vast knowledge regarding the missions, operational activities and supporting system functions are readily available for the LSI. This turns into a challenge and a disadvantage if the LSI cannot manage such a large volume of information in a rapid and timely manner. Therefore, in this environment, systems engineers do not only face challenges to find out ways to adopt SoSE to handle SoS specific problems but also have to improve agility in their SE processes.

### 1.1. Motivation for the Study

It can no longer be an excuse for the systems engineers to just say that agile is not for large scale complex system projects, but it is for small scale -purely- software projects. The real problem is not the adoption of the current agile methods in large scale system projects, or it is not the problem of tailoring the light weighted software engineering methods to SE. The real problem is that we need agility in SoSE, we need agile SE approaches and methodologies to be applied in complex, large scale, software intensive SoS projects. Hence this study aims to address the problem of achieving agility in large scale SoSE, not only at a theoretical level but also in terms of practical and applicable techniques.

The new agile SE approach to be developed shall obviously make use of the agile software engineering principles which are already proven in some context. However,

special attention should be paid for adopting some techniques and methodologies from the architecture-driven development, knowledge management and Model-based Systems Engineering (MBSE) fields.

### 1.2. Research Objectives

The ultimate goal of this research is to define and implement an agile SoS SE approach. The term "approach" corresponds here to a set of related activities, techniques, and conventions that implement one or more SE processes and is generally supported by a set of tools (Friedenthal, 2011). Our understanding of "agile" is scoped by the agility attributes that are defined for the SE processes: Flexibility, learning attitude, focus on customer value, short iterations delivering value, continuous integration and test driven development, lean attitude and team ownership (Turner, 2007). Therefore, our aim is to define and implement an approach which can improve one or more agility attributes in a SoS environment. Following research objectives are formulated:

**RO1:** Identify and validate agility related issues and challenges of the traditional SE approach in SoS projects,

**RO2:** Formulate and qualitatively confirm a SE approach with improved agility attributes.

As previously stated, agile methodologies have gained mainstream status in the software development field. Therefore, our first proposition is based on the existing agile software methods:

**Proposition 1:** Traditional SE approach can adapt techniques and principles from the existing agile software engineering literature and these principles can be applied in SoS environments.

Agility is the common term that is used to refer to the problems of the traditional SE. However, one future trend is considered very important in trying to improve SE practices because it completely changes the environment, scope and the stakeholders of the SE processes. It is quite often that systems of systems are constructed by the integration of the independently developed systems. Such systems are not developed by a single supplier. Systems are developed in a more collaborative approach lead by the lead systems integrator (LSI) company that is responsible for the overall delivery of the SoS level capabilities. Collaborative systems development is not a scaled version of development with a lot of subcontractors. It is a completely different concept. Therefore, the new SE approach equipped with more agility attributes should present such agility in a collaborative systems development environment which is different than developing a system internally.

(Bosch & Bosch-Sijtsema, 2010) highlights a need for transition from traditional integration centric development to a more composition based development. They put the integration centric approaches to one extreme of the system development spectrum and the open ecosystems to the other end. Successful software product lines are stimulating environments for software ecosystems. Such SPLs turn into platforms for external, 3rd party developers. Our second proposition is based on the observation that SPLs and open ecosystems present the state of the art of

collaborative software development. In fact the platforms in the SPL ecosystems are very similar to the architectures in the SoS projects:

**Proposition 2:** Architecture-driven and collaborative approaches can improve agility attributes of SoSE.

Based on our research objectives and the propositions, we have the following research questions:

**RQ1:** What are the agility related issues and challenges of the traditional SE approach in the SoS projects?

**RQ2:** Can the agile software principles and processes be adapted to increase the agility of the traditional SE approach?

**RQ3:** Is it possible to improve agility by adapting an architecture-driven and collaborative SE approach?

### 1.3. Research Plan and Methodology

The research was planned in two main stages: first, to uncover the agility related problems in military SoS projects and then to address some of these with a novel approach with improved agility attributes. Overall research plan and methods are summarized in Figure 1 below. The dashed lines between the research objectives, research questions and research steps indicate their relationship. For example, research question 1 is defined based on the research objective 1. Literature survey is one of the research methods that will be used in studying the research question 1. Similarly arrows between the research steps and the outputs indicate input/output relationships which are explained in the following paragraphs. Following main activities constitute the research steps:

- literature survey,
- exploratory, interview-based survey,
- formulation of the new agile approach, and
- validation through a confirmatory case study.

Literature survey aims to identify and prioritize the issues and challenges with the traditional systems engineering approach in the large scale software intensive system development environments. The output of this phase will be the prioritized list of current issues and challenges of the traditional SE approach.

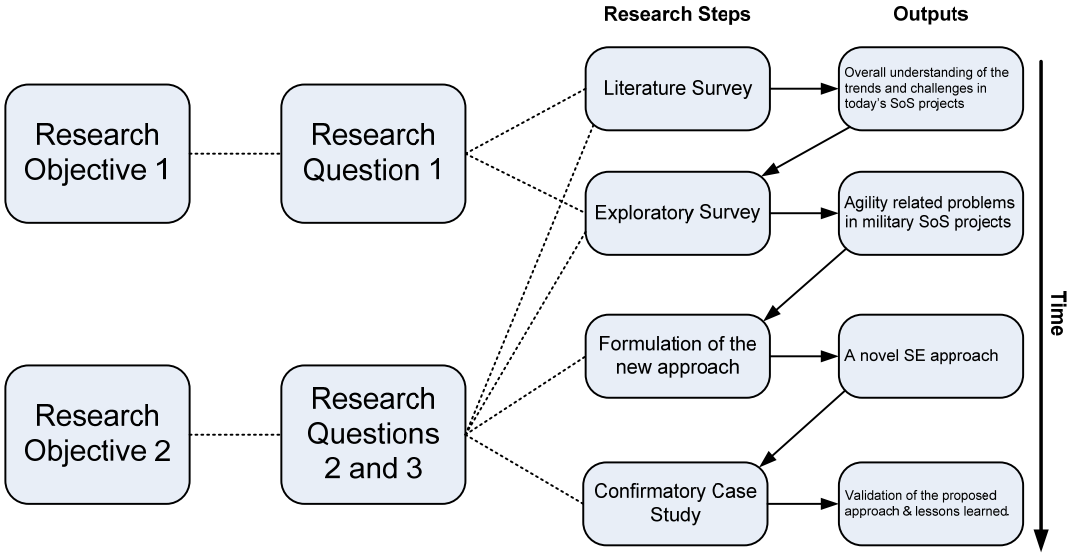


Figure 1: Research Plan

In the next step, we made an empirical start in identifying problems of the traditional SE approach. The intention was not to study all kinds of problems but the ones that are related to agility attributes. Survey research, which allows researchers to collect information from a specific population, usually, but not necessarily by means of a questionnaire or interview, is selected as the empirical method (Robson, 2011). During the analysis of the findings improvement objectives were defined to be used in formulating the new approach.

A novel agile SE approach will be formulated to mainly address the agility problems uncovered during the survey research. Formulated steps will also be implemented and assessed in a confirmatory case study. Evaluation of the proposed approach and development of the lessons learned will be the output of the case study.

### 1.4. Main Contributions

The first phase of this study where an exploratory survey was performed has several contributions: (a) for the first time, agility related problems of traditional SE in SoS projects are explored and validated in an empirical research, (b) findings surprisingly revealed that main agility problem was not the traditional SE process but in contrary weak application of the principles of the classical SE, (c) our interviews with the systems engineers to explore the reasons and nature of changes in SoS projects resulted in a novel model which relates changes to the discovery of new knowledge, (d) analysis of the model contributes to the SE literature with the idea of improving agility in SE by identifying the valuable but missing knowledge early in the project and by tailoring the SE processes to focus on the required knowledge.

Main contribution of this research is the proposal of the knowledge-driven, collaborative and model-based architecture development approach that can be

applied in SoS projects by the system of systems integrators. Proposed approach is implemented and assessed in a case study where the outcomes confirm that the novel approach has the potential to increase the agility attributes of the SE process at the SoS level.

From the practitioner's point of view, lessons learned and the success factors presented in this study can help LSI companies to increase the speed of shift from traditional SE approaches to architecture-driven, knowledge-focused and model-based SoSE practices.

## 1.5. Structure of the Thesis

This thesis is organized in chapters which reflect the phases of the research that is discussed in Section 1.3. Chapter 2 presents the literature survey focusing on the current and future trends and challenges in SoS projects. In Chapter 3, we present context, findings, threats to validity and analysis results of the exploratory survey. Thereafter, the proposed approach is formulated in Chapter 4. In Chapter 5 the setting, findings, independent evaluations, threats to validity and lessons learned in the confirmatory case study are presented. Chapter 6 concludes the thesis by presenting a summary of the study and main contributions.

# CHAPTER 2

# LITERATURE SURVEY

The focus of this section is to gather and synthesize the current literature to highlight the need for an agile SE and characterize its agility attributes. We first define the future trends in systems projects and examine the agile software engineering methods as a software engineering domain's response to these trends. Later we provide the characteristics of the future SE practices and present some SE process models and lifecycle management approaches from the exiting literature.

## 2.1. Characteristics of Today's System Projects

In 1987, Brooks told us that there was no silver bullet for the software problem. He also stated that even in the future there will not be one. This was because of the essential problems originating from the nature of the software, like complexity, conformity, changeability and invisibility. Even back in 1987, software was holding the most of the functionality of a system and it is the functionality that one frequently wants to change (Brooks, 1987).

A lot of things have changed since 1987. Changeability and complexity, essential properties of software, are still there and killing silver bullets. The only thing that is changing about this fact is the increasing size and complexity of the software. Today we are talking about large scale, complex software intensive systems in almost all sectors.

Information technology and software systems became the largest component of the capital investment. In the commercial market this caused a new concept which is called “the digital firm”. In today’s business environment most of the firms perform their business operations using software systems. All of their relationships with the customers, suppliers and employees are digitally enabled and mediated (Laudon & Laudon, 2011).

The technological advances and the information systems not only transformed our businesses (both in commercial and military domain) but also our daily lives. The systems of systems (SoS) became the only way of our daily existence in every part of our life. Especially the advances in the internet, communication and networking technology have completely changed our perception of doing business. In this new world even for a simple daily business process, a large number of systems, users and services are interacting. We can say that the Software Intensive Systems of Systems (SISOS) concept was not created but evolved as a natural

technological response to the customer demands and needs (Wade J., Madni A., Neill C., Cloutier R., Turner R., Korfiatis P., Carrigy A., Boehm B., 2010).

In a standalone complex system failures were locally critical. Today, complex systems are connected to many other standalone complex systems and a failure in one of such systems turns into a broadly/globally critical challenge (Turner, 2007).

It is true that the driving force behind today's complex systems of systems is our demands and needs. However, today, it is open to discussion which one is driving the other. SISOS has also changed the concept of time and its value. In the past, the army which has more information sources has the superiority in the field of war. However, today the ability of using the acquired information in rapid decision making is the determining factor of the success. It can sometimes turn into a disadvantage if the huge amount of information you have slows down your decision making process.

Every change in the operating environment is either a threat or an opportunity. Each of these threats/opportunities requires a response capability. However, in order to survive in today's dynamic business environment (whether it is commercial or military) such a response to change in the environment must be timely. From the systems engineering point of view, this means rapid fielding of the required capabilities.

One of the primary sources of change in the systems engineering environment is the change in the technology. According to a survey report by state of Michigan, almost half of the companies renew their computer systems (the hardware) every 14 months. Microsoft plans to release a new operating system (software) every two years. It is also possible to take the subject to the military domain. Results of a report prepared by the US Army War College states that estimated refresh rate in the commercial electronics is 12-18 months (Kennedy & Umphress, 2011).

On the other hand, for the major information systems acquired by the US government (DoD), average duration between the definition of the operational requirements and the delivery of the initial system capabilities is 91 months. The conclusion is not surprising: Considering such a huge schedule gap between the requirements definition phase and the initial system delivery in an environment with a technology refresh rate about 12-18 months indicates a final system delivery which will not be an up to date system (Kennedy & Umphress, 2011). Therefore, with the traditional systems engineering approaches we arguably provide some response capability but we are not providing it in a timely manner.

Another major change in today's system projects is that the assumption of relatively stable requirements is no more valid. Traditional systems engineers were dealing with completely defined standalone systems with specific functionality. The requirements are assumed to be well known from the beginning and the evolution of the systems is under the control of the systems engineers (Turner, 2007).

In the current/future systems everything is connected. Such systems are complex, adaptive, emergent systems of systems (Turner, 2007). Increasing complexity originates from both the increasing number of interacting and interconnecting systems (users, COTS products, legacy systems, re-used systems, etc.) and the increasing number of interdependencies (Wade J., Madni A., Neill C., Cloutier R., Turner R., Korfiatis P., Carrigy A., Boehm B., 2010).

Emergent behavior and the changing requirements is an obvious result of the increased emphasis on users and end value in today's systems. As the systems get more user-intensive, as the users considered part of the systems, it becomes almost impossible to pre-specify the requirements of such systems. The requirements of such systems emerge with the familiarity/use of the systems (Boehm, 2006).

In today's systems: the scope and complexity of the systems have increased; interfaces, connections and interdependencies have increased; emergent capabilities and unknowns have increased; but in addition to all these challenges, the expected duration for the fielding of the demanded system is decreased. Such trends exponentially increased the risks in system development projects.

In response to these trends we have somehow changed the way we conduct the system projects. In the past a complex standalone system was being developed by a single company. A small part of the system was being outsourced and only limited amount of COTS usage were preferred. In traditional systems engineering processes, the systems engineers were spending most of their time in designing the system in a top-down manner. In such a traditional approach requirements analysis, design and implementation processes were the main systems engineering processes (Bourque & Dupuis, 2004) (Haskins, Forsberg, Krueger, Walden, & Hamelin, 2010).

Today, such SISOS are developed by a set of companies using significant amount of COTS/re-use. One prime contractor has the overall system responsibility. However, this prime contractor (or the lead systems integrator) outsources significant amount of system's capability. Projects turn into somehow integration projects rather than pure top-down development projects.

In the past systems engineers preferred COTS/re-use/outsourcing as a risk mitigation approach. Today, this is no more a preference but the dominant way of developing complex systems. In such a system development approach, systems engineers spend most of their time in performing analysis on COTS/reuse issues and interoperability and integration problems with the legacy systems and the sub-systems developed by the subcontractor companies (Boehm, 2006).

COTS-based design approaches do not only emphasize some of the processes like decision management (or make/buy/re-use analysis) or subcontract management but also changes the overall top-down systems engineering processes. In the traditional systems engineering process, requirements determine the system capabilities. However, design with COTS products constrains your solution with the capabilities of the COTS products. Therefore, COTS products, re-used systems or capabilities of the subcontractors determine the requirements (Boehm 2006). In addition to this, you will have no (or very little) control over the evolution of the COTS products, re-used systems or capabilities of the subcontractors which will limit your control in the evolution of the overall system (Turner, 2007).

Another trend in today's systems engineering environment is the increased integration of systems engineering and software engineering. In the traditional approaches, the activities are organized in a more linear fashion so that systems engineers work at the early parts of the development lifecycle and provide the output of their processes to the software and hardware engineers for further design and development. This approach is mostly based on the assumption that the requirements

of the systems are pre-specified at the beginning of the project and these requirements are good and mature enough to develop a system that satisfies the operational needs of the customer. As the systems get more user-intensive, it becomes almost impossible to pre-specify the requirements of such systems. This fact undermines the fundamental approach of sequentially specifying the system (mostly systems engineering) and then implementing it (mostly software engineering) (Boehm, 2006).

Concepts and approaches like integrated product and process development (IPPD), concurrent engineering, incremental development and Integrated Capability Maturity Model (CMMI) are a result of the effort in dealing with such a need to integrate systems engineering and software engineering (Boehm, 2006).

## 2.2. Responses to Change in the Software Engineering Domain

The failure of projects due to the inability to respond to changes (whatever the source of the change is) and the pitfalls of the traditional lifecycle models are already recognized and have driven the development of methods like Agile and Software Product Line approaches (Tian & Cooper, 2006).

### 2.2.1. Agile Software Development Methods

Agile development methods such as Extreme Programming (Beck, 2001), Adaptive Software Development (Highsmith, 2000), Crystal (Cockburn, 2006), Scrum (Schwaber & Beedle, 2001) and DSDM (Consortium, 2008) are no more young in the software development field. They are widely accepted and have found a lot of chances in the different industries.

In February 2001, representatives of the agile methods (XP, scrum, Crystal, ASD, Feature-Driven Development and several other) met at Utah to discuss the principles of the light weighted software development methodologies. They wrote and signed the Agile Manifesto as (Highsmith & Cockburn, 2001):

*"We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:*

- *Individuals and interactions over processes and tools,*
- *Working software over comprehensive documentation,*
- *Customer collaboration over contract negotiation,*
- *Responding to change over following a plan*

*That is, while there is value in the items on the right, we value the items on the left more."*

Boehm discusses and compares the agile methods with the traditional plan driven methods. He concludes that it is not possible to say one way is better than the other. Each approach has its own home ground of project characteristics. For example, Agile methods work best when the customer works with the team in a dedicated

mode and the size of the project is small (both the team size and the schedule). However, as the size and complexity of the project gets higher and higher, the project starts to move to the home ground of the plan driven methods (Boehm, 2002).

In another study, Boehm and Turner define five critical factors in characterizing a project: personnel, dynamism, culture, size and criticality. Personnel is the skill levels of the developers and agile methods requires highly skilled personnel. Dynamic environments are more proper for agile methods. Culture is mostly related with the freedom of the developers. Agile methods are easy to adapt for the teams who feel more comfortable when provided more degrees of freedom. Agile methods also recommended for small sized teams and for systems with low criticality. One can use these factors as the five dimensions to analyze the project and to find the proper approach (Boehm & Turner, 2003).

### 2.2.2. Agility and Agile Attributes

Although the agile concept is no more a young concept, the perception of agile has almost been limited to "fast", "unplanned" and "document-free". Without an inclusive definition of "agility" and its attributes, it is not feasible to discuss the effectiveness and value of agile methods and compare them with the traditional methods. Five agility attributes (flexibility (FY), speed (SD), leanness (LS), learning (LG) and responsiveness (RS)) are identified and a definition for “agility” in terms of these attributes is proposed: *"Agility is a persistent behavior or ability of a sensitive entity that exhibits flexibility to accommodate expected or unexpected changes rapidly, follows the shortest time span, uses economical, simple and quality instruments in a dynamic environment and applies updated prior knowledge and experience to learn from the internal and external environment"* (Qumer & Henderson-Sellers, 2006).

Flexibility is the ability or behavior of an entity that allows adapting to changes whenever it is required. A method or a phase in a method may demonstrate flexibility by accommodating expected or unexpected changes. Speed of an entity characterizes rapid and quick behavior to get to the desired destination or to achieve goals. A speedy method may help to show the results quickly by following a specific approach. Leanness refers to compactness and tidiness. A lean method gives the desired quality output, economically, in the shortest possible time frame by applying simple and quality means of development. Learning refers to knowledge and improvement and is an indispensable ability of an entity, which is achieved primarily by using up-to-date knowledge and experience, gained from previous practices. A learning method shows continuous improvement over the period of time. Responsiveness refers to life, reaction and sensitivity. A responsive method is method that does not remain silent when response is required in different situations (Qumer & Henderson-Sellers, 2006).

An object needs to possess one or more of the agile attributes in order to demonstrate the agility. Although Qumer proposes this approach for evaluating the levels of agility in the software development methods, an object in this definition is not necessarily a complete software development methodology. It can be an

organization, a process, a document, a system, software or a method. Furthermore, agility may exist in such an object at different degrees and at different lifecycle phases. For example, a process or a method may have agility in the design phase but does not demonstrate any agility in the requirements phase. If all five attributes (FY, SD, LS, LG and RS) are supported by an object it is called full agility. If at least one of them is missing, it is called partial agility. Qumer proposes such a scheme to compare and discuss the levels of agility in different software development methodologies (Qumer & Henderson-Sellers, 2006).

### 2.2.3. What is an "Agile Software Development Method"

According to the above definition, the concept of “agile software development method (or methodology)” can be discussed. What makes a software development method an agile method? Five agile attributes can be interpreted from the perspective of a software development methodology as (Qumer & Henderson-Sellers, 2008):

- Flexibility: Can the method deal with the expected and unexpected changes?
- Speed: Is the method quick in delivering the demanded capabilities?
- Leanness: How compact is the method? Does it deliver the value within the shortest time span with the less but high quality effort?
- Learning: Does it the method improve itself with the new information and experience?
- Responsiveness: Is the method a reactive one or a passive one?

In parallel with this interpretation a definition for an "agile method" is proposed: "A software development method is said to be an agile software development method when a method is mainly people focused and communication-oriented, flexible (ready to adapt to expected or unexpected change at any time), speedy (encourages rapid and iterative development of the product in small releases), lean (focuses on shortening timeframe and cost and on improved quality), responsive (reacts appropriately to expected and unexpected changes), and learning (focuses on improvement during and after product development)" (Qumer & Henderson-Sellers, 2006).

## 2.3. Need for "Agility" in Large Scale, Complex System Projects

Kennedy and Umphress discuss the software intensive systems development from three aspects: Business, system and software. Business aspect is responsible from the overall strategy, operational capabilities, the contracts, funding and overall approach. System aspect is the interface between the management and the engineers. It is responsible for the overall technical definition and management. Software aspect is responsible for the software items (Kennedy & Umphress, 2011).

Business layer has already defined the need for agility in the development and acquisition of software intensive systems. In the section 804 of “National Defense

Authorization Act for Fiscal Year 2010, US congress directed the US Secretary of Defense to, “develop and implement a new acquisition process for IT systems”, especially for the rapid fielding of urgent operational capabilities. National Defense Authorization Act also states that the new process should be based on the March 2009 report of the Defense Science Board Task Force on Department of Defense Policies and Procedures for the Acquisition of Information Technology, which states “The conventional DOD acquisition process is too long and too cumbersome to fit the needs of the many IT systems that require continuous changes and upgrades” (Kennedy & Umphress, 2011) (*One Hundred Eleventh Congress of the United States of America*, 2010)

Project management field has already started to question the traditional project management approaches (Koskela & Howell, 2002). It is good news that PMI declared a new certification program in agile project management (Project Management Institute, 2011). "Agile Process" has already found itself a place in the "Chaos Success Factors" list in the Standish group's chaos report in 2009 (The Standish Group International, 2009).

As mentioned above, the pitfalls of the traditional system development approaches are also addressed in the software layer and several agile development methodologies are developed. Since the declaration of the agile manifesto (maybe earlier than that), agile methods have been criticized such that they were not proper for the large scale, complex system projects. However, while approaching agile methods as software engineering methodologies and comparing them with the plan driven methods, we have spent relatively less effort in trying to identify and solve the actual problem which was the need for agility in the large scale, software intensive system projects or putting in another way, the “need for agile systems engineering”.

Although we avoid explicitly mentioning, the common understanding in the project management and systems engineering domains is: dealing with the problems of the large scale complex projects are the main playground of project management and systems engineering. It is also the main driving force behind the developments in these areas. The intention is obviously not to say that systems engineering and project management methodologies, processes, tools and techniques are not for small scale projects but rather to indicate that the project management and systems engineering effort needed by the large scale projects are relatively greater compared to small scale ones.

Standish Group's Chaos reports clearly indicate that only a small portion of the projects are successfully delivered (Dominguez, 2009). In addition to this, the characteristics of the failing projects exposes that large scale projects are the ones that are mostly failing (Jørgensen & Moløkken-Østvold, 2006).

It is obvious that a lot of things have changed in the systems engineering environment. Change is inevitable and also requires a change in the systems engineers’ perception of "change". Traditional engineering culture and the accompanying systems/software engineering approaches try to achieve stable concepts and requirements early in the project. Main idea is "effort must be spent to eliminate change". In today's environment ignoring the change, or following a strategy to close the projects to changes would only mean unresponsiveness to the

changing business conditions which has the same meaning as a business failure. If trying to eliminate changes is not realistic and no more a valid option, then new approaches are needed to reduce the cost of responding to changes (Highsmith & Cockburn, 2001).

Current systems engineering literature (guidelines, handbooks and standards) is still based on the waterfall like approach and expects the defined systems engineering processes to be applied in a more sequential way. A new "Agile Development" section is added to the lifecycle approaches chapter in the current version of the INCOSE systems engineering handbook. However, it is still explained as a software development methodology (Haskins et al., 2010). The literature does not provide a framework that can be used by the systems engineers to plan and manage their tasks towards a more agile systems engineering approach. Unless such a framework is provided and systems engineers adapt the rapidly changing environment, the developments in the business area and the benefits of agile development methods in the software engineering domain will be ineffective in complex system projects (Kennedy & Umphress, 2011).

## 2.4. Vision for Future Systems Engineering

The problem of developing a new approach in systems engineering or an agile systems engineering framework is not a trivial one. Although there is no single proposed method promising to deal with all aspects of the current systems engineering problems, several attempts are made to define the need for and the characteristics of agile systems engineering. In this section, we categorize the previous work under three different categories: Characteristics of agile systems engineering, agile systems engineering process/lifecycle models and future research areas.

### 2.4.1. Characteristics of Agile Systems Engineering

In the introduction section of his book "Competitive Engineering" (Gilb, 2005), Tim Gilb, justifies the reasons for a new systems engineering approach. He explains several fundamental concepts that we need in today's systems engineering approach. "Rapid feedback" is a powerful tool to monitor and control your projects. As soon as feedback is early, it provides you information on how the things are working in practice and gives you the opportunity to fix things. Of course, your approach and your projects should be able to respond to the feedback. "Dynamic adaptability" requires that as you learn through rapid feedback, you need to modify your strategies and requirements as necessary. He admits that stability is good but it is unreachable. Therefore, the way to survive is to adapt dynamically to the changes. The new systems engineering approach must focus on the delivery of the required results, not on the technology and the processes. We must ensure that we are delivering critical and profitable results. Interdisciplinary communication, receptiveness to organizational change, capturing the critical success factors (critical requirements) and practical strategies for systems engineering are among the other fundamental concepts he explained (Gilb, 2005).

Turner starts his analysis in characterizing the agile systems engineering with providing his own definition of agile attributes (Turner, 2007):

- Learning Attitude: Adapt both the systems and the processes using the updated information and lessons learned to meet customer needs.
- Focus on customer value: Success and progress is measured against the prioritized requirements by the customer and operational features.
- Short iterations delivering value: Released are planned to deliver working systems. Realistic and risk based iterations planned in a rolling-wave manner.
- Neutrality to change: Change is inevitable. Design processes and system for change.
- Continuous integration: Integration and testing are ongoing activities.
- Test-driven: Demonstrable progress. Tests are developed before designing or coding any other artifacts. Capabilities are defined by tests.
- Lean attitude: Remove no value-added activities. Delay decisions as much as possible.
- Team ownership: Allow teams to have their own plans and processes.

The traditional systems engineering approaches are based on the assumption that most of the information to plan and execute the projects is readily available to all stakeholders at the beginning of the project. This assumption is unrealistic for the complex system projects. Systems engineering is a learning process. In real projects, the delivered system is never the one that is initially described. Throughout the project lifecycle, systems engineers seek information to fill the gaps in the initial description. All the practices in the systems engineering process are ways to learn about the system being developed. However, current systems engineering process, the V-diagram in the most primitive way, does not view systems engineering process as a learning process (Turner, 2007).

Another unrealistic consideration in the traditional approaches is that the needs of the customers are perfectly represented by the customer requirements and the concept documents. This results in a project environment in which the systems engineers are mostly isolated from the customers. Most of the time, the customer requirements are value neutral and do not include any relative ranking indicating the relative value of a single requirement to the customer. Without such information, any analysis or any trade study performed by the systems engineers will ignore the perspective of the customer. Systems engineers need to have more focus on the value to customer. This should be practically included in the systems engineering approach (Turner, 2007).

