Developing and Securing Software for Small Space Systems

Brandon L. Shirley
Utah State University

DEVELOPING AND SECURING SOFTWARE FOR SMALL SPACE SYSTEMS

by

Brandon L. Shirley

A dissertation submitted in partial fulfillment
of the requirements for the degree

of

DOCTOR OF PHILOSOPHY

in

Computer Science

Approved:

Stephen W. Clyde, Ph.D. Major Professor	David E. Brown, Ph.D. Committee Member
Curtis E. Dyreson, Ph.D. Committee Member	Chad D. Mano, Ph.D. Committee Member
Dan W. Watson, Ph.D. Committee Member	Richard S. Inouye, Ph.D. Vice Provost for Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2019

ABSTRACT

Developing and Securing Software For Small Space Systems

by

Brandon L. Shirley, Doctor of Philosophy

Utah State University, 2019

Major Professor: Stephen W. Clyde, Ph.D.
Department: Computer Science

This research addresses two problems associated with developing smaller multi-vendor satellites for Small Space. These two problems interrelate and this research addresses them together. The *Development Problem* deals with the development of modular, reusable, and secure space systems while the *Security Problem* encompasses securing these space systems.

This research addresses the *Development Problem* by conducting a series of five surveys, referred to as Space Industry Software Development Practices and Attitudes (SISDPA), to asses current attitudes and state of practice among space system developers. This crystallized a need in space system development — modular reusable open networks can help Small Space realize its potential, but there is still a need to address certain security threats.

This research addresses the *Security Problem* by creating Secured Space Plug-and-Play Services Manager (SSSM), a secure modular reusable open-network software development framework based off of Space Plug-and-Play Services Manager (SSM). SSSM adds security provisions while minimizing the impact on developers using the framework. An evaluation of SSSM shows that it preserves the ease-of-use of SSM while adding policy enforcement in the form of authentication, access control, and encryption provisions.

(284 pages)

PUBLIC ABSTRACT

Developing and Securing Software For Small Space Systems

Brandon L. Shirley

The space systems industry is moving towards smaller multi-vendor satellites, known as Small Space. This shift is driven by economic and technological factors that necessitate hardware and software components that are modular, reusable, and secure. This research addresses two problems associated with the development of modular, reusable, and secure space systems: developing software for space systems (the *Development Problem*) and securing space systems (the *Security Problem*). These two problems are interrelated and this research addresses them together.

The *Development Problem* encompasses challenges that space systems developers face as they try to address the constraints induced by reduced budgets, design and development lifecycles, maintenance allowances, multi-vendor component integration and testing timelines. In order to satisfy these constraints a single small satellite might incorporate hardware and software components from dozens of organizations with independent work forces and schedules. The *Security Problem* deals with growing need to ensure that each one of these software or hardware components behaves according to policy or system design as well as the typical cybersecurity concerns that face any information system.

This research addresses the *Development Problem* by exploring the needs and barriers of Small Space to find the best path forward for the space systems industry to catch up with the methodology advancements already being widely used in other software fields. To do this exploration a series of five surveys, referred to as SISDPA, was conducted to asses current attitudes and state of practice among space system developers. This crystallized a need in space system development — modular reusable open networks can help Small Space realize its potential, but there is still need to address certain security threats.

This research addresses the *Security Problem* by augmenting a modular reusable open-network software development framework, called SSM, by adding policy enforcement in the form of authentication, access control, and encryption provisions, to create a new development framework, SSSM. This design and implementation adds security provisions while minimizing the impact on developers using the framework. SSSM is evaluated in terms of developer and system resource burden and shows that SSSM does not significantly increase developer burden and preserves the ease-of-use of SSM.

To Allyson, we would not be here without you.

ACKNOWLEDGMENTS

It has been a long road, I would like to acknowledge those who have helped or carried me along the way.

- The Space Dynamics Laboratory, for the PhD Fellowship and support via internal research and development funding.
- Allyson, for sharing the burden and prodding me along when I got complacent.
- Dr. Stephen Clyde, for letting me struggle when I needed to struggle and helping when I really needed help even though he was busy beyond belief.
- Brook Mckenna for her help and patience throughout the PhD Fellowship process.
- The various members of the Utah State University (USU) Computer Science faculty that I have pestered over the years, for their patience and help.
- Parents, for instilling a resilient stubbornness that helped get me through.
- Katelyn, Hannah, and Nora, for helping me remember what is important.
- Mark Greenman, for letting me vent, and generally giving me good advice.
- David Anderson and Ian Karlinsey, for helping stay on track during all those years down in Albuquerque.
- Brandon Holdaway, for his help in implemented SSSM.
- All those in Albuquerque and Logan that have put up with my struggle over the years.
- All the others who helped get me here.

Brandon L. Shirley

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SAMPLES

ACRONYMS

AMSAT-NA	Radio Amateur Satellite Corporation
ACAS	assured compliance assessment solution
ADCS	attitude determination and control system
AE	aerospace engineer
AES	Advanced Encryption Standard
AFB	Air Force Base
AFRL	Air Force Research Laboratory
AIAA	American Institute of Aeronautics and Astronautics
API	Application Programming Interface
AS	Authentication Server
ASCII	American Standard Code for Information Interchange
CAN	Controller Area Network
CCSDS	Consultative Committee for Space Data Systems
CDH	Command and Data Handling
CAS	Central Address Service
CFS	Core Flight System
COMSEC	Communications Security
CC	*Core Concepts*
CI	confidence interval
COTS	commercial of-the-shelf
CPU	Central Processing Unit
DARPA	Defense Advanced Research Projects Agency
DMA	direct memory access
DoD	Department of Defense
DoS	denial-of-service

EE	electrical engineer
EOL	end-of-life
FIPS	Federal Information Processing Standards
GB	gigabyte
GCM	Galois/Counter Mode
GHz	gigahertz
GPS	Global Position System
HACMS	High-Assurance Cyber Military Systems
HACSS	High-Assurance Cyber Space Systems
Hz	hertz
IA	information assurance
ICS	industrial control system
IDA	Interactive Disassembler
IDS	intrusion detection system
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
IPC	Inter-Process Communication
IPS	intrusion prevention system
IOT	Internet of Things
IRB	Internal Review Board
ITU	International Telecommunication Union
IV	Initialization Vector
I^2C	Inter-Integrated Circuit
KDC	Key Distribution Center
LA	logical address
LOI	letter of information
LEO	low earth orbit
LS	Lookup Service

MA	malicious application
MB	megabyte
MBps	megabytes per second
ME	mechanical engineer
MIT	Massachusetts Institute of Technology
MONA	Modular Open Network Architecture
MSV	Modular Space Vehicle
MTU	maximum transmission unit
NASA	National Aeronautics and Space Administration
NICTA	National Information and Communications Technology Australia
NISTIR	National Institute of Standards and Technology Interagency Report
Nmap	Network Mapper
OS	operating system
OSA	Open Systems Architecture/Approach
OSAL	operation system abstraction layer
OSAM	*Open Systems Architecture and Modularity*
OSI	Open Systems Interconnection
ORS	Operationally Responsive Space
OpenVAS	Open Vulnerability Assessment System
PI	principal investigator
PM	program manager
RAM	random access memory
RIPCC	*Reuse*, *Interoperability*, *Portability*, *Code Complexity*
RST	Responsive Space Testbed
SBC	single board computer
SCADA	supervisor control and data acquisition
SDL	Space Dynamics Laboratory
SDM	Satellite Data Model

SE	software engineer
seL4	security enhanced L4
SFS	Space Flight Software
SFSYS	Space Flight System
SGS	Space Ground Software
SGSYS	Space Ground System
SISDPA	Space Industry Software Development Practices and Attitudes
SpaWar	Space and Naval Warfare Systems Command
SMC	Space and Missile Systems Center
SM-1	SPA-1 subnet-manager
SM-E	SPA-E subnet-manager
SM-L	SPA-L subnet-manager
SM-S	SPA-S subnet-manager
SNAP	Standard Network Adapter for Payloads
SOIS	Spacecraft Onboard Interface Services
SPA	Space Plug-and-Play Architecture
SPA-1	SPA I^2C
SPA-E	SPA ethernet
SPA-L	SPA local
SPA-S	SPA SpaceWire
SPI	Serial Peripheral Interface
SSM	Space Plug-and-Play Services Manager
SSSM	Secured Space Plug-and-Play Services Manager
STL	Standard Template Library
STS	space test software
STSYS	space test system
SWaP	size, weight, and power
SysE	systems engineer

TE	thermal engineer
TEDS	Transducer Electronic Data Sheet
TGS	Ticket Granting Server
TGT	Ticket Granting Ticket
TL	technical lead
USB	Universal Serial Bus
USU	Utah State University
UUID	Universally Unique Identifier
VM	virtual machine
VMware	Virtual Machine Ware
XML	Extensible Markup Language
xTEDS	Extensible Transducer Electronic Data Sheet
XUUID	Extensible Transducer Electronic Data Sheet Universally Unique Identifier

CHAPTER 1
INTRODUCTION

The space systems industry is moving away from large monolithic satellites, known as Big Space, to smaller multi-vendor satellites, known as Small Space. This shift, driven by demand for lower costs, shorter schedules, new technology, and the ongoing government capability space race, requires hardware and software components to be substantially more modular, reusable, and secure than in the past. For example, a single small satellite might incorporate hardware and software components from dozens of organizations with independent work forces and schedules. To be efficient, not only do these organizations need to complete their components with minimal coordination, the components need to be decoupled as much possible so they are isolated from change in other parts of the system. Also, if the components, be it hardware or software, are modular with good abstractions and encapsulation, then they will be more reusable and thereby help reduce development costs in future space systems. The downside of having multiple independent organizations, with varying levels of trust, provide components for a space system is that the system's security provisions have to ensure that each component behaves according to policy or system design. This research addresses two problems associated with the development of modular, reusable, and secure space systems: developing software for space systems (the *Development Problem*) and securing space systems (the *Security Problem*). These two problems interrelate and this research addresses them together.

The *Development Problem* encompasses challenges that space systems developers face as they try to address the constraints induced by reduced budgets, design and development lifecycles, maintenance allowances, multi-vendor component integration and testing timelines. Big Space vehicles have had long development cycles, low risk postures, and require high information assurance; this typically results in high cost, difficulty with untrusted parts and software sourcing, and lag behind other industries [1–3]. Developers often develop one-off solutions while leveraging past successes, particularly with respect to achieving reliability and power efficiency [1]. There will likely always be a place for these types of systems, but there are a growing number of applications where this approach is no longer the gold standard, e.g. low earth orbit (LEO) applications, applications with short lifecycles, swarms, constellations, and high-risk applications. The space industry is realizing this new standard under the banner of Small Space.

The idea of using modularity and reuse to reduce cost in space system development is not a new concept; it has been tried before. Modularity has been explored in many aspects of space system design going back to the 1970s [4]. However, the various attempts do not seem to advance much past their originator or impact external development efforts. This could be due to the protected nature of space systems, especially when developed by government agencies, or it could be due to a failure to adopt development practices and tools that could facilitate this transition. Is it possible that the technology was not available yet to truly realize the reuse and modularity needed? Can Small Space ideals be coupled with reuse, modularity, and security to push and sustain this next evolutionary step in space system development? To explore these and other questions and to fully understand the development problem as well as the best path-forward for the space systems industry, this research designed and conducted a series of five surveys, referred to as SISDPA, to asses current attitudes and state of practice among space system developers. Chapter 3 describes the five SISDPA surveys and Appendix A shows the actual survey instruments.

To enable space systems development to be more modular, reusable, and secure, it is first necessary to better understand the current software development practices and perceptions among space system developers. Open networks are well-understood solutions for integrating systems of systems, as evidenced by the Internet of Things (IOT) and the Internet at large. An open network can enable high degrees of reuse, flexibility, and extensibility, because it lends itself to good abstraction, modularity, and encapsulation [5]. However, its openness can lead to additional challenges when it comes to security. The results of the SISDPA surveys support these assessments of the benefits and barriers for open network adoption in space systems development. See Chapter 4. Understanding the barriers, in particular, can help the industry eventually overcome these barriers and catch up with the methodology advancements already being widely used in other fields.

More specifically, Chapters 3 and 4 crystallize a need in space system development — modular reusable open networks can help Small Space realize its potential, but they still need to address certain security threats. This is the second problem this dissertation addresses and is simply referred to here as the *Security Problem*. It encompasses challenges that relate to the secure integration of components from different vendors and organizations.

To address the *Security Problem*, this research augments a modular reusable open-network software development framework, called SSM. SSM is set of software services that allow components to communicate over heterogeneous networks without knowledge of the network protocols or addressing schemes [6, 7]. This research adds policy enforcement to SSM in the form of authentication, access control, and encryption provisions, to create a new framework, SSSM. Its design adds security provisions while aiming to minimize the impact on developers using the framework. See Chapter 5.

Chapter 6 evaluates SSSM in terms of developer and system resource burden, because an increase in developer burden affects how easy a component is to use and therefore reuse and an increase in system resource burden reduces the set of applications where a component can be applied when criteria like size, weight, and power (SWaP) are driving factors. The evaluation shows that SSSM does not significantly increase developer burden. In other words, SSSM preserves the ease-of-use of SSM. Chapter 6 also shows that both SSM and SSSM have upper bounds on their network throughput that are tied to CPU and memory limitations and not the actual network. SSSM tops out before SSM and generally uses slightly more resources under nominal operation. The net result is a decrease in reusability from a system resource perspective. Therefore, an extremely resource-limited system that might be able to to use SSM might not be able to use SSSM. Chapter 6 identifies some ways around this problem by tweaking how a developer uses the API.

Finally, Chapter 7 summarizes the contributions of this dissertation relative to both the *Development* and *Security Problems*. It argues that the SISDPA surveys provide an in-depth understanding of the attributes and current practices in space systems development and that this has led to a better understanding of the benefits and barriers associated with using open networks in space systems. It also concludes that SSSM effectively addresses both problems. From a development perspective, SSSM provides an easy-to-use framework that allows developers to create space systems with better abstraction, modularity, and encapsulation. From a security perspective, it addresses the security concerns that were barriers to the adoption of open networks, even when there are multiple vendors involved.

CHAPTER 2
RELATED WORK

2.1 Introduction

The *Development Problem* and the *Security Problem* can be addressed by a solution that encompasses reuse, modularity, and security. Modularity enables reuse, and good abstraction and encapsulation enable modularity [5]. Developing software with clear boundaries helps lead to designs with good abstraction and encapsulation. Network architectures or development frameworks naturally define boundaries that help with abstraction and encapsulation enabling modularity. MONAs are a realization of this idea and there are examples of these in space systems development, [8–10] but MONAs are struggling to see widespread adoption. Increasing the usage of these architectures and approaches would help bring more modern software development processes and tools to spaces systems development while benefiting from the modularity, abstraction, and encapsulation that enable reuse.

A literature review of work related to the *Development Problem* did not uncover any other research that had undertaken the task of understanding the problem by surveying space system developers to understand their current practices and attitudes, particularly in relation to modularity, reuse, and security. Nevertheless, these important software engineering principles are common themes in space system development literature. Section 2.2 provides an overview of modularity and reuse in space systems literature, while Section 2.3 covers concepts that relate to security.

2.2 Modularity and Reuse in Space Systems

The idea of using modularity and reuse to reduce cost in space vehicle development is not a new concept; it has been tried before. Modularity has been explored in many aspects of space vehicle design going back to the 1970s [4]. The United States National Aeronautics and Space Administration (NASA) started developing a multi-mission spacecraft standard in 1978 [11] that was used in six space vehicles over a twelve year period starting in 1980 [12]. However, the various attempts do not seem to advance much past their originator or proliferate to external development efforts. This could be due to the protecting nature of space systems especially when developed by government agencies, or it could be due to a failure to realize the development practices and tools that could really facilitate this transition. For example, not too long ago the case was that software for space systems developed in C++ might not be allowed to use dynamic memory allocation or the Standard Template Library (STL) [7]. And it was not until the success of the Spirit and Opportunity Mars rovers in 2004, running VxWorks, that operating systems were even used on space systems [13]. Being able to use an operating system (OS) on a space system is a major reusability and modularity enabler.

This paradigm is making a resurgence as more and more commercial entities get into space and development gets more competitive. It is also making a resurgence on the government side as the need to test and produce new capabilities in shorter more cost effective time-spans is driven by both science and defense needs. The Department of Defense (DoD) sees open systems architectures as a way to reduce the cost to develop, maintain, and update systems [14, 15]. The DoD acquisition policy has even required that system providers use open system architectures where feasible [16], but exceptions, especially in space systems development, have been common. The DoD goes on to say "Open systems and modular architectures provide valuable mechanisms for continuing competition and incremental upgrades, and to facilitate reuse across the joint force." [17] Open systems architectures are viewed, by the DoD, as contributing to increased competition, reduced life-cycle cost, enhanced interoperability, cheaper and faster maintenance, schedule reduction, and increased innovation [18]. This relates to efforts like Northrop Grummans Modular Space Vehicle (MSV) Bus, Space and Missile Systems Center (SMC)s Standard Network Adapter for Payloads (SNAP), NASA's Core Flight System (CFS) System, and Air Force Research Laboratory (AFRL)'s SPA [8–10, 19] which also look to promote and create various pieces of reusable software.

CFS, the Spacecraft Onboard Interface Services (SOIS) architecture, and the SPA standards are examples of research and effort that has been put into developing reusable development frameworks that support open systems development and reuse across applications by various entities within the United States government. These endeavors have had various degrees of success within their respective organizations, but have not propagated much beyond the organizational boundaries.

NASA developed CFS to reduce repetition of effort in systems design and development that was resulting in soaring costs and extended development schedules [19–21]. CFS is built around a reusable core flight executive that uses a layered architecture to provide a standard middleware/bus and an API. This allows for a reusable component library to be built up that achieves some level of plug-and-play capability [22]. Layered approaches can be key to reusing software and well defined APIs are also critical. CFS provides a software bus that applications can use to relay data. Adding a new piece of hardware entails developing a new device driver. A software application must also be written to communicate with the new device, but an application can communicate with one or more hardware devices. The software application acts as the link between the hardware and the software bus and relays data between the two; the software bus is essentially a channel or pipe that applications may use to pass messages. The message passing protocol can stay the same even as new hardware is added. Once the application and driver is created it can, in theory, be reused with minimal effort from a software development perspective in other space systems that leverage CFS. CFS has been used in various NASA missions [23].

The SOIS architecture is an effort by an international committee, called Consultative Committee for Space Data Systems (CCSDS), to standardize interfaces for various functions and space systems components. This effort was again looking to reduce cost and promote reuse in space systems development. The CCSDS SOIS committee considering making SOIS SPA compliant to share development effort and promote reuse [7], but SOIS largely remains an architecture without an implementation due to a lack of a consensus, e.g. should SOIS be CFS or SPA compliant? The common thread here is that there has been a decent amount of development within each kingdom, but not much cross-pollination or consensus between kingdoms; the failure here might simply be tied to the nature of government agencies and budget infighting or it might stem from the difficulty of developing systems for the harsh space environment.

There are some issues with space systems development that are not present in terrestrial systems development and there has been research into the differences between software developed for space [6, 24] but there has not been any significant research into how developer preference and attitude about development practices differs from space systems development to terrestrial systems development. It may be that there are practices that can be done away with during the transition to lower cost shortened development cycles with high risk postures. This research furthers the understanding of why reuse has lagged and how terms like modularity and interoperability are perceived by the space systems development community to understand what can or should be coupled with Small Space development in order to realize a more agile and cost effective development process.

Finally, SSM is a development framework that implements the SPA networking standard [6, 25, 26]. It also adds an API to make is easy to develop with while adhering to the standard. AFRL developed the SPA standards for reasons very similar to the reasons NASA endeavored in developing CFS. This effort was nurtured by AFRL to support the Operationally Responsive Space (ORS) effort. This work was continued through the coming years [27, 28], and SPA got its first mention by name in the Responsive Space conference in 2005 [29]. The Responsive Space Testbed (RST) [30] was started at Kirtland Air Force Base (AFB) as a testbed for plug-and-play technology for space systems. These efforts were led by Dr. Lyke and Dr. Cannon [26, 31] and AFRL continued as a center for plug-and-play development for the next 7 years.

The idea was to provide a unified set of protocols that provide for the discovery and exchange of data and commands between modular space system components [7]. SSM made the networking component of SPA a reality and addressed some of its weaknesses via updates to the protocols. Components can now communicate on a heterogeneous open-network without the need for specialized hardware using standard networking buses and well defined software interfaces. This is a significant enabler for reuse as it provides for modularity, interoperability, and openness. SSM is starting to see more usage within government entities, but has seen little use outside the government due to restrictions on availability and limitations in implementation like determinism. Section C.1 of Appendix C covers SSM in more detail if more context is needed to understand the modifications to SSM that Chapter 5 details.

A sizable amount of work has gone into making space software systems more reusable. SSM has taken an approach that leverages an open-network topology that lends itself to building modular components enabled by good abstraction and encapsulation. The approach also opens the door to leveraging technology, protocols, and methodology for securing the systems. Adding security, if done right, can actually increase reusability because it can be reused in more applications, i.e. systems with more stringent information assurance (IA) requirements or concerns. Chapter 4 analysis suggest that this lack of security in development frameworks like SSM might be one of the factors restricting use. SSM is well-situated as a open-network development framework to address security concerns because security mechanisms can be added to software component egress points to protect and regulate traffic.

2.3 Security for Space Systems

Assessing and surveying the currently available solutions and ongoing research relate to the space system cybersecurity problem did not yield any research or existing technology that provides a secure modular reusable open network software development framework for space systems. Much of the research falls into policy, process, security tools, hardening, and different types of solutions for space systems and similar systems like the smart grid or IOT systems. These solutions and research are all solving pieces of the larger problem: how to secure heterogeneous networks of heterogeneous systems in diverse, sometimes harsh, and inaccessible environments. Everything is interconnected and space is increasingly becoming part of ground infrastructure from Global Position System (GPS) to satellite Internet to power grids [32]; compromising a space system has very real consequences on the ground [32–36].

The related research can be broken down into a few different camps: policy and principles, process and tools, and actual security provisions. There is a lot of interest in this area and body of research, this section covers a representative body of work but is not meant to be comprehensive. First, this section reviews some of the policy and principles research around cybersecurity for space systems as well as the specific set of security principles that fed into the design of SSSM and whose importance is validated by results for the survey analysis. Next comes process and tools research that has and is going into making the development of space systems more secure by helping to define how space systems software can be developed more securely and how that security can be tested. Finally, this section covers security provisions that are being developed or researched for space systems generally and reusable space system specifically.

2.3.1 Policy and Principles

Cybersecurity for space systems has largely been unregulated, the main international body that regulates space systems is the International Telecommunication Union (ITU), and they are concerned mainly with frequency interference. The ITU did put forth an agenda to begin to address cybersecurity in space systems in 2007, but it has not be updated since then [32]. This is a huge issue as it means there is not a set of standards for defining and validating a secure space system. This means that space system are often the weakest link in any infrastructure or data system. This is complicated further by the difficultly in patching or upgrading these system and the ballooning number of sources for hardware and software that might go into a single space system all developed to different security standards, no security standards at all, or possibly with actual malicious intent.

Figure 2.1 illustrates a compromise via a satellite-to-satellite attack using a malicious payload with a separate radio. This is one example, the payload does not need its own radio, the compromise could come anywhere along the chain once the satellite is exposed to terrestrial and space-based networks; also note that core components, like an attitude determination and control system (ADCS), might be sourced from third parties and provide the same type of backdoor into a space system. In Figure 2.1, the malicious payload would have been sourced from a third-party and even then the software and components that make up the payload might have come from many different sources. This issue is complicated by a

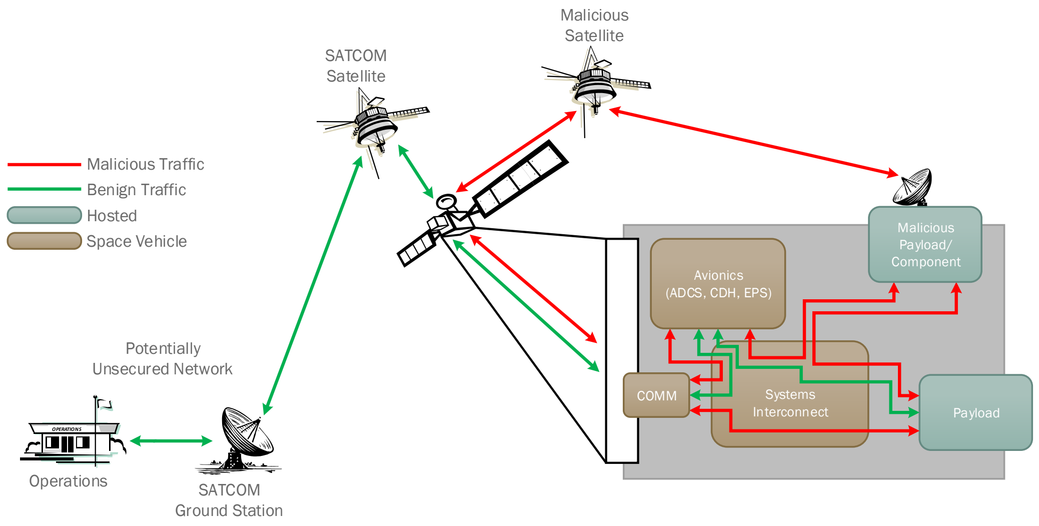


Fig. 2.1. Malicious Satellite to Satellite Scenario. Potential attack scenario illustrating issues with sourcing components and software from various providers with differing security standards.

trend, especially in small satellites or *CubeSats*, towards the integration multiple commercial of-the-shelf (COTS) parts to reduce cost and schedule [32,33]. NASA even goes as far as to have a catalog of approved vendors, but cybersecurity at the level of firmware and software is not always addressed [32]. There is ongoing research going into addressing this lack of cohesive security standards and component vetting or validation process, some of this falls under policy and principles and will be covered here, and some of it falls under process and tools and will be covered in Section 2.3.2.

A good place to start is with a set of security principles that can be used to secure a space system or set of space systems. A working set of principles for security provides a lens for understanding security for space systems and systems like them. Figure 2.1 illustrates a few points, but on of the key ones is a lack of access control, if the malicious payload comes from an untrusted source, but it is required and cannot be further vetted then a space system should have a way to implement the concept of least privilege where access is only granted to the level the component or piece of software needs to do its job. If it is a sensor then it is only allowed to provide a telemetry stream, but cannot issue commands or access other telemetry streams. This is a form of access control, and the first set of principles covered are from smart grid cybersecurity and deal with access control. "National Institute of Standards and Technology Interagency Report (NISTIR) 7628 Guidelines for Smart Grid Cybersecurity" establish a set of principles for providing security [37]. These principles are listed below along with brief description of how they relate to access control specifically:

1. **Identity Management** — Communicating entities need to establish their identities so that authentication can proceed
2. **Mutual Authentication** — Once identity is established, the communicating entities need to authenticate each other's identity
3. **Authorization** — Once authenticated permissions and rules can be used to authorize or deny access based on entity identities
4. **Auditing** — Tracking pertinent events for troubleshooting system issues and for detecting unauthorized behavior
5. **Confidentiality** — Interactions between entities need to be private, otherwise controlling access has little effect
6. **Integrity** — Interactions between entities need to be free of alteration or other subversion, otherwise the resource was not truly accessed
7. **Availability** — Services or assets need to be available when expected

These principles where developed with power and utility grids in mind, but these systems face many of the same problems facing space systems as they become more interconnected and accessible [38–42]. Space and the grid previously enjoyed some level of security due to their inaccessibility, and both now need to address security as they become more accessible and expand their component sourcing.