Systems engineering should change the inherited attitude towards change. Change needs to be considered as a new dimension in the analysis and trade studies. Especially the architecture and design considerations and processes should adopt a way for always looking for an easy to modify or easy to extend solution (Turner 2007). One should interpret the "solution" as to include both the system and the processes used to develop that system.

It is one of the agile principles to empower the individuals. Turner highlights this point under team ownership. Systems engineers should be provided an environment in which they can both develop their own process, plan and execute them. This will obviously increase their ownership and performance (Turner 2007).

Systems engineering is mostly considered as a one pass application of the strict V-model. Therefore, iterations and especially short iterations is not a concept that the systems engineers are familiar with. Turner states that iterations can be integrated into the systems engineering approach and the techniques used in systems engineering like prototypes, models, simulations, demonstrations and testing can all be iterative. In the agile software development methodologies, the true agile iterations result in operable system with value to customer. Of course in systems engineering this does not need to be like this. At an early stage of the project, the customer would be more interested in the operational concepts and the interoperability issues rather than a truly operational system. Therefore, the systems engineering approach should iteratively provide such outputs to the customer at the early stages. The traditional systems engineering approach which is using the systems engineers early in the project to define the requirements and the solution; and perform their trade studies and go away during the implementation phase until the validation starts is no more a valid approach. The more realistic and viable view of systems engineering should be similar to Barry Boehm's systems engineering definition. He characterizes systems engineering as a Command and Control, Intelligence, Surveillance and Reconnaissance (C2ISR) activity, consisting of several Observe, Orient, Decide, Act (OODA) loops and ongoing intelligence, surveillance and reconnaissance tasks (Boehm & Lane, 2006; Turner, 2007).

As the systems engineering adopts an iterative approach, it will not be so difficult to integrate continuous integration and test driven development into the systems engineering approach. As the focus on customer value increases and the systems engineers starts iteratively demonstrate the operational capabilities or provide deliveries of some artifacts which have value for the customer, the integration and testing of the outputs will no more be a luxury but a requirement.

### 2.4.2. Agile Systems Engineering Process/Lifecycle Models

A recent study highlights the problems in the traditional integration-centric or integration at the end approaches. The increasing complexity and unpredictability in large scale software development is analyzed and a need for the transition from traditional integration centric development to a more composition based development is identified. Three trends in the software intensive system development are highlighted: widespread adoption of software product lines (SPL), broad globalization in software development and building software ecosystems (Bosch & Bosch-Sijtsema, 2010).

*"A software product line (SPL) is a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way"* (Clements & Northrop, 2001). The principle approach in an SPL is

to create a family of software systems using a shared set of software assets (also called the "platform"). Each individual system in the product line can be created by configuring the platform for its own purposes. Additional software development is required for the specific functionality that is not covered by the common platform.

SPLs are widely adopted and providing order of magnitude improvements in time to market, cost, producibility, quality and many others. In the scope of this study, SPLs also enable rapid system development and flexible responses to changes (SEI, 2011), which are the main problems of today's systems engineering approaches. However, Bosch discusses that such an intra-organizational reuse of software assets and the platform also creates dependencies to the platform. These new dependencies, which do not exist in the product-centric approach, also increase the complexity of software development (Bosch & Bosch-Sijtsema, 2010).

Global software development requires software development in distributed team environment which again increases the complexity of software development due to different level of dependencies. Differences in culture, time zone, skill sets and maturity levels are sources of conflicts. Effort is needed to adapt the processes in transitioning from local development to global development (Bosch & Bosch-Sijtsema, 2010).

Finally, software ecosystem, which is simply *"a community of 3rd party application developers, around a successful product"*, also increases the complexity in software development due to the dependencies between the platform company and the 3rd party application developers. While the environment for software product lines is intra-organizational, software ecosystems include both internal and external developers (Bosch & Bosch-Sijtsema, 2010).

As a result of analyzing these trends, a need to transition from an integration centric approach to a more composition centric approach is identified. Following a too integration-centric approach results in pure decoupling of the assets and this increases the dependency between the teams and between the system elements. In an integration-centric approach, traditional sequential activities take place. Requirements are defined and allocated to subsystems and components, subsystems are designed and developed and finally the project enters the integration phase to integrate all the system components to start the system level acceptance tests. In such an approach, a lot of problems are found during the integration phase and at this late stage it is very costly to resolve the problems –if it is possible to resolve. The results may require significant amount of re-design and re-implementation (Bosch & Bosch-Sijtsema, 2010).

It is possible to put the integration centric approaches to one extreme of the system development spectrum and the open ecosystems to the other end. Successful software product lines are stimulating environments for software ecosystems. Such SPLs turn into platforms for external, 3rd party developers. The final systems include different functionalities based on the applications added/integrated to the SPL platform by the external developers (Bosch & Bosch-Sijtsema, 2010).

As stated above US congress directed the US Secretary of Defense to “develop and implement a new acquisition process for IT systems”. The new approach should aim for: continuous interaction with the user; rapid implementation of the

increments; an evolutionary approach supported with successive prototyping; and a modular, open-systems approach (Kennedy & Umphress, 2011). It is worth noting the statement of "open-systems approach" in a formal military document. This is an inevitable result of the evolution of the system projects towards a more collaborated (like software ecosystems) development approaches.

It is very important to analyze such trends in the complex software intensive system projects in order to propose improvements to the currents systems engineering approach. Future systems are getting more complex and becoming more software intensive. They are constructed by the integration of the independently developed systems (so-called systems of systems). Moreover, they are very large, dynamically evolving, unprecedented due to emerging requirements/behavior, and include significant amount of socio technical issues. Such systems are not developed by a single supplier. Generally, there is a lead systems integrator who wins the contract and is responsible for the overall delivery of the capability. However, the system development effort from the perspective of this lead systems integrator company includes development of the architecture, selection of the suppliers, identification of the COTS systems and components, planning the whole system lifecycle and the associated activities and planning and controlling the integration and test activities for the whole system. Of course this company is also responsible for the evolution of the system level architecture and design due to the emerging/changing requirements, subcontractor limitations and other issues raised during the system development (Boehm & Lane, 2006).

These companies should be very aware of the risks in developing systems of systems within such an environment. A prioritized list of risks in developing software intensive systems of systems is provided in (Boehm and Lane 2006): Acquisition management and staffing; requirements/architecture feasibility; achievable software schedules; supplier integration; adaptation to rapid change; systems and software quality factor achievability; product integration and electronic upgrade; commercial off-the-shelf (COTS) software and reuse feasibility; external interoperability; and technology readiness.

Some of the risks stated above cannot be handled by the traditional systems engineering approaches. A hybrid process with a blend of plan driven and agile approaches is required. The new approach cannot only focus on developing a system internally but it needs to consider the plan-driven management of the integration with the subcontractors and also an extremely agile process to concurrently and continuously watch the market, competition, change in technology, COTS refresh and renegotiate the next increment's scope and prioritized content and the interfaces between the supplier's next-increment interface specifications (Boehm and Lane 2006).

Boehm and Lane propose the Scalable Spiral Process model for developing 21st century systems. The model proposes a three-team cycle: a team of lean, plan driven, stabilized developers; a team of thorough V&Vers (verification and validation) and a team of agile, proactive re-baseliners. While the plan-driven development team performs the "Act" part of the OODA loop for the current increment, the agile team performs the "Observe", "Orient" and "Decide" parts for the next iteration.

To address the software intensive systems of systems development risks explained above, interpretation of the OODA activities is provided. In the observe part of the cycle, one needs to scan the environment for the changes in the technology, in the COTS capabilities and the systems that we are somehow interacting. The act of "observe" also requires continuous monitoring of the process on the current increment for any problems or improvement opportunities. Orienting is the part that we analyze the information gathered in the observe part. Change impact analyses, trade studies, risk analyses are performed during orienting. "Deciding" part is the part that the stakeholders negotiate on the decisions regarding any updates for the future increments, any need for architecture re-baselining or the validity of the feasibility rationales for the future increments (Boehm & Lane, 2006).

An agile systems engineering action model is proposed in (Stelzmann et al., 2010). Key agile characteristics proposed by (Turner, 2007)are adopted. Similar to (Boehm & Lane, 2006) they use the OODA model for visualizing the agility. They propose that any process enhancing any phase of the OODA loop provides agility in the systems development. However, the whole cycle and all phases of the OODA loop should be considered to provide a comprehensive systems engineering approach. Therefore, a proposed agile systems engineering approach should provide ability:

- to gather new information (Observe),
- to analyze this information (Orient),
- to decide for its relevance and how to react (Decide),
- to respond, which can also be called “flexibility” (Act).

There are four principles underlying the proposed agile systems engineering action model. First of all the approach gives more value to the engineers than the process itself. They are doing the job. Every suggestion and feedback from the engineers should be considered with care.

Second principle is the incremental development with close customer/user interaction. It is widely experienced that it is not realistic to assume that all the requirements are well known at the beginning of a project. Therefore, we need to expect some requirements to change, some new requirements to emerge and some of them to be deleted. This fact takes product vision concept into this model. At the beginning of a project, instead of trying to find a comprehensive set of stable requirements, we need to begin with a product vision. Using tools and methods for customer/user interaction (such as prototypes and simulation models), new information should be elicited. This will increase the level of detail in the requirements in an incremental manner

Third principle is the iterative development of increments. Each development item will be developed through the iterative development process. If the development item passes the test at the end of the iterative process, it is presented to the customer. Fourth and the final principle is the flexible design. This principle can be applied to the system under development (architecture) but it can also be applied to the processes, organization and supply chain (Stelzmann et al., 2010).

### 2.4.3. Key Research Areas

Boehm highlights evolutionary development, business model-based user programming, product line development and network centric systems of systems among the key topics to focus on in developing a new systems development approach (Boehm, 2006).

The Office of the Deputy Assistant Secretary of Defense for Systems Engineering is leading the research initiatives to fundamentally change the capabilities for the design, adaptation, and manufacture of defense systems. The Systems-2020 initiative is intended to demonstrate and transition 21st century engineering practices and tools. Systems-2020 Final report (Booz Allen Hamilton, 2010) is one of the inputs to the planning of the Systems 2020 Program. The purpose of Systems-2020 initiative is defined as *"rapid development and fielding of systems capable of adapting post deployment"*. In order to achieve this goal, three main areas for improvements are defined: model-based engineering MBE, platform-based engineering (PBE) and capability on demand (Booz Allen Hamilton, 2010).

Traditional systems engineering is a linear process consisting of sequential activities mostly performed with different teams. Each team gets an input from the previous process and delivers an output to the following process. A lot of information is lost during such handoffs. Significant amount of rework is required and it is really difficult to keep a common understanding from beginning to end. MBE provides a good communication medium and facilitates the interaction between different teams and domains. It is possible to use models for all aspects of the complex systems. Concept engineering, virtual design and modeling; and model driven manufacturing are the further areas highlighted within the MBE (Booz Allen Hamilton, 2010).

Platform-Based Engineering, emphasizes development around a system architecture/platform based on commonalities and planned variability. Such a platform based approach is proven to be successful in the field of software product lines (Clements and Northrop 2001).

One of the main findings of the Systems-2020 initiative was that the DoD does not develop product line architectures effectively. In order to be successful in complex systems projects, future systems engineering approaches should address areas of architectural patterns, software product line techniques and open systems architectures. Successful adaption of PBE in systems engineering will provide significant benefits such as reduced system development times, reduced cost, simplified training, high quality, agility and flexibility (Booz Allen Hamilton, 2010).

As described in previous sections, the most significant threat for future systems engineering approaches is emerging requirements and the need for an evolving system. Although platform based engineering helps systems engineers to adapt changes in pre-deployment, it does not help any support for the adaptation of the changing user needs after the system is deployed. Capability on demand concept refers to designing systems to support adaptation to changing needs in the user environment. Systems engineers should spend effort both on performing agile

"systems engineering" and to engineer systems that are agile (i.e. "agile systems" engineering).

## 2.5. SoS Challenges & SoSE

### 2.5.1. Definition and SoS Characteristics

A SoS is generally described as the integration of a set of new and existing systems. Each component system has its own operational objectives and corresponding functionality (Lane & Boehm, 2008). The USA Department of Defense (DoD) defines a SoS as "a set or arrangement of systems that results when independent and useful systems are integrated into a larger system that delivers unique capabilities" (ODUSD(A&T)SSE, 2008). Maier postulated five key characteristics of SoS: operational independence of component systems, managerial independence of component systems, geographical distribution of component systems, emergent behavior, and evolutionary development processes (Maier, 1998).

Based on a recognized taxonomy, there are four types of SoS:

- Directed - Created and managed to fulfill specific purposes. The component systems maintain an ability to operate independently; however, their normal operational mode is subordinated to the central managed purpose;
- Acknowledged - The SoS has recognized objectives, a designated manager, and resources for the SoS; however, the constituent systems retain their independent ownership, objectives, funding, and development and sustainment approaches. Changes in the systems are based on cooperative agreements between the SoS and the system;
- Collaborative - Component systems interact more or less voluntarily to fulfill agreed upon central purposes. The Internet is a collaborative SoS; and
- Virtual - Lacks a central management authority and a centrally agreed upon purpose for the SoS. Large-scale behavior emerges -and may be desirable- but this type of SoS must rely on relatively invisible mechanisms to maintain it.

This characterization offers a framework for understanding SoS based on the origin of the SoS capability objectives and the relationships among the stakeholders for both the SoS and the systems. Although there are examples of all four types of SoS, in the scope of this study, we will focus on acknowledged type of SoS, where customers try to obtain new capabilities through the integration of existing systems. The SoS is developed by collaborating companies. One prime contractor called the lead systems integrator (LSI) or SoS integrator has the overall system responsibility. An LSI cannot consider component systems as independent bounded entities, but rather as components in larger, more variable, ensembles of interdependent systems. Interaction between the component systems are based on end-to-end operational processes and networked information exchange.

### 2.5.2. SoSE

In traditional system projects, systems engineers are able to define boundaries clearly and able to control the development environment so that requirements can be optimally allocated to components based on technical trade analyses. This is seldom the case in the SoS environment (Lane & Dahmann, 2008). Existing systems are used as the components of the SoS. Systems engineers are challenged with an allocation of functionality and implementation details which may not be optimal from the SoS perspective. In addition, the system of systems integrator lacks control over the component systems that retain their independent ownership, funding, and development processes. The overall SoS architecture must be designed to maximize the overall delivered capability and the individual systems must be designed to adequately perform their role in the architecture.

SoSE evolved as a result of recognizing the need for systems engineering at the SoS level. SoSE is defined as: "The process of planning, analyzing, organizing, and integrating the capabilities of a mix of existing and new systems into a system-of-systems capability that is greater than the sum of the capabilities of the constituent parts" (ODUSD(A&T)SSE, 2008). In the Systems Engineering Guide for SoS, DoD characterizes SoSE with seven elements and their relationship (the trapeze model): 1) translating SoS capability objectives into SoS requirements and 2) assessing the extent to which these capability objectives are being addressed, as well as 3) anticipating and assessing the impact of external changes on the SoS. Central to SoSE is 4) understanding the systems which contribute to the SoS and their relationships and 5) developing a design for the SoS which acts as a persistent framework for 6) evaluating new SoS requirements and solution options. Finally the SoS systems engineer 7) orchestrates enhancements to the SoS, monitoring and integrating changes made in the systems to improve the performance of the SoS (Lane & Dahmann, 2008; ODUSD(A&T)SSE, 2008).

Although the trapeze model presents a good conceptual view of SE in SoS environment, it is found less useful by practitioners who are trying to implement this model (Dahmann, Rebovich, Lane, & Baldwin, 2011). Wave model is proposed as an improvement by mapping the seven core elements of the trapeze model to a series of six time-sequenced major steps in implementing a SoS SE process. Both the trapeze model and its transformation to the wave model are depicted in Figure 2 (Dahmann et al., 2011).

Main purpose of initiate SoS is to establish critical needs in terms of SoS capability objectives and create SoS level CONOPS to provide a context for these critical needs. Gathering information about any systems that are likely to affect the SoS capability objectives is one of the main activities.

Once the SoS is initiated, conduct SoS analysis is performed to provide analysis of the "as is" SoS. The aim is to establish an initial SoS baseline and plan for the SoS engineering efforts for the required improvements. Needed SoS functions are defined in this process independent of the component systems for various capability options.

Develop and evolve SoS architecture develops/evolves the persistent technical framework for addressing SoS evolution. SoS architecture is the key artifact created

and used in this element. The architecture provides a shared representation of the SoS by capturing information about the component systems, key SoS functions, relationships, as well as end-to-end functionality and data flow. It defines the way in which the component systems work together.

Once the architecture is available SoS update is planned (plan SoS update) by evaluating priorities and options. In this element risks and mitigations are identified and agreements are developed to plan for the next SoS upgrade cycle. The focus is on developing an integrated master schedule with the consideration of the detailed development schedules maintained by constituent systems. Finally, updates are actually implemented in implement SoS update. The constituent systems implement and test changes at their level while SE teams of the system of systems integrator lead integration and test, monitor progress and update the integrated master schedule. The output of this element is the

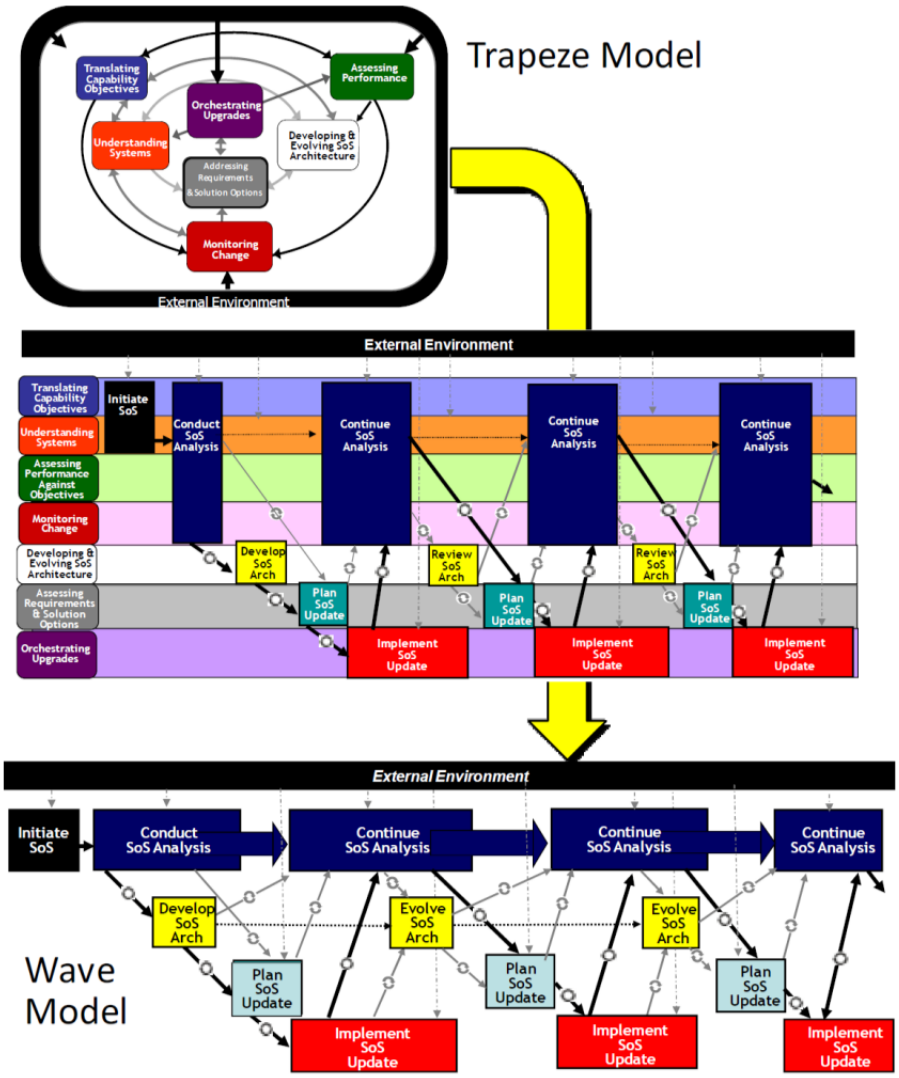


Figure 2: Unwrapping the Trapeze Model to Create the Wave Model[1]

[1] Reproduced with J. Dahmann's permission.

new baseline for the SoS. Continuous SoS analysis is the key element in the wave model which involves ongoing analysis that revisits key information on the state of the SoS (Dahmann et al., 2011).

### 2.5.3. SoS Pain Points

SoS and SoSE are topics that interest a significant number of INCOSE members globally, both individuals and organizations. As a result a working group is established by INCOSE to share understanding of SoS and SoSE issues, good practice and background, and contribute to maturing the understanding of the available body of knowledge in this area. Primary purpose of the working group is to advance and promote the application of SE to SoS, often referred to as SoSE (Harding & Dahmann, 2012).

In response to the need to understand the SoS issues of importance, SoS working group conducted a "SoS Pain Point Survey" in a recent INCOSE event in January 2012 in Jacksonville Florida. The objective of the survey was to collect information on major issues or "pain points" in the area of SoS operation, management and SE to support planning for activities of the INCOSE SoS working group. The survey identified seven "pain points" raising a set of questions for SE of SoS(Dahmann, 2013).

***Constituent Systems:*** Coordination and management of multiple independent constituent systems in SoS. Legacy systems which were not configured or managed to allow insertion into the overall SoS. This creates interoperability concerns between the older and newer systems. Managerial and evolutionary independence can mean that constituent systems change in response to the perceived goals for that system, usually with little regard for the impact on SoS goals or behaviors. **Question:** What are effective approaches to integrating constituent systems?

***Leadership:*** SoS organizational and technical complexity, multiple independent stakeholders with their own interests and independence makes the role of leadership in SoS even more important. Without the type of traditional top down control, there are clear challenges for application of SE at the SoS level. **Question:** What are the roles and characteristics of effective SoS leaders?

***Capabilities and Requirements:*** Traditionally, SE process begins with a clear, complete set of user requirements and SE provides a disciplined approach to develop a system to meet these requirements. Typically, SoS are comprised of multiple independent systems with their own requirements working towards broader capability objectives. In the best case the SoS capability needs are met by the systems as they meet their own local requirements, but in many cases the SoS needs may not be consistent with the needs of the constituent systems. In these cases, the SoS systems engineers need to identify alternative approaches to meeting those needs through changes to the constituent systems or additions of other systems to the SoS. This is in effect asking the systems to take on new requirements with the SoS acting in a way as the "user". **Question:** How can SE address SoS capabilities and requirements?

***Testing, Validation and Learning:*** Findings highlight the fact that most defense SoS cannot be tested thoroughly prior to fielding leading to approaches like incremental validation. With multiple constituent systems on asynchronous development cycles, finding ways to conduct traditional testing across the SoS can be difficult. It is necessary to reflect a perspective that looks at significant learning going on over the life of a SoS. **Question:** How can SE approach SoS validation, testing, and continuous learning in SoS?

***SoS Principles and Thinking Skills:*** The survey revealed lack of formalized processes and examples of SoS success. This area is one where progress in identifying and articulating SoS principles, SoS Thinking and examples, could have benefit to the discipline. **Question:** What are the key SoS thinking principles?

***Interdependence and Emergence:*** Combining component systems into SoS produce unexpected behavior. Well-structured approaches for "design for emergence" are not generally available. Complex relationships among systems in a SoS are often poorly understood and difficult to analyze. Systems often have interdependencies that are either unknown or unacknowledged. This is exacerbated by interdependencies between systems in development, a system in development and fielded systems, and fielded systems; further, this is compounded by multiple combinations of all of these. There is a need for SoSE methods and tools to support the modeling and prediction of complex SoS behaviors including analysis and architecting methods. **Question:** How can SE address the complexities of SoS interdependencies and emergent behaviors?

***SoS Authority:*** In defense projects, authority conflicts often dominate discussion of SoS. The focus in this area is on creation of the incentives and development environment which allow the systems to proceed to meet their own objectives while working cooperatively to support broader objectives at the SoS level. **Question:** What are effective collaboration patterns in SoS?

# CHAPTER 3

# EXPLORATORY SURVEY

In this chapter, the context, findings and analysis results of the exploratory survey is presented. The overall objective of the exploratory survey is to uncover the agility related problems of the SE in the large scale, software intensive, socio-technical military SoS projects.

## 3.1. Scope and Research Questions

Our intention is not to study all kinds of problems but the ones related to the agility attributes as described below. Although the concept of agility has been in the agenda, specifically in the context of software engineering, since the early 2000's (Beck, 2001), its perception has almost been limited to "fast", "unplanned" and "document-free". Without an inclusive definition of "agility" and its attributes, it is not feasible to discuss the effectiveness and value of agile methods and compare them with traditional methods. Five agility attributes are identified for software engineering: flexibility, speed, leanness, learning and responsiveness (Qumer & Henderson-Sellers, 2006).

Flexibility is the ability or behavior of an entity that allows adapting to changes whenever it is required. A method or a phase in a method may demonstrate flexibility by accommodating expected or unexpected changes. Speed of an entity characterizes rapid and quick behavior to get to the desired destination or to achieve goals. A speedy method may help to show the results quickly by following a specific approach. Leanness refers to compactness and tidiness. A lean method gives the desired quality output, economically, in the shortest possible time frame by applying simple and quality means of development. Learning refers to knowledge and improvement and is an indispensable ability of an entity, which is achieved primarily by using up-to-date knowledge and experience. Responsiveness refers to life, reaction and sensitivity. A responsive method is method that does not remain silent when response is required in different situations (Qumer & Henderson-Sellers, 2006).

Turner follows the same approach to characterize agile SE. He provides his own definition of agile attributes but this time for the SE process (Turner, 2007):

- Learning Attitude: Adapt both the systems and the processes using the updated information and lessons learned to meet customer needs.

- Focus on customer value: Success and progress is measured against the prioritized requirements by the customer and operational features.
- Short iterations delivering value: Releases are planned to deliver working systems. Realistic and risk based iterations planned in a rolling-wave manner.
- Neutrality to change: Change is inevitable. Design processes and system for change.
- Continuous integration: Integration and testing are ongoing activities.
- Test-driven: Demonstrable progress. Tests are developed before designing or coding any other artifacts. Capabilities are defined by tests.
- Lean attitude: Remove no value-added activities.

In the present study, our interpretation will be based on the agility attributes defined by Turner. Since a key characteristic of any agile approach is its ability to deal with changes (i.e. flexibility), we shall first analyze the changes in large scale software intensive system projects. Research Question 1 that was introduced in Chapter 1 is extended with the following research questions as an entry point to the subject:

RQ1.1: What are the main reasons for changes in the large scale, software intensive, socio-technical military system projects?

RQ1.2: In which lifecycle phase do most of the requirements changes occur?

RQ1.3: Which of these changes could be categorized as expected changes?

## 3.2. Research Method

The research was designed as an iterative, interview-based exploratory survey which allowed us to come up with new research questions during the iterations between data collection and data analysis processes. In such an approach, researchers develop categories and meaning from data through an iterative process that starts by developing an initial understanding of the perspectives of those being studied. That understanding is then tested and modified through cycles of additional data collection and analysis (Kaplan & Duchon, 1988).

### 3.2.1. Data Collection

The main data collection method of the study was semi-structured open-ended interviews. Interviews were initiated on a template of pre-defined questions (questionnaire) but new questions are discussed according to the answers obtained. The questionnaire used during the semi-structured interviews is given in the Appendix section of this thesis. Most of the initial interviews were face to face or telephone interviews but due to the iterative nature of the study, re-discussion of the answers, summary of the interview results and discussions of the new issues were conducted via telephone and email exchanges.