The design and implementation of SSSM relied on these principles, along with the following additional principles from Ingols et al. at Massachusetts Institute of Technology (MIT) for developing secure kernels [42].

1. **Fail Slowly** — Commands that are dangerous for a space system should take a while before they execute so there is time to recover before there is irreversible damage.
2. **Crypto Beyond Communications Security (COMSEC)** — Cryptographically enforce role-based access control.
3. **Root of Recovery** — Have a way to reassert control of the vehicle given that physical access is generally not possible.
4. **Ablative Defense** — Ensure the defense mechanisms do not jeopardize the space systems utility in terms of it purpose
5. **Reboot and Succeed Quickly** — ensure the process to recovery, if it requires restarting the system, is quick so that normal operations be resume quickly.

There is a body of research that sees the lack of internal regulation and standards for space systems as a growing area of concern; the assertion of this research is that space systems need an international regulatory body to drive standards and regulations to bring the same level of protection and control that is present for terrestrial systems [32–34]. Falco et al. [32] do an in-depth dive on the special problem area that is space system's cybersecurity. They review the ramifications and difficult problems of space systems and suggest that a good deal of it can be addressed by bringing space systems regulation up to par with terrestrial standards and regulations. They put the onus on governments, space asset organizations, and policy makers to develop these standards and suggest that common ground practices like having a center for sharing threat information and maintaining documentation of space system cybersecurity principles and standards.

2.3.2 Process and Tools

Even if policies, principles, and standards are put in place, there is a need for processes and development tools that lead to secure systems and validation tools that validate the security of these systems. There are various groups putting effort towards developing processes and integrating tools that allow for secure space systems development even when parts and software are sourced from many different vendors or repurposed from different industries. One such group is Space and Naval Warfare Systems Command (SpaWar) [41,43]. A current goal of this group is to securely address traditional space system cost drivers. Integrating relatively low cost COTS components while addressing IA is essential to being able to develop and field systems quickly. To do this they are looking at net-centric technologies that keep prices down and promote reuse while trying to borrow concepts and components from supervisor control and data acquisition (SCADA) and industrial control system (ICS) technology, which have security problems [39, 44]. SpaWar is mapping a cybersecurity overlay to the typical small satellite life-cycle so that security is baked into the development process as shown below [43]:

- **Concept Development and Design** — Assess vulnerabilities and plan security controls
- **Payload and Subsystem Development** — Incorporate security controls, and gray box and black box testing of interfaces
- **Bus, Payload, and Ground Subsystem Acceptance** — static and dynamic analysis and reverse engineering
- **System-level Integration and Testing** — dynamic analysis and testing of vulnerability to signal interference, inception, and injection
- **Launch, On-orbit Operations and Maintenance** — monitor and defend network and components

The idea being to also take a layered approach where hardware, firmware, OS, flight software, and network interfaces are assessed. They propose integrating standard vulnerability scanners, and static and dynamic analysis tools, like assured compliance assessment solution (ACAS), Open Vulnerability Assessment System (OpenVAS), Network Mapper (Nmap), Metasploit, and Interactive Disassembler (IDA) Pro. They are also looking into ICS monitoring tools like Sophia to provide continuous monitoring of network flow. These tools, modeling, and testbeds can help ensure that the aggregate system is secure as well as test the efficacy of a software development framework like SSSM.

There is a lot similar work with SCADA and ICS in terms of the industrial automation and smart grids that is looking at modeling, analysis, scanning, and monitoring that looks at analyzing and testing systems during development as well as actively and passively monitoring them once they are deployed using various tools [38, 39, 44].

2.3.3 Implementations

One of the fundamental tenants of cybersecurity for network systems is “CIA”, which stands for Confidentiality, Integrity, and Availability. However, many small space systems, reusable systems, modular open systems, IOT systems, and smart grid systems have prioritized these terms as “AIC” [41], where the access to the data or connectivity to the asset is more important than protecting the data or securing the asset. This might not be as critical when a system is on the ground and it is turning off a light switch, but it becomes very critical when an attacker is stealing sensitive information, causing a blackout, or redirecting a satellite into another satellite. Once control is lost on a remote space system it might not be impossible to to recover, there typically is no way to physically reassert control [42], it is very important to make sure control is never lost or that there is a robust path to recovery or graceful end-of-life (EOL). There is various ongoing research into adding security layers or mechanisms to space systems; and a specific assertion that space systems need encryption for use on-board satellites as another layer of protection beyond the COMSEC boundary [42, 45]. This is the bucket of research that into which SSSM falls. First, other research in this area is discussed and then SSSM is briefly discussed in terms of the related technology and research it utilizes, this is of course covered in more detail in Chapter 5. There is various ongoing research into implementing cybersecurity space systems, there is also some relevant research into IOT and smart grids.

SpaWar, in conjunction with their procedure and tools efforts, are taking the Defense

Advanced Research Projects Agency (DARPA) High-Assurance Cyber Military Systems (HACMS) program and applying it to their High-Assurance Cyber Space Systems (HACSS) approach to securing space systems [41]. This starts with a secure kernel that secure software components can be run against and the goes back to their process and tools for generating and validating secure software. The idea would be to leverage a secure separation kernel that would provide separate containers for running the flight components and a virtual machine for running legacy Linux applications, third-party software and interfacing with third-party hardware. There would still be an issue when these components need to talk and certain access needs to be available between the secure and unsecured domains. This type of secured foundation would be an excellent base for SSSM as SSSM could provide for a secured interface between the flight software components and COTS hardware and software on the local subnet, or across heterogeneous space system networks with systems on other processors.

In a very related vein, Ingols et al. at MIT are also looking at using a secure microkernel called security enhanced L4 (seL4) [42]. seL4 was developed by National Information and Communications Technology Australia (NICTA) and the DARPA HACMS program. seL4 is piece of the their secure development platform that encompasses Zynq 7000 series parts, Inter-Integrated Circuit (I^2C) and Serial Peripheral Interface (SPI) based devices for communication, and are actively porting CFS as reusable flight software. CFS provides an operation system abstraction layer (OSAL) layer for Linux, FreeRTOS, and VxWorks, but in the case of seL4 they have to and are implementing this layer themselves. OSAL expects some features that seL4 does not provide, but they were able to work around this for the support they needed and opt to ignore any networking support. If all this work can come together then it will provide a reusable development platform. This effort is the closet in spirit to the SSSM research and starts from the ground up, it however does not support networking and is limited to the buses described and its Inter-Process Communication (IPC) message passing for component communication and appears to be very tied to the set of hardware they have chosen to support.

SSSM is modeled after the security protocols in Kerberos. It was chosen after looking at various options that would have required invasive modification of SSM that might have affected ease-of-use or that might have required external management of policy. Kerberos was initially developed by MIT in conjunction with Project Athena and is still under active development [46, 46, 47]. The protocols from Kerberos are combined with OpenSSL for symmetric Advanced Encryption Standard (AES) encryption to round out the solution. SSSM does not provide for process isolation and makes no claims about the security of the OS it resides on so software components would need to reside on separate processing nodes and/or leverage a secure separation kernel or something like seL4 microkernel as previously described. Section C.2 of Appendix C covers Kerberos in more detail.

CHAPTER 3
SURVEY SERIES DESCRIPTION

3.1 Introduction

As mentioned in Chapter 1, space systems development needs to become more modular, reusable, and secure. To achieve this transition it is first necessary to understand the current software development practices and perceptions in the space system industry. To this end, this research includes a series of five surveys, labeled: *CC*, *OSAM*, *Security*, *RIPCC*, and *Network*. Collectively, the surveys include questions that aim to shed light on the following, for practitioners in the space system industry:

1. Perception of and experience with reuse, portability, interoperability, and security; along with development area allocation. The *CC* Survey looks at these areas.

2. Attitudes toward modular open-network software development frameworks as they impact or relate to interoperability, integration, adaptability, and reusability. The *OSAM* Survey covers this area.

3. Experience with and the perceived importance of various security mechanisms and open source software. The *Security* Survey addresses these areas.

4. Perception and experience with reuse, interoperability, portability, and code complexity in relation to various aspects of space systems and space systems development. The *RIPCC* Survey addresses these topics.

5. Experience with and perception of various network types in space systems and how these networks affect space system aspects like security, code complexity, adaptability, fault tolerance, and interoperability. The *Network* Survey addresses these areas.

3.2 Survey Questions

In the spirit of good modularity and reuse, the surveying itself was broken up into five independent surveys, each with its own instruments, but the instruments relied on common types of questions:

Likert Scale — These question used either a 3-point or 5-point Likert scale, and typically asked participants to rate items in terms of importance or difficulty.

Ranking — These questions asked participants rank a set of items, and typically asked participants to rank items in term of importance or difficulty.

Polar — These questions asked participants to answer a question with a binary choice of answers, typically a yes-no question.

Percentage — These questions ask participants to give a percentage or share of an area.

Check-all-that-apply — These questions allow participants to provide multiple answers to a single question.

Pro-neutral-con — These questions gave participants a item and asked them to categorize its effect on a common set of items related to space system development by qualifying the items effect as either beneficial or Pro, neutral, negative or Con. The common set of items is always the same, Appendix A, Section A.10 gives a full listing of items and definitions that were available to the participants.

Appendix A presents the full instrumentation for each survey as presented to the participants. Here is the breakdown and description of the questions used for each of the five surveys:

***CC* Survey** — Used in questions related to code complexity, reuse, networking, and security. The full survey instrumentation is covered in Appendix A, Section A.5.

- ***CC* 2.1** — *Likert Scale.* Asks participants to rate the importance of reuse, portability, interoperability, minimal code complexity, rapid development, cost of ownership, and security on a one-to-five Likert scale in relation to Space Flight Software (SFS), other software fields, Space Ground Software (SGS), and space test software (STS).

- ***CC* 2.2** — *Ranking.* Asks participants to rank reuse, portability, interoperability, minimal code complexity, rapid development, cost of ownership, and security.

OSAM **Survey** — Used in questions related to modular open-network system approaches, networking, and security. The full survey instrumentation is covered in Appendix A, Section A.6.

OSAM **2.4** — *Polar.* Asks participants to indicate yes or no if their organization has employed, is employing, or will be employing open system architecture or open network architecture.

OSAM **2.5** — *Percentage.* Asks participants to indicate the percentage of project/missions use OSA, MONA, closed proprietary systems, or other.

OSAM **2.6** — *Check-all-that-apply.* Ask participants to indicate all the factors that might prohibit their organization from using open systems approaches. The full listing of factors is shown in Table 4.34 of Chapter 4 as well as the Section A.6 of Appendix A.7.

Security **Survey** — Used in questions related to networking and security. The full survey instrumentation is covered in Appendix A, Section A.7.

Security **2.1** — *Pro-neutral-con.* Asks participants to indicate the effect of internal security on each of the common items.

Security **2.2** — *Polar.* Asks participants to indicated if they have direct security experience with Space Flight System (SFSYS)s, Space Ground System (SGSYS)s, space test system (STSYS)s, penetration testing, or other.

Security **2.6** — *Likert Scale.* Asks participants to rate the importance of Identify Management, Mutual Authentication, Authorization, Auditing, Confidentiality, Integrity, Availability, Well-defined Interfaces, Abstraction layers, Access Control, Network Segmentation, Compliance, Testing, Recovery, Migration, and Other in relation to Open-network SFSYSs and Traditional SFSYSs.

Security **2.7** — *Likert Scale.* Asks participants to rate the difficulty of providing for Identify Management, Mutual Authentication, Authorization, Auditing, Confidentiality, Integrity, Availability, Well-defined Interfaces, Abstraction layers, Access Control, Network Segmentation, Compliance, Testing, Recovery, Migration, and Other in relation to Open-network SFSYSs and Traditional SFSYSs.

***Security* 2.8** — *Ranking.* Asks participants to rank Identify Management, Mutual Authentication, Authorization, Auditing, Confidentiality, Integrity, Availability, Well-defined Interfaces, Abstraction layers, Access Control, Network Segmentation, Compliance, Testing, Recovery, Migration, and Other in terms of importance with relation to SFSYSs.

***RIPCC* Survey** — Used questions related to code complexity and reusability. The full survey instrumentation is covered in Appendix A, Section A.8.

***RIPCC* 2.10** — *Pro-neutral-con.* Asks participants to indicate the effect of software reuse on each of the common items.

***RIPCC* 2.13** — *Pro-neutral-con.* Asks participants to indicate the effect of code complexity on each of the common items.

***RIPCC* 2.15** — *Likert Scale.* Asks participants to rate the importance of cyclomatic complexity, depth of inheritance, class coupling, methods per class, lock of cohesion, data complexity, data flow complexity, decisional complexity, language complexity, interface complexity, lines of code, and other with regard to measuring code complexity.

***Network* Survey** — Used a question related to modular open-network system approaches, networking, and security. The full survey instrumentation is covered in Appendix A, Section A.9.

***Network* 2.3** — *Pro-neutral-con.* Asks participants to indicate the effect of internally networked space systems on each of the common items.

The *CC* and *RIPCC* Survey questions used show that developers perceive minimal code complexity to be important and beneficial for space systems. These surveys also show that developers feel that reusability is important and beneficial for space systems. The *Network* and *OSAM* Survey questions used show that developers see open-network systems to be largely beneficial and increasing in use. The *CC*, *OSAM*, *Security*, and *Network*

Surveys questions used show that developers see open-network space systems as having a negative impact on security, but an overall positive impact.

Each survey also includes a common set of background questions, these questions used the same question types as above. Appendix A includes the specific background questions for each survey, and Appendix B provides an overview of the background question results. These questions ask participants about the roles they have participated in, years of experience in various areas, perception of development phase, duration, total mission or project count, typical concurrent project or mission count, and organization type affiliation.