We have conducted 59 interviews. Interviewees were selected from 10 different companies and from 18 different SoS projects. Selected companies are mid-to-large scale companies from around the world (4 from Turkey, 2 from USA, 1 from

Australia and 3 from Europe). Our target population is systems engineers with more than 5 years of SE experience of which at least 3 years is gained in a large scale SoS project in applying traditional systems engineering approaches. At least 3 systems engineers and at most 4 systems engineers are selected from each project. Only 2 of the systems engineers had previous agile development experience.

### 3.2.2. Case Selection

Since our intention is to explore the agility problems in the large scale, software intensive SoS projects in the defense industry, we applied some selection criteria while forming the study set. The context of the exploratory survey is summarized in Table 1.

Table 1: Study Context

| Attributes | Criteria |
|---|---|
| Project Team Size | > 30 |
| Project Duration | > 2 years |
| Project Budget | > 100 million $ |
| Product vs. Project based? | Project based business |
| New system vs. improvement? | New system development |
| Sector | All defense projects |

All the systems engineers defined their life cycle approach as the traditional waterfall model. Although there are slight differences in the adapted waterfall model between the companies and the projects, all systems engineers agree on the following basic illustration (Figure 3):


Figure 3: Traditional Waterfall Lifecycle

## 3.3. Exploratory Survey Findings[2]

### 3.3.1. Reasons for Change

Agility attribute "flexibility" is about dealing with changes. In today's systems engineering environment, changes occur in the concept, scope, requirements, design and even in the already implemented part of the system during the system lifecycle. However, in the scope of this study, we limited our focus to the changes in the requirements (especially system requirements) with the assumption that all the other major changes should be reflected by the changes in the requirements.

Each interviewee was asked to identify 3 major reasons for the requirements change. In addition to this, in order to keep the set more understandable and workable, similar reasons are grouped under more general definitions. For example, different answers like:

- Late understanding of the operational processes;
- Late identification of the complete set of users, their responsibilities and interactions;
- Increased understanding of the customer's decision making mechanism;
- Late discovery of the existing system's performance limitations on the operations

are all grouped under "increased understanding of the operational processes". Table 2 summarizes the major reasons for requirements change.

[2] The original answers collected from the participants during the exploratory survey are captured in a technical report (METU/II-TR-2014-20) and not included in this thesis for proprietary reasons. The readers who are interested in the original data can contact to the author to obtain the technical report.

Table 2: Reasons for Requirements Changes

| ID | Reasons for Change | Total | % |
|---|---|---|---|
| R1 | Increased understanding of the initially defined operational problem | 45 | 76 |
| R2 | Late clarification/definition of the non-functional requirements | 37 | 63 |
| R3 | Late understanding of the capabilities/constraints of the COTS/outsourced products | 36 | 61 |
| R4 | Changes in the technology | 23 | 39 |
| R5 | Initial problem has changed as a consequence of other changes | 22 | 37 |
| R6 | Late understanding of the legacy/external system constraints. | 14 | 24 |

### 3.3.2. Operational Domain Knowledge

The primary reason for requirement changes is the increased understanding of the initially defined operational problem. R1, R5 and R6 are considered as closely related to the operational domain knowledge. Many different answers are collected addressing this problem from different perspectives but all were related to the customer's operating environment and the missing knowledge regarding this environment such as customer's business processes, organizational structure, decision hierarchy, user types and their interaction, etc. During the interviews, all systems engineers stated that their main problem was to understand the customer's operating environment and their problems in this environment. For example, only one systems engineer stated that he had real difficulty in developing the technical solution after the customer's operational problem was clearly understood.

On the other hand, the survey results showed us that although the systems engineers are having problems due to the lack of operational knowledge, most of them do not make use of the relevant SE processes and methods that are already defined in the traditional SE literature to address this knowledge gap. Traditional systems engineering literature proposes processes and methods for better understanding of the customer's operational needs and the problem domain such as concept of operations (ConOps) development, scenario development, prototype development, stakeholder analysis, etc. However, systems engineers do not consult them.

Following are some of the findings from the survey:

- Only 24% of the systems engineers stated that they have a defined ConOps development (or a similar one) process in the project/organization. However, the resources and time allocated to this process is not enough to conduct this process properly.

- None of the projects/organizations has a defined process (or a guideline) for:
  - Operational scenario development,
  - Prototype development,
  - Stakeholder analysis.
- None of the organizations has a business analyst/business process engineer (or a similar) role in the team. In any of the projects, there isn't a task defined in the WBS to capture and document the existing business/operational processes of the customer.
- None of the systems engineers establish traceability from the requirements to the source stakeholder and his/her responsibilities in the organization.
- No task or document exists for the complete definition of the stakeholder and user types.
- None of the systems engineers has ever defined effectiveness measures for the system.

Such findings were surprising as the traditional systems engineering approach receives most critiques on the heavy processes and resulting documentation. However, our observations indicate that documents are produced but the process steps to produce such documents are not properly conducted. As a result, systems engineering artifacts were produced without the valuable information in them.

**Highlight1:** Operational domain knowledge is defined as the most valuable knowledge by the systems engineers. This knowledge is indicated as a key to understand the real problem to be solved.

**Highlight2:** Systems engineers do not perform the requirements/knowledge elicitation and analysis processes that are proposed by the traditional systems engineering literature to address the operational domain knowledge.

These findings led to the following objectives for the formulation of the new approach:

**Objective1:** Extend the scope of SE activities to cover operational domain.

**Objective2:** New approach shall focus on the analysis activities, especially knowledge elicitation activities, which are primarily addressing the operational domain knowledge.

### 3.3.3. Customer Interaction

As the interviewees highlighted the need for the operational domain knowledge, we also discussed the interaction of the systems engineers with customers. The findings are summarized in Table 3, 4 and 5.

The results are mostly surprising. It was expected that most of the systems engineers would have difficulty in accessing the customer, especially since these are all military projects. However, the results showed that more than half of the systems engineers (53%) were able to communicate with the customer whenever they needed customer support. Similarly, a significant portion of the interviewees (39%) stated

that they could reach the customers when a serious problem occurs. In total, 92% of them did not have real difficulty in reaching the customer when they needed.

On the other hand, we observed that the real problem about customer collaboration is not the difficulty in accessing the customer but the difficulty in communicating with the correct customer representative with the required knowledge, experience and decision authority. 75% of the systems engineers stated that they did not have a chance to work with the specific customer representative but with the one that is provided by the customer. Such kind of situations did not help the systems engineers to resolve their problems.

Table 3: Ability to Reach Customer

| I can communicate/collaborate with the customer | Total | % |
|---|---|---|
| Whenever I want | 31 | 53 |
| Whenever a significant problem occurs | 23 | 39 |
| When the customer wants | 5 | 8 |
| Not at all | 0 | 0 |

Table 4: Ability to Select Customer Representative

| I can chose the customer representative according to my needs | Total | % |
|---|---|---|
| Yes | 15 | 25 |
| No | 44 | 75 |

Table 5: Need for the Customer Collaboration vs. Project Phase

| I need the customer collaboration mostly during the: | Total | % |
|---|---|---|
| Requirements Phase | 42 | 71 |
| Design Phase | 13 | 22 |
| Development Phase | 4 | 7 |

The survey also revealed that there is no specific effort to increase the level and effectiveness of communication and collaboration with the customer representatives. Apart from the planned review meetings with the customer (such as requirements

review, preliminary design review, critical design review, etc), all the communication and collaboration activities are ad-hoc upon the request of the systems engineering team. For example, in all projects there were detailed training requirements in the contract. However, in any of the projects trainings were not planned and detailed until the end of the development phase and were not performed after the whole system is tested.

Another observation indicates that systems engineers need customer collaboration mostly during the requirements phase. This finding is in parallel with the observation which indicated the operational domain knowledge as the main reason for the requirements changes. We observed that the contract documents do not provide sufficient information about the operational problem and the organizational processes of the customers. As the requirements definition phase is when systems engineers first meet the customer and try to understand the problem as defined in the customer requirements, they require significant customer support in this phase.

It was seen that requirements phase provides an environment with dense customer interaction. It was expected that the systems engineers learn a lot from the customers about the operational problem such as business processes, interacting legacy systems, types of users etc. However, it is found that the systems engineers mostly interact with the customers to clarify the ambiguities in the customer requirements. Customer requirements are considered as the scope of the project and no extra effort is spent to understand the operational objectives and current limitations of the customer in the mission environment to reach those objectives.

**Highlight3:** The real problem in collaborating with the customer is not to reach the customer but to interact with the customer with the required skills, experience and decision authority.

**Highlight4:** Systems engineers do not proactively plan and manage their interaction with the customer to increase the effectiveness of the collaboration.

**Highlight5:** Customer requirements are considered as the scope of the project and no extra effort is spent to understand the operational objectives and current limitations of the customer in the mission environment to reach those objectives.

Hence, **Objective3** was formulated as: Establish an environment to increase the level and effectiveness of the interaction with the customer and to reach correct customer representative.

### 3.3.4. Non-Functional Requirements

Late clarification/definition of the non-functional requirements is the other main reason for requirements changes. Non-functional requirements are known as constraints or quality requirements. They can be further classified according to whether they are performance requirements, maintainability requirements, safety requirements, reliability requirements, or one of many other types of system requirements (Bourque & Dupuis, 2004).

Non-functional requirements are sometimes addressed under the specialty engineering title in the systems engineering literature. INCOSE handbook defines 12

areas (Table 6) in the specialty engineering section. Our discussions regarding the problems with the non-functional requirements are based on these areas (Haskins et al., 2010).

The survey also revealed that specialty engineering tasks are almost always performed by the systems engineering group. However, the systems engineering groups do not have a fixed, well-defined position for a specialty engineer. We have observed only 4 titles defined for specialty engineering within the 10 companies and 18 projects: 2 logistics analyst, 1 human engineer and 1 specialty engineer. All these positions are allocated to a single project (no matrix organization for specialty engineers). In these companies there is no separate group for specialty engineering but they are working within the systems engineering group.

In the rest of the projects, the tasks required for specialty engineering is performed by the systems engineers. However, these systems engineers do not have the required training and experience to perform such tasks (summarized in Table 6). For example, among the 59 interviewees, only 2 of them had training on electromagnetic compatibility analysis and only 3 of them have past experience on this area. Similarly, only 1 systems engineer have previously performed training needs analysis. However, this systems engineer did not have any education for this task.

Table 6: Specialty Engineering Competencies

| Specialty Engineering Area | Training | Experience |
|---|---|---|
| Cost effectiveness analysis | - | - |
| Electromagnetic compatibility analysis | 2 | 3 |
| Environmental impact analysis | - | - |
| Interoperability analysis | - | - |
| Lifecycle cost analysis | - | - |
| Manufacturing and producibility analysis | - | - |
| Mass properties engineering analysis | - | - |
| Safety and health hazard analysis | 1 | 3 |
| Sustainment engineering analysis | - | - |
| Training needs analysis | - | 1 |
| Usability/human systems integration analysis | - | - |
| Value engineering analysis | - | - |

Most of the systems engineers find it relatively difficult to define non-functional requirements compared to defining functional requirements (Table 7). This is the major reason for addressing the non-functional requirements very late during the project lifecycle.

We observed that systems engineers do not like dealing with the non-functional aspects of the requirements. In all of the investigated projects, the document templates that are used for the requirements and design artifacts have dedicated sections that are addressing the non-functional requirements. Our findings indicate that unless a non-functional requirement is defined directly by the customer in the stakeholder requirements document, systems engineers leave those document sections as TBD (to be determined) as long as possible.

Table 7: Relative Difficulty in Defining Different Types of Requirements

| Difficult | Easy | Requirements Types |
|---|---|---|
| 1 | 58 | Writing functional requirements |
| 50 | 9 | Defining performance requirements |
| 56 | 3 | Defining system security requirements |
| 54 | 5 | Defining system safety requirements |
| 54 | 5 | Defining maintainability, reliability, availability requirements |

**Highlight6:** Although most of the systems engineers do not have the required training and experience, non-functional requirements are considered in the responsibility of the systems engineering teams.

**Highlight7:** As the systems engineers do not feel comfortable working on the non-functional requirements, definition/clarification of the requirements related to specialty engineering is delayed late in the project lifecycle.

### 3.3.5. Time of Requirements Change

Discussions on the time of the requirements changes provided us with interesting findings. The results of the interviews are summarized in Table 8. There are two things to note about the findings. Firstly, since the changes during the requirements phase is expected and mostly welcome without any side effects, the changes during the requirements phase is not discussed during the interviews.

Secondly, the finding that no requirements change was reported by the systems engineers during the maintenance phase may mislead to a conclusion that customers do not demand any change in the system and its functions during maintenance, which is obviously not true. We identified two main reasons for this perplexing finding.

Firstly, the changes during the maintenance phase are not reflected as requirements changes. Secondly, in most of the projects the final customer acceptance is obtained before the maintenance phase starts. Therefore, the systems engineers we interviewed are not knowledgeable about the maintenance period as they have already started to work for other projects.

According to the results in Table 8, majority of the requirements changes occur after the design phase. Therefore, it is possible to interpret that most of the requirements changes have significant impacts as they occur late during the project. They result in rework both in the design and in the developed product.

Table 8: Time of Requirements Change

| # | Design | Development | Verification |
|---|---|---|---|
| **Total** | 9 | 34 | 16 |
| **%** | 15 | 58 | 27 |

A significant number of changes were expected in the verification phase. Since the final system is tested against the system requirements together with the end users, it is the first time that the users interact with the overall system. It is quite normal that the users discover what they really needed and what they did not like as a result of this interaction.

On the other hand, large amount of changes in the development phase was an unexpected result for us. During this phase, SoS level requirements and SoS architecture is already baselined and the overall system is realized by implementing necessary changes in the constituent systems and integrating them into SoS. This is the phase that software, hardware and integration engineers are mostly on the scene.

Re-discussing the findings with the interviewees showed us that there are several reasons for this. We found that keeping the system, software and hardware engineers of the constituent systems out of the decision making process during the requirements and architecture phases of the SoS is the main reason. During the requirements phase where most of the fruitful customer interaction occurs and LSI systems engineers try to understand the problem, engineers of the constituent systems are not consulted. As a result, software and hardware engineers try to understand the problem to be solved from the artifacts of the requirements and design phases.

As the system developers try to implement the abstract design and requirements into a physical and functioning system, they discover that significant amount of details for this transition is missing in the LSISE documents or conflicting with the characteristics of the constituent systems. Questions raised by the system developers against the LSI systems engineers to understand the problem domain results in the discovery of new knowledge which in turn initiates changes to the SoS level

requirements. Similarly, as the functioning systems get more visible, systems engineers understand that this cannot be the feature/behavior that customer was asking for and asks for changes.

Following is the observation of one of the interviewees: *"During the projects, we (LSI systems engineers) act as the proxy of the customer/user. System/Software engineers of the subcontractors sometimes find it difficult to design/implement the functionalities against the requirements defined by us. They try to imagine the real operational scenario to satisfy the customer's operational needs. Our requirements with 'the system shall do this, the software shall do that' kind of expressions do not help them much. Their questions in an attempt to understand the real problem reveal that we often do not really understand and capture the operational problem or ignore the limitations originating from the existing systems".*

Although most of the changes occur during the development phase (Table 8), the results presented in Table 5 show that only 7% of the participants highlighted the need for customer collaboration during the development phase. During the discussions on this finding we have observed that when the systems engineers discovers a need for customer collaboration to bridge some knowledge gap, their approach towards the resolution of this problem changes from phase to phase.

If they discover that they need customer support during the requirements definition phase, they prefer to contact the customer without any hesitation. On the other hand when they face to a similar situation during the design or development phases, their behavior is somewhat different. It is clear that as the time passes and as the decision point gets far and far from the requirements definition phase, systems engineers prefer the decisions that result in less rework and less changes in the previous artifacts. Especially, if the decision point is in the development phase they prefer not to consult the customer.

Most of the systems engineers state that they discover new knowledge related to the problem domain as a result of the interaction between the development teams and the systems engineering team during the development phase. Most of the time, such new knowledge make them think that the requirements and design should change in a way to provide better solution with more customer satisfaction. However, they are reluctant to discuss such major changes with the customer since such changes are considered to have significant cost and schedule impact.

Table 9 summarizes how the change decisions are supported during different phases. As the decision point gets away from the requirements phase, systems engineers prefer making assumptions towards the option which requires less changes instead of asking for customers support and opinion. On the other hand, participants note that most of these decisions are rejected by the customer in later stages and had more serious cost consequences.

Although significant amount of the participants stated that they would execute decision support processes with the necessary analysis during the design and development phases, in practice only 4 of the 59 systems engineers have previously performed a formal analysis study with properly planned evaluation criteria and with properly documented analysis results.

Table 9: Change Decisions

| | Requirements | Design | Development |
|---|---|---|---|
| Make assumptions | 5 | 23 | 30 |
| Consult the customer | 53 | 20 | 9 |
| Execute decision support processes supported with the required analysis work | 1 | 16 | 20 |

**Highlight8:** Since the majority of the changes occur after the design phase, it can be interpreted that the costs of the changes are high.

**Highlight9:** Interactions between the LSI systems engineers and the system developers during the development phase create a great environment for improving the understanding of the customer's problem and identifying the knowledge gaps in the requirements and design artifacts.

**Highlight10:** Nature of the traditional systems engineering approach discourages the collaboration between the systems engineers and the customer/system developers.

These finding led to the following objective for the new approach: **Objective4:** Involve the engineers of the constituent systems early in the project, especially during the definition of the SoS level requirements.

### 3.3.6. Validation Process & Customer Focus

All the participants were theoretically aware of the difference between the verification and validation. They all referred to the widely known definition of verification and validation as: "…While verification proves whether 'the system was done right'; validation proves whether 'the right system was done' (NASA, 2007).

This definition is the main reason to include validation process in this case study. Validation is about user's operational needs and expectations (Wasson, 2005). Therefore, if there is a problem in understanding the true operational needs of the customer, then it is not possible to properly validate the system because the criteria to perform such validation against is missing.

Following are the highlights:

- 54% of the interviewees were performing requirements analysis activities in defining the system requirements from the stakeholder requirements.
- 46% were defining the system requirements from the stakeholder requirements based on their experience without performing any requirements analysis activities.
- 20% were performing requirements validation activities such as reviews, analysis, prototyping, etc.

- 10% were utilizing a requirements prioritization scheme and any of them did not use customer's value or stakeholder expectations as criteria. All criteria were chosen based on the developer's technical, budget and cost considerations.
- Verification & validation plans were not prepared in any one of the cases before the implementation started.
- 100% assume that the operational needs of the customer are documented in the stakeholder requirements (customer technical requirements provided with the contract) and a final system passing the tests that are defined against the stakeholder requirements in the operational environment results in a validated system.
- All agree that the technical review meetings (apart from the requirements review meeting) do not effectively provide customer feedback to systems engineers. Most of the time, customer's do not understand much from these technical review meetings.

Validated set of acquirer requirements are defined as a requirement/prerequisite for the end system validation in the ANSI/EIA 632 (ANSI/EIA, 1999). Therefore, in order to perform a proper validation process, one needs to perform requirements validation activities. The survey results indicate that requirements validation is not a well-known concept and systems engineers do not perform this process properly.

**Highlight11:** Validation activities are poorly understood and weakly performed in practice.

**Highlight12:** Current systems engineering practice is not customer and value focused. No special attention is given to understanding the specific needs of the stakeholders and their relative importance.

### 3.3.7. Project Documentation

Traditional systems engineering approaches are also known as "document-driven" or "document-heavy" processes. Systems engineers defined different reasons for the usage of documentation in their systems engineering processes:

- We communicate the information via documentation.
- We are paid for the documentation.
- Documentation is required during the operation and maintenance of the system after delivery.
- Documentation is necessary if some of the engineers leave the project or the company?
- Documentation is used for future projects to help estimating and planning.
- Documentation helps us to prevent/control changes.
- Progress is monitored by the completion of the documents.

Documentation we refer in this section is the technical documentation such as requirements, design and interface documents. The user documentation like the manuals has not been discussed during the survey.

We observed that producing a lot of documentation is not the main agility problem of the traditional systems engineering processes. Systems engineering document artifacts are the places for capturing the requirements and the design. Every piece of information in the documents somehow reflects a decision. System development phases have pre-determined timeframes ending with a review together with the customer and getting approval for the documentation. Systems engineers need to deliver the documents with the required decisions within this pre-determined timeframe. They are expected to make the decisions when the necessary information is missing. The rest of the design and development activities (and the associated documentation) are based on ill-made decisions. Systems engineers state that most of their time is spent on trying to make the decisions without required information and then changing the decisions when the required information is available and re-working on the documents in the later phases of the projects.

Findings indicate that the existing processes are weak in measuring the quality and completeness of the documents. Systems engineers use the majority of the time to write further requirements on the problem domain that they are familiar with. While the details of the well-known areas increase with a lot of redundant requirements, the unknown space remains to be unknown. Time is used to increase the thickness of the documents. However, knowledge creation and hence the value provided to the systems engineers is not directly proportional with the thickness of the documents.

**Highlight13:** Current document-oriented systems engineering practice forces systems engineers to make decisions when the necessary information is not mature or available. This causes serious rework and lack of team's ownership.

**Highlight14:** Current SE practice focuses on document generation and is weak in creating value for the systems engineers. In such an approach it is not easy to control whether the processes are redundantly capturing the already known information or new and valuable knowledge is being created. Such an approach makes it difficult to monitor the progress in the project.

Hence: **Objective5:** Shift from document based approach to model-based systems engineering (MBSE) approach.

## 3.4. Analysis of the Findings

### 3.4.1. Survey Highlights

In this section, we summarize the survey findings in terms of highlights from the previous sections. Instead of repeating long observations in the following sections we will shortly refer to these highlights presented in Table 10.

### 3.4.2. Knowledge-Based Model for Change Initiation

Our interviews with the systems engineers to explore the reasons and nature of changes in the SoS projects resulted in a simple model (Figure 4) which relates changes to the discovery of new knowledge. Our model is based mainly on the categorization of the knowledge. Within the scope of this research, the term

*knowledge* refers to anything one needs to know for SoS level decisions, especially the ones that are related to the architecture. In this respect Known Knowns (KK) are the available knowledge and Known Unknowns (KU) constitute the necessary but missing knowledge. Unknown Unknowns (UU) represent the knowledge that is required for the development of the solution but project staff are not aware of the fact that this knowledge exists and is necessary. This type of knowledge remains completely in the unknown space.

KKs represent our knowledge in relation to the project. Every piece of information that we have and we are planning to use (i.e. value adding information) during the SoS development is in this category. Technical experience, operational knowledge, functional and non-functional characteristics of the constituent systems, etc. can be considered as KKs.

KUs are the missing information that we need. This is defined according to our understanding of the initial problem. We decide that to understand and solve the problem we need some information but we don't have it yet. For example, we know that we need to know all the user types and their operational responsibilities in order solve their operational problem. However, initial project documents do not include such information. Therefore, this missing information is identified (i.e. known) as necessary but we don't have it yet (i.e. unknown at the moment).

Table 10: Highlights from the Survey

| ID | Highlights |
|---|---|
| HL1 | Operational domain knowledge is defined as the most valuable knowledge by the systems engineers. This knowledge is indicated as a key to understand the real problem to be solved. |
| HL2 | Systems engineers do not perform the requirements/knowledge elicitation and analysis processes that are proposed by the traditional systems engineering literature to address the operational domain knowledge. |
| HL3 | The real problem in collaborating with the customer is not to reach the customer but to find and reach the customer with the required skills, experience and decision authority. |
| HL4 | Systems engineers do not proactively plan and manage their interaction with the customer to increase the effectiveness of the collaboration. |
| HL5 | Customer requirements are considered as the scope of the project and no extra effort is spent to understand the operational objectives and current limitations of the customer in the mission environment to reach those objectives. |
| HL6 | Although most of the systems engineers do not have the required training and experience, non-functional requirements are considered in the responsibility of the systems engineering teams. |
| HL7 | As the systems engineers do not feel comfortable working on the non-functional requirements, definition/clarification of the requirements related to specialty engineering is delayed late in the project lifecycle. |
| HL8 | Since the majority of the changes occur after the design phase, it can be interpreted that the costs of the changes are high. |
| HL9 | The interactions between the systems engineers and the system developers during the development phase create a great environment for improving the understanding of the customer's problem and identifying the knowledge gaps in the requirements and design artifacts. |
| HL10 | Nature of the traditional systems engineering approach discourages the collaboration between the systems engineers and the customer/system developers. |
| HL11 | Validation activities are poorly understood and weakly performed in practice. |
| HL12 | Current systems engineering practice is not customer and value focused. No special attention is given to understanding the specific needs of the stakeholders and their relative importance. |
| HL13 | Current document-oriented systems engineering practice forces systems engineers to make decisions when the necessary information is not mature or available. This causes serious rework and lack of team's ownership. |
| HL14 | Current SE practice focuses on document generation and is weak in creating value for the systems engineers. In such an approach it is not easy to control whether the processes are redundantly capturing the already known information or new and valuable knowledge is being created. Such an approach makes it difficult to monitor the progress in the project. |

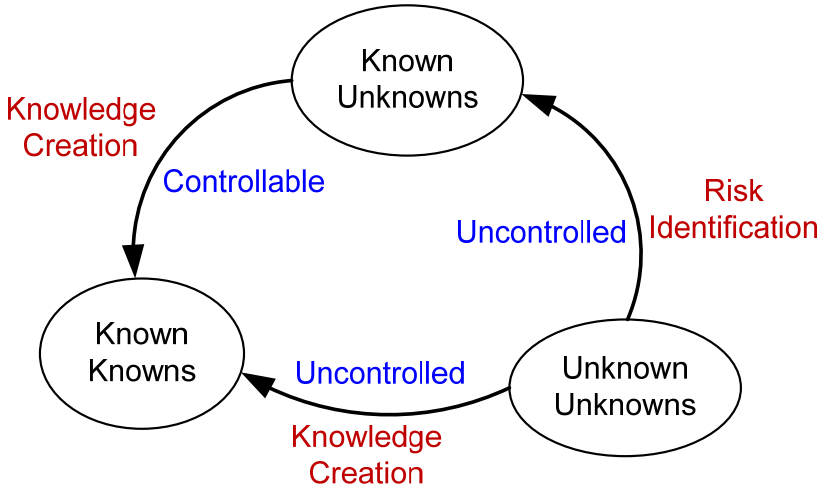


Figure 4: Knowledge-based change initiation model

UUs are the missing information that is required for the solution development but we are not aware of the fact that this piece of information exists and is necessary. This type of knowledge remains completely in the unknown space. In fact, there is another type of knowledge: Unknown Knowns (UK). This is the knowledge we have but we are not aware at the moment that this knowledge should be used during the solution development. Within the context of this study we treat UKs in the category of UUs.

During the survey, it has been agreed with the participants that every change request is a result of new knowledge (i.e. new KKs). Of course not necessarily that every new information triggers changes. According to our model there are two ways to create new knowledge. The first is a result of the transition of some KUs into KKs. We acquire new information which was already identified as valuable but missing. The second is a result of the transition of some UUs into KKs. Some new information is discovered and at the same time we discover that this new information affects the project. In both transitions, new valuable information is available. As we define the KKs as the valuable information from the project point of view, it is expected that this new information will trigger changes on the previously made decisions.

There is one other transition in the model which is from the UUs to the KUs. This occurs when we discover that we need some information which will affect our decisions. We discover the need for that information but we don't have it yet. This is the transition that systems engineers are supposed to define risks.

### 3.4.3. Agility with Focus on Learning

The key characteristic of agile approaches is flexibility. This requires dealing with expected and unexpected changes. In today's system projects, trying to

eliminate changes is not realistic and no more a valid option, then what we need is to reduce the cost of responding to changes (Highsmith & Cockburn, 2001).

The exploratory survey has shown that most of the reasons for change can be associated to missing knowledge (unknowns) such as operational domain knowledge, knowledge on the legacy systems and knowledge on the COTS/outsourced products, etc. In addition to this, participants categorize this missing information as KUs. For example, all the systems engineers know that they will need information on all types of users and the business processes they are involved (HL1). They all know that their system design will be affected by the legacy systems that are in use. They all know that they need to know about the security, interoperability, safety, etc. needs of the customer (HL6). Traditional ConOps templates and requirements templates all address such information. However, current linear, document –oriented practice does not focus on getting or creating the required knowledge (HL2, HL4). Most of the new knowledge comes uncontrolled and hence comes late in the project. Therefore, the changes initiated by such late and uncontrolled knowledge have costly consequences (HL8).

According to our model, all the transitions from the UUs are uncontrollable. However, transitions from the KUs to KKs can be controlled. By identifying the valuable but missing knowledge (i.e. KUs) early in the project and by defining/tailoring the SE processes to focus on getting/creating the required knowledge can shift these transitions to early phases and thus reduce the costs of changes. Such an approach will trigger learning and knowledge focused systems engineering and will also be flexible from the agility point of view.

### 3.4.4. Focus on Customer Value and Lean Behavior

Although the systems engineers agree and are very well aware of the fact that the customers do not know what they need at the beginning of the project and the system requirements derived from these customer requirements do not reflect the true operational need of the customer, they still try to develop a system based on the ill-defined customer/system requirements (HL5). Therefore, it is observed that the systems engineering process in practice does not have the agility attribute defined as customer focused process. As a result, it does not focus on validation process which is directly related to customer value and his operational satisfaction.

The agility attributes "focus on customer value" and "lean attitude" is closely related to each other. Lean attitude is about removing non-value added activities and delaying decisions as much as possible (Turner, 2007). Lean thinking defines three conditions for value added activities (Oppenheim, Murman, & Secor, 2011; Womack & Jones, 2003):

- The external customer is willing to pay for it explicitly or implicitly (that is if the customer understood the details he would support this activity),
- Transforms information or material, or reduce uncertainty,
- Provides specified performance right the first time.

The first bullet is directly related to customer value. In order to develop something valuable for the customer they need to understand the customer's real problem. It's found that systems engineers are having difficulty in understanding what is valuable for the customer and what is not. They state that they need problem domain information to understand the real operational problem but they are not consulting the necessary processes for this need (HL1, HL12). Difficulty in understanding the real problem and real value for the customer makes it also difficult for the systems engineers to properly define and perform the validation activities (HL11).

In real life, operational problems are mostly non-functional oriented rather than functionally oriented. Whether a software system will satisfy a customer will mostly be determined by its quality attributes but not its functions. Every competing system will more or less provide similar functionality (Chung, Cesar, & Leite, 2009). The key characteristics of the successful systems differentiating them from the others are their non-functional requirements. A system's utility and effectiveness will be determined by its non-functional requirements (HL12). For example, every bank has a web page providing online banking services. The functions or features provided to the customer are almost the same. However, some of the customers are very happy with their bank's online banking services whereas others almost hate theirs.

Main distinguishing characteristics of systems engineering is its interdisciplinary nature. Systems engineers are not expected to be experts in all of the areas related to whole system lifecycle. However, it is the main contribution of the systems engineers to coordinate the activities of different disciplines for the success of the overall system. Coherent integration of the specialty engineering into the project/program at the right time is one of the responsibilities of the systems engineers (INCOSE UK Systems Engineering Competencies Working Group, 2010). Developing specialty engineering competencies in the systems engineering teams and changing the current processes in a way to integrate the specialty engineering efforts early in the lifecycle is necessary for developing valuable systems (HL6, HL7 and HL12).

The second one is addressing value from both the customer and the systems engineering perspective (i.e. internal and external customer). The value for the systems engineers is the timely and accurate knowledge. However, with the current document-oriented SE practices they cannot focus on the value/knowledge creating activities. In contrast, current SE practices encourage ignoring the valuable knowledge in decision making since this knowledge comes uncontrolled and late (HL8).

In addition to this, linear and document-driven approach forces systems engineers to make decisions early in the project lifecycle when the necessary information is missing. This is in contrast to the lean behavior which supports delaying decisions as much as possible (HL13).

Instead of focusing on the documentation and document-based processes, systems engineers should focus on their own needs. In such an approach, the side effects of the document heavy approach will be reduced and the agility attribute team ownership will be increased. Systems engineers will find the opportunity to plan and

perform tasks to reach what they really need instead of trying to fill out the document templates with poorly made decisions (HL14).

### 3.4.5. Collaborative Systems Engineering and Trainings

In this exploratory survey we have clarified that traditional, document-oriented, linear systems engineering approach does not emphasize and support the collaboration and interaction both internally and externally. Therefore, agile systems engineering approach should be designed as a collaborative one. It should support collaboration of the systems engineers with the customer, with the internal system developers and with the external parties (COTS companies and subcontractors). HL1, HL2, HL3, HL4, HL5, HL9 and HL10 can all be associated with the problem of timely and effective collaboration.

Increased understanding of the initial operational problem by both the customer's team and the supplier's team is found to be the primary reason for changes (HL1). We think that trainings can help systems engineers on this problem. As the users get more familiar with the concept and the technical capabilities they can address their problems better. This is the primary source for the changes requested by the customers. In the current practice, trainings are used to teach the users and maintainers how to operate and sustain the delivered system. That is why the trainings are not planned until almost all the features of the system are designed and implemented. This is always considered as something valuable to the customer but a non-value added activity from the supplier's point of view.

Considering the knowledge-based change model and the other findings of this phase of our study, we believe that trainings can be used to achieve several agility attributes by starting the training sessions very early in the project and conducting in a periodic manner. The objectives of the trainings will be defined in a large spectrum such as systems engineering and system lifecycle training, concept of operations training, user interface training, COTS capabilities trainings, etc.

We expect several benefits from such a training dense systems engineering approach:

- Trainings will provide an effective knowledge sharing and creating environment,
- It is possible to increase the concept and technology awareness of the customer so that the customer can discover his operational needs early in the project. Consequently, change requests will occur early in the project with relatively reduced costs.
- It will be possible to interact with different types of customer representatives with the required skill set and experience. Customers will not see this collaboration activity as a burden because this is a training session and they will benefit from this.
- It will be possible to give the context and system view to the subcontractors and obtain early feedback about their limitations.
- Remove or reduce the paper based engineering efforts,

- Better understanding of customer value.
- Early feedback from the systems developers regarding the feasible technical options and innovative improvement suggestions.

### 3.5. Threats to Validity

There are four widely accepted categories of validity related to qualitative research: construct validity, internal validity, external validity and reliability (Runeson & Höst, 2008). Internal validity is relevant for the explanatory case studies (Tellis, 1997).

Construct validity reflects how much the findings represent what the researcher is really investigating. Researcher's subjectivity is always a threat. The results may be misleading if the interview questions are misunderstood or misinterpreted by the participants. We used both written questionnaires and interviews for data collection. By selecting the systems engineers with more than 5 years systems engineering experience we tried to minimize the possibility of the misunderstanding in the concepts used in the interviews. In addition to this, we explained every key concept using the definitions and examples from the well-known IEEE and INCOSE systems engineering literature. As we discovered some ambiguity we iteratively re-discussed the findings with the participants. As suggested by (Yin, 2003), all the findings and our interpretations were reviewed by the participants and corrected if necessary according to their feedback.

External validity determines whether the findings can be generalized beyond the setting of the group studied (Runeson & Höst, 2008). In order to increase the external validity of our study we carefully planned and selected multiple case interview set. We have conducted 59 interviews. Interviewees were selected from 10 different companies and from 18 different projects. Selected companies are mid-to-large scale companies from around the world (4 from Turkey, 3 from Europe, 2 from USA and 1 from Australia). At least 3 systems engineers and at most 4 systems engineers are selected from each project. All projects are large scale software intensive defense projects. It is argued in the relevant literature that cross-case analysis involving 4 to 10 case studies may provide a good basis for analytical generalization (Eisenhardt, 1989). Yin also states that these case studies can be from the same organization (Yin, 2003). Since we have worked on 18 different cases from 10 different organizations, the findings are expected to be generalizable to other cases.

We tried to increase the reliability of findings by both using an interview protocol based on a template questionnaire and open interviews as data sources. On the other hand, due to the nature of the military projects we unfortunately did not have a chance to review the documents of the projects or make observations in the projects. For example, the results would be more realistic and reliable if we could have a chance to review the change request forms of the projects.

# CHAPTER 4

# AGILE AND COLLABORATIVE SE APPROACH

In this chapter we first present the examples and depict abstractions formulated within the scope of the present study to lay the foundation of our novel approach. Thereafter, the proposed knowledge-driven, agile and collaborative SE approach is formulated.

## 4.1. Foundations for a Knowledge-driven SoSE

### 4.1.1. Handling Complexity with Decomposition Approaches

Decomposition is widely used in divide-and-conquer strategies to deal with complexity. A wide variety of strategies such as decomposition by structures, behaviors, goals etc. are available. SoS projects involve groups of stakeholders representing different domains and coming from different disciplines. These stakeholders have different problems originating from different complexity dimensions. Therefore, each stakeholder tries to apply decomposition differently. Following are some examples.

"Classic" SE methods apply the dominant decomposition along physical units (Figure 5a). It is the natural match when complexity and cost is driven by physical units or their integration. That is the reason that traditional SE literature advises to build the WBS over the PBS. As we are talking about the software intensive SoS, one of the major stakeholder groups is software engineers. Contemporary software engineering methods apply the dominant decomposition along operational and functional units (Figure 5b). It is the natural match when the driver of complexity are use cases/functions and their development. Many other disciplines (e.g. rollout, logistics, and training) apply the dominant decomposition along topological units (Figure 5c), which is the natural match when complexity and cost is driven by topographic units and deployment architecture. Customers or system of systems integrators often apply the dominant decomposition along social units (Figure 5d). It is the natural match when complexity and cost is driven by governance and by the need to manage a large amount of people coming from different departments.

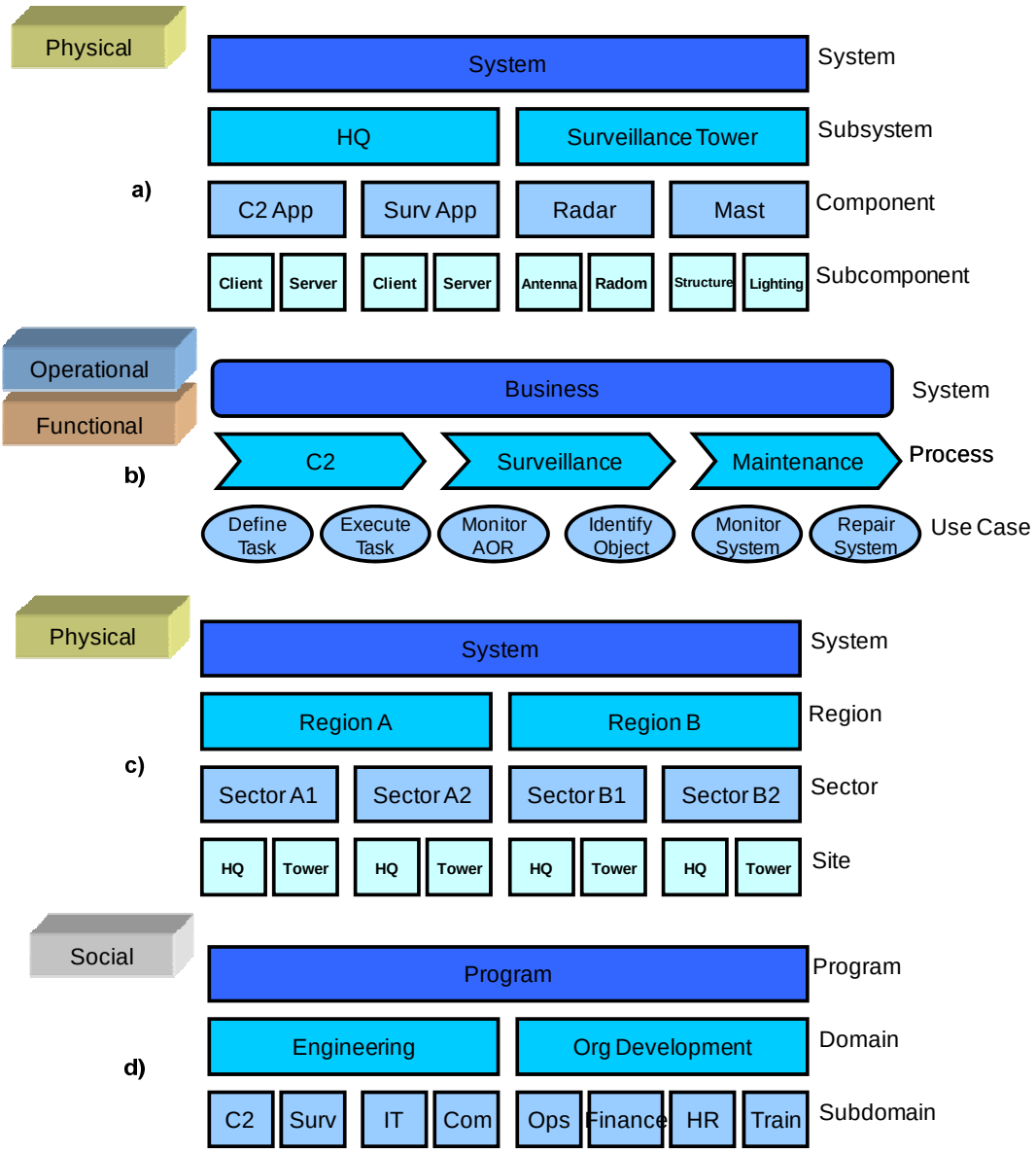


Figure 5: Handling Complexity with 2D Decomposition Approaches

### 4.1.2. Multi-aspect Decomposition and Enterprise Architecture Frameworks

Based on the specific concerns of the stakeholders, each dimension of complexity is addressed with its own decomposition. In SoS environments, such as border security projects, the solution has many complex aspects. Therefore, no particular decomposition is dominant. An alternative to classical dominant decomposition schemes can be a multi-aspect decomposition. For one aspect (e.g. functional), starting with an abstract level, one can increase the level of detail by decomposition (e.g. system functions, system element functions, component functions, etc.). Different aspects can be introduced as semantic layers to separate decomposition in different domains as shown in Figure 6. In enterprise architecture frameworks (AF) such a layered approach can be observed in terms of views and viewpoints.

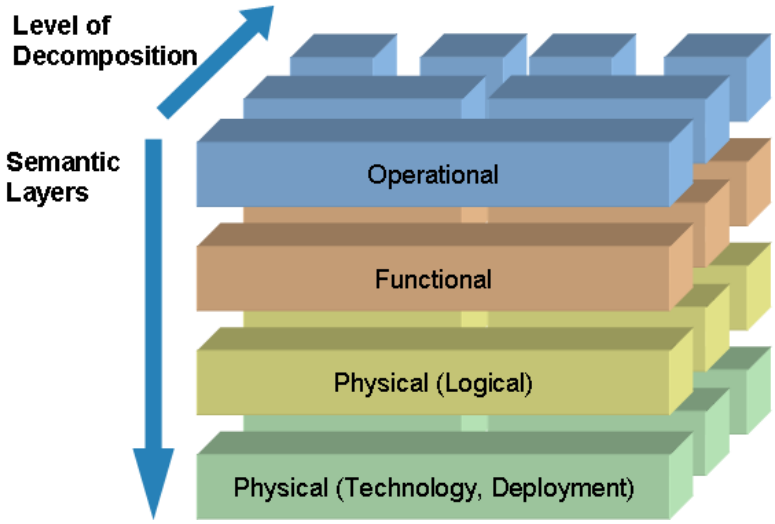


Figure 6: Layered Decomposition Approach to Handle Complexity

Layered decomposition is a result of the top-down nature of traditional SE. In such a layered approach, dependency between elements of different layers represents a "cause-effect" or "problem-solution" relationship. Each layer depends on the layer above it (Figure 7). Each element in a layer needs to be justified and validated against the layer above it. This is a way of performing analysis and there is no problem with this approach if one prefers to view the problem in different layers or from different aspects. However, this approach becomes problematic if one tries to perform the SE tasks in the same manner that he/she analyzes the problem. This results in a waterfall like process and prevents systems engineers from focusing on value adding activities and prevents an engineering organization from effectively reacting to changing needs and constraints in each of -but mostly the top- design layers.

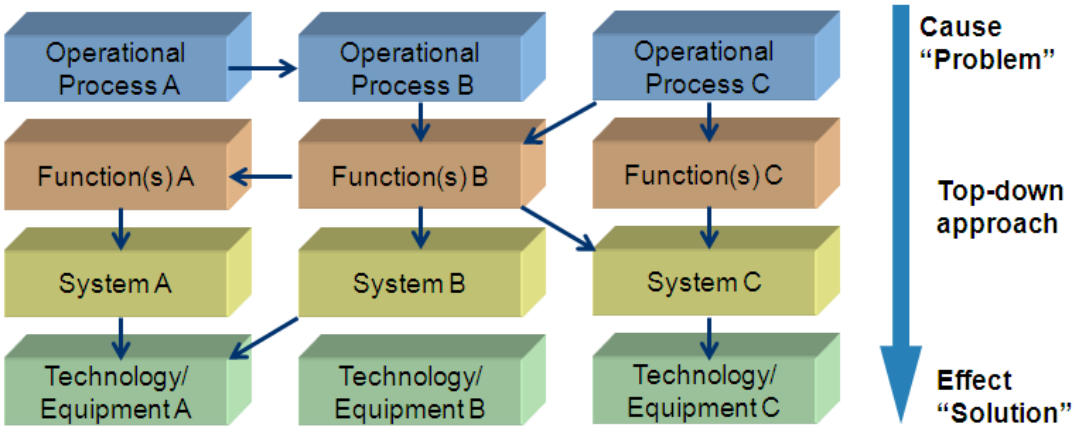


Figure 7: Cause-Effect Relationship between Layers

As shown in Figure 8, in such a classic approach phases are linearly planned. In every phase a layer is completed and the results are captured in documents. For the next phase, the knowledge of this layer that is captured in the documents is an input for the next layer. Gray areas in Figure 8 represent the unknown areas or the knowledge that is ignored; and colored areas represent documented knowledge.

The engineers that are responsible for each layer are separated in different teams. The WBS and plans are constructed in a way that first one layer is completed and then activities for the next layer start. Completion is determined by the release of the documents at the project milestones. Most of the inter-team interaction occurs at the milestone reviews where document hand over occurs.

Such an approach implicitly ignores the existing knowledge at the beginning of the project until it is captured in a formally released documentation. As a result, operational requirements are developed with little understanding of the system and system level decisions are made with little understanding of subsystems (i.e. ignoring existing knowledge in the next layer). When the next layer's knowledge becomes available in the process, it becomes necessary to change the decisions made in the upper level. Since earlier decisions are generally the most critical ones, and changes in those decisions require significant rework, projects are reluctant to make changes in those decisions.

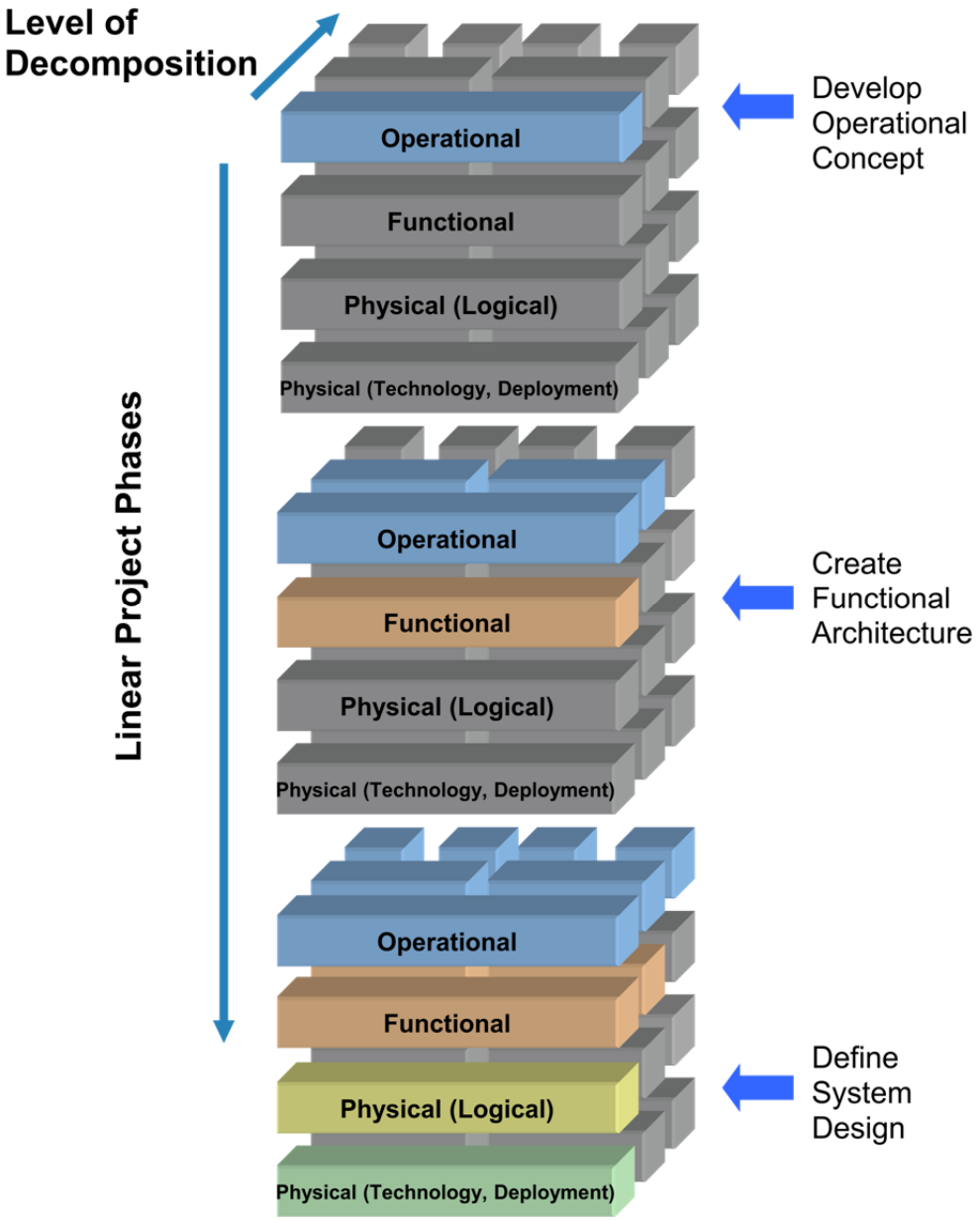


Figure 8: Layered Decomposition Approach Results in Waterfall-like Process

### 4.1.3. Lead Systems Integrators & Architecture-driven Approaches

Acknowledged SoS are developed by collaborating companies that are responsible for the constituent systems. One prime contractor which is called the lead systems integrator (LSI) or system of systems integrator has the overall responsibility for the SoS level capabilities. Projects turn into integration projects rather than pure top-down development projects. In such an environment, systems engineers of the LSI company spend most of their time in capturing the architectural knowledge that spans capability objectives, operational activities, major physical systems and their interfaces, end-to-end functional behavior of the integrated SoS, etc.

LSI SE team does not have the full control over the constituent systems. However, it is possible to set some constraints on their development, structure or operation in order to achieve SoS level capability objectives. This highlights the importance of the LSI activity **understanding systems** which is mapped to the **initiate SoS** in the wave model. Under such a scenario, the level of detail of the information captured in terms of architectural knowledge is much less than is relevant to the detailed development of each individual system in the SoS (Maier, Emery, & Hilliard, 2004). As a result, the SE activities in this environment are more of a **knowledge management** kind than top-down design activities. This is one of the reasons that architecture frameworks and architecture-driven approaches are becoming more popular among companies with LSI responsibilities.

Our discussions with the systems engineers working for LSI companies revealed that significant amount of the required knowledge to develop the SoS is already available at the beginning of the project. Majority of the constituent systems exist and are serving for the well-defined operational objectives. In addition to this, for various reasons such as customer preference, buying strategy of the company or technology used in legacy systems there were significant constraints (lower layer known knowns) on each layer of the system design at the very beginning of the project. Therefore, initial knowledge presented in Figure 9 depicts a more realistic case in comparison to the one presented in Figure 8.

**Success Factor 1:** It is critical that especially the LSI company understands the specific characteristics of the SoS projects and the shortcomings of the top-down processes. Adopt a knowledge-based and architecture-driven SE approach.

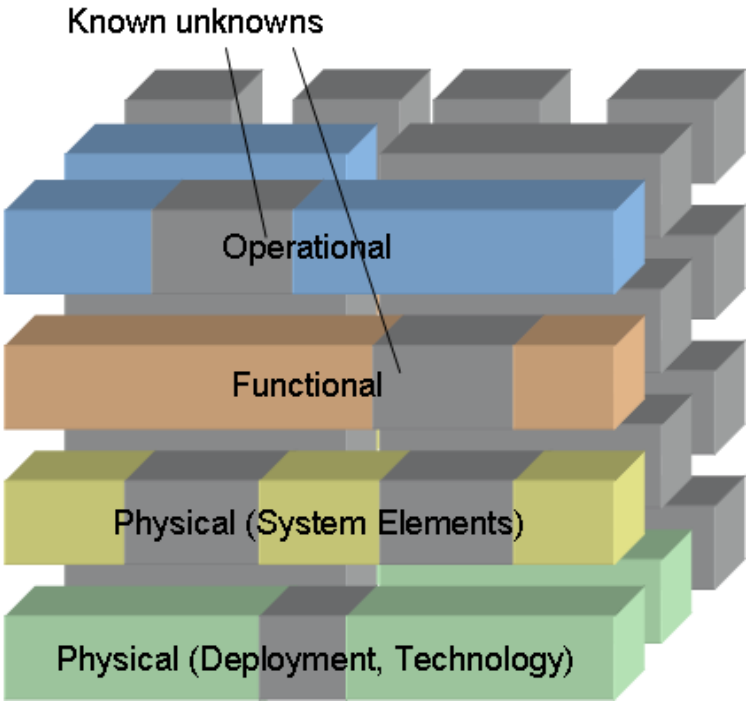


Figure 9: Initial Knowledge at the Beginning of a Project is Spotty

#### 4.1.4. Architecture Knowledge vs. Rubik's Cube

Although Figure 9 depicts a more realistic situation, from the LSI perspective, it still does not emphasize the importance of the interaction/relationships between different domains or semantic layers. Recall the classical decomposition schemes discussed in the previous sections. The "parts" in decompositions have relationships to parts in other decompositions (e.g. Figure 10). From the system of systems integrator perspective, identification and management of these relationships in a SoS project is one of the main problems (as part of the **conduct SoS analysis** in the wave model). LSI needs to consider all of these aspects together.

Our understanding of "architecture knowledge" may be best viewed as a Rubik's cube. Consider each face of the Rubik's cube as representing the knowledge in one domain. It is not possible to justify knowledge in one domain in isolation from the other knowledge domains. The correct strategy to solve a Rubik's cube is to focus on fixing all the faces at once. Beginners try to solve the faces one by one. Once they fix one face they think that this is a step forward. With this strategy, it is almost impossible to fix other faces without breaking the already fixed one.