3.3 Survey Distribution

To reach out to various developers, engineers, and managers within the space systems development, the participant solicitation targeted multiple distribution lists. The potential participant pool makeup consisted of about 900 from the American Institute of Aeronautics and Astronautics (AIAA)/USU Conference on Small Satellites, 785 from the CubeSat Mailing list, 1500 from the Radio Amateur Satellite Corporation (AMSAT-NA) Bulletin Board Mailing List list, and about 20 – 40 from the Space Dynamics Laboratory (SDL) software developers mailing list. Outlined below are each of the mailing lists used to distribute the surveys:

1. AIAA/USU Conference on Small Satellites: The Commerce of Small Satellites 2014 Participation — provides direct access to email addresses from the 2014 conference. Invitation to participate was sent to the addresses on this list directly via the Qualtrics email system.
2. Cubesat Mailing List — provides access to educators, developers, and some vendors that have a specific interest in small satellites, specifically cubesats or U-class spacecraft [48]. Access to this list is available via the cubesat@cubesat.org email address.
3. AMSAT-NA Bulletin Board Mailing List — provides access to a group that is primarily made up of amateur satellite builders and users, with a particular emphasis on radio. Access to this list is available via the amsat-bb@amsat.org email address and

coordination with the maintainers.

4. SDL — provides access to software developers within SDL.

The method of invitation to participate varied based on the distribution list. For example, the participation in List 1 described above were sent email invitation directly. Those in Lists 2 and 3 were invited through email messages sent to their respective group email addresses. Those in List 4 where invited by email message sent by an SDL employee who had access to all employee email addresses.

Although each survey focuses on a specific set of ideas, each was designed to stand on its own. Therefore, for the series to gather useful information, it is not necessary for all participants to complete all surveys.

The main incentive for respondents to participate is helping to increase the body of knowledge that relates to software development for space systems and helping to direct future research. Also, taking a survey provides participants with an opportunity to reflect on their current software development practices and how these practices affect the projects on which they have worked in the past, are working currently, and will work in the future. The survey-specific drawings gave two randomly selected participants a $25 gift card. The overall drawing gave two randomly selected participants from the pool of all participants of any survey a $200 gift card. At the end of each of the surveys, a participant was redirected to a web-page that asks for an email address. A participant had to enter a valid email address to be considered for that surveys drawing or the overall drawing; email addresses were tracked separately from surveys.

CHAPTER 4
SURVEY SERIES RESULTS AND ANALYSIS

4.1 Introduction

This chapter presents the primary findings from a detailed analysis of responses from the five SISDPA surveys. The findings of this analysis establish the importance of securing space systems and developing secured modular reusable software development frameworks, like SSSM, for space systems. This shows that a secured modular reusable software development framework is key in addressing the *Development* and *Security* Problems. Section 4.2 presents an overview of the participation numbers for each survey, with a more in depth breakdown of the participant demographic covered in Appendix B. Sections 4.3 through 4.6 analyze results related to code complexity, reusability, modular open-network architectures or open systems approaches, and security. Each of these sections covers one of these topic areas, e.g. code complexity, and analyzes the set of survey questions that relate to the area and had significant results. Typically each section starts with an overview of the findings in terms of the high-level topic area, e.g. code complexity, and then describes the sub-areas that are analyzed and make up this high-level topic area. Each subsection covers one of these sub-topics, e.g. Code Complexity Importance and Benefit, and has sub-subsections that present the survey question analysis as it relates to the sub-topics and the high-level topic. Section 4.7 concludes with a summary of the findings and some general insights.

All the results tables use a color coding scheme to help visualize strong-positive, weak, and strong-nevative t-values. Table 4.1 shows the typical t-value ranges designated for each color. Very strong negative t-values are shown in red, this is typically below -4. Strong negative t-values are shown in orange, this is typically from below the negative critical t-value, usually around -2, down to -4; this will vary based on the degrees of freedom for the test.

Table 4.1. Color Key for t-value Row Coloring in Question Results Tables. The -2 and 2 values vary a bit and are dependent on the degrees of freedom for the given test.

Strengthof t-value	Rough t-value
Very strong negative	$t \leq -4$
Strong negative	$-4 \leq t \leq -2$
Weak	$-2 \leq t \leq 2$
Strong positive	$2 \leq t \leq 4$
Very strong positive	$t \geq 4$

Larger numbers of participants require a lower t-value to be significant. Neutral t-values are shown in gray. Strong positive t-values are shown in green, this is typically from above the positive critical t-value, usually around 2, up to 4. Very strong positive t-values are shown in blue, this is typically above 4.

4.2 Survey Series Participation

Table 4.2 illustrates that the usable response rate is between 0.9% and 3.1%. The average usable response rate for the surveys is ~1.5% with a standard deviation of ~0.5. Each survey went through multiple rounds of email solicitations to achieve this response rate.

The *CC* survey saw the best yield with a total of 97 usable participants, this is even after removal of the invalid and partial responses. Table 4.2 shows that the *CC* Survey started with a total of 176 survey responses. A total of 64 blank responses left 112 potentially usable participant responses. The *CC* survey was the first survey administered and as a result suffered from the problem where participants only took the background portion of the survey. Later surveys had the order of the background and the survey specific sections reversed to mitigate this problem. Segregating out the background-only participation left 97 participants, meaning 15 participants only filled out the background section of this survey; this is why 97 participants are shown as usable in Table 4.2. This leaves a usable response rate of 3.1%, which is double the next best response rate.

The *CC* Survey was available online for about 6 months, this was arguably too long a duration to let the survey run, but this survey did get the most responses. This was not a sustainable duration to let the remaining surveys run as administering all 6 would have taken 3 years. Another option would have been to let them overlap, but there was already confusion about which survey was which and questions from participants about whether or not they had taken a survey. Survey participation had to be anonymous so participants were not able to be removed from the distribution list once they participated in a given survey, also some of the distribution lists were blind lists, so email addresses could not be removed. USU's Internal Review Board (IRB) thought it would be too stressful for participants take the survey series as one long survey. In total 5 recruitment emails were sent out over that 6 month duration with a thank you email at the end. The other surveys were delivered at a much higher cadence, typically with 4 recruitment emails about a week apart over the course of a month.

Table 4.2. Survey Participant Response and Usability Rates

Survey	Participant Pool	Respondent Count	Usable Responses	Response Rate	Usable Rate
CC	3173	176	97	5.5%	3.1%
OSAM	3085	115	44	3.7%	1.4%
Security	3060	107	31	3.5%	1.0%
RIPCC	3038	83	29	2.7%	0.9%
Network	3024	93	32	3.1%	1.0%

Table 4.2 shows that the other 4 surveys all had similar respondent counts and usable rates leaving each of them with about 30 to 40 usable responses. Care was taken during analysis so that the groupings used to understand participant responses did not over segment the respondents to the point where only one or two respondents were left in a dominant category. For example, a question that used an 1-to-5 importance scale might be reduced to a 1-to-3 importance scale.

Over the course of the survey 149 potential participants opted out of the future survey participation, this number was only driven by those who could be solicited via direct

Qualtrics delivery. There was not a way to opt out participants that belong to the Cubesat or AMSAT mailing lists. This opt out rate did not have a notable effect on *Usable Rate* or *Usable Responses*, i.e. the 3173 that the pool started with versus the 3024 that the pool ended with did not really have a significant effect on percentages or *Usable Responses*. It is not clear how many participants mentally opted out of the survey mailing lists.

It should be noted that the responses to the *Background* sections of the survey series show that software engineer (SE)s had strong representation and the participants have good development experience mixtures, and that certain management roles tended to be multi-role. The *Background* results where explored but not formally analyzed, so the findings from this exploration are covered in Appendix B.

4.3 Participant Perception of Code Complexity

This section presents findings related to software code complexity. These findings stem from analyzing response to questions in the *CC* and *RIPCC* surveys. These findings present the participating space systems developers' perception of the importance and benefit of minimal code complexity as well as important code complexity measurement methods.

Section 4.3.1 shows a a general consensus among participants that minimal code complexity is important and beneficial to space systems and space systems development. Further, the results show that minimal code complexity is the most important for SFS. Section 4.3.2 covers participating developers' perception that some code complexity metrics are more important for measuring code complexity for space systems software.

Developer perception of the importance of minimal code complexity relates to the security portion of this research because is shows that developers believe that minimizing code complexity is important to space systems development. This gives strength to the idea that adding security to a software development framework, like SSM, with minimal increase to code complexity is a valuable contribution to the space systems development community. This analysis also shows that participants believe that the lack of cohesion and cyclomatic complexity metrics are the most important for measuring code complexity. This information is generally useful to the space systems development community, but also relates to the code complexity metrics that could be used to measure the delta in code complexity between SSM and SSSM.

4.3.1 Code Complexity Importance and Benefit

The consensus among the space systems developers that participated in the *CC* and *RIPCC* Surveys is that minimizing code complexity is important and beneficial to software developed for space systems. This is generally relevant as it suggests at least one way of quantitatively measure the quality of code.

This section analyzes the responses to *CC* Survey, Questions 2.1 and 2.2, as they relate to the importance of code complexity. This section also analyzes the responses to *RIPCC* Survey, Question 2.13. Analysis of Question 2.1 shows that participants perceive minimizing code complexity to be generally important for S*S and other software, and that they found it to be the most important for SFS. The participants found code complexity to be generally less important than reusability. Section 4.3.1.1 presents the analysis of Question 2.1. Analysis of Question 2.2 shows that participants perceive code complexity to have an average importance ranking mean. Section 4.3.1.2 presents the analysis of Question 2.2. Analysis of Question 2.13 shows that participants perceive minimizing code complexity to be generally beneficial, and specifically beneficial to aspects that related to usability of the code, aspects that, in theory, make the software easier to use. Section 4.3.1.3 presents the analysis of Question 2.13.

4.3.1.1 *CC* Survey, Question 2.1 Analysis, Part 1

This section shows that the participants perceive code complexity to be important for SFS, SGS, STS, and other software fields. This section also shows that participants perceive code complexity to be more important for SFS than for SGS, STS, or other software fields. This section shows that minimizing code complexity is seen to be generally more important for space systems software than for other software. Finally, this section shows code complexity is generally seen as less important than reuse, but equal in importance to security for space systems related software and other software fields.

CC Survey, Question 2.1 is a Pros-neutral-cons rating question, for these questions, a special caveat was listed for how neutrally impacted aspects would be counted:

> "All items should be placed for the answer to be considered complete by the system. In considering partial or incomplete answers, unplaced items will be considered neutral."

This means that aspects that were not placed can be counted as neutral, if no aspects where placed then the response was dropped. If the participant placed some of the items then the analysis proceeded with two interpretations, i.e. one where non-placed aspects were assumed neutral, and one where non-placed aspects were dropped. There was little statistical difference between the two interpretations so the assumed neutral variant is used for all Pros-neutral-cons rating questions. This analysis covers three statistical tests. Test 1, a one-sample t-test, looks at code complexity importance for software in general. Test 2, a matched-pairs t-test, compares code complexity across different software areas. Test 3, a second matched-pairs t-test, compares code complexity to other software aspects.

Test 1

Summary: Participants see code complexity as being important for SFS, SGS, STS, and other software fields.

Question: Question 2.1 asks each participant to rate the importance of code complexity for SFS, SGS, STS, and other software fields on a 5-point Likert scale.

Analysis: One-sample t-test with test value of 3, which represents a neutral value.

Hypothesis: Software developers will find code complexity to have a non-neutral importance.

H_0: $\mu\text{CodeComplexityImportance} = 3$

H_A: $\mu\text{CodeComplexityImportance} \neq 3$

Table 4.3 shows the statistics for this test. Table 4.3 shows that all the means for code complexity importance are above neutral, including the *overall* mean.

Table 4.3. *CC* 2.1, Part 1 — Code Complexity Importance Mean One-sample Test Statistics, Test Value 3, 5-point Likert Scale

Area	N	Mean	Std. Deviation	Std. Error Mean
SFS	90	3.98	0.960	0.101
SGS	90	3.36	0.975	0.103
STS	89	3.51	1.035	0.110
Other Software	79	3.44	1.010	0.114
Overall	93	3.61	0.746	0.077

Table 4.4 shows that all of the categories, S*S individually and overall, as well as *other software* fields, show strong evidence against the hypothesis that participants perceive code complexity importance to be neutral. This means that code complexity is generally seen as important for all of the software areas under consideration by the participants.

The general importance of code complexity means that it should be considered and weighed against the gains provided by reuse and security that a open-network software development framework might offer.

Table 4.4. *CC* 2.1, Part 1 — Code Complexity Importance Mean, One-sample Test Results, Test Value 3, 5-point Likert Scale

						95% CI	
Area	Critical t-value (2-tailed)	Sample t-value (2-tailed)	df	Sig. (2-tailed)	Mean Difference	Lower	Upper
SFS	1.9870	9.666	89	0.000	0.978	0.78	1.18
SGS	1.9870	3.459	89	0.001	0.356	0.15	0.56
STS	1.9873	4.609	88	0.000	0.506	0.29	0.72
Other Software	1.9909	3.901	78	0.000	0.443	0.22	0.67
S*S Overall	1.9861	7.853	92	0.000	0.608	0.45	0.76

Test 2

Summary: Participants perceive code complexity to be more important for SFS than for SGS, STS, or other software fields. The participants perceived code complexity to be more important for space systems in general than for other software. They also perceive code complexity to be less important than reuse and equivalent to security for both space systems software and other software fields.

Question: Question 2.1 asks each participant to rate the importance of code complexity for SFS, SGS, STS, and other software fields on a 5-point Likert scale.

Analysis: Matched-pairs t-test to compare means of various combinations of SFS, SGS, STS, and other software fields.

Hypothesis:

H_0: μS*S = μOtherSoftware

H_A: μS*S $\neq$ μOtherSoftware

Table 4.5 shows that the mean for SFS is above all the other means. Table 4.5 also shows that S*S in general is above other software.

Table 4.5. *CC* 2.1, Part 1 — Code Complexity Importance Mean, Matched-pairs Test Statistics, 5-point Likert Scale

	Area	Mean	N	Std. Deviation	Std. Error Mean
Pair A1	SFS	3.93	76	0.998	0.114
	Other Software	3.46	76	1.012	0.116
Pair A2	SGS	3.43	77	0.952	0.108
	Other Software	3.44	77	1.019	0.116
Pair A3	STS	3.53	76	1.013	0.116
	Other Software	3.45	76	1.012	0.116
Pair A4	SFS	3.98	87	0.964	0.103
	STS	3.52	87	1.044	0.112
Pair A5	STS	3.51	87	1.044	0.112
	SGS	3.36	87	0.988	0.106
Pair A6	SFS	3.97	88	0.964	0.103
	SGS	3.38	88	0.975	0.104
Pair A7	S*S	3.62	78	0.758	0.086
	Other Software	3.44	78	1.014	0.115

Table 4.6 shows the results for this test. The low p-values and strong positive t-values for Pairs *A1*, *A4*, *A6*, and *A7* give strong evidence that the group perceives code complexity

to be more important for SFS than for any of the other software categories considered by the survey. Minimizing code complexity helps keep software systems simple and often more robust; this can help with resiliency, integration and testing, and development speed. Simpler systems can also translate into reduced resource load for systems that are typically constrained in terms of SWaP, as well as processing and memory resources. This suggests that any system the might look to promote reuse, e.g. SSM, or add security, e.g. SSSM, needs to balance these gains against their effects on code complexity when it comes to SFS.

Pairs *A2*, *A3*, and *A5* did not show a significant difference in importance averages; this means that participants attribute a similar amount of importance to code complexity for STS, SGS, and other software.

Table 4.6. *CC* 2.1, Part 1 — Code Complexity Importance Mean, Matched-pairs Test Results, 5-point Likert Scale

				95% CI					
Security Area	Mean	Std. Dev.	Std. Error	Lower	Upper	Critical t-value (2-tailed)	Sample t-value (2-tailed)	*df*	Sig. (2-tailed)
Pair A1: SFS vs. Other Software	0.474	1.194	0.137	0.201	0.747	1.9921	3.458	75	0.001
Pair A2: SGS vs. Other Software	-0.013	0.851	0.097	-0.206	0.180	1.9917	-0.134	76	0.894
Pair A3: STS vs. Other Software	0.079	0.935	0.107	-0.135	0.293	1.9921	0.736	75	0.464
Pair A4: SFS vs. STS	0.460	1.228	0.132	0.198	0.721	1.9879	3.493	86	0.001
Pair A5: STS vs. SGS	0.149	0.922	0.099	-0.047	0.346	1.9879	1.512	86	0.134

(continued on next page)

Table 4.6. *CC* 2.1, Part 1 — Code Complexity Importance Mean, Matched-pairs Test Results, 5-point Likert Scale (continued)

				95% CI					
Security Area	Mean	Std. Dev.	Std. Error	Lower	Upper	Critical t-value (2-tailed)	Sample t-value (2-tailed)	*df*	Sig. (2-tailed)
Pair A6: SFS vs. SGS	0.591	1.238	0.132	0.329	0.853	1.9876	4.479	87	0.000
Pair A7: S*S vs. Other Software	0.184	0.780	0.088	0.008	0.360	1.9913	2.080	77	0.041

Test 3

Summary: The participants perceive code complexity to be less important than reuse and equivalent to security for both space systems software and other software fields.

Question: Question 2.1 asks each participant to rate the importance of code complexity, reusability, and security for SFS, SGS, STS, and other software fields on a 5-point Likert scale.

Analysis: Matched-pairs t-test to compare means of various combinations of code complexity, reusability, and security.

Hypothesis:

H_0: μSoftwareAspect = μOtherSoftwareAspect

H_A: μSoftwareAspect $\neq$ μOtherSoftwareAspect

Table 4.7 shows that the mean for code complexity is noticeably less important than it is for reuse in space systems software and other software while roughly equivalent for security importance.

Table 4.7. *CC* 2.1 — Code Complexity vs. Reuse vs. Security Importance Mean, Matched-pairs Test Statistics, 5-point Likert Scale

	Area	Mean	N	Std. Deviation	Std. Error Mean
Pair D1	S*S Reuse	3.86	93	0.733	0.076
	S*S Code Complexity	3.61	93	0.746	0.077
Pair D2	S*S Reuse	3.86	93	0.733	0.076
	S*S Security	3.73	93	0.864	0.090
Pair D3	S*S Code Complexity	3.61	93	0.746	0.077
	S*S Security	3.73	93	0.864	0.090
Pair D4	Other Software Reuse	3.75	79	0.967	0.109
	Other Software Code Complexity	3.44	79	1.010	0.114
Pair D5	Other Software Reuse	3.73	79	0.970	0.109
	Other Software Security	3.49	79	1.061	0.119
Pair D6	Other Software Code Complexity	3.44	78	1.014	0.115
	Other Software Security	3.50	78	1.066	0.121

Table 4.8 shows, in the case of *Pair D1* and *Pair D4*, that reuse is generally seen as more important than code complexity for space systems software and for other software. This suggests that while code complexity needs to be strongly considered for SFS it needs to be balanced against the overall importance of reuse. This suggests a benefit from open-network software development frameworks that can promote reuse and security while abstracting or minimizing code complexity for the end developer. Finally, the *D** pairs that relate to security show very weak evidence against the null meaning that the importance of security in relation to reuse and complexity is equal with regard to space systems software and other software.

Table 4.8. *CC* 2.1 — Code Complexity vs. Reuse vs. Security Importance Mean, Matched-pairs Test Results, 5-point Likert Scale

				95% CI					
Security Area	Mean	Std. Dev.	Std. Error	Lower	Upper	Critical t-value (2-tailed)	Sample t-value (2-tailed)	*df*	Sig. (2-tailed)
Pair D1	0.256	1.015	0.105	0.047	0.465	1.9861	2.434	92	0.017
Pair D2	0.129	1.027	0.107	-0.083	0.342	1.9861	1.211	92	0.229
Pair D3	-0.127	1.110	0.115	-0.356	0.101	1.9861	-1.106	92	0.271
Pair D4	0.304	1.202	0.135	0.035	0.573	1.9909	2.246	78	0.028
Pair D5	0.241	1.157	0.130	-0.019	0.500	1.9909	1.847	78	0.068
Pair D6	-0.064	1.231	0.139	-0.342	0.213	1.9913	-0.460	77	0.647

4.3.1.2 *CC* Survey, Question 2.2 Analysis, Part 1

This section shows that participants see code complexity as having an "average" ranking using a one-sample t-test. This test shows that the participants do not perceive minimal code complexity to be the most important but it is still very relevant.

Summary: Participants perceive code complexity to have a mean that is very close to 4 or an "average" ranking.

Question: *CC* Survey, Question 2.2 asks each participant to rank the provided software characteristics from 1 to 7, 1 being the highest ranking.

Analysis: One-sample t-test with test value of 4, which represents a middle-of-the-pack ranking.

Hypothesis: Participants will perceive code complexity to have a non-average ranking when compared against the 7 other characteristics.

H_0: μCharacteristicRanking $= 4$

H_A: μCharacteristicRanking $\neq 4$

Table 4.9 shows that minimal code complexity has a mean very close to 4 or the average value for the ranking scale; this does not mean that code complexity is unimportant, in fact participants found complexity to be important as indicated by analysis already presented in this section. Participants just found some of the other aspects to be more important overall, e.g. the $D*$ pairs from Section 4.3.1.1 indicate that reusability is more important.

Table 4.9. *CC* 2.2, Part 1 — Software Characteristic Importance Ranking Means, One-sample Test Statistics, Test Value 4, 1 to 7 Ranking Scale

Aspect	N	Mean	Std. Deviation	Std. Error Mean
Reuse	96	2.83	1.560	0.159
Portability	96	4.38	1.564	0.160
Interoperability	96	3.93	1.773	0.181
Minimal Code Complexity	96	3.97	2.250	0.230
Rapid Development	96	4.23	2.034	0.208
Cost of Ownership	96	4.81	1.860	0.190
Security	96	3.85	2.312	0.236

The ranking of code complexity indicates that it should be considered, but that development aspects like reusability should be given more weight when developing software for space systems. This follows the trend towards MONAs and OSAs that promote reuse. This trend is resulting in a increased need to protect these systems as shown in Section 4.6.3.2. Section 4.6.3.2 that shows security is more important for open-network space systems. This indicates there is a benefit from open-network software development frameworks that can promote reuse and security, frameworks like SSSM.

Table 4.10. *CC* 2.2, Part 1 — Software Characteristic Importance Ranking Means, One-sample Test Results, Test Value 4, 1 to 7 Ranking Scale

						95% CI	
Area	Critical t-value (2-tailed)	Sample t-value (2-tailed)	df	Sig. (2-tailed)	Mean Difference	Lower	Upper
Reuse	1.9852	-7.325	95	0.000	-1.167	-1.48	-0.85
Portability	1.9852	2.349	95	0.021	0.375	0.06	0.69
Interoperability	1.9852	-0.403	95	0.688	-0.073	-0.43	0.29

(continued on next page)

Table 4.10. *CC* 2.2, Part 1 — Software Characteristic Importance Ranking Means, One-sample Test Results, Test Value 4, 1 to 7 Ranking Scale (continued)

						95% CI	
Area	Critical t-value (2-tailed)	Sample t-value (2-tailed)	df	Sig. (2-tailed)	Mean Difference	Lower	Upper
Minimal Code Complexity	1.9852	-0.136	95	0.892	-0.031	-0.49	0.42
Rapid Development	1.9852	1.104	95	0.272	0.229	-0.18	0.64
Cost of Ownership	1.9852	4.280	95	0.000	0.813	0.44	1.19
Security	1.9852	-0.618	95	0.538	-0.146	-0.61	0.32

4.3.1.3 *RIPCC* Survey, Question 2.13 Analysis

This section shows that participants perceive reducing or minimizing code complexity to have an overall positive affect on the space systems aspects in question.

Summary: Participants see minimizing code complexity as beneficial overall. There is very strong evidence against the null for the *Overall* mean. This is true because there are a lot of aspects with very strong, strong, or neutral t-values and no aspects that have a significant negative t-value.

Question: *RIPCC* Survey, Question 2.13 asks each participant to record the impact of minimizing code complexity on a range of system aspects in terms of Pros, Neutral, and Cons. Pros is coded as a 3, Neutral is coded as a 2, and Cons is coded as a 1.

Analysis: One-sample t-test with test value of 2, which represents a neutral value.

Hypothesis: Participants will perceive code complexity to have a non-neutral affect on space system aspects.

H_0: μCodeComplexityImpact $= 2$

H_A: μCodeComplexityImpact $\neq 2$

Table 4.11 shows that most of the aspects have a mean benefit rating at or above 2.

Table 4.11. *RIPCC* 2.13 — Minimal Code Complexity Benefits on System-aspects Means, One-sample Test Statistics, Test Value 2, 1 to 3 Cons-Neutral-Pros Scale

Aspect	N	Mean	Std. Deviation	Std. Error Mean
Regression reduction	21	2.8095	0.51177	0.11168
Code design	21	2.5238	0.81358	0.17754
Development cost	21	2.4286	0.81064	0.17690
Maintenance cost	21	2.6190	0.66904	0.14600
Development productivity	21	2.4286	0.81064	0.17690
Development efficiency	21	2.2381	0.83095	0.18133
Code complexity	21	2.8571	0.47809	0.10433
Maintainability	21	2.6190	0.66904	0.14600
Integration	21	2.4762	0.60159	0.13128
Adaptability	21	2.1429	0.79282	0.17301
Documentation/Examples	21	2.3810	0.58959	0.12866
Encapsulation	21	2.2857	0.56061	0.12234
Bug detection	21	2.8571	0.35857	0.07825
Code quality	21	2.6190	0.66904	0.14600
Code robustness	21	2.5238	0.60159	0.13128
Best practices	21	2.5714	0.50709	0.11066
Schedule	21	2.3333	0.73030	0.15936
Code or algorithm optimization/efficiency	21	2.0476	0.86465	0.18868
Uniformity of coding style	21	2.2857	0.78376	0.17103
Domain knowledge	21	2.0952	0.53896	0.11761
Code readability	21	2.6667	0.57735	0.12599
Security	21	2.0952	0.62488	0.13636
I/0 efficiency	21	1.9524	0.66904	0.14600
Radiation hardness	21	2.0000	0.44721	0.09759
Fault tolerance	21	1.7619	0.62488	0.13636
Hardware complexity	21	2.1429	0.47809	0.10433
Latency	21	1.9048	0.70034	0.15283
Determinism	21	2.1905	0.60159	0.13128
Interoperability	21	2.0476	0.74001	0.16148

(continued on next page)

Table 4.11. *RIPCC* 2.13 — Minimal Code Complexity Benefits on System-aspects Means, One-sample Test Statistics, Test Value 2, 1 to 3 Cons-Neutral-Pros Scale (continued)

Aspect	N	Mean	Std. Deviation	Std. Error Mean
Portability	21	2.0476	0.80475	0.17561
Testing	21	2.5238	0.67964	0.14831
Reusability	21	2.1905	0.74960	0.16358
Software upgradability	21	2.2381	0.70034	0.15283
Hardware changes/flexibility	21	2.0476	0.66904	0.14600
Adoption rates/software proliferation	21	2.2857	0.64365	0.14046
Ease of use	21	2.5238	0.60159	0.13128
Mission/Project requirement changes	21	2.2381	0.76842	0.16768
Information Assurance	21	2.1905	0.51177	0.11168
Mission Assurance	21	2.1905	0.60159	0.13128
Overall	21	2.3175	0.32295	0.07047

Table 4.12 shows that 18 of the 39 aspects are positively affected by minimizing code complexity Table 4.12 shows 21 aspects that participants did not think code complexity affected. Finally, Table 4.12 shows that there are no aspects that are negatively affected. The results also show that the *Overall* mean affect had a very strong positive t-value, meaning that overall minimizing code complexity was seen as beneficial.