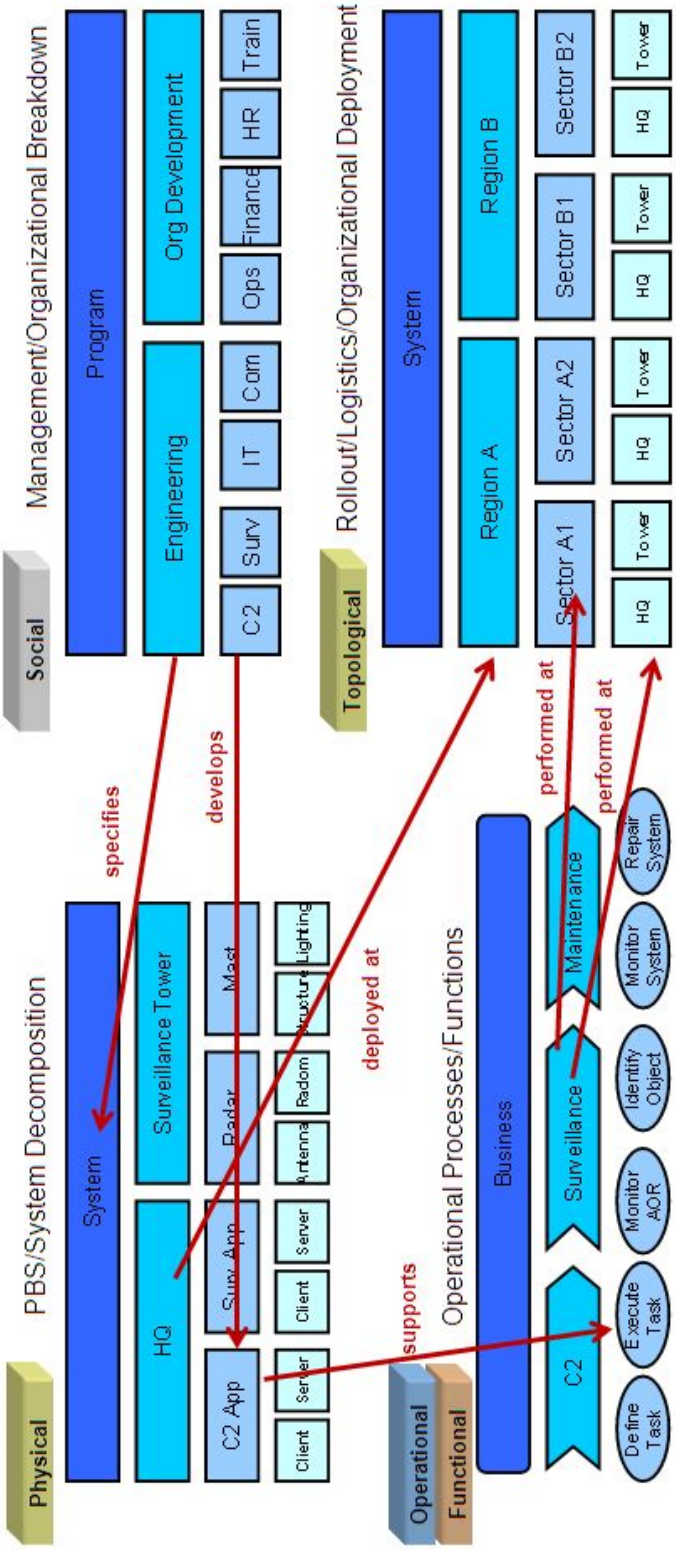


Figure 10: Project Effort Reflects Different Decompositions

Traditional SE never advises to follow a linear and waterfall-like approach. Even for the large scale projects, highly iterative, incremental and risk based approaches which propose concurrent engineering are available in the SE literature. On the other hand, application of such methods in large scale SoS projects depend on many factors like politics, maturity of the customer, maturity of the system providers, type of the contract, understanding the specifics of SoS projects, etc. As a result, in a recent research, we have observed that many of the SoS projects still exhibits waterfall characteristics. Unfortunately, they are trying to solve a Rubik's cube one face at a time.

For example, stakeholder requirements analysis phase ends when the operational knowledge is captured in the CONOPS document and in operational requirements document. Once these documents are completed (i.e. this operational domain face is fixed), system requirements phase starts and uses these documents as input. However, according to the situation depicted in Figure 9, we already have significant knowledge (i.e. known knowns) about existing system/subsystem functions, equipment and technology during the stakeholder requirements analysis phase. Top-down driven culture has a tendency to ignore this. As a result, SoS level operational architecture is developed with little understanding of the constituent systems. When the next layer's knowledge becomes available in the process, it becomes necessary to change the decisions made in the upper level. This is similar to the need to break the already fixed face of the Rubik's cube while trying to fix the next face.

**Success Factor 2:** Develop SoS level solution architecture like solving a Rubik's cube. Operational, technical, organizational, etc. aspects should be developed simultaneously considering the big impact of the available knowledge from the existing constituent systems.

### 4.1.5. Organizational Aspects in SoS Level Knowledge Management

There is also an organizational aspect of this problem. Knowledge is bound to individuals and in today's system projects no individual team member has the complete picture of the solution. As demonstrated in Figure 9, majority of the required knowledge to understand the problem and to formulate the solution consists of known knowns. This means that the required knowledge to design and describe the solution mostly exists in the engineering organization. However, clusters of known knowns are distributed among the individuals and across the teams (Figure 11).

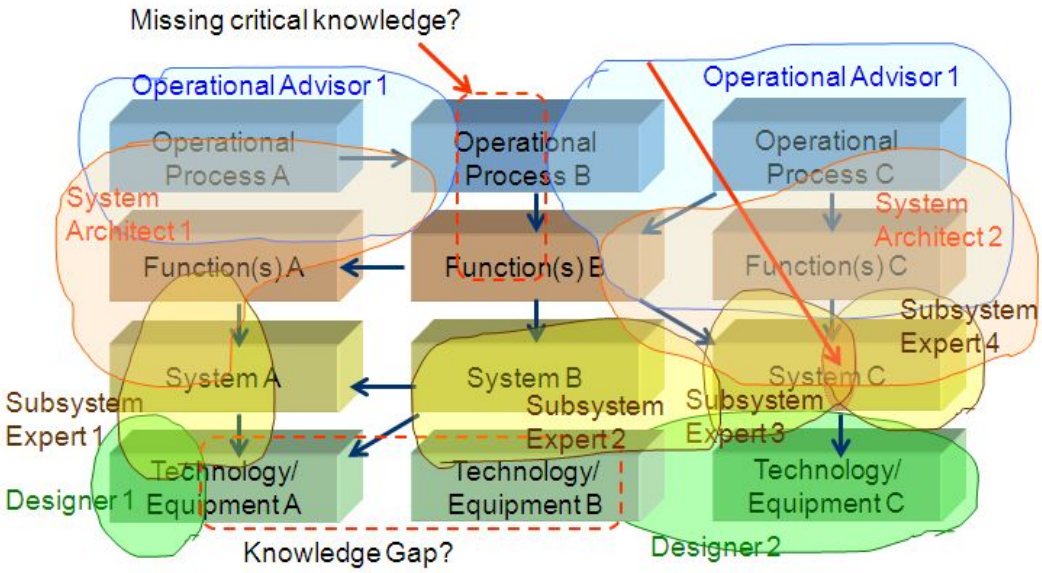


Figure 11: Knowledge is bound to individuals

The situation depicted in Figure 11 represents a system project. In a SoS environment, the problem is even more complicated. Let's try to illustrate the difficulties using Figure 12. As already described, LSI is responsible for the delivery of the SoS level capabilities. On the other hand, capability objectives at the enterprise level or SoS level depends on the constituent systems. Having existing systems that are already being used by some organizations to support their operational problems may be considered as an advantage. This means that we have significant knowledge around that system.

As an example, consider the search capability that is supported by an AWACS system that is operated by the Air Force organization. The company that is responsible for the development of the AWACS system knows how the system operates to support different operational users in order to help them perform search related operational activities. They are knowledgeable about the operational units involved in these activities and how they interact. They also know how the operations related to search capability interacts with the other mission domains such as command & control and intelligence.

Now let's consider a scenario where the customer wants to achieve a higher level search capability by being able to operate different operational units, within the same scenario and by combining their individual capabilities to create a synergistic effect. This requires integration of technical and nontechnical resources. As an example, consider constituent systems as an AWACS system operated by the Air Force, a border security system that is operated by the Border Guard (or Ministry of Defense) and a satellite system that provides services to the government's Intelligence Agency. In addition to these existing systems,

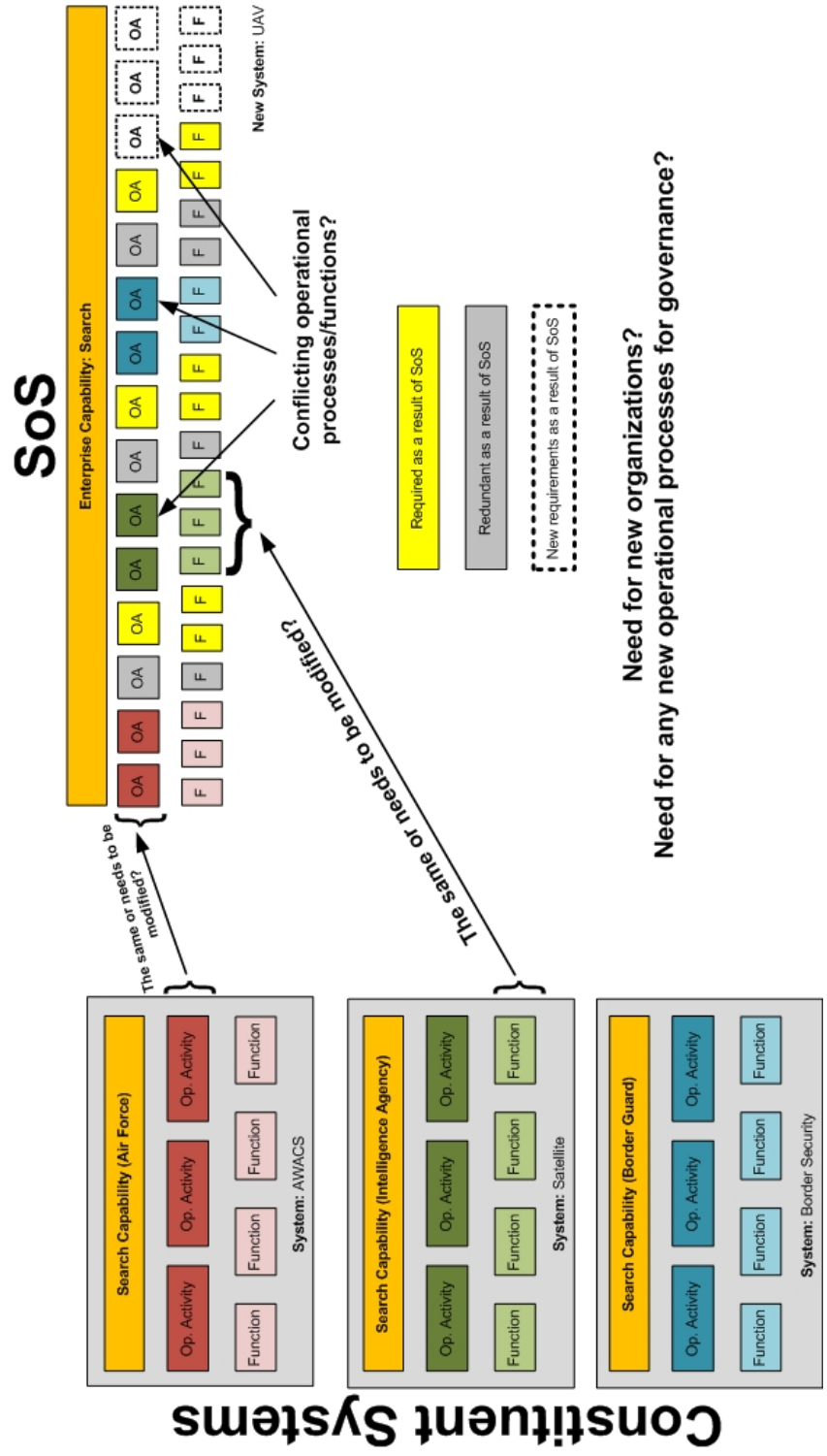


Figure 12: Challenges in eliciting SoS level knowledge from available systems

acquisition of an UAV system is also considered within the SoS project. Each of these systems is operated by different organizations to support their search capabilities.

When an LSI company gets the responsibility to integrate the technical systems into a SoS to help achieving enterprise level (i.e. government level) capabilities, the most critical knowledge he will be interested in will be the knowledge about the interaction/relationship between the architectural elements of different domains, missions and organizations. LSI will try to understand the interaction between the operational activities of the different organizations. In the best case, the knowledge of how the Air Force performs search missions (e.g. operational process A in Figure 11) using AWACS provided system functions (e.g. function A in Figure 11) and how the Border Guard performs search missions (e.g. operational process C in Figure 11) using functions provided by the border security system (e.g. function C in Figure 11) are already available to the LSI. LSI needs to use this available knowledge (**understanding systems** as part of **initiate SoS** in the wave model) to figure out SoS level operational processes and SoS level functions (e.g. need for a new operational process/system function like operational process B/function B in Figure 11). In such a situation, traditional team structures will not help the LSI. Such organizations are set up to work in fixed predetermined knowledge areas and not to address the particular knowledge gaps of a SoS project. Please note that the example used in this example covers the same mission domain like surveillance. The problem will be even more difficult when different domains and different organizations are involved.

**Success Factor 3:** It is vital to determine the nature of a project by charting available and missing knowledge and understanding which knowledge resources are needed to fill the SoS level knowledge gaps. Instead of fitting the problem(s) to pre-existing organizations the most efficient use of knowledge owners is to arrange them according to the problem. Even the project team of the customer may not be able to have a vision on the SoS level emergent capabilities. This knowledge can only be elicited in **collaborative environments**.

### 4.1.6. Agility in SoSE via Focus on Knowledge

According to our model there are two ways to acquire new knowledge. The first is a result of the transition of some KUs into KKs. New knowledge becomes available which was already identified as valuable but missing. The second is a result of the transition of some UUs into KKs. New knowledge becomes available and at the same time we discover that this new knowledge affects the project. In both cases it is expected that new knowledge triggers changes on the previously made decisions. There is one other transition in the model which is from the UUs to the KUs. Missing valuable knowledge that would affect decisions is discovered. This is the transition showing that systems engineers are supposed to define risks.

The transitions from UUs are uncontrollable. On the other hand, transitions from KUs to KKs can be controlled. In the SoS projects, customers want to be on the safe side by having a firm fix-price contracts. On the other hand, LSI companies try to fix the concept, requirements, expectations, etc. as much as possible to reduce their risk

due to such contracts. As explained above, the result is a waterfall-like process culture that is not neutral to changes. In reality, in the SoS projects, the delivered system is never the one that is initially described. Throughout the project lifecycle, systems engineers should seek information to fill the gaps in the initial description. All the practices in the SE process consist of ways to learn about the problem and the solution being developed (Turner, 2007). Current linear and document oriented SE practice cannot satisfactorily focus on identifying and capturing required knowledge. Most new knowledge arises in an uncontrolled fashion and hence comes late in the project. Therefore, the changes initiated by such late and uncontrolled knowledge have costly consequences. The surprising aspect of this result is that most of these costly changes can be categorized as expected changes as they are triggered from the KUs.

**Success Factor 4:** By identifying the valuable but missing knowledge (i.e. KUs) early in the project and by tailoring the SE processes to focus on the required knowledge, shift these transitions to early phases and thus reduce the costs of changes.

## 4.2. Formulation of the Approach

Based on the foundations presented in the previous section, a knowledge-driven architecture approach that demonstrates improvements in the agility attributes was the overall objective of the present study. The wave model is selected as a reference SoS process. A collaborative and knowledge-driven approach is further formulated by seeking an answer to: How can an LSI company implement the wave model at the lowest, that is, the most practical, level? Success factors that are defined in the previous sections and the improvement objectives that are defined during the survey part of this research are used as high level objectives and guidelines. Following steps represent the outline of the proposed approach addresses **Objective2** by clearly focusing on knowledge elicitation activities:

- Identify critical knowledge for SoS level (i.e. KKs and KUs),
- Capture available/existing knowledge (i.e. KKs),
- Identify and plan the effort to acquire missing knowledge

### 4.2.1. Modeling the Solution Knowledge Space

In order to define and manage SoSE activities that focus on acquiring missing knowledge, the first step is to identify the knowledge space of the socio-technical solution to be developed. This is the knowledge that is needed for the elicitation of the problem and for the development of the SoS architecture that addresses the problem. The SoS knowledge space (SKS) will be the union of KKs and KUs. The problem is how to formulate the SoS knowledge. What are the alphabet, words and sentences that will be used in describing the problem and the solution? We need to develop a logical framework for organizing and representing the knowledge about a SoS architecture. Such a logical framework defines the knowledge elements as building blocks, attributes of these elements and the relationships among these

knowledge elements. This logical framework or the SKS will be used as a common guideline by the LSI SE team to capture the SoS architecture knowledge in a consistent manner.

For the development of the SKS, domains and nodes defined by (Bartolomei, Hastings, Neufville, & Rhodes, 2011) were adopted. They first define a conceptualization to formalize the identification of the domains that are common to all engineering projects (e.g. environmental domain, social domain, functional domain, etc.). They then define six classes of nodes that are important in describing an engineering system (e.g. stakeholders, functions, objectives, activities, etc.). Each node corresponds to one of the "engineering systems" domains.

In order to model the SKS, the first step for the LSI SE team is to establish a common understanding around the concept of socio-technical system. This can be achieved by a storyline around the problem and the solution. For example, in general, customers' ultimate purpose is to increase the effectiveness of their enterprise. In the defense environment, effects are formulized by the definition of missions and tasks. Tasks are assigned to organizations and are achieved as a result of operational activities. A group of coherent operational activities can be logically grouped under the responsibility of an operational node (e.g. intelligence node, surveillance node, headquarters, etc). Lowest level operational activities can also be performed by the operational users. Technical systems support operational nodes and users in performing the operational activities by the functions they provide. As a result, realization of the operational success (i.e. desired effect) depends not only on the technical system but also on operational users and operational processes. Based on the socio-technical solution concept described above, six knowledge domains were defined: environmental, social, operational, functional, technical and physical. For each of these domains we further defined the knowledge elements which will be used for representing the knowledge in that domain. Environmental domain represents the exogenous factors (SoS drivers) that are to be provided to the LSI systems engineers. Operational domain knowledge will be captured using the missions, tasks and operational activities as knowledge elements. Functional and technical domains represent the logical and physical architecture of the system respectively. Physical domain represents the deployment architecture for the operational units (operational nodes and users).

The resulting knowledge space is captured in a multiple-domain matrix (Figure 13). On the matrix, diagonal cells represent the relationship among the same knowledge elements and off-diagonal cells represent the relationships between different knowledge elements. In our approach we use only half of the matrix as we are only interested in the relationship between the knowledge elements but not interested in which knowledge element affects the other. Responsibility of the LSI SE team is highlighted with red dotted line as the system drivers represent the exogenous drivers on the solution. Including operational domain knowledge within the scope of SE activities addresses **Objective1**.

| | Environmental | Social | | Operational | | | Functional | Technical | Physical | |
|---|---|---|---|---|---|---|---|---|---|---|
| | System Drivers | Operational Node | Operational User | Mission | Task | Operational Activity | System Function | System | Location | Physical Workplaces |
| System Drivers | SDxSD | ONxSD | OUxSD | MxSD | TxSD | OAxSD | SFxSD | SxSD | LxSD | PWxSD |
| Operational Node | | ONxON | OUxON | MxON | TxON | OAxON | SFxON | SxON | LxON | PWxON |
| Operational User | | | OUxOU | MxOU | TxOU | OAxOU | SFxOU | SxOU | LxOU | PWxOU |
| Mission | | | | MxM | TxM | OAxM | SFxM | SxM | LxM | PWxM |
| Task | | | | | TxT | OAxT | SFxT | SxT | LxT | PWxT |
| Operational Activity | | | | | | OAxOA | SFxOA | SxOA | LxOA | PWxOA |
| System Function | | | | | | | SFxSF | SxSF | LxSF | PWxSF |
| System | | | | | | | | SxS | LxS | PWxS |
| Location | | | | | | | | | LxL | PWxL |
| Physical Workplaces | | | | | | | | | | PWxPW |

Figure 13: Knowledge elements represented in a Multi Domain Matrix

### 4.2.2. Identification of the Relationships in a Meta-model

Matrix representation of the knowledge elements improves our ability to visually observe relationships between all knowledge elements. However, in order to handle some of the complex relationships some artificial knowledge elements are created. For example, in order to handle the relationship between operational activities, operational scenarios are defined as a knowledge element. Similarly, use cases are defined to capture the relationship between systems, functions, operational activities and operational users. In order to have a better visualization of the overall knowledge space the results are captured in a meta-model (Figure 14). Please note that the meta-model defined for the SoS level includes significant number of knowledge elements from the operational domain addresses **Objective1.**

### 4.2.3. Collaborative Modeling with Deep Dive Architecture Workshops

One of the important contributions of the trapeze and wave models is to introduce understanding systems as a core element for the SoSE. In the wave model, this element is performed as part of conduct SoS analysis in the first wave. It is important to note that conduct SoS analysis comes before the develop SoS architecture. This point is quite important to understand the role of the LSI SE team. The success of the SoS architecture depends significantly on the understanding of the constituent systems. Another aspect of this point is the need for agility in understanding systems. Efforts of the providers of the systems will be defined based on the SoS architecture. In the SoS projects this must be done as early as possible.

Deep dive architecture workshops are defined to help LSI SE teams to deal both with organizational issues and the problems due to linear project phases arising from cultural and contractual factors. Collaborative modeling with deep dive architecture workshops consists of four main activities:

- Eliciting existing knowledge,
- Visualizing existing knowledge through modeling,
- Verifying consistency of the knowledge by continuous reviews, and
- Identifying & planning for missing knowledge.

Improvement in the agility of the process is mainly targeted with 1) performing these activities concurrently within the same workshop and 2) with the collaboration of all available knowledge sources. Workshops are full day meetings with attendance of all architecture team and other required knowledge sources. Discussions originate from the known operational capabilities that are supported by the existing constituent systems. In our approach, capabilities are characterized by a set of operational scenarios demonstrating the well-known operational problems. Primary objective in the workshops is to first capture the available system knowledge in the SoS model repository in terms of the knowledge elements defined in the SKS. This is done by capturing knowledge in all different domains and in different layers (i.e. like solving a Rubik's cube). At the end of the day, one can validate the capability requirements

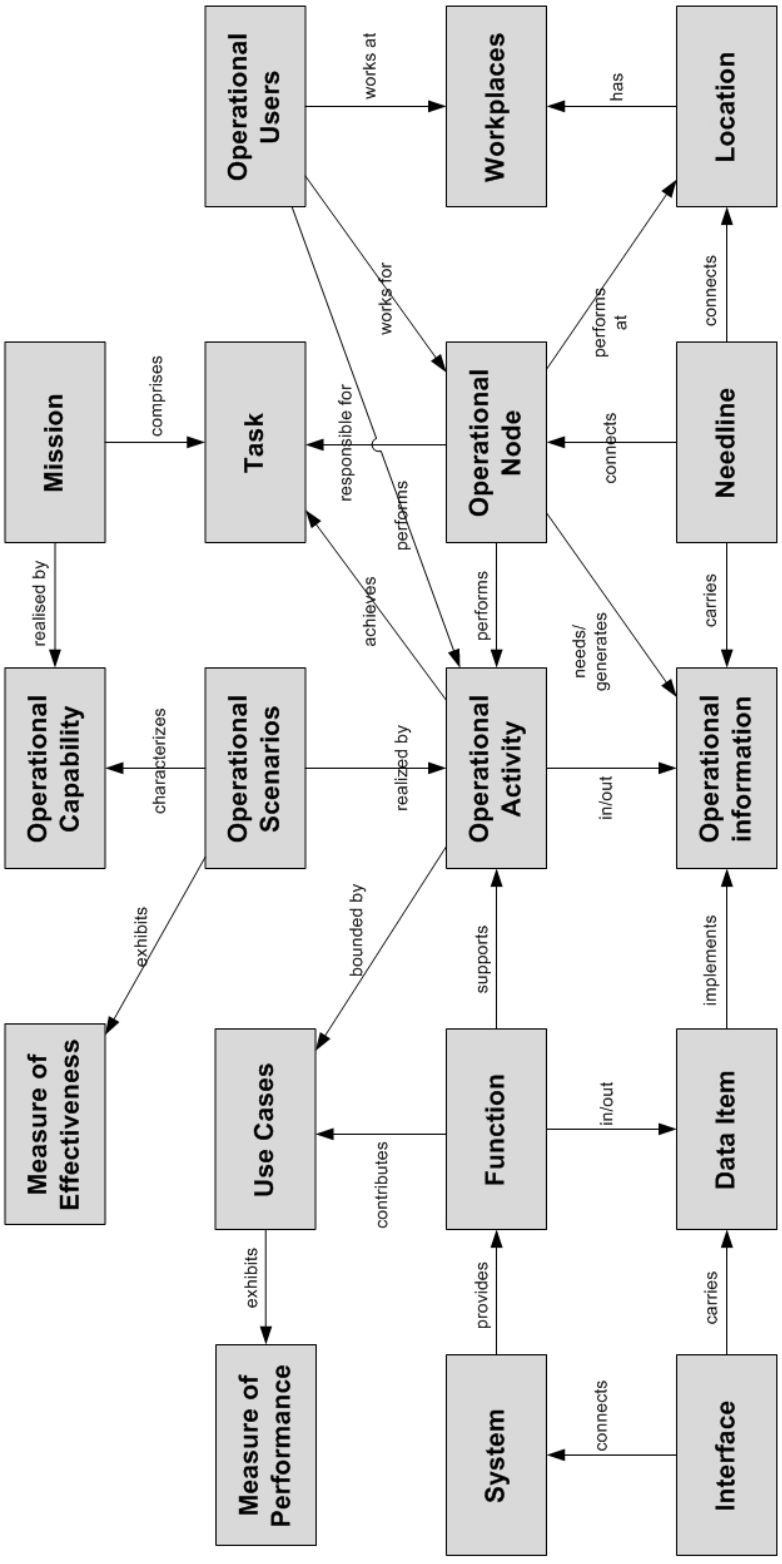


Figure 14: SKS represented as a Meta-model

in a top down manner starting from missions to the system functions and interfaces. That is why these workshops are named as deep dive architecture workshops. Please recall that the designs of the constituent systems are under the responsibility of the system providers and main interest and responsibility of the LSI SE team is the knowledge around the architecture.

Starting from the high level capability definitions (e.g. fixed surveillance) we develop a model of the initial solution concept in terms of existing operational nodes, their information needs, the activities performed by them and how the existing systems support these capabilities with their functionalities. Available knowledge captured in terms of UML or SysML models facilitates communication and establishes a shared understanding among the stakeholders on the high level as-is architecture. Such an understanding of the available knowledge (i.e. KKs) will also initiate identification of the missing necessary knowledge at the SoS level. Based on the missing knowledge, LSE SE team identifies the required knowledge sources and domain experts that are necessary for the following activities (Figure 15).