Table 4.12. *RIPCC* 2.13 — Code Complexity Benefits on System-aspects Means, One-sample Test Results, Test Value 2, 1 to 3 Cons-Neutral-Pros Scale

						95% CI	
Aspect	Critical t-value (2-tailed)	Sample t-value (2-tailed)	df	Sig. (2-tailed)	Mean Difference	Lower	Upper
Regression reduction	2.0860	7.249	20	0.000	0.80952	0.5766	1.0425
Code design	2.0860	2.950	20	0.008	0.52381	0.1535	0.8941

(continued on next page)

Table 4.12. *RIPCC* 2.13 — Code Complexity Benefits on System-aspects Means, One-sample Test Results, Test Value 2, 1 to 3 Cons-Neutral-Pros Scale (continued)

						95% CI	
Aspect	Critical *t*-value (2-tailed)	Sample *t*-value (2-tailed)	df	Sig. (2-tailed)	Mean Difference	Lower	Upper
Development cost	2.0860	2.423	20	0.025	0.42857	0.0596	0.7976
Maintenance cost	2.0860	4.240	20	0.000	0.61905	0.3145	0.9236
Development productivity	2.0860	2.423	20	0.025	0.42857	0.0596	0.7976
Development efficiency	2.0860	1.313	20	0.204	0.23810	-0.1401	0.6163
Code complexity	2.0860	8.216	20	0.000	0.85714	0.6395	1.0748
Maintainability	2.0860	4.240	20	0.000	0.61905	0.3145	0.9236
Integration	2.0860	3.627	20	0.002	0.47619	0.2024	0.7500
Adaptability	2.0860	0.826	20	0.419	0.14286	-0.2180	0.5037
Documentation/ Examples	2.0860	2.961	20	0.008	0.38095	0.1126	0.6493
Encapsulation	2.0860	2.335	20	0.030	0.28571	0.0305	0.5409
Bug detection	2.0860	10.954	20	0.000	0.85714	0.6939	1.0204
Code quality	2.0860	4.240	20	0.000	0.61905	0.3145	0.9236
Code robustness	2.0860	3.990	20	0.001	0.52381	0.2500	0.7976
Best practices	2.0860	5.164	20	0.000	0.57143	0.3406	0.8023
Schedule	2.0860	2.092	20	0.049	0.33333	0.0009	0.6658
Code or algorithm optimization/ efficiency	2.0860	0.252	20	0.803	0.04762	-0.3460	0.4412
Uniformity of coding style	2.0860	1.671	20	0.110	0.28571	-0.0711	0.6425
Domain knowledge	2.0860	0.810	20	0.428	0.09524	-0.1501	0.3406
Code readability	2.0860	5.292	20	0.000	0.66667	0.4039	0.9295
Security	2.0860	0.698	20	0.493	0.09524	-0.1892	0.3797
I/0 efficiency	2.0860	-0.326	20	0.748	-0.04762	-0.3522	0.2569

(continued on next page)

Table 4.12. *RIPCC* 2.13 — Code Complexity Benefits on System-aspects Means, One-sample Test Results, Test Value 2, 1 to 3 Cons-Neutral-Pros Scale (continued)

						95% CI	
Aspect	Critical *t*-value (2-tailed)	Sample *t*-value (2-tailed)	df	Sig. (2-tailed)	Mean Difference	Lower	Upper
Radiation hardness	2.0860	0.000	20	1.000	0.00000	-0.2036	0.2036
Fault tolerance	2.0860	-1.746	20	0.096	-0.23810	-0.5225	0.0463
Hardware complexity	2.0860	1.369	20	0.186	0.14286	-0.0748	0.3605
Latency	2.0860	-0.623	20	0.540	-0.09524	-0.4140	0.2236
Determinism	2.0860	1.451	20	0.162	0.19048	-0.0834	0.4643
Interoperability	2.0860	0.295	20	0.771	0.04762	-0.2892	0.3845
Portability	2.0860	0.271	20	0.789	0.04762	-0.3187	0.4139
Testing	2.0860	3.532	20	0.002	0.52381	0.2144	0.8332
Reusability	2.0860	1.164	20	0.258	0.19048	-0.1507	0.5317
Software upgradability	2.0860	1.558	20	0.135	0.23810	-0.0807	0.5569
Hardware changes/ flexibility	2.0860	0.326	20	0.748	0.04762	-0.2569	0.3522
Adoption rates/software proliferation	2.0860	2.034	20	0.055	0.28571	-0.0073	0.5787
Ease of use	2.0860	3.990	20	0.001	0.52381	0.2500	0.7976
Mission/Project requirement changes	2.0860	1.420	20	0.171	0.23810	-0.1117	0.5879
Information Assurance	2.0860	1.706	20	0.104	0.19048	-0.0425	0.4234
Mission Assurance	2.0860	1.451	20	0.162	0.19048	-0.0834	0.4643
Overall	2.0860	4.505	20	0.000	0.31747	0.1705	0.4645

Minimizing code complexity was largely seen as beneficial to aspects that related to the usability of the code, i.e. code readability, ease of use, best practices, code robustness, code quality, bug detection, encapsulation, documentation/examples, integration, maintainability, development productivity, and code design. These are all aspects that, in theory, make the software easier to use. If the software is easier to use then it translates into some of the other benefits the participants identified, i.e. schedule, regression reduction, development cost, and testing. It follows that the perceived benefits of code complexity address a lot of the issue facing software-systems development for space systems, and designers should consider code complexity during design and implementation of software for space systems.

This does have to be balanced against other aspects that help reduce cost and schedule like reusability and adaptability that can drive code complexity. Software development frameworks like SSM that may be internally complex help to limit the complexity for an end user by abstracting the complexity and allowing reuse and adaptability.

4.3.2 Code Complexity Metrics

There is a consensus among *RIPCC* Survey participants on the more important ways to measure code complexity for software developed for space systems. The previous section, Section 4.3.1, discussed developer perception of the importance and benefit of minimal code complexity, and it follows that the development community would want to agree on ways to measure code complexity. The participants did not seem to agree on one particular code complexity metric as being the most important, but coalesced around two metrics that would be the most important in determining a piece of software's code complexity. This by itself is useful to the space systems software development community because it suggests a couple of metrics that are more important for measuring code complexity.

Participants in the *Security* Survey believe that internal security provisions have a negative impact on code complexity, as detailed in Section 4.6.5. Analysis of code complexity in SSM versus SSSM will be left for future work and may show SSSM as a way to add internal security provisions without a significantly increasing code complexity.

The front running code complexity metrics where cyclomatic complexity and lack of cohesion. They would therefor be the most important metrics to target or measure software against when attempting to minimize code complexity.

4.3.2.1 *RIPCC* Survey, Question 2.15 Analysis

This section shows that participants perceive cyclomatic complexity and lack of cohesion to be the only code complexity metrics with positive non-neutral importance for measuring code complexity.

Summary: Participants perceive cyclomatic complexity and lack of cohesion to have above average importance; the participants felt that all of the other metrics have neutral or average importance.

Question: Question 2.15 asks the participants to rate the importance of each code complexity metric on a 5-point Likert scale.

Analysis: One-sample *t*-test with test value of 3, which represents a neutral value.

Hypothesis: Participants will perceive the different code complexity metrics to have a non-neutral importance.

H_0: μCodeComplexityMetricImportance $= 3$

H_A: μCodeComplexityMetricImportance $\neq 3$

Table 4.13 shows that most of the code complexity metrics have a mean importance rating at or slightly above 3.

Table 4.13. *RIPCC* 2.15 — Code Complexity Importance Means, One-sample Test Statistics, Test Value 3, 5-point Likert Scale Scale

Aspect	N	Mean	Std. Deviation	Std. Error Mean
Cyclomatic complexity (McCabe Metric)	27	3.59	1.366	0.263
Depth of inheritance	27	3.15	1.064	0.205
Class coupling	26	3.04	0.958	0.188
Methods per class	27	2.89	0.892	0.172
Lack of cohesion	27	3.67	1.359	0.261
Data complexity (Chapin Metric)	27	3.33	1.000	0.192
Data flow complexity (Elshof Metric)	27	3.41	1.047	0.202

(continued on next page)

Table 4.13. *RIPCC* 2.15 — Code Complexity Importance Means, One-sample Test Statistics, Test Value 3, 5-point Likert Scale (continued)

Aspect	N	Mean	Std. Deviation	Std. Error Mean
Decisional complexity (Mcclure Metric)	27	3.44	1.251	0.241
Language complexity (Haltsead Metric)	27	3.07	1.269	0.244
Interface Complexity (Henry fan-in/fan-out Metric)	27	3.33	1.301	0.250
Lines of code	27	3.04	1.224	0.236
Other	13	2.54	1.664	0.462
Overall	27	3.24	0.723	0.139

Table 4.14 shows that 2 of the 12 metrics, including *Other*, had means on the important side of the scale. However, only lack of cohesion and cyclomatic complexity have strong evidence against the null. This means that lack of cohesion and cyclomatic complexity are the only two metrics that have non-neutral or important means. Lack of cohesion is when a class represents more than one abstraction and cyclomatic complexity deal with the number of linearly independent paths through a program.

Table 4.14 shows that the remaining metrics had neutral means. The results also show that *Overall* mean importance had a neutral mean.

Other portions of this analysis show that participants believe that minimizing complexity is generally important for space systems software and other software; and that it is specifically more important for SFS. Putting these points together it follows that the participants perceive minimizing code complexity to be important and so achieving good lack of cohesion and cyclomatic complexity scores would be the most important metrics to manage, particularly in the case of SFS where minimizing code complexity was shown to be the most important.

Table 4.14. *RIPCC* 2.15 — Code Complexity Importance Means, One-sample Test Results, Test Value 3, 5-point Likert Scale

						95% CI	
Area	Critical t-value (2-tailed)	Sample t-value (2-tailed)	df	Sig. (2-tailed)	Mean Difference	Lower	Upper
Cyclomatic complexity (McCabe Metric)	2.0555	2.254	26	0.033	0.593	0.05	1.13
Depth of inheritance	2.0555	0.724	26	0.476	0.148	-0.27	0.57
Class coupling	2.0596	0.205	25	0.840	0.038	-0.35	0.43
Methods per class	2.0555	-0.648	26	0.523	-0.111	-0.46	0.24
Lack of cohesion	2.0555	2.550	26	0.017	0.667	0.13	1.20
Data complexity (Chapin Metric)	2.0555	1.732	26	0.095	0.333	-0.06	0.73
Data flow complexity (Elshof Metric)	2.0555	2.021	26	0.054	0.407	-0.01	0.82
Decisional complexity (Mcclure Metric)	2.0555	1.847	26	0.076	0.444	-0.05	0.94
Language complexity (Haltsead Metric)	2.0555	0.303	26	0.764	0.074	-0.43	0.58
Interface Complexity (Henry fan-in/fan-out Metric)	2.0555	1.331	26	0.195	0.333	-0.18	0.85
Lines of code	2.0555	0.157	26	0.876	0.037	-0.45	0.52
Other	2.1788	-1.000	12	0.337	-0.462	-1.47	0.54
Overall	2.0555	1.692	26	0.103	-0.235	-0.05	0.52

4.4 Participant Perception of Reusable Software

This section presents the findings related to developing reusable software for space systems. These findings stem from analysis of answers to questions in the *CC* and *RIPCC* surveys. This analysis shows that participants view reuse as important and beneficial for software developed for space systems.

Developer perception of the importance and benefits of software reuse relates to benefits and importance of SSM, a software development framework designed to enable and promote reuse. Other portions of the survey series analysis show that developers perceive that open-network systems, such as SSM, are beneficial and important and ultimately the direction space systems development is heading, but that these same systems have a stronger need for security. This gives strength to the idea that adding security to a software development framework, like SSM, which promotes reuse and leverages OSA and MONA concepts, in the from of SSSM, is a valuable contribution to the space systems development community.

4.4.1 Reusability Importance and Benefit

There is a consensus among the space systems developers that participated in the *CC* Survey that reusability is generally beneficial to space systems and space systems development. This section analyzes the responses to *CC* Survey, Questions 2.1 and 2.2; as they relate to the importance of reusability. This section also analyzes the responses to *RIPCC* Survey, Question 2.10. Analysis of Question 2.10 shows that participants perceive reusability to be generally important for S*S and other software, and that they found it to be the most important for SGS. The participants found reusability to be generally more important than code complexity. Section 4.4.1.1 presents the analysis of Question 2.1. Analysis of Question 2.2 shows that participants perceive reusability to have an above average importance ranking, it was the only system-characteristic measure to achieve this. Section 4.4.1.2 presents the analysis of Question 2.2. Analysis of Question 2.10 shows that participants perceive maximizing reuse to be generally beneficial, especially in terms of aspects typically thought of as relating to reuse. There are some aspects that the participants perceived to be unaffected that would normally be thought as having a symbiotic relationship with reuse. Coupling the perceived benefits of networking with a reusable software development framework allow a system like SSM to address some of the shortfalls. Section 4.4.1.3 presents the analysis of Question 2.10. These findings show the importance of open-network software development frameworks like SSM that promote reusability with the additional security provisions in SSSM to secure them.