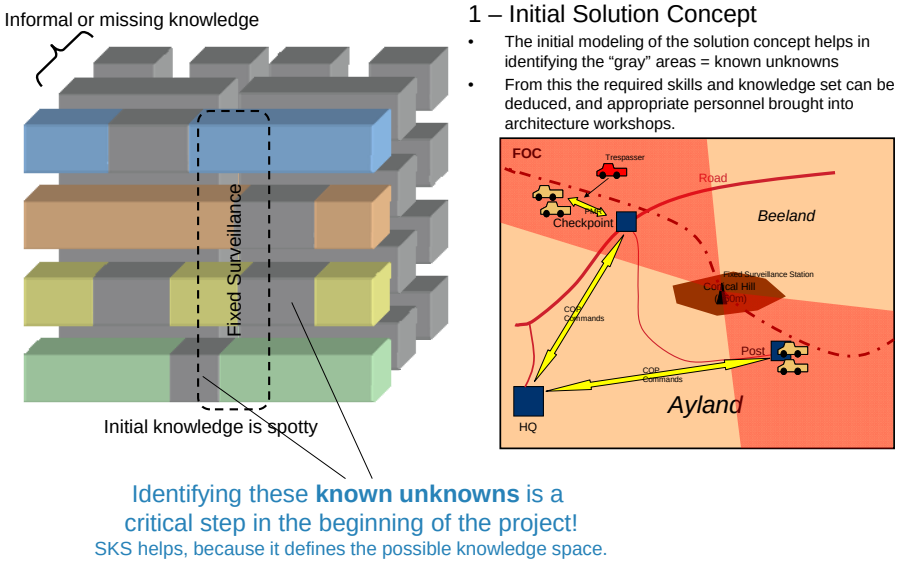


Figure 15: Start with initial modeling of the SoS concept

Deep dive architecture workshops bring different sources of knowledge in the same meeting. Discussions on the problem and solution scenarios are the key to externalize such knowledge embedded with the individual experts. Modeling experts serve to immediately develop models during the workshops and makes the elicited knowledge visible to all participants. Progressively elaborating the details in the SoS model turns unknowns into available knowledge but also helps discovering new known unknowns (Figure 16). Each system function, each operational activity, etc. defined in the model reflects a decision. Immediately visualizing the decisions (in terms of model elements and their interaction) helps further identifying new discussion topics and identification of the missing knowledge.

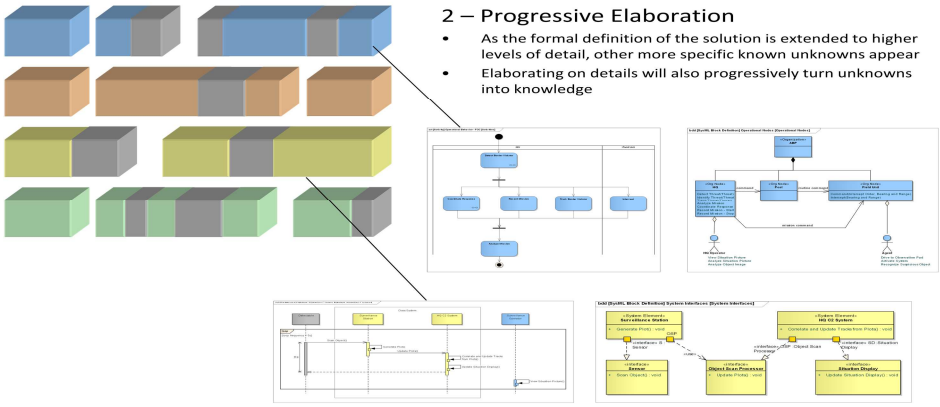


Figure 16: Progressive elaboration of the solution by modeling

It is not realistic to target capturing all the missing knowledge in a SoS project. However, there is a point where the LSI systems engineers decide that it is captured knowledge and the modeled SoS architecture is good enough to baseline. The activities performed in the deep dive architecture workshops are summarized in Figure 17.

### 4.2.4. Knowledge Circles and Time-boxed Planning

Lead SoS engineer's main responsibility is to make sure that the workshops focus on modeling the existing knowledge in a steady manner. Whenever the team cannot proceed due to missing knowledge, it is his responsibility to stop discussions and plan for the missing knowledge. In our approach, every KU has a consumer and a supplier within the team. For example, consider a scenario that operational node A operates constituent system A and operational node B operates constituent system B. In the SoS level architecture these two nodes need to collaborate to achieve enterprise level capability goals. In order for systems engineers of the constituent system A and B to decide whether they need to implement new functions or modify the existing ones they need to understand how operational node A and B will behave in the SoS environment (missing knowledge). In this scenario consumer is the system engineers A and B; and suppliers are the operational advisors that are responsible for the operational domain knowledge. Involving the systems engineers of the constituent systems in SoS architecture team and using them to identify necessary knowledge sources from their organizations and involve them in knowledge circles clearly addresses the **Objective4**.

Consumer defines the scope of the knowledge, level of detail and the format to capture the knowledge. Lead SoS architect validates the requested knowledge and level of detail. Assigned supplier is expected to outline a plan for the delivery of the required knowledge during the workshop. Supplier also defines the needed resources for the activities and gets the approval of the lead SoS architect.

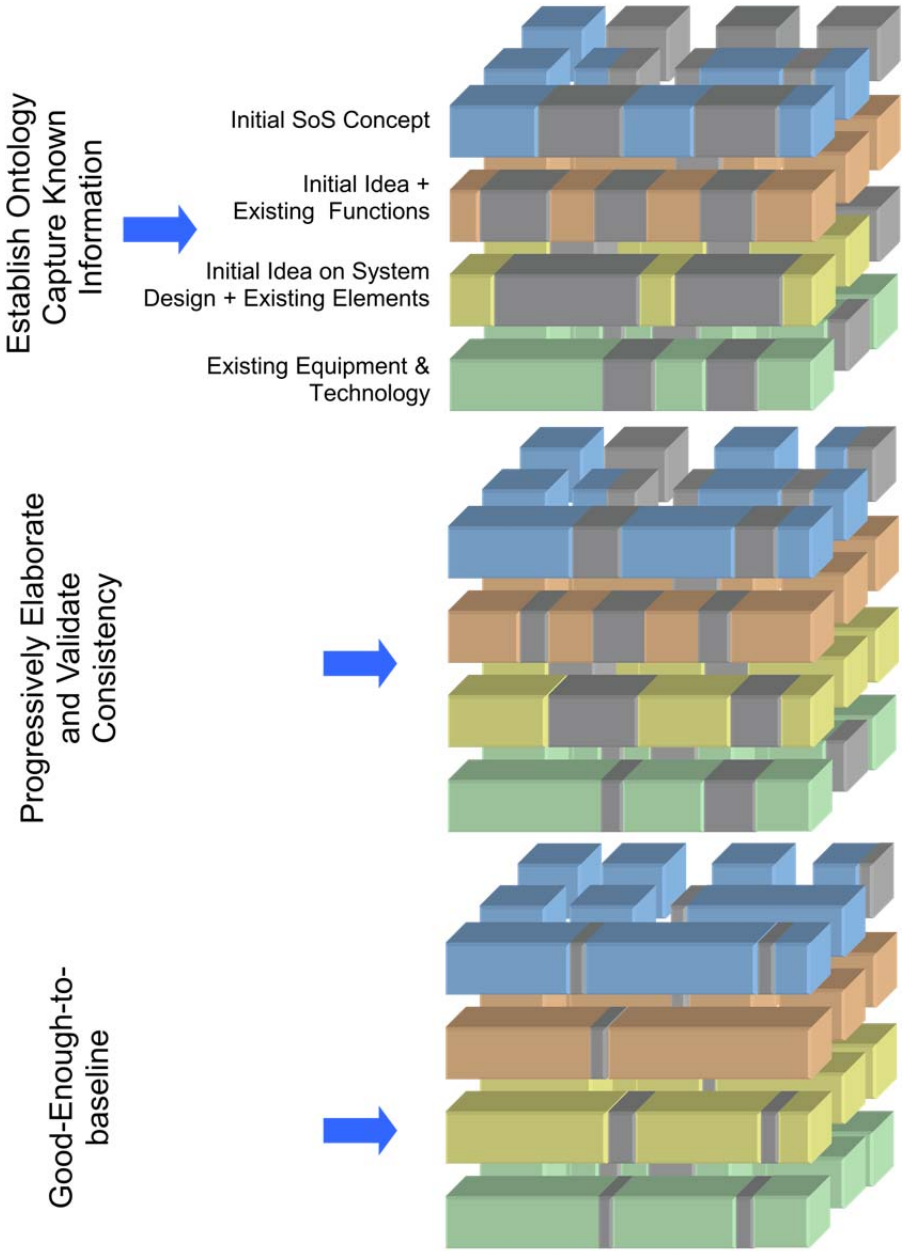


Figure 17: Overview of the activities in deep dive architecture workshops

As soon as the consumer and supplier agree on the due date based on the scope, required effort and available resources, KU is recorded into the knowledge backlog by assigning a prioritization. Supplier is responsible to establish a team using the identified resources. These teams are called “knowledge circles”. They are temporary IPTs that are arranged according to the knowledge gaps at the SoS level.

Our approach also adopts the concept of time-boxing from agile software engineering. The plan for acquiring required knowledge can take at most two weeks. For example, for an interoperability issue or some missing performance data, the architecture team can perform some tests in the field or they can plan for a prototype to acquire the necessary knowledge. Such activities can be completed in short time. On the other hand, some known unknowns require more effort and planning, especially if it involves more than one knowledge domain and the required knowledge source is not inside the SoS architecture team. If during the planning activities within the workshop, it is understood that the effort requires more than two weeks, this KU is escalated to senior management (e.g. by creating a new risk in program level risk board). Similarly, if a KU which is planned for at most two weeks cannot be acquired within four weeks this KU is also escalated. Escalation means that LSI SE team cannot handle the situation alone and needs some management support. In this approach, all the critical issues are escalated to the attention of the senior managers without losing time. Deep dive architecture workshops should be performed at least once every week starting with the presentation of the analysis results and summary of the knowledge backlog status; and ending with the overview of the planned activities.

### 4.2.5. MBSE and Requirements Last Approach

Traditional SE approaches are also known as "document-driven" or "document-heavy" processes. Producing a lot of documentation is not the main problem of the traditional SE processes. Every piece of information in the documents somehow reflects a decision. Systems engineers need to deliver the documents with the required decisions within a pre-determined timeframe ended at a project milestone. In this approach they are expected to make the decisions when the necessary information is missing. The rest of the design and development activities (and the associated documentation) are based on such ill-made decisions. Significant effort and time is spent on changing the decisions when the required information is available and re-working on the documents in the later phases of the projects.

In order to deal with such issues, a requirements last approach is adopted. This is in contrast to the traditional approach where every activity starts with a set of requirements from the layer above. In this completely model driven approach no requirements are written until the model is frozen (i.e. good enough to baseline in Figure 17). All the knowledge management activities are performed on the model. Existing knowledge is captured in terms of models that are supported by textual explanations when necessary. Until it is evaluated that the knowledge captured in the model repository is mature and stable enough, no requirements are written. As the model is frozen, documents are generated based on the knowledge captured in the

model repository. Adopting requirements last approach by MBSE addresses **Objective5**

### 4.2.6. Continuous Customer Trainings

In the new approach continuous customer trainings are used as a tool to increase the effectiveness of the customer collaboration by starting the training sessions very early in the project and conducting them in a periodic manner. In the current practice, trainings are used to teach the users and maintainers how to operate and sustain the delivered system. That is why the trainings are not planned until almost all the features of the system are designed and implemented. Conducting training activities in this manner does not provide any benefits to the LSI company other than fulfilling contractual obligations.

In the proposed approach trainings are performed every three weeks starting from the contract award. First session is the SE training to provide the customer with an understanding of the system life cycle, relevant SE activities, challenges of the SoS and the approach to address the challenges (introduction to the trapeze and wave models). In the second training session, initial understanding of the enterprise capability objectives and how proposed SoS solution matches to these objectives is shared. Starting from the third training session, the operational users are trained for their specific roles based on the operational scenarios that span more than one organization and more than one mission domain (e.g. interaction between surveillance and C2). For validation purposes, system behavior is presented to the users by simulated user interfaces.

It is worth emphasizing that the operational scenarios used for the trainings are cross organizational scenarios. System providers already know how the standard missions are performed by the operational units of an organization. On the other hand, in a SoS level scenario like the one depicted in Figure 12, even the users that are coming from different organizational units and from different mission domains do not know how they will operate in a collaborative way to satisfy enterprise capability objectives. Different operational units will have overlapping capabilities, conflicting situations, lack of coordination, etc. during this kind of scenarios. These are the real issues for the LSI company.

In the proposed approach, **Objective3** is achieved by using trainings as a tool to:

- Initiate interaction very early with the customer,
- Increase the effectiveness of the interaction with the customer,
- Elicit and resolve stakeholder conflicts (especially cross organization/mission conflicts) within the customer organization,
- Continuously validate the solution concept and evolving the SoS architecture

### 4.2.7. Overview of the Proposed Solution

Main agility improvement in the proposed approach is targeted by performing different knowledge elicitation activities concurrently with the collaboration of available resources. The proposed approach constitutes a contrast against the

traditional SE where the activities are linearly defined in a waterfall like process. On the other hand, in order to help the practitioners to understand and implement the proposed approach, core elements are transformed to a more familiar process view as a series of time-sequenced major activities.

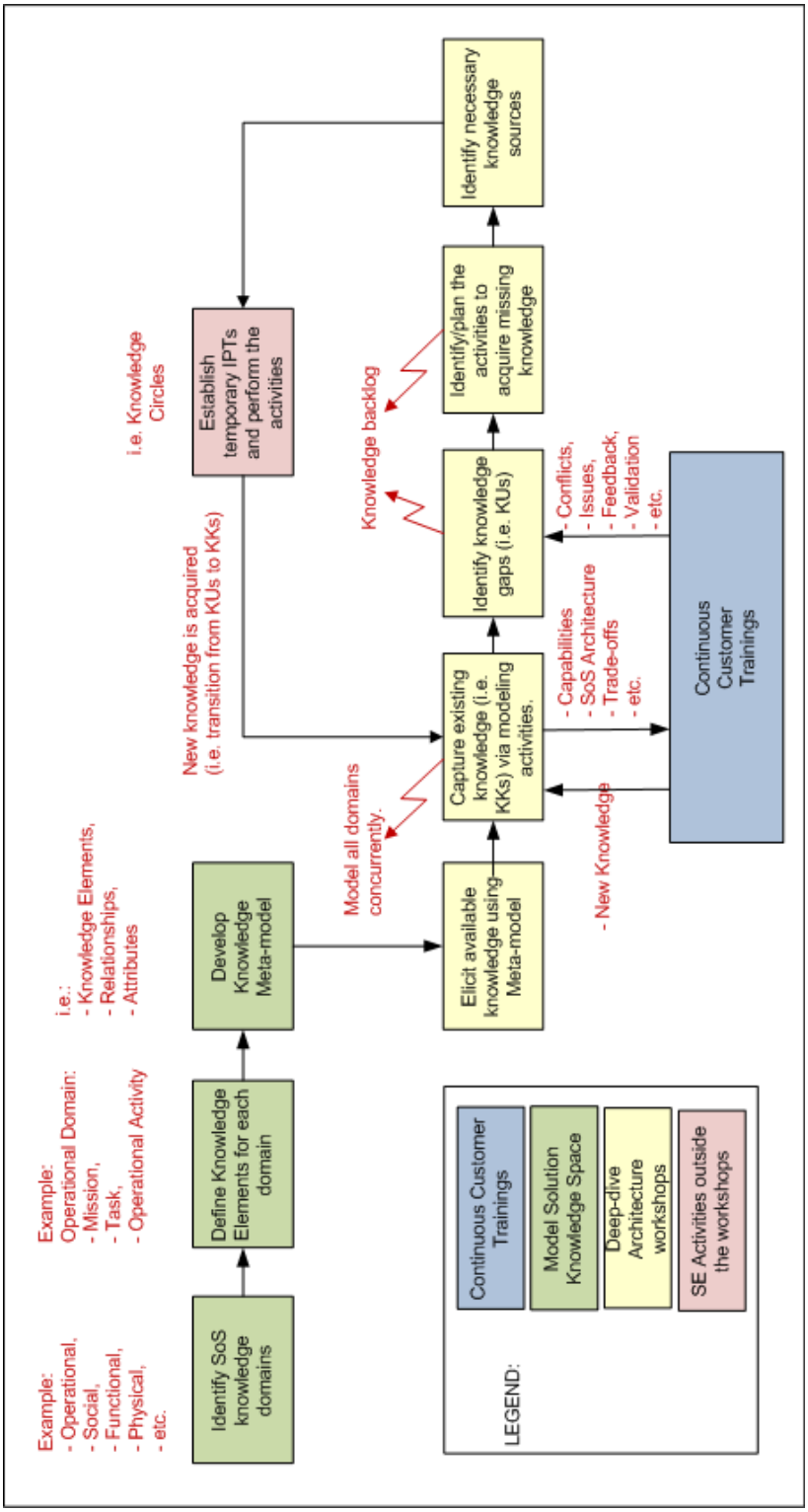


Figure 18: Overview of the proposed approach

# CHAPTER 5

# CONFIRMATORY CASE STUDY

The knowledge-driven and collaborative approach that is formulated in the previous section has been applied and evaluated in a confirmatory case study (Boudreau, Gefen, & Straub, 2001). The analytical research paradigm is not sufficient for investigating complex real life issues, involving humans and their interactions with technology. The case study methodology is well suited for many kinds of SE research, as the objects of study are contemporary phenomena, which are hard to study in isolation. Case studies do not generate the same results on e.g. causal relationships as controlled experiments do, but they provide deeper understanding of the phenomena under study. Reconsidering that knowledge is more than statistical significance; it is possible to learn significantly from case studies by applying proper research methodology practices (Flyvbjerg, 2006; Runeson & Höst, 2008). In this section we shall present the setting, findings and evaluations derived in that context.

## 5.1. Case Study Design

### 5.1.1. Case Study Context

The Company[3] is a worldwide SoS integrator providing large scale solutions to civil and military customers around the globe. Air systems (aircraft and unmanned aerial systems), land, naval and joint systems, intelligence and surveillance, cyber security, secure communications, test systems, missiles, services and support solutions are among the solution portfolio of The Company. Border security projects constitute the top strategic objectives of The Company. Although it has been almost 10 years since The Company started to invest in this area, they have not yet demonstrated success in managing border security projects as programs or as a business line.

The Company manages the border security business line by projects. There were five ongoing border security projects in The Company. The proposed approach was validated in one of these border security projects. Comparison of the project characteristics are summarized in Table 11.

[3] Actual name of the organization is disguised for confidentiality reasons.

Table 11: Comparison of Border Security Projects

| | Planned Duration | Budget (Million Euro) | Constituent Systems | Customer | End User | Border Length | User/System Requirements |
|---|---|---|---|---|---|---|---|
| Project 1 | 4 | 580 | 7 | Ministry of Interior | Border Police | 3100 | 800/1800 |
| Project 2 | 5 | 800 | 11 | Ministry of Interior | Joint | 900 | 950/4400 |
| Project 3 | 5 | 2100 | 10 | Ministry of Interior | Joint | 1400 | 1150/2400 |
| Project 4 | 6 | 2600 | 11 | Navy | Joint Armed Forces | 3500 | 1050/2000 |
| Case Study Project | 5 | 2100 | 8 | Department of Defense | Joint | 4500 | 1400/2200 |

In the case study project, The Company was the lead system integrator for a large scale software intensive border security project. Eight subsystems were standalone systems developed for different domains such as command and control, communication, intelligence, surveillance, etc. All these constituent systems were available in the market as almost COTS systems. Minor customizations on the hardware and software were done according to project specific needs. Obviously the end system was a SoS. The solution was foreseen as an integration of the already existing systems and products with as little as possible new development. This was the understanding both on the customer side and on The Company side.

Five of these subsystems were developed by The Company itself and the other three were subcontracted to external companies. As expected, both the subcontractor teams and the teams of The Company that were responsible from these systems were reluctant to implement new functionality or to modify the existing ones. The Company and the subcontractors were using traditional SE processes and no specific effort was performed to tailor the processes according to the SoS context. As a result, requirements of the constituent systems were used to define the SoS level requirements. For example, the requirements of the communication system were sampled to generate a set of SoS level requirements to capture the communication aspects of the overall integrated system. It was assumed that each constituent system was mature enough to cover that aspect of the integrated system and the requirements already developed for that existing system would serve as SoS level requirements.

The project was planned for 5 years. Some of the constituent systems were already deployed on the customer site to demonstrate functionality. In the second year, an internal system design review (SDR) was performed to evaluate the maturity of the SoS level architecture before performing the real SDR with the customer. It was a failure for the project. Results demonstrated that the SoS architecture and the SoS requirements were not properly defined. Most of the requirements were defined at the wrong level. Majority of the SoS level requirements were talking about the details of the constituent systems' design but did not give an idea about what the SoS shall do and why. The project review board decided that although the system was partially deployed at the customer's operating environment and demonstrated some functionality, the logical link between the functioning system and the operational need was missing. Additionally, a heavy criticism of the board was that little -if any- consideration had been devoted towards non-functional characteristics of the system that would be necessary to ensure fit for purpose. More technically, top-down requirements traceability was not consistent and justifiable.

As a result, this internal decision gate resulted in a failure. The review with the customer was canceled. It was decided by the review board that the overarching system (i.e. SoS) architecture was missing. A set of action items which constituted a new work package for defining the SoS architecture were agreed upon with the key stakeholders and the chairman of the SDR review board. As the system was already partially deployed, it was not possible to define system architecture from the scratch by a pure top-down approach by ignoring the existing implementation.

### 5.1.2. Case Selection and Research Questions

The selected project for the case study has some specific characteristics that is important for the evaluation of the proposed approach. First of all, as described in the previous section, the selected project for the case study clearly demonstrates main SoS characteristics such as:

- Operational Independence of Elements
- Managerial Independence of Elements
- Geographical Distribution of Elements

There are eight constituent systems contributing to the SoS. Five of these systems are developed by The Company and the other three are developed by subcontractors. Each of the constituent systems are standalone systems that are independently developed using different processes and different life-cycles.

The Company and the subcontractors were using traditional SE processes and no specific effort was performed to tailor the processes according to the SoS context. Therefore, the results of the case study will clearly help us to compare the proposed approach that is developed specifically for the SoS projects with the results of the traditional SE approach.

Internal system design review was performed to evaluate the maturity of the SoS level architecture before performing the real SDR with the customer. It was a failure for the project. Results demonstrated that the SoS architecture and the SoS requirements were not properly defined. The project review board decided that although the system was partially deployed at the customer's operating environment and demonstrated some functionality, the logical link between the functioning system and the operational need was missing. Such findings of the review board are very important for the evaluation of the proposed approach because the findings have clearly indicated that the problem is at the SoS level and specifically related to the SoS architecture. The chairman also highlighted the fact that since many of the constituent systems were already deployed and functioning, primary need is to develop the overarching SoS architecture using the available knowledge rather than trying to design the SoS from scratch. This makes the case study project a proper environment for the evaluation of the proposed approach as the proposed approach is designed as a knowledge-focused approach to develop SoS architecture by primarily capturing the knowledge of the available systems.

Finally, due to the upcoming system design review with the customer, the work package for the SoS architecture is defined for six months. Therefore, the management was looking for an agile architecting approach that can demonstrate success within six months. Moreover, as the review board findings indicated, the expected outcome was not only the architecture but also the formulation of the operational problem. Hence, the expected approach should develop both operational architecture and the solution architecture within six months. This again makes this project very important for us as the proposed approach claims to develop knowledge of different domains concurrently within the deep dive architecture workshops.

Hence the following research questions are identified for the case study:

**RQ1:** Is it possible to develop both operational architecture and system architecture and the corresponding requirements (i.e. operational requirements and SoS requirements) concurrently within six months?

**RQ2:** Can the proposed approach improve agility attributes in a real SoS project that has already failed using the traditional SE approach?

**RQ3:** Can Model-based Systems Engineering approach be implemented in a SoS environment where the constituent systems are designed and developed using different SE processes and lifecycles?

**RQ4:** Is it possible to reduce rework on the documentation updates by using requirements last approach and MBSE?

### 5.1.3. Data Collection

Observations of the results and data collection were planned based on four major venues of information exchange between the researchers and staff from The Company:

- SDR with the customer,
- Review with the SE steering group,
- Re-usability workshop within the border security business line, and
- Interviews

Other than the interviews, evaluation of the proposed approach was performed independently from the researchers. During the SDR, customer is evaluated the SE artifacts generated using the proposed approach. SE steering group, consisting of senior systems engineering experts, performed an independent review and evaluated both the proposed approach and the documents generated during the project. Finally, in a re-usability workshop, the approach and the model elements are evaluated by the systems engineers of the other border security projects from the re-usability perspective.

## 5.2. Case Study Results

### 5.2.1. System Design Review with the Customer

The primary expectation of The Company from the new approach was to perform the SDR successfully against the customer. Therefore, application of the proposed approach is limited to the system level architecture and ends with the definition of the subsystem requirements. Duration of the work package defined for this task was six months. Within six months:

- Operational problem was modeled in terms of SoS capability objectives, operational scenarios and operational activities,
- SoS and system level behavior was modeled in terms of use cases and system functions.

- Operational requirements (211 requirements), SoS requirements (577 requirements consisting of functional, physical and non-functional) and system requirements (2679 requirements consisting of functional, physical and non-functional) were defined based on the model.
- Non-functional requirements for SoS and system level were defined based on international standards and based on the expertise of the specialty engineers.
- SoS architecture documents were generated based on the model.

All the requirements and design documents that were generated by the new approach were subject to SDR. SDR was performed against the customer and resulted in a conditional pass. Assessment of the SE artifacts resulted in 78 discrepancies of which 5 were category A, 41 were category B and 32 were observation type of discrepancies. Category A represents critical problems meaning that SDR cannot be passed without fixing those problems. Category B type of discrepancies represent significant issues that need to be solved before the next technical review (i.e. critical design review for our case). SoS architecture team resolved all category A discrepancies within 5 weeks and passed the SDR successfully. All category B discrepancies were resolved in 17 weeks.

### 5.2.2. Review with the SE Steering Group

In addition to the review of the requirements and design documents with the customer, SE steering group performed an internal panel evaluation to review both the process and the resulting documentation. The panel board consisting of the senior technical experts and managers from different business lines within The Company evaluated that the resulting model for the SoS architecture has a high potential to serve as a role model and can be/should be re-used in comparable enterprises. Panel board assigned the lead SoS architect to another problematic border security project to establish the same knowledge driven approach for that project. This independent evaluation of senior experts validated that the proposed approach was successfully confirmed by the case study.

### 5.2.3. Re-usability Workshop

Based on the recommendations of the panel board, a re-usability workshop was performed within the border security business line. In a three-day workshop, SE teams of other four border security projects (27 systems engineers) were briefed by the SoS architecture team of the case study project. Parallel sessions were performed to evaluate the re-usability of the resulting model and knowledge elements by the other projects. Participants evaluated that 62% of the operational activities, 45% of the system functions and 23% of the subsystem functions could be re-used in at least one of the other border security projects. Review of the model was based on reviewing the knowledge elements and their textual descriptions. All the participants agreed that more time was required to evaluate the model elements within specific operational scenarios.

### 5.2.4. Interviews

In addition to the above mentioned independent reviews of the architecture description and the process, data was collected through interviews with all architecture team, project manager of the case study and three operational users from the customer team. Data collection method was open-ended interviews to discuss evaluations of the participants on different parts of the proposed approach.

By the help of the trainings, customer was always involved in the SE process. They were familiar with the emerging functionalities of the SoS starting from the beginning. Every three weeks they had a chance to review the changes in the SoS architecture based on the feedback they have provided in the previous training session. Due to their continuous interaction with the problem and the solution concept, their understanding of solution and their role as part of this solution has been improved. As a result, architecture team evaluated customer comments as always relevant and valuable.

This approach was observed to be more feasible and effective than inviting customer to ad-hoc meetings and directly asking them about directions. As trainings were organized based on specific operational roles, system architects found the chance to get feedback from the correct customer representative. One lesson learned about the trainings was to plan them as early as possible and inform the customer side about the scope, the date and the users to be involved. Some of the trainings were delayed because the expected users were not available for the planned dates. In an approach like this which aims to improve agility attributes, delaying a training session for several weeks would have significant effects.