4.4.1.1 *CC* Survey, Question 2.1 Analysis, Part 2

This section shows that the participants perceive reuse to be important for SFS, SGS, STS, and other software fields. This section also shows that participants perceive reuse to be more important for SGS than for SFS, STS, or other software. The other disciplines did not have any distinct separation from each other, this includes the difference between space systems software and other software; this suggests that reuse is generally seen as important.

This analysis covers two statistical tests. Test 1, a one-sample t-test, looks at reusability importance for software in general. Test 2, a matched-pairs t-test, compares reusability importance across different software areas.

Test 1

Summary: Participants perceive reuse to be important for SFS, SGS, STS, and other software fields.

Question: Question 2.1 asks each participant to rate the importance of reusability for SFS, SGS, STS, and other software fields on a 5-point Likert scale.

Analysis: One-sample t-test with test value of 3, which represents a neutral value.

Hypothesis: Software developers will find reusability to have non-neutral importance.

H_0: μReuseImportance $= 3$

H_A: μReuseImportance $\neq 3$

Table 4.15 shows that all the importance means are above 3. All of the categories had very strong t-values and very low p-values. This gives very strong evidence against the null, meaning reuse is generally seen as important for all of the software areas under consideration by the participants.

Table 4.15. *CC* 2.1, Part 2 — Reuse Importance Mean, One-sample Test Statistics, Test Value 3, 5-point Likert Scale

Area	N	Mean	Std. Deviation	Std. Error Mean
SFS	92	3.60	1.038	0.108
SGS	92	4.27	0.915	0.095
STS	91	3.64	0.961	0.101
Other Software	80	3.74	0.964	0.108
S*S Overall	93	3.86	0.733	0.076

The general importance of reuse explains the trend towards MONAs and OSAs that promote reuse and suggests that systems like SSM will increase in terms of relevance and adoption. This in turn increases the need to protect these systems as evidenced by the findings in Section 4.6.3.2 that say security is more important for open-network space systems.

Table 4.16. *CC* 2.1, Part 2 — Reuse Importance Mean, One-sample Test Results, Test Value 3, 5-point Likert Scale

						95% CI	
Area	Critical *t*-value (2-tailed)	Sample *t*-value (2-tailed)	df	Sig. (2-tailed)	Mean Difference	Lower	Upper
SFS	1.9864	5.522	91	0.000	0.598	0.38	0.81
SGS	1.9864	13.330	91	0.000	1.272	1.08	1.46
STS	1.9867	6.330	90	0.000	0.637	0.44	0.84
Other Software	1.9904	6.839	79	0.000	0.737	0.52	0.95
S*S Overall	1.9861	11.370	92	0.000	0.864	0.71	1.01

Test 2

Summary: Participants perceive reusability to be more important for SGS than for SFS, STS, or other software. Participants do not perceive there to be a distinction between the importance of reuse for space systems in general and other software. Participants do perceive reuse to be generally more important than code complexity.

Question: Question 2.1 asks each participant to rate the importance of reusability for SFS, SGS, STS, and other software fields on a 5-point Likert scale.

Analysis: Matched-pairs t-test to compare means of various combinations of SFS, SGS, STS, and other software fields.

Hypothesis:

H_0: μS*S $=$ μOtherSoftware

H_A: μS*S $\neq$ μOtherSoftware

or

H_0: μS*S $=$ μS*S

H_A: μS*S $\neq$ μS*S

Table 4.17 shows the various means for SFS, SGS, STS, and *other software* fields as they compare to each other. Table 4.17 shows that the mean for SGS is above all the other means. Table 4.18 shows that *Pair B2* and *Pair B6* have strong evidence against the null meaning the group perceives reuse to be more important for SGS than for *other software* and STS.

Pair B4 shows a strong negative t-value, but because of the direction of the comparison, this is actually still showing that reuse is more important for SGS than for SFS. This means that reuse is perceived by the participants to be more important for SGS, than any other software category under consideration. This suggests a strong emphasis on reuse for SGS, but Test 1 shows that it is still important for all the areas, just more important for SGS.

Table 4.17. *CC* 2.1, Part 2 — Reuse Importance Mean, Matched-pairs Test Statistics, 5-point Likert Scale

	Area	Mean	N	Std. Deviation	Std. Error Mean
Pair B1	SFS	3.65	78	1.030	0.117
	Other Software	3.74	78	0.973	0.110
Pair B2	SGS	4.33	79	0.858	0.097
	Other Software	3.75	79	0.967	0.109
Pair B3	STS	3.70	79	0.965	0.109
	Other Software	3.75	79	0.967	0.109

(continued on next page)

Table 4.17. *CC* 2.1, Part 2 — Reuse Importance Mean, Matched-pairs Test Statistics, 5-point Likert Scale (continued)

	Area	Mean	N	Std. Deviation	Std. Error Mean
Pair B4	SFS	3.61	90	1.002	0.106
	SGS	4.27	90	0.922	0.097
Pair B5	SFS	3.61	90	1.002	0.106
	STS	3.63	90	0.965	0.102
Pair B6	SGS	4.62	91	0.917	0.096
	STS	3.64	91	0.961	0.101
Pair B7	S*S	3.90	79	0.711	0.080
	Other Software	3.75	79	0.967	0.109

The finding that reuse is more important for SGS than SFS is curious because SFS can have very constrained schedules and financing, as well as an emphasis on flight heritage; one takeaway from this finding is that the need for heritage in SFS might be waining. This might be due higher risk postures in terms of mission assurance. Another thought stems from code complexity, Section 4.3.1.1 shows that participants perceived code complexity to be the most important for SFS. Both ground and flight are complex space systems and often reusability can increase code complexity, at least internally and sometimes in terms of what must be configured. Therefore, something like reusability would be less important for SFS if code complexity is more important. The other issue is the cost associated with developing a reusable system and with on-boarding new developers that must learn the system. If only one mission or satellite is being developed then the complexity and cost is not worth it, this might be another reason that reusability is less import for SFS: developers might see SFS as more of one-off software than they do SGS. Keep in mind that reuse for SFS is still seen as important, but this finding suggests a greater need for reusable SGS and potentially the ability to handle multiple missions with the same SGS.

Table 4.18. *CC* 2.1, Part 2 — Reuse Importance Mean, Matched-pairs Test Results, 5-point Likert Scale

Area	Mean	Std. Dev.	Std. Error	95% CI Lower	95% CI Upper	Critical t-value (2-tailed)	Sample t-value (2-tailed)	*df*	Sig. (2-tailed)
Pair B1: SFS vs. Other Software	-0.090	1.219	0.138	-0.365	0.185	1.9913	-0.650	77	0.517
Pair B2: SGS vs. Other Software	0.582	1.008	0.113	0.357	0.808	1.9909	5.136	78	0.000
Pair B3: STS vs. Other Software	-0.051	0.932	0.105	-0.259	0.158	1.9909	-0.483	78	0.631
Pair B4 — SFS vs. SGS	-0.656	0.950	0.100	-0.855	-0.457	1.9870	-6.546	89	0.000
Pair B5: SFS vs. STS	-0.022	1.199	0.126	-0.273	0.229	1.9870	-0.176	89	0.861
Pair B6: SGS vs. STS	0.626	1.132	0.119	0.391	0.862	1.9867	5.279	90	0.000
Pair B7: S*S vs. Other Software	0.148	0.847	0.095	-0.042	0.337	1.9909	1.550	78	0.125

Table 4.18 shows that the rest of the $B*$ pairs to not have significant separation. This means, for example, that reuse is not perceived to be more important for SFS than it is for STS or vice versa.

4.4.1.2 *CC* Survey, Question 2.2 Analysis, Part 2

This section shows that participants perceive reuse to have an above average importance ranking.

Summary: Participants perceive reuse to have an above average ranking mean, reuse has the strongest negative t-value, indicating a strong distinction between it and the average ranking of 4.

Question: *CC* Survey, Question 2.2 asks each participant to rank the provided software characteristics from 1 to 7, 1 being the highest ranking.

Analysis: One-sample t-test with test value of 4, which represents a middle-of-the-pack ranking.

Hypothesis: Participants will perceive reusability to have a non-average ranking when compared against the 7 other characteristics.

H_0: $\mu\text{CharacteristicRanking} = 4$

H_A: $\mu\text{CharacteristicRanking} \neq 4$

Table 4.19 shows the reusability measurement has a mean below the test value, on the more important side of the ranking scale. Table 4.20 shows that the reusability measurement has strong evidence against the null, meaning it has a higher than average ranking. Keep in mind that 1 is the highest ranking so a strong negative t-value means a "higher" average ranking. This means that reusability has an above average importance ranking, this coupled with the D^* pairs from Section 4.3.1.1 indicates that reusability is one of the more important development characteristics of a software development framework.

Table 4.19. *CC* 2.2, Part 2 — Software Characteristic Importance Ranking Means, One-sample Test Statistics, Test Value 4, 1 to 7 Ranking Scale

Aspect	N	Mean	Std. Deviation	Std. Error Mean
Reuse	96	2.83	1.560	0.159
Portability	96	4.38	1.564	0.160
Interoperability	96	3.93	1.773	0.181
Minimal Code Complexity	96	3.97	2.250	0.230

(continued on next page)

Table 4.19. *CC* 2.2, Part 2 — Software Characteristic Importance Ranking Means, One-sample Test Statistics, Test Value 4, 1 to 7 Ranking Scale (continued)

Aspect	N	Mean	Std. Deviation	Std. Error Mean
Rapid Development	96	4.23	2.034	0.208
Cost of Ownership	96	4.81	1.860	0.190
Security	96	3.85	2.312	0.236

This higher than average importance ranking for reuse is in line with the trend towards MONAs and OSAs that promote reuse and suggests that systems like SSM will increase in terms of relevance and adoption. This in turn increases the need to protect these systems as evidenced by the findings in Section 4.6.3.2 that say security is more important for open-network space systems. This indicates there is a benefit from open-network software development frameworks that can promote reuse and security, frameworks like SSSM.

Table 4.20. *CC* 2.2, Part 2 — Software Characteristic Importance Ranking Means, One-sample Test Results, Test Value 4, 1 to 7 Ranking Scale

						95% CI	
Area	Critical t-value (2-tailed)	Sample t-value (2-tailed)	df	Sig. (2-tailed)	Mean Difference	Lower	Upper
Reuse	1.9852	-7.325	95	0.000	-1.167	-1.48	-0.85
Portability	1.9852	2.349	95	0.021	0.375	0.06	0.69
Interoperability	1.9852	-0.403	95	0.688	-0.073	-0.43	0.29
Minimal Code Complexity	1.9852	-0.136	95	0.892	-0.031	-0.49	0.42
Rapid Development	1.9852	1.104	95	0.272	0.229	-0.18	0.64
Cost of Ownership	1.9852	4.280	95	0.000	0.813	0.44	1.19
Security	1.9852	-0.618	95	0.538	-0.146	-0.61	0.32

4.4.1.3 *RIPCC* Survey, Question 2.10 Analysis

This section shows that participants feel that maximizing reuse is generally beneficial in terms of the development process and the developed product.

> **Summary**: Participants perceive maximized reusability in space systems to be generally beneficial in terms of development process and the developed product. Only two aspects where seen as being negatively affected, namely latency and code or algorithm optimization/efficiency as shown in Table 4.22 in orange. All other aspects where either neutrally or positively affected.
>
> **Question**: *RIPCC* Survey, Question 2.10 asks each participant to record the impact of maximizing reuse on a range of system aspects in terms of Pros, Neutral, and Cons. Pros is coded as a 3, Neutral is coded as a 2, and Cons is coded as a 1.
>
> **Analysis**: One-sample t-test with test value of 2, which represents a neutral value.
>
> **Hypothesis**: Participants will perceive maximized reuse to have a non-neutral effect or impact on system-aspects.
>
> H_0: μReuseImpact $= 2$
>
> H_A: μReuseImpact $\neq 2$

Table 4.22 shows that 24 of the 39 aspects that are seen as benefiting from reusability by the participants. There are 2 aspects that are seen as being negatively affected by reusability. Finally, there were 13 aspects that the participants perceive reusability to not affect. The results also show that the *Overall* mean effect has a significant positive t-value, meaning that overall reuse is seen as beneficial.