One of the customer representatives stated that he was really feeling that his opinions were taken into consideration. Another one mentioned that he had no difficulty in reviewing the SDR package as he was already knowledgeable about the overall concept and the parts that are related to his operational role. Users also highlighted the effect of trainings on the early identification of the possible conflicts among different operational groups within the customer organization. All the participants agreed that trainings increased the agility of the SE process by obtaining the customer feedback early. Early feedback helped the LSI SE team to identify and plan the necessary changes in constituent systems.

Deep dive architecture workshops were evaluated to be the most valuable constituent of the proposed approach. All team members stated that they had a clear understanding of what is expected from them and how their effort contributes to the overall architecture. Lead SoS architect evaluated that focusing on the missing critical knowledge improved the definition and allocation of the analysis tasks. Planning the efforts within time-boxes limited to two weeks helped the team to plan and manage their activities effectively. Since every analysis activity addressing KUs had a defined customer within the team, all the activities were generating value for the others and team members stated that they felt this. Knowledge circles (temporary IPTs) were established based on this value expectation and each member had a clear definition for his/her contribution.

During the case study, it was not possible to resolve all knowledge issues within two weeks after the identification due to limited resources in the architecture team. A "knowledge backlog" was defined by the team and analysis activities were allocated to the team members based on the priority of the missing knowledge. In such a way the team produced 14 SoS level analysis papers within six months. In addition to the time-boxed analysis tasks handled within the architecture team, 8 analysis issues were escalated to the program level risk board. It was evaluated that these issues could not be solved within the architecture team and the team needed management support. Four out of 5 category A discrepancies identified during the SDR were already identified by the team within those 8 analysis issues. Since the required resources were obtained and the necessary analysis activities were already started, the team was able to resolve all category A problems within 5 weeks and successfully passed the SDR.

In total, 14 short term and 8 long term analysis studies were identified and performed within seven months (and an additional one was identified during the SDR by the customer). This was evaluated as a significant improvement in addressing the real problems as the total number of analysis papers produced in other three border security projects was only three. In addition to this, 8 long term analysis issues were also discussed with the other 4 border security projects and it was determined that 6 of these issues were applicable to all other projects. However, no analysis tasks were defined for these issues in the other projects.

As all the decisions were made in a collaborative environment with the contribution of all knowledge sources starting from operational domain to subsystem functions, every team member had a chance to effect any decision. Changes in any decision were effective immediately without losing time in cross team and sometimes cross organization communication paths. The feedback obtained from the customer during the trainings was not only evaluated by a few system engineers but all the architecture team including the experts from the subcontractors were also involved in the decisions on necessary changes.

The team evaluated that collaborative environment provided within the workshops was the main agility enabler that led to the team being able to deliver SoS architecture within six months. Table 11 compares the case study project with the other 4 border security projects in The Company in terms of SE deliverables and required time.

One challenge in performing the deep dive architecture workshops was getting all team members in the same meeting room. Due to global structure of The Company, team members were participating from other cities and even from other countries. It was mandatory to have 70% of the members in the same meeting room. Online conferencing system of The Company helped the team to overcome this difficulty to a certain extent. Online conferencing system provides services that enable meetings, training sessions and seminars to be held online from anywhere in the world.

Table 12: Projects vs. Milestone Times

| | Operational (Stakeholder) Requirements Review | System Requirements Review | System Design Review |
|---|---|---|---|
| Project 1 | T0 + 3 Months | T0 + 7 Months | T0 + 15 Months |
| Project 2 | T0 + 4 Months | T0 + 9 Months | T0 + 17 Months |
| Project 3 | T0 + 3 Months | T0 + 6 Months | T0 + 20 Months |
| Project 4 | T0 + 3 Months | T0 + 7 Months | T0 + 22 Months |
| Case Study Project | T0 + 6 Months | | |

## 5.3. Observations & Lessons Learned

### 5.3.1. Agility Attributes vs. Proposed Approach

As previously stated, in this study, our interpretation of agile SE is based on the agility attributes defined by Turner: learning attitude, focus on customer value, short iterations delivering value, neutrality to change, continuous integration, test-driven, lean attitude and team ownership.

First of all, proposed approach is a knowledge-driven approach. Primary focus is on acquiring valuable but missing knowledge as early as possible so that the decisions, especially the ones that are related to SoS architecture, can be more accurate and stable. Therefore, learning attitude is the main driver behind the proposed approach.

The agility attributes "focus on customer value" and "lean attitude" are closely related to each other. Lean attitude is about removing non-value added activities and delaying decisions as much as possible (Turner, 2007). Lean thinking defines three conditions for value added activities (Oppenheim et al., 2011; Womack & Jones, 2003):

- The external customer is willing to pay for it explicitly or implicitly (that is if the customer understood the details he would support this activity),
- Transforms information or material, or reduce uncertainty,
- Provides specified performance right the first time.

The first bullet is directly related to customer value. In order to develop something valuable for the customer it is necessary to understand the customer's real problem. SoS Knowledge Space and the meta-model are structured with the critical knowledge elements from the LSI point of view. It is clearly visible from the constructed meta-model that operational elements dominate the logical knowledge framework in the proposed approach. Therefore, the proposed approach primarily

focuses on understanding the operational problem of the customer. This is a clear indication that the proposed approach is focusing on customer value.

Since the meta-model is the driver behind all the activities in the deep dive architecture workshops, all the activities are generating value based on the above lean definition. All the activities are targeting missing knowledge. In addition to this, all knowledge elicitation activities and all analysis activities have a consumer and a supplier in the team. The consumer needs to justify his/her knowledge needs and if the team agrees a supplier is assigned as the leader of the knowledge circle. Therefore, everybody in the team knows what they are doing and for what reason. This is obviously a lean approach and also increases the team ownership. Turner defines team ownership as allowing teams to have their own plans and processes. This is exactly in line with the proposed approach. As soon as a supplier is assigned to a task the first activity he performs is the identification of the knowledge resources to establish the knowledge circle. This is done in the deep dive architecture workshop together with most of the stakeholders. As soon as the team is decided, they plan the required activities to capture the known unknowns within 2 weeks.

Proposed approach is more than neutral to changes. In this approach we are actively attacking the changes. New knowledge means changes. Therefore, by focusing on the missing knowledge and trying to acquire it as early as possible, the objective in the proposed approach is to trigger changes as early as possible. As a result, the expectation is not to close the SoS project to changes but to reduce the cost of changes.

Time-boxed planning and continuous trainings can be linked to test-driven, continuous integration and short iterations delivering value.

### 5.3.2. Requirements Last Approach & Model Driven Document Generation

In the case study, all the architecture documents were generated automatically using the features of the modeling software (Sparx Enterprise Architect with UPDM plug-in). Architecture team worked only on the modeling software and did not write even a single word into the design documentation. Everything was handled by the modeling software. Due to contractual obligations, the team needed to generate traditional requirements with shall statements. In the new approach only two weeks were allocated to the requirement definition activity. No requirements were written before the model was frozen one month before the system design review. Within two weeks "shall" based requirements were developed manually based on the model (i.e. drawings and explanatory text). Requirements were defined in the modeling tool and then exported to DOORS for contractual obligations. As soon as the requirements were completed at different levels, all the architecture and requirements documents were generated from the modeling software automatically for peer review. We have experienced that this approach reduced rework significantly and therefore increased the agility of SE activities.

**Success Factor 5:** Convey the message that requirements, or any other design document, do not constitute the ultimate goal. Documents are tools to communicate knowledge. Experience showed us that documenting stable knowledge does not take

time. First stabilize the common and consistent knowledge through models and then generate the documents. This reduces the time lost in updating documentation.

### 5.3.3. Definition of the problem is an important part of solution

In the traditional SE culture, especially in large scale projects, customer is expected to perform all the necessary analysis activities before the project to identify their operational problem. The problem must be captured in the contractual requirements and maybe in the accompanying ConOps document. Systems engineers are expected to develop the solution based on this well-defined and well-scoped problem definition. On the other hand this is never the case and this deadly assumption is one of the main reasons for failure of classical SE approaches.

This assumption is still widely accepted in defense industry projects. Following are some of the findings from the exploratory survey:

- Only 24% of the systems engineers stated that they have a defined ConOps process in their organization.
- None of the organizations has a defined process or a guideline for operational scenario development, prototype development or stakeholder analysis.
- None of the organizations has a business analyst, business process engineer or a similar role in the team.
- In any of the projects, there isn't a task defined in the WBS to capture and document the existing business/operational processes of the customer.

**Success Factor 6:** It is the responsibility of the system developer to understand the operational problem and justify the solution against the problem. Operational problem is never fully and consistently defined in the contract. Project-wide visibility of this fact can be achieved by explicitly defining work packages in the WBS for capturing the operational domain knowledge.

### 5.3.4. Systems Engineers are not Business Analysts

Including tasks in the WBS for eliciting and capturing the operational problem does not solve the problem. Teams and individuals are needed to be allocated to these tasks. Assigning these tasks to SE teams is of course one of the options. However, the case study revealed that it is not easy to perform these tasks effectively with traditional SE teams. First of all, systems engineers do not have the necessary operational domain knowledge. Secondly, systems engineers are engineers. Engineers are not educated for identifying the problems but solving “well-defined” problems. In order to elicit the operational needs of the customers, one needs very good communication and question asking skills. The knowledge resides on the customer side and you will get what you ask for. Asking a question like “what is your problem” will not help you at all.

**Success Factor 7:** Operational domain must be handled by business analysts, concept engineers or operational advisors. If these tasks will be performed by the SE

team, develop business analysis skills in your SE competency framework. Define specific roles for analyzing the problem space.

### 5.3.5. Model-based Systems Engineering is "Systems Engineering"

Models and documents are two different ways of representing knowledge. In documents, knowledge is captured in terms of requirements, design explanations, etc. These documents become the sources of necessary knowledge when a decision is to be made. Graphical models have many advantages over text based representations of knowledge. This is one of the benefits promised by MBSE. However, it is critical to understand that MBSE is not the art of drawing diagrams. MBSE is model based + systems engineering. First of all it is SE. MBSE is a shift from document-centric to model-centric SE. The SE process remains essentially the same. In other words, requirements that constitute a good application of SE remain unchanged.

The case study pointed out that in most of the cases (i.e. border security projects in The Company) systems engineers do not know what to expect from MBSE. That is one of the reasons that they define their MBSE attempts as unsuccessful. In reality, the part they were unsuccessful was not the MBSE but the SE. Modeling or MBSE can help in communicating the existing knowledge in an easier and faster way. However, modeling cannot acquire the missing knowledge. MBSE cannot perform the tradeoff analysis for you. If some knowledge is missing, it can be relatively easier to identify missing knowledge in MBSE. However, if prototyping is required to elicit some knowledge or some operational analysis is required to identify the needs of the users MBSE is not the solution for this. As a result, whether a document-based approach or a model-based one is adopted, the problem is the missing knowledge to justify the SE decisions. Decisions (i.e. the architecture and design of the system) can be captured in models or in documents. MBSE cannot turn bad decisions into good ones or architecture frameworks cannot make a bad architecture a good one.

**Success Factor 8:** Apply SE principles to your own problems. First elicit your problems related to SE (or SoSE) processes and understand the benefits promised by MBSE. Clearly identify and communicate your expectations from MBSE in dealing with your problems and measure your projects against those expectations.

### 5.3.6. Definition of the SoS Knowledge Space and Meta-Model

Engineers from different backgrounds and organizations quite often tend to misunderstand each other, because they associate different concepts with deceptively simple words. Especially, if one is trying to implement an agile and knowledge driven approach, one must minimize the communication barriers as much as possible. In the course of the present study it was shown that joint elaboration of a logical knowledge framework in terms of a meta-model can provide significant support to SE teams at this point. It helps engineers to establish a common understanding about the key concepts and knowledge items in their shared problem. In a knowledge-driven SE approach, such a meta-model can guide the systems engineers to develop complete and consistent knowledge on the SoS architecture.

In Figures 19, 20 and 21, we present some examples that have been modeled, in the course of the workshops, with the collaboration of domain experts and modeling experts using UML, all based on the knowledge meta-model depicted in Figure 14.

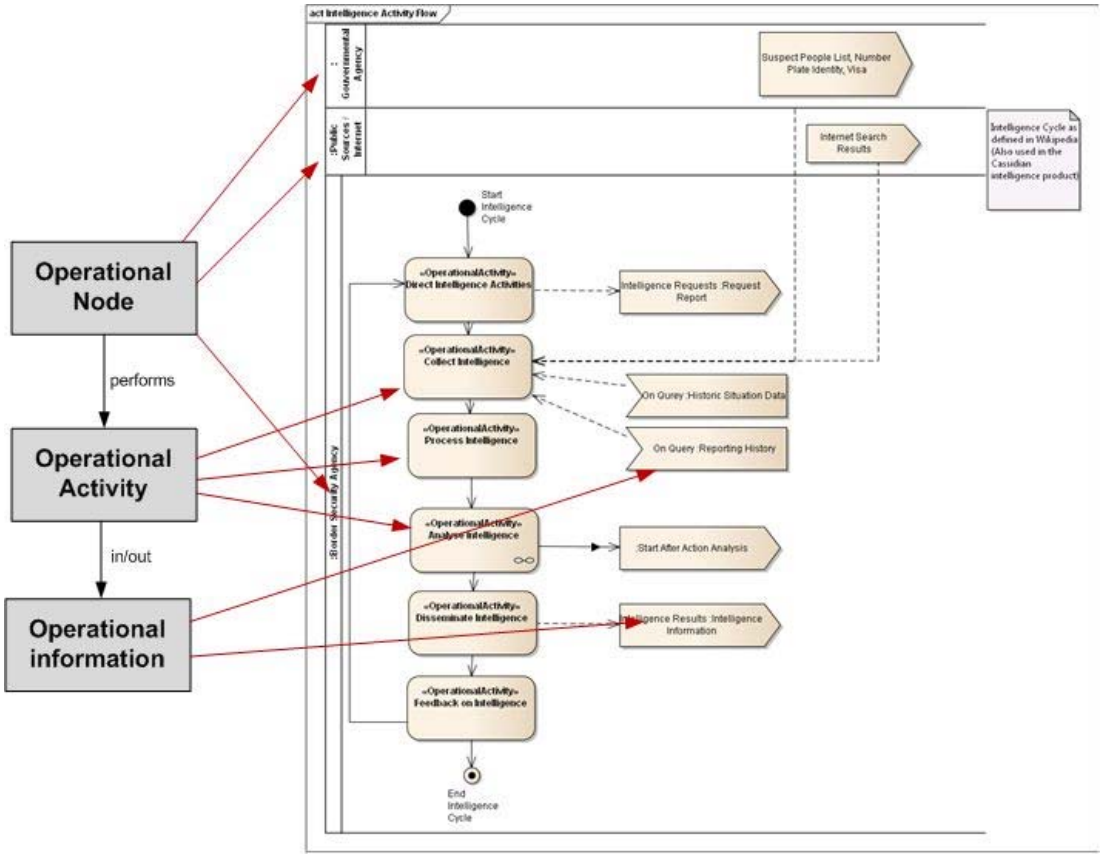


Figure 19: Model instances of knowledge elements Operational Node, Operational Activity and Operational Information

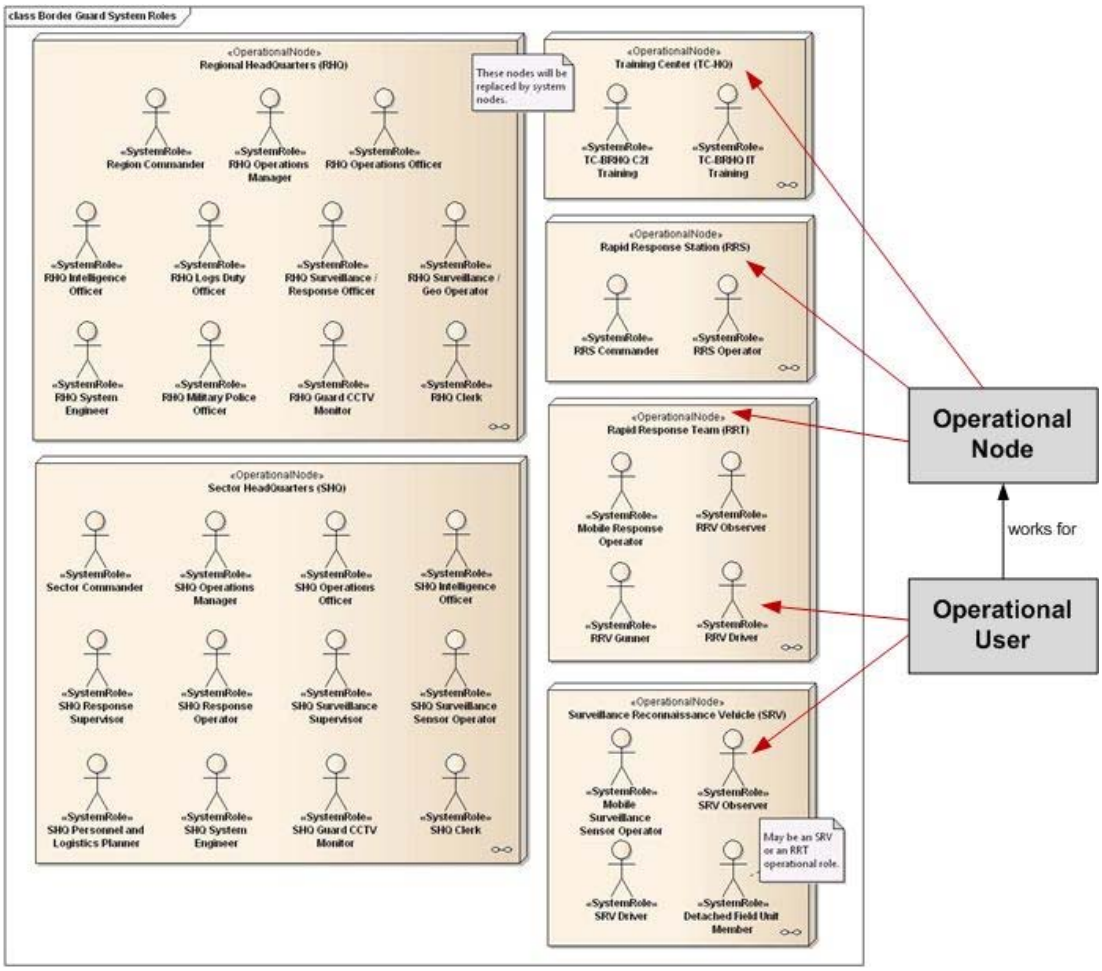


Figure 20: Model instances of knowledge elements Operational Node and Operational User

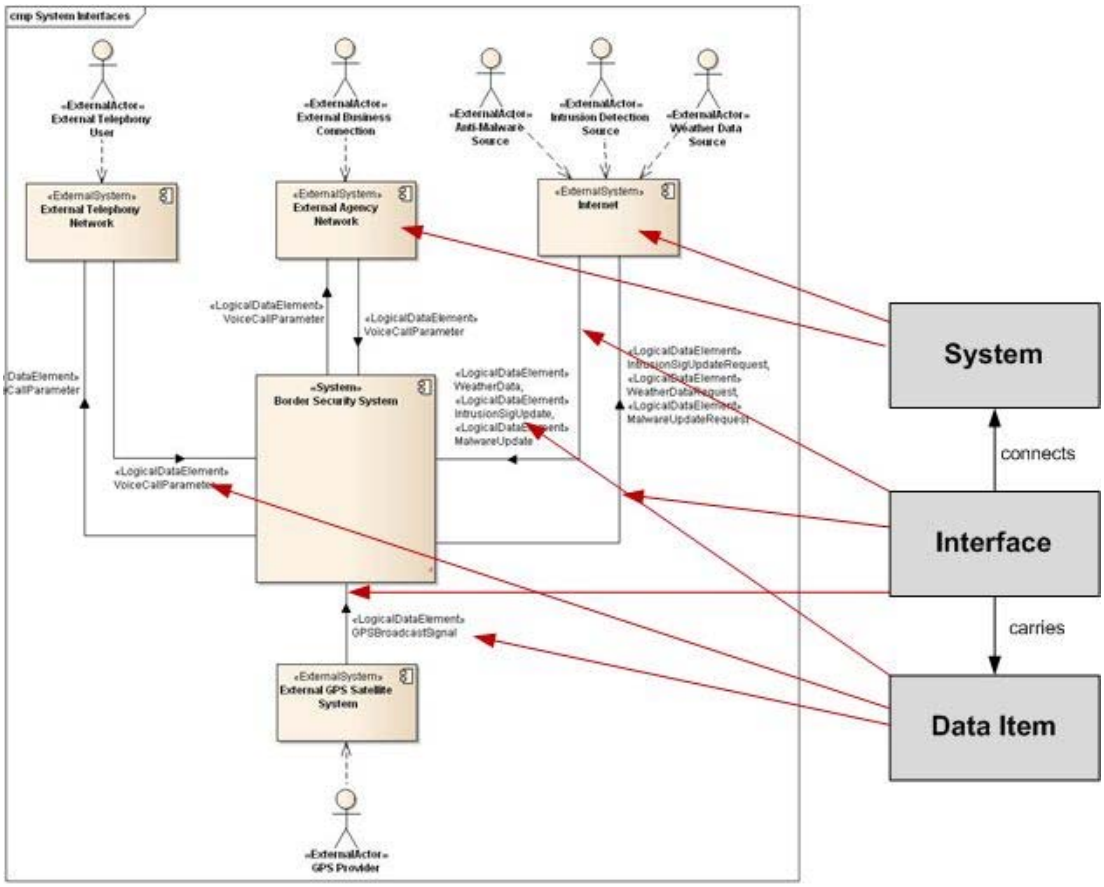


Figure 21: Model instances of knowledge elements System, Interface and Data Item

### 5.3.7. Use Models and Modeling Experts to Facilitate Collaboration

During the case study we experienced that models facilitate the collaborative environment provided by the deep dive architecture workshops by making it easier to elicit and communicate knowledge. On the other hand, we also observed that, understanding the models that are developed by others is not easy when one reads them alone. The challenge here is to find a modeling language that is easily understandable and usable by all participants. Even the systems engineers who were familiar with the modeling languages (e.g. UML and SySML) and architecture frameworks (e.g. NAF, DODAF and MODAF), experience difficulties in reading and understanding the models developed by others.

In case study, the deep-dive architecture workshops, knowledge-driven efforts, time-boxed planning, knowledge circles, requirements last approach and modeling experts are implemented as an integrated approach. However, this was not the first time The Company tried to adopt MBSE and model-driven document generation approach. In the other border security projects, we observed that, understanding the models that are developed by others is not easy when one reads them alone. The challenge here is to find a modeling language that is easily understandable and usable

by all participants. Even the systems engineers who were familiar with the modeling languages (e.g. UML and SySML) and architecture frameworks (e.g. NAF, DODAF and MODAF) were having difficulty in reading and understanding the models developed by others.

On the other hand, this was the first time that The Company used collaborative environment and modeling experts. Therefore, it is concluded that, the real improvement in the agility of the knowledge-driven activities comes with the deep dive architecture workshops that are facilitated by modeling experts. Models, modeling languages and modeling experts can facilitate efforts when they are used in such environments. A sequence diagram may improve the speed and quality of the discussions in a workshop in comparison to pages of textual description. On the other hand, using models in an architecture workshop is like taking notes in a conference. If you have already participated to the conference and listened to the presentations, the notes will obviously help you to go through the proceedings in a quick manner. On the other hand, notes taken by somebody else will not help you much if you haven't participated in the conference (e.g. similar to not attending to architecture workshops).

The case study revealed that using modeling experts also increases the acceptance of the MBSE approach by systems engineers. Instead of asking the engineers to start modeling from the first day of the new approach, we asked modeling experts to perform the modeling activities using the modeling software. Systems engineers served as the knowledge sources and analysis experts. When the systems engineers do not feel the pressure to learn new modeling languages and new modeling tools they do not react negatively to the MBSE approach. In time, they develop skills to use the language and the tool themselves.

### 5.4. Threats to Validity

Threats to the validity of qualitative research are generally accepted as: construct validity, internal validity, external validity and reliability (Runeson & Höst, 2008). Internal validity is relevant for explanatory type of research (Tellis, 1997) and therefore is not applicable to our evaluation of threats. In this section, we assess the validity of the reported research in terms of construct validity, external validity and reliability.

Construct validity reflects how much the findings represent what the researcher is really investigating. Researcher's subjectivity is always a threat. The results may be misleading if the interview questions are misunderstood or misinterpreted by the participants. We tried to minimize the possibility of misunderstanding in the concepts used in the interviews by explaining every key concept using the definitions and examples from the well-known IEEE and INCOSE SE literature. As suggested by (Yin, 2003), all the findings and interpretations, including the final papers, were reviewed by the participants and updated if necessary according to their feedback.

External validity determines whether the findings can be generalized beyond the setting of the group studied. The proposed approach is formulated based primarily on an analysis of the SoS problems of the defense industry. Similarly the approach is

evaluated exclusively according to the results observed in a military environment. In order to develop an understanding on the applicability of the proposed approach in other business domains such as health, finance, commercial, etc. further empirical research is needed in these application domains. On the other hand, many of the core elements that are part of the proposed approach are based on general principles such as focusing on missing critical information, requirements last approach based on MBSE or increasing effectiveness of the interaction with the customer via continuous customer trainings. The author believes that these elements can be applied in any business context to increase agility.

Reliability is concerned with repeating the research with different researchers. Best mitigation strategy is planning the research before conducting it. For the case study, multiple sources of evidence were used to reach reliable results. Regarding both external validity and reliability, interview results were only one of the evaluation methods. Evaluations from the SDR with the customer, review of the SE steering group and re-usability workshops were completely independent from the researchers and have been pointed out where relevant in this thesis. We have compared the results with the sister projects in The Company to increase reliability (Verner, Sampson, Tosic, Bakar, & Kitchenham, 2009).

During the planning phase of the case study, the case study context (The Company, case study project and its problems) and the formulated approach were presented to seven INCOSE CSEP certified systems engineers in a focus group and the agility expectations and defined success factors were validated (Yin, 2003). Interviews were performed with staff bearing different roles from inside the LSI company and outside the LSI company (e.g. customer representatives and systems engineers of the subcontractor companies).

# CHAPTER 6

# CONCLUSION

## 6.1. Summary

In this dissertation, empirical research aiming to improve agility in SoS architecting by a model-based, knowledge-driven and collaborative approach is reported. The research was performed mainly in two stages. In the first stage, agility related problems in traditional SE were identified via an exploratory survey. In the second stage, a novel approach was formulated to address the agility problems pointed out by the survey. The approach is based on eliciting existing knowledge, visualizing it by immediate modeling, verifying consistency of the knowledge by continuous review, and identifying and planning for missing knowledge. Agility is mainly targeted with 1) performing these activities concurrently in modeling workshops, where operational, functional and physical aspects of the solution are developed simultaneously; and 2) with the collaboration of cross domain/organization knowledge sources. Continuous customer training is another tool adopted for enhancing collaboration effectiveness with the customer. Formulated steps were implemented and assessed in a confirmatory case study in which it was observed that the proposed approach has the potential to increase agility of the SE process in SoS projects.

## 6.2. Contributions

The first phase of this study where an exploratory survey was performed has several contributions: (a) for the first time, agility related problems of traditional SE in SoS projects are explored and validated in an empirical research, (b) findings surprisingly revealed that main agility problem was not the traditional SE process but in contrary weak application of the principles of the classical SE, (c) our interviews with the systems engineers to explore the reasons and nature of changes in SoS projects resulted in a novel model which relates changes to the discovery of new knowledge, (d) analysis of the model contributes to the SE literature with the idea of improving agility in SE by identifying the valuable but missing knowledge early in the project and by tailoring the SE processes to focus on the required knowledge.

The main contribution of this research is the proposal of the knowledge-driven, collaborative and model-based architecture development approach that can be applied in SoS projects by the SoS integrators. The proposed approach is

implemented and assessed in a case study where the outcomes confirm that the novel approach has the potential to increase the agility attributes of the SE process at the SoS level. The proposed approach clearly demonstrates success in many of the agility attributes defined for SE.

INCOSE SoS working group identified major issues in terms of pain points in the area of SoS operation, management and SE to support planning for future activities of the INCOSE SoS working group. The survey identified seven "pain points" raising a set of questions for SE of SoS (Dahmann, 2013). This study contributes at least 3 of the pain points.

***SoS Principles and Thinking Skills:*** *The survey revealed lack of formalized processes and examples of SoS success. This area is one where progress in identifying and articulating SoS principles, SoS Thinking and examples, could have benefit to the discipline.* ***Question:*** *What are the key SoS thinking principles?* First of all, proposed approach is a knowledge-driven approach. The driving idea behind the proposed approach is the fact that in the SoS projects, the level of detail of the information captured in terms of architectural knowledge is much less than is relevant to the detailed development of each individual system in the SoS. As a result, SE activities in SoS environment are more of a **knowledge management** kind than top-down design activities. Therefore, this study contributes to this pain point in two different ways. First, this study proposed knowledge-focused SoS thinking and secondly confirmed this idea in a case study as an example of SoS success.

***SoS Authority:*** *In defense projects, authority conflicts often dominate discussion of SoS. The focus in this area is on creation of the incentives and development environment which allow the systems to proceed to meet their own objectives while working cooperatively to support broader objectives at the SoS level.* ***Question:*** *What are effective collaboration patterns in SoS?* This study contributes to this pain point by successfully applying deep dive architecture workshops and continuous customer trainings as two different collaborative environments. Both of these environments were led by the LSI teams with the collaboration of the system providers and customer representatives.

***Testing, Validation and Learning:*** *Findings highlight the fact that most defense SoS cannot be tested thoroughly prior to fielding leading to approaches like incremental validation. With multiple constituent systems on asynchronous development cycles, finding ways to conduct traditional testing across the SoS can be difficult. It is necessary to reflect a perspective that looks at significant learning going on over the life of a SoS.* ***Question:*** *How can SE approach SoS validation, testing, and continuous learning in SoS?* Proposed approach confirmed that it is possible to put learning at the heart of a SE approach by adopting a knowledge-driven approach. Continuous trainings make customer a main part of this learning process. With the case study, this research confirmed that it is possible to continuously validate SoS concepts and architecture using the continuous trainings.

Both SoSE and MBSE are considered as two major areas in the future of contemporary SE practices. For an organization, adoption of new approaches such as wave model or MBSE requires careful consideration of the introduction mechanisms. From the practitioner's point of view, lessons learned and the success factors

presented in this study can help LSI companies to increase the speed of shift from traditional SE approaches to architecture-driven, knowledge-focused and model-based SoSE practices.

### 6.3. Limitations

In order to validate new approaches via qualitative research, it is suggested to conduct multiple case studies. This remains to be a challenge in our case, as it is quite cumbersome, if at all possible, to reach cases of large scale SoS projects and investigate them at the necessary level of detail.

### 6.4. Future Work

While the proposed approach creates an initial foundation for knowledge-focused and collaborative architecting that exhibits improved agility in SoS environments, there are many areas for future work. Understanding the existing systems is the key element in the wave model. This element corresponds to capturing the knowledge of existing systems in the proposed approach. The faster this knowledge is made available for the SoS level decisions, the more agile the overall SoS architecting process is.

In this study it is confirmed that model-based approaches improves communication and increases agility of the processes in comparison to document-driven approaches. On the other hand, developing system models in every project from the scratch is neither agile nor lean. Therefore, re-use of architecture artifacts, especially the model-based ones, is one of the critical activities with the potential to improve agility further. However, variety of the modeling approaches, modeling tools and modeling languages increases the effort to re-use existing architecture models. Research is needed to develop methods, best practices and standards to facilitate model integration and re-use so that the effect of the collaborative modeling environments can be improved. Specific attention should be paid to variability modeling during the development of the system architectures.

## REFERENCES


ANSI/EIA. (1999). ANSI/EIA-632 Standard, Processes for Engineering a System.

Bartolomei, J. E., Hastings, D. E., Neufville, R. De, & Rhodes, D. H. (2011). Engineering Systems Multiple-Domain Matrix: An Organizing Framework for Modeling Large-Scale Complex Systems, 41–61. doi:10.1002/sys

Batra, D., Xia, W., VanderMeer, D., & Dutta, K. (2010). Balancing Agile and Structured Development Approaches to Successfully Manage Large Distributed Software Projects: A Case Study from the Cruise Line Industry. *Communications of the Association for Information Systems*, *27*(1).

Beck, K. (2001). *Extreme programming explained: embrace change*. Addison-Wesley.

Boehm, B. (2002). Get ready for agile methods, with care. *Computer*, *35*(1), 64–69. doi:10.1109/2.976920

Boehm, B. (2006). Some future trends and implications for systems and software engineering processes. *Systems Engineering*, *9*(1), 1–19. doi:10.1002/sys.20044

Boehm, B., & Lane, J. A. (2006). 21st Century Processes for Acquiring 21st Century Software-Intensive Systems of Systems. *Crosstalk: Journal of Defence Software Engineering*.

Boehm, B., & Turner, R. (2003). *Balancing Agility and Discipline: A Guide for the perplexed*. Addison-Wesley Professional.

Booz Allen Hamilton. (2010). *Systems-2020 Study Final Report*. Retrieved from www.acq.osd.mil/se/docs/BAH-Systems-2020-Report-Final.pdf

Bosch, J., & Bosch-Sijtsema, P. (2010). From integration to composition: On the impact of software product lines, global development and ecosystems. (H.-J. Bullinger & A.-W. Scheer, Eds.)*Journal of Systems and Software*, *83*(1), 67–76. doi:10.1016/j.jss.2009.06.051

Boudreau, M.-C., Gefen, D., & Straub, D. M. (2001). Validation in Information Systems Research: A State-of-the-Art Assessment. *Management Information Systems Quarterly*, *25*(1), 1–16. doi:10.2307/3250956

Bourque, P., & Dupuis, R. (2004). Guide to the Software Engineering Body of Knowledge 2004 Version. *SWEBOK 2004 Guide to the Software Engineering Body of Knowledge*, *1*(1). doi:10.1109/SESS.1999.767664

Brooks, F. P. (1987). No Silver Bullet. *IEEE Computer*, *20*(4), 10–19. doi:10.1016/S0197-2510(10)70174-1

Chung, L., Cesar, J., & Leite, P. (2009). On Non-Functional Requirements in Software. In *Conceptual Modeling: Foundations and Applications* (pp. 363–379). Springer Berlin / Heidelberg.

Clements, P., & Northrop, L. (2001). *Software Product Lines: Practices and Patterns* (p. 608). Addison-Wesley Professional.

Cockburn, A. (2006). *Agile Software Development: The Cooperative Game* (p. 504). Addison Wesley.

Consortium, D. (2008). *DSDM Atern The Handbook* (p. 201). DSDM Consortium.

Dahmann, J. (2013). Systems of Systems: A Systems Engineering Perspective. In *Complex System Design & Management Conference*. Paris: Springer Berlin Heidelberg.

Dahmann, J., Rebovich, G., Lane, J., & Baldwin, K. (2011). An Implementers' View of Systems Engineering for Systems of Systems. Retrieved from www.acq.osd.mil/se/docs/ImplementerViewSE-SoS-Final.pdf,

Dominguez, J. (2009). The Curious Case of the CHAOS Report 2009. Retrieved December 19, 2013, from http://www.projectsmart.co.uk/the-curious-case-of-the-chaos-report-2009.html

Eisenhardt, K. M. (1989). Building Theories from Case Study Research. (A. M. Huberman & M. B. Miles, Eds.)*Academy of Management Review*, *14*(4), 532–550. doi:10.2307/258557

Flyvbjerg, B. (2006). Five Misunderstandings About Case-Study Research. *Qualitative Inquiry*. doi:10.1177/1077800405284363

Friedenthal, S. (2011). *A Practical Guide to SysML: The Systems Modeling Language (The MK/OMG Press)* (p. 666). Morgan Kaufmann.

Gilb, T. (2005). Competitive Engineering. In *Competitive Engineering*. Elsevier.

Harding, A., & Dahmann, J. (2012). INCOSE - System of Systems Working Groups. Retrieved December 20, 2013, from http://www.incose.org/practice/techactivities/wg/details.aspx?id=sos

Haskins, C., Forsberg, K., Krueger, M., Walden, D., & Hamelin, R. D. (2010). *Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities*. (C. Haskins, Ed.) (3.2 ed., Vol. 3.2). INCOSE.

Highsmith, J. (2000). *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems* (p. 358). Dorset House Publishing.

Highsmith, J., & Cockburn, A. Agile software development: the business of innovation. , 34 Computer 120–127 (2001). IEEE. doi:10.1109/2.947100

INCOSE UK Systems Engineering Competencies Working Group. (2010). *Systems Engineering Competencies Framework*.

Jørgensen, M., & Moløkken-Østvold, K. (2006). How large are software cost overruns? A review of the 1994 CHAOS report. *Information and Software Technology*, *48*(4), 297–301. doi:10.1016/j.infsof.2005.07.002

Kaplan, B., & Duchon, D. (1988). Combining Qualitative and Quantitative Methods in Information Systems Research: A Case Study. *MIS Quarterly*, *12*(4), 571–586. doi:10.2307/249133

Kennedy, M. R., & Umphress, D. A. (2011). An Agile Systems Engineering Process The Missing Link□? *CrossTalk The Journal of Defense Software Engineering*, *4*(3), 16–20.

Koskela, L., & Howell, G. (2002). The underlying theory of project management is obsolete. *IEEE Engineering Management Review*, *36*(2), 22–34. Retrieved from http://usir.salford.ac.uk/9400/

Lane, J. A., & Boehm, B. (2008). System of systems lead system integrators: Where Do they spend their time and what makes them more or less efficient? *Systems Engineering*, *11*(1), 81–91. doi:10.1002/sys.20085

Lane, J. A., & Dahmann, J. (2008). Process evolution to support system of systems engineering. *Proceedings of the 2nd international workshop on Ultra-large-scale software-intensive systems - ULSSIS '08*, 11–14. doi:10.1145/1370700.1370704

Laudon, K. C., & Laudon, J. P. (2011). *Management Information Systems: Managing the Digital Firm*. Prentice Hall.

Maier, M. W. (1998). Architecting principles for systems-of-systems. *Systems Engineering*, *1*(4), 267–284. doi:10.1002/(SICI)1520-6858(1998)1:4<267::AID-SYS3>3.0.CO;2-D

Maier, M. W., Emery, D., & Hilliard, R. (2004). ANSI/IEEE 1471 and systems engineering. *Systems Engineering*, *7*(3), 257–270. doi:10.1002/sys.20008

NASA. (2007). NASA Systems Engineering Handbook. (C. Haskins, Ed.)*Systems Engineering*, *6105*(June), 360. doi:10.1016/0016-0032(66)90450-9

ODUSD(A&T)SSE. (2008). *Systems Engineering Guide for Systems of Systems* (1.0 ed.). Washington, DC: Office of the Deputy Under Secretary of Defense for Acquisition and Technology, Systems and Software Engineering.

*One Hundred Eleventh Congress of the United States of America*. (2010). Retrieved from http://www.gpo.gov/fdsys/pkg/BILLS-111hr6523enr/pdf/BILLS-111hr6523enr.pdf

Oppenheim, B. W., Murman, E. M., & Secor, D. A. (2011). Lean enablers for systems engineering. *Systems Engineering*, *43*(1), n/a–n/a. doi:10.1002/sys.20161

Project Management Institute. (2011). PMI Agile Certified Practitioner. Retrieved November 07, 2011, from http://www.pmi.org/en/Certification/New-PMI-Agile-Certification.aspx

Qumer, A., & Henderson-Sellers, B. (2006). Crystallization of agility back to basics. In *ICSOFT* (pp. 121–126).

Qumer, A., & Henderson-Sellers, B. (2008). An evaluation of the degree of agility in six agile methods and its applicability for method engineering. *Information and Software Technology*, *50*(4), 280–295. doi:10.1016/j.infsof.2007.02.002

Robson, C. (2011). *Real World Research 3e* (p. 608). John Wiley & Sons.

Runeson, P., & Höst, M. (2008). Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, *14*(2), 131–164. doi:10.1007/s10664-008-9102-8

Schwaber, K., & Beedle, M. (2001). *Agile Software Development with SCRUM* (p. 158). Prentice Hall.

SEI. (2011). Software Product Lines. Retrieved December 25, 2013, from http://www.sei.cmu.edu/productlines/

Shatil, A., Hazzan, O., & Dubinsky, Y. (2010). Agility in a Large-Scale System Engineering Project: A Case-Study of an Advanced Communication System Project. *2010 IEEE International Conference on Software Science, Technology & Engineering*, 47–54. doi:10.1109/SwSTE.2010.18

Stelzmann, E. (2012). Contextualizing agile systems engineering. *IEEE Aerospace and Electronic Systems Magazine*, *27*(5), 17–22. doi:10.1109/MAES.2012.6226690

Stelzmann, E., Kreiner, C., Spork, G., Messnarz, R., & Koenig, F. (2010). Agility Meets Systems Engineering : A Catalogue of Success Factors from Industry Practice, 245–256.

Tellis, W. (1997). Introduction to Case Study. *The Qualitative Report*, *3*(2), 1–11. doi:10.1016/j.jvolgeores.2009.02.004

The Standish Group International, I. (2009). *CHAOS Summary 2009*. *Chaos* (pp. 1–4). The Standish Group. Retrieved from http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:CHAOS+Summary+2009#0

Tian, K., & Cooper, K. (2006). Agile and software product line methods: are they so different. … *with the 10th International Software Product Line* …. Retrieved from http://www.lsi.upc.edu/events/aple/TianCooperpresent.pdf

Turner, R. (2007). Toward Agile Systems Engineering Processes. *CrossTalk The Journal of Defense Software Engineering*, (April), 11–15. Retrieved from http://www.stsc.hill.af.mil/CrossTalk/2007/04/0704Turner.html

Verner, J. M., Sampson, J., Tosic, V., Bakar, N. A. A., & Kitchenham, B. A. (2009). Guidelines for industrially-based multiple case studies in software engineering. *2009 Third International Conference on Research Challenges in Information Science*. doi:10.1109/RCIS.2009.5089295

Wade J., Madni A., Neill C., Cloutier R., Turner R., Korfiatis P., Carrigy A., Boehm B., T. S. (2010). Development of 3-Year Roadmap to Transform the Discipline of Systems Engineering Final Technical Report – SERC-2009-TR-006.

Wasson, C. S. (2005). *System Analysis, Design, and Development: Concepts, Principles, and Practices*. *Development* (Vol. 22, p. 832). Wiley-Interscience.

Womack, J. P., & Jones, D. T. (2003). *Lean thinking: banish waste and create wealth in your corporation*. *Action Learning Research and Practice* (Vol. 4, p. 384). Free Press. doi:10.1080/14767330701233988

Yin, R. K. (2003). *Case study research - design and methods*. *Applied Social Research Methods Series* (p. 200). SAGE Publications.

## APPENDIX

### Exploratory Survey Interview Questions

Optional:

| | |
|---|---|
| Name: | |
| Company: | |
| Title/department: | |
| email/phone | |

Your project experience:

| | | |
|---|---|---|
| Sector | ☐ Military | ☐ Commercial |
| Team size (overall) | ☐>30 engineers | ☐<30 engineers |
| Project Duration | ☐ Shorter than 2 years | ☐ Longer than 2 years |
| Project Environment | ☐ Distributed | ☐ Collocated team |
| Subcontractor Usage | ☐ Yes | ☐ No |
| Separate Systems Engineering Team | ☐ Yes | ☐ No |

Questions:

1. Do you find a chance to work together with the customers/users/maintainers during the project?

   ☐ YES ☐ NO

2. Which one best describes your situation.

   I can discuss with the customer/user/maintainer:

   ☐ whenever I want,

   ☐ whenever a significant problem occurs

   ☐ only when they want

   ☐ not at all

   other:

3. I need the customer collaboration mostly during the (select only one):

   ☐ requirements definition phase

   ☐ design phase

   ☐ implementation phase

   ☐ maintenance/operational support

   other:

4. Which one is valid for you (select only one):

   ☐ I can generally find the opportunity to choose the specific stakeholder type from the customer side (e.g. the user, maintainer, engineer, sales person, etc.). I can communicate with any user type with specific training and domain knowledge. This is not a problem.

   ☐ I generally accept the customer provided representative. I have no chance to select the person with specific domain knowledge.

5. Which one is valid for you:

   ☐ I am a systems engineer but I am working for the specific phases of the system lifecycle. For example, I am a systems engineer but in fact I am a requirements engineer with specific requirements engineering skills, training and experience. Therefore I am mostly working on the requirements definition and analysis phase. I am not responsible for systems architecture or system design.

   ☐ I am a systems engineer and working almost in all phases of the system lifecycle.

6. Does your project team include any specific position (not a role, well defined position with a title) as:

   ☐ Specialty engineer

   ☐ Human engineer

   ☐ Security engineer

   ☐ Safety engineer

   ☐ Business analyst

   ☐ Requirements engineer

   ☐ Concept engineer

   ☐ Logistics analyst

7. Which one(s) are relatively difficult or easy tasks for you (D for Difficult E for Easy)

| | | |
|---|---|---|
| ☐ D | ☐ E | Writing functional requirements |
| ☐ D | ☐ E | Defining performance requirements |
| ☐ D | ☐ E | Defining system security requirements |
| ☐ D | ☐ E | Defining system safety requirements |
| ☐ D | ☐ E | Defining maintainability, reliability, availability requirements |

8. I have performed the following type of analysis (I planned the analysis and captured the results in a formal project document or in an analysis paper):
   - ☐ Cost effectiveness analysis
   - ☐ Electromagnetic compatibility analysis
   - ☐ Environmental impact analysis
   - ☐ Interoperability analysis
   - ☐ Lifecycle cost analysis
   - ☐ Manufacturing and producibility analysis
   - ☐ Mass properties engineering analysis
   - ☐ Safety and health hazard analysis
   - ☐ Sustainment engineering analysis
   - ☐ Training needs analysis
   - ☐ Usability analysis/human systems integration analysis
   - ☐ Value engineering analysis

9. Which one is valid for you (select only one):

   ☐ There is a specific process and defined methods in my company/organization/project to derive system level requirements (not HW or SW level) from the customer/user/stakeholder requirements.

   ☐ No defined process and/or methods for system requirements definition. We take the contract technical requirements and write the system level requirements according to our experience.

   other:

| |
|---|
| |

10. What is the common way of moving forward when you notice that some information is needed from the customer?

    ☐ Make assumptions,

    ☐ Arrange a meeting with the customer,

    ☐ Consult an experienced engineer in an another project?

    ☐ Write an email and wait for the written information from the customer,

    ☐ Follow the defined communication process defined in the communication plan to obtain the required information,

    other?

| |
|---|
| |

11. Is there a specific process in your organization to develop a "concept of operations" (CONOPS) document? Which project team is responsible for the CONOPS definition?

    ☐ NO ☐ YES, Which team:

12. What are the primary reasons for the **requirements changes** in your projects? Please do not do any brainstorming! Just provide your experience. You can do multiple selections.

☐ Changes in the technology

☐ Changes due to increased understanding of the problem during the development.

☐ User's needs changed/evolved as a consequence of changes in business policies and procedures

☐ Changes in the market conditions

☐ Changes in the regulations and laws.

☐ Late understanding of the capabilities/constraints of the COTS products,

☐ Late understanding of the capabilities/constraints of the subcontractors,

☐ Late definition of the new user types and their responsibilities

☐ Late definition of the maintenance needs,

☐ Late clarifications/definition of the non-functional requirements,

☐ The problem our system was intended to solve has changed as a consequence of other changes.

other:

13. In which part of the project life cycle do the requirements generally change?

☐ requirements definition

☐ design

☐ implementation

☐ verification

☐ maintenance

14. Does your organization use a **defined process** for change analysis? Do you analyze the effects of a possible change (change in requirements, change in the stakeholder user types, change in the architecture, change in the operational flow, etc.) in a methodological way?

☐ YES ☐ NO

15. Have you ever been into a situation that the problem was very well defined and customer has provided all the necessary information timely but you were having great difficulty in producing a technical solution? What was the reason(s)?

☐ YES ☐ NO

The reasons:

| |
|---|
| |

16. Do you use a defined process for utilizing scenarios to elicit/understand customer's operational needs and the desired properties of the system?

☐ YES ☐ NO

17. Do you use a defined process for utilizing prototypes to elicit/understand customer's operational needs and the desired properties of the system?

☐ YES ☐ NO

18. Does the customer's documentation package that is provided to you at the beginning of the project provide enough information about the organizational structure, different roles/user types and their expected interaction with the system?

☐ YES ☐ NO

19. Do you organize your requirements based on different stakeholder roles? Is there a traceability of each system requirement to the source stakeholder and his/her goal in the organization?

☐ YES ☐ NO

20. Which one is valid for you:

☐ All the customer and system level requirements are the same for us. They are value neutral. We do not follow a prioritization scheme for the requirements.

☐ Every requirement is prioritized. The primary criteria that is used for prioritization is defined by the customer.

other:

21. If you use a prioritization scheme for your requirements, which team in your organization is responsible for this prioritization:

22. In my organization, validation activities are based on the following documents:

23. In my organization, verification activities are based on the following documents:

24. In my organization, verification and validation activities are planned:

☐ Very early in the project,

☐ During the design phase

☐ Very late in the project, as we start implementing the system

25. Do you perform requirements validation? What method is used?

☐ YES ☐ NO

The method used is:

| |
|---|
| |

26. Which one is valid for you:

☐ In most of my projects, there is **no** concept definition phase, business process re-engineering phase or an operational analysis phase.

☐ In most of my projects, there is a concept definition phase, business process re-engineering phase or an operational analysis phase but the resources and the time allocated for that phase is not enough.

☐ In most of my projects, there is a concept definition phase, business process re-engineering phase or an operational analysis phase. This phase is the most critical phase to understand how the customer is operating in the mission environment and what his needs are from our system. We have great support (in terms of time and resources) from both the customer and the project management for this phase.

27. Which one is valid for you:

Whenever I need some information:

☐ I prefer personal communication and I ask my question to the relevant person,

☐ I know what information is in which project document. Therefore, I look at the relevant project documents.

☐ I know what information is in which project document. However, most of the time other people do not prepare the documents properly, so that I can't find the information in the documents and I need to ask my question to the document owner.

28. Which one is valid for your organization/projects:

☐ Customer/user trainings are planned at the end of the project near the system delivery.

☐ Customer/user trainings are planned early in the project but the content is prepared at the end of the project. Trainings are performed after the system is accepted.

☐ We are continuously training the customer/users/maintainers on the concept and on the system usage via system prototypes and incremental system delivery.

other:

29. What are the three most significant problems you are dealing with, in large scale software intensive projects (you can select from the list or you can use your own words):

- ☐ Communication problems both with the customer and the subcontractors
- ☐ Changing requirements
- ☐ COTS limitations
- ☐ Changing technology
- ☐ Lack of leadership
- ☐ Lack of systems engineering processes
- ☐ Lack of quality control
- ☐ Customer does not know what s/he needs
- ☐ Can't get answers from the customer
- ☐ Not enough time and budget

Your own problems:

# VITA

PERSONAL INFORMATION


Surname, Name : Aşan, Emrah
Nationality : Turkish,
Date and Place of Birth : 14 September 1977, Ankara
Phone : +49 170 5736410
Email : emrah.asan@gmail.com


EDUCATION

| **Degree** | **Institution** | **Year** |
|---|---|---|
| MS | METU, Electrical & Electronics Engineering | 2008 |
| BS | Bilkent, Electrical & Electronics Engineering | 2003 |

WORK EXPERIENCE

| Year | Organization | Position |
|---|---|---|
| 2013-Present | Airbus Defense and Space | Project Manager |
| 2011-2013 | EADS | Lead System Architect |
| 2008-2011 | HAVELSAN-BOEING | Lead Systems Engineer |
| 2005-2008 | HAVELSAN | Systems Engineer |
| 2004-2005 | Creative Lab. | Research Engineer |
| 2003-2004 | Agilent Technologies | Application Engineer |
| 1999-2001 | Software Arts | Technical Manager |

FOREIGN LANGUAGES

English (advanced), German (Beginner)

PUBLICATIONS

Asan E. & Bilgen S. (2013). *Improving Agility by Knowledge- Driven & Collaborative System Engineering: A Case Study,*submitted to INCOSEJournal of Systems Engineering.

Asan E., Allbrecht O. & Bilgen S. (2013). *Handling Complexity in System of Systems Projects – Lessons Learned from MBSE Efforts in Border Security Projects*, Proc. Fourth Int. Conf.Complex Systems Design & Management, CSD&M 2013, December 2013, Paris, France **(Best Paper Award)**.

Asan E. & Bilgen S. (2012). *Agility Problems in Traditional Systems Engineering – A case study*. Proc. Third Int. Conf.Complex Systems Design & Management, CSD&M 2012, December 2012, Paris, France.

Asan E. & Bilgen S. (2012). *Agile Collaborative Systems Engineering – Motivation for a Novel Approach to Systems Engineering*. 22nd Annual INCOSE International Symposium, June 2012, Rome, Italy.

Asan E. (2011). *Video Shot Boundary Detection by Graph Theoretic Approaches: Graph Theory & Shot Boundary Detection*, LAP LAMBERT Academic Publishing, 2011.

Asan E. & Alatan A. (2009). *Video Shot Boundary Detection by Graph Theoretic Approaches.* ISCIS -2009 Conference Proceedings Cyprus, September 2009. 24th International Symposium on Computer and Information Sciences (ISCIS).

Asan E. & Alatan A. (2009). *Baskın Kümeler Yöntemi ile Video Çekim Sınırı Sezme*. IEEE SIU-2009 Conference Proceedings Antalya, April 2009. Paper presented at IEEE 17th Signal Processing and Communications Applications Conference (SIU-2009).

# TEZ FOTOKOPİ İZİN FORMU

**ENSTİTÜ**

Fen Bilimleri Enstitüsü ☐

Sosyal Bilimler Enstitüsü ☐

Uygulamalı Matematik Enstitüsü ☐

Enformatik Enstitüsü ☐

Deniz Bilimleri Enstitüsü ☐

**YAZARIN**

Soyadı : AŞAN
Adı : EMRAH
Bölümü : BİLİŞİM SİSTEMLERİ

**TEZİN ADI** (İngilizce) : AGILE AND COLLABORATIVE SYSTEMS ENGINEERING

**TEZİN TÜRÜ** : Yüksek Lisans ☐ Doktora ☐

1. Tezimin tamamı dünya çapında erişime açılsın ve kaynak gösterilmek şartıyla tezimin bir kısmı veya tamamının fotokopisi alınsın. ☐
2. Tezimin tamamı yalnızca Orta Doğu Teknik Üniversitesi kullancılarının erişimine açılsın. (Bu seçenekle tezinizin fotokopisi ya da elektronik kopyası Kütüphane aracılığı ile ODTÜ dışına dağıtılmayacaktır.) ☐
3. Tezim bir (1) yıl süreyle erişime kapalı olsun. (Bu seçenekle tezinizin fotokopisi ya da elektronik kopyası Kütüphane aracılığı ile ODTÜ dışına dağıtılmayacaktır.) ☐

Yazarın imzası ..... Tarih .....