Table 4.21. *RIPCC* 2.10 — Reusability Benefits on System-aspects Means, One-sample Test Statistics, Test Value 2, 1 to 3 Cons-Neutral-Pros Scale

Aspect	N	Mean	Std. Deviation	Std. Error Mean
Regression reduction	26	2.5769	0.70274	0.13782
Code design	26	2.7308	0.60383	0.11842
Development cost	26	2.6923	0.67937	0.13323
Maintenance cost	26	2.5769	0.64331	0.12616

(continued on next page)

Table 4.21. *RIPCC* 2.10 — Reusability Benefits on System-aspects Means, One-sample Test Statistics, Test Value 2, 1 to 3 Cons-Neutral-Pros Scale (continued)

Aspect	N	Mean	Std. Deviation	Std. Error Mean
Development productivity	26	2.6154	0.49614	0.09730
Development efficiency	26	2.3846	0.75243	0.14756
Code complexity	26	1.8077	0.80096	0.15708
Maintainability	26	2.6154	0.63730	0.12499
Integration	26	2.4615	0.58177	0.11410
Adaptability	26	2.0385	0.72004	0.14121
Documentation/Examples	26	2.5385	0.64689	0.12686
Encapsulation	26	2.4231	0.57779	0.11331
Bug detection	26	2.5000	0.50990	0.10000
Code quality	26	2.3846	0.57110	0.11200
Code robustness	26	2.5385	0.64689	0.12686
Best practices	26	2.6154	0.49614	0.09730
Schedule	26	2.3462	0.79711	0.15633
Code or algorithm optimization/efficiency	26	1.6154	0.75243	0.14756
Uniformity of coding style	26	2.4231	0.64331	0.12616
Domain knowledge	26	2.1154	0.58835	0.11538
Code readability	26	2.3077	0.54913	0.10769
Security	26	2.1923	0.69393	0.13609
I/0 efficiency	26	1.8077	0.63367	0.12427
Radiation hardness	26	2.0000	0.40000	0.07845
Fault tolerance	26	2.1538	0.67482	0.13234
Hardware complexity	26	1.8846	0.32581	0.06390
Latency	26	1.6538	0.56159	0.11014
Determinism	26	2.1154	0.51590	0.10118
Interoperability	26	2.6538	0.56159	0.11014
Portability	26	2.4615	0.64689	0.12686
Testing	26	2.7692	0.42967	0.08427
Reusability	26	2.8077	0.40192	0.07882
Software upgradability	26	2.1154	0.71144	0.13953

(continued on next page)

Table 4.21. *RIPCC* 2.10 — Reusability Benefits on System-aspects Means, One-sample Test Statistics, Test Value 2, 1 to 3 Cons-Neutral-Pros Scale (continued)

Aspect	N	Mean	Std. Deviation	Std. Error Mean
Hardware changes/flexibility	26	2.1923	0.74936	0.14696
Adoption rates/software proliferation	26	2.5385	0.64689	0.12686
Ease of use	26	2.2692	0.60383	0.11842
Mission/Project requirement changes	26	2.0385	0.72004	0.14121
Information Assurance	26	2.1923	0.56704	0.11121
Mission Assurance	26	2.3846	0.63730	0.12499
Overall	26	2.3215	0.22226	0.04359

As one might expect, aspects that are typically thought of as relating to reusability show a very strong beneficial trend, i.e. aspects that relate to recombining components or using components as building blocks that can be moved from system to system or application of the software to application of the software, are positively affected by reusability. These are aspects like development cost, maintenance cost, maintainability, integration, encapsulation, interoperability, portability, testing, adoption, and mission assurance. These are aspects that reusable software development frameworks, like SSM, try to address; it is good that perception of the reusability benefits matches that effort.

Oddly, adaptability was not perceived to be positively affected. Normally software that is not adaptable is harder to reuse, as it likely only has one specific use case. Adaptability is typically thought of as one of the precepts of reusability. It may be that participants felt that the scope of space systems software was more limited or specialized and so adaptability suffered even as developers target reuse; this might be especially true for traditional point-to-point systems where intra-space system communication might be very specific and tied to hardware. In this case reuse of the software might mean reuse of the hardware.

Table 4.22. *RIPCC* 2.10 — Reusability Benefits on System-aspects Means, One-sample Test Results, Test Value 2, 1 to 3 Cons-Neutral-Pros Scale

						95% CI	
Area	Critical *t*-value (2-tailed)	Sample *t*-value (2-tailed)	df	Sig. (2-tailed)	Mean Difference	Lower	Upper
Regression reduction	2.0596	4.186	25	0.000	0.57692	0.2931	0.8608
Code design	2.0596	6.171	25	0.000	0.73077	0.4869	0.9747
Development cost	2.0596	5.196	25	0.000	0.69231	0.4179	0.9667
Maintenance cost	2.0596	4.573	25	0.000	0.57692	0.3171	0.8368
Development productivity	2.0596	6.325	25	0.000	0.61538	0.4150	0.8158
Development efficiency	2.0596	2.606	25	0.015	0.38462	0.0807	0.6885
Code complexity	2.0596	-1.224	25	0.232	-0.19231	-0.5158	0.1312
Maintainability	2.0596	4.924	25	0.000	0.61538	0.3580	0.8728
Integration	2.0596	4.045	25	0.000	0.46154	0.2266	0.6965
Adaptability	2.0596	0.272	25	0.788	0.03846	-0.2524	0.3293
Documentation/ Examples	2.0596	4.244	25	0.000	0.53846	0.2772	0.7997
Encapsulation	2.0596	3.734	25	0.001	0.42308	0.1897	0.6565
Bug detection	2.0596	5.000	25	0.000	0.50000	0.2940	0.7060
Code quality	2.0596	3.434	25	0.002	0.38462	0.1539	0.6153
Code robustness	2.0596	4.244	25	0.000	0.53846	0.2772	0.7997
Best practices	2.0596	6.325	25	0.000	0.61538	0.4150	0.8158
Schedule	2.0596	2.214	25	0.036	0.34615	0.0242	0.6681
Code or algorithm optimization/ efficiency	2.0596	-2.606	25	0.015	-0.38462	-0.6885	-0.0807
Uniformity of coding style	2.0596	3.353	25	0.003	0.42308	0.1632	0.6829
Domain knowledge	2.0596	1.000	25	0.327	0.11538	-0.1223	0.3530

(continued on next page)

Table 4.22. *RIPCC* 2.10 — Reusability Benefits on System-aspects Means, One-sample Test Results, Test Value 2, 1 to 3 Cons-Neutral-Pros Scale (continued)

						95% CI	
Area	Critical *t*-value (2-tailed)	Sample *t*-value (2-tailed)	df	Sig. (2-tailed)	Mean Difference	Lower	Upper
Code readability	2.0596	2.857	25	0.008	0.30769	0.0859	0.5295
Security	2.0596	1.413	25	0.170	0.19231	-0.0880	0.4726
I/0 efficiency	2.0596	-1.547	25	0.134	-0.19231	-0.4483	0.0636
Radiation hardness	2.0596	0.000	25	1.000	0.00000	-0.1616	0.1616
Fault tolerance	2.0596	1.162	25	0.256	0.15385	-0.1187	0.4264
Hardware complexity	2.0596	-1.806	25	0.083	-0.11538	-0.2470	0.0162
Latency	2.0596	-3.143	25	0.004	-0.34615	-0.5730	-0.1193
Determinism	2.0596	1.140	25	0.265	0.11538	-0.0930	0.3238
Interoperability	2.0596	5.937	25	0.000	0.65385	0.4270	0.8807
Portability	2.0596	3.638	25	0.001	0.46154	0.2003	0.7228
Testing	2.0596	9.129	25	0.000	0.76923	0.5957	0.9428
Reusability	2.0596	10.247	25	0.000	0.80769	0.6454	0.9700
Software upgradability	2.0596	0.827	25	0.416	0.11538	-0.1720	0.4027
Hardware changes/ flexibility	2.0596	1.309	25	0.203	0.19231	-0.1104	0.4950
Adoption rates/software proliferation	2.0596	4.244	25	0.000	0.53846	0.2772	0.7997
Ease of use	2.0596	2.273	25	0.032	0.26923	0.0253	0.5131
Mission/Project requirement changes	2.0596	0.272	25	0.788	0.03846	-0.2524	0.3293
Information Assurance	2.0596	1.729	25	0.096	0.19231	-0.0367	0.4213
Mission Assurance	2.0596	3.077	25	0.005	0.38462	0.1272	0.6420

(continued on next page)

Table 4.22. *RIPCC* 2.10 — Reusability Benefits on System-aspects Means, One-sample Test Results, Test Value 2, 1 to 3 Cons-Neutral-Pros Scale (continued)

						95% CI	
Area	Critical *t*-value (2-tailed)	Sample *t*-value (2-tailed)	df	Sig. (2-tailed)	Mean Difference	Lower	Upper
Overall	2.0596	7.376	25	0.000	0.32150	0.2317	0.4113

Section 4.5.1.1 shows that networking is seen as having a very positive affect on adaptability and coupling networking and reusability addresses these developer perceptions. An open-network software development framework, like SSM, is directly in line with this concept. Progressing open-network software development frameworks that enable or promote reusability, like SSM, allows more space systems to realize the reusability benefits perceived by the participants while addressing adaptability; adding security in the from of SSSM protects the open-network space systems where security it perceived to be more important.

4.5 Participant Perception of Modular Open-network System Approaches

This section presents findings related to modular open-network system approaches. These findings stem from analysis of the results of questions in the *Network* and *OSAM* surveys. These findings are presented in terms of the perception among participating space systems developers with regard to the benefit of networking systems and the trend towards OSAs and MONAs. There is a general consensus among participants that networking is beneficial to space systems and space systems development as covered in Section 4.5.1. Section 4.5.2 covers the perception among the participating developers that space systems design and implementation is trending towards OSAs and MONAs at an organizational level.

Developer perception of the benefits of networking relates to the benefits of SSM as a software development framework that leverages networking for interprocess and inter-component communication. Further, the participating developers perception of a network and open systems centric implementation relates to SSM for the same reasoning. Other portions of the survey series analysis show that developers perceive security to be more important and more difficult for open-network systems, such as SSM. This gives strength to the idea that adding security to a open-network software development framework, like SSM, which leverages OSA and MONA concepts, in the from of SSSM, is a valuable contribution to the space systems development community.

4.5.1 Networking Benefits

There is a consensus among the space systems developers that participated in the *Network* Survey that networking is generally beneficial to space systems and space systems development. This section analyzes the responses to *Network* Survey, Questions 2.3. Analysis of Question 2.3 shows that participants perceive open-network systems to be largely beneficial. Section 4.5.1.1 presents the analysis of Question 2.3. Additional analysis as it relates directly to security, as one of the few negatively impacted aspects is covered in Section 4.6.2.2. These perceptions of networking's positive affect on systems-aspects show that there is a need for open-network software development frameworks that allow developers to realize all the benefits of networking. This coupled with the findings in Section 4.6.2.2 show the importance of addressing security for open-network software development frameworks like SSSM does for SSM.

4.5.1.1 *Network* Survey, Question 2.3 Analysis, Part 1

This section shows that participants perceive the effects of open-network space systems to be largely beneficial to the various aspects of that system. *Network* Survey, Question 2.3 is a Pros-neutral-cons rating question.

Summary: Participants perceive open-network space systems to have a net-positive impact on the aspects of that system. Only three aspects where seen as being strongly negatively affected, namely security, latency, and determinism as shown in Table 4.24 in red.

Question: *Network* Survey, Question 2.3 asks each participant to record the impact that a open-network space system has on a range of system aspects in terms of Pros, Neutral, and Cons. Pros is coded as a 3, Neutral is coded as a 2, and Cons is coded as a 1.

Analysis: One-sample t-test with test value of 2, which represents a neutral value.

Hypothesis: Participants will perceive networking to have a non-neutral effect or impact on system-aspects.

H_0: μNetworkImpact $= 2$

H_A: μNetworkImpact $\neq 2$

Table 4.24 shows that 19 of the 39 are positively affected. Table 4.24 shows that 17 aspects are neutrally affected, and that only 3 aspects are negatively affected. The results also show that *Overall* effect was positive.

Table 4.23. *Network* 2.3 — Network Benefits on System-aspects Means, One-sample Test Statistics, Test Value 2, 1 to 3 Cons-Neutral-Pros Scale

Aspect	N	Mean	Std. Deviation	Std. Error Mean
Regression reduction	29	1.8621	0.63943	0.11874
Code design	29	2.2069	0.81851	0.15199
Development cost	29	2.1724	0.80485	0.14946
Maintenance cost	29	2.4483	0.73612	0.13669
Development productivity	29	2.4483	0.57235	0.10628
Development efficiency	29	2.4138	0.62776	0.11657
Code complexity	29	1.7241	0.79716	0.14803
Maintainability	29	2.2759	0.75103	0.13946
Integration	29	2.5862	0.68229	0.12670
Adaptability	29	2.8621	0.44111	0.08191
Documentation/Examples	29	2.1724	0.53911	0.10011
Encapsulation	29	2.3793	0.56149	0.10427
Bug detection	29	2.0000	0.80178	0.14889
Code quality	29	2.2069	0.49130	0.09123
Code robustness	29	2.2759	0.52757	0.09797
Best practices	29	2.3448	0.48373	0.08983

(continued on next page)

Table 4.23. *Network* 2.3 — Network Benefits on System-aspects Means, One-sample Test Statistics, Test Value 2, 1 to 3 Cons-Neutral-Pros Scale (continued)

Aspect	N	Mean	Std. Deviation	Std. Error Mean
Schedule	29	2.0345	0.62580	0.11621
Code or algorithm optimization/efficiency	29	2.0345	0.73108	0.13576
Uniformity of coding style	29	2.1034	0.48879	0.09077
Domain knowledge	29	2.2759	0.64899	0.12051
Code readability	29	2.1034	0.40925	0.07600
Security	29	1.4828	0.63362	0.11766
I/0 efficiency	29	1.8276	0.80485	0.14946
Radiation hardness	29	2.1034	0.30993	0.05755
Fault tolerance	28	2.2500	0.70053	0.13239
Hardware complexity	29	1.7241	0.75103	0.13946
Latency	29	1.2759	0.52757	0.09797
Determinism	28	1.3929	0.56695	0.10714
Interoperability	29	2.7586	0.51096	0.09488
Portability	29	2.5862	0.68229	0.12670
Testing	29	2.2759	0.88223	0.16383
Reusability	29	2.6897	0.54139	0.10053
Software upgradability	29	2.4483	0.63168	0.11730
Hardware changes/flexibility	29	2.5862	0.68229	0.12670
Adoption rates/software proliferation	28	2.2500	0.51819	0.09793
Ease of use	29	2.2759	0.70186	0.13033
Mission/Project requirement changes	28	2.3929	0.73733	0.13934
Information Assurance	29	1.8621	0.69303	0.12869
Mission Assurance	28	2.2500	0.51819	0.09793
Overall	29	2.1899	0.22063	0.04097

Aspects that relate to reusability show a very strong beneficial trend, i.e. aspects that relate to recombining components or using components as building blocks are positively affected by networking. These aspects, largely shown in blue in Table 4.24 all had strong positive t-values above 4, and low p-values. Adaptability was the aspect perceived to benefit the most from network-approaches to system design and implementation. The ability to adapt during development as well as between different development efforts allows for more agile development as well as shared effort between programs or missions. This can help to reduce cost and reduce schedule.

Table 4.24. *Network* 2.3 — Network Benefits on System-aspects Means, One-sample Test Results, Test Value 2, 1 to 3 Cons-Neutral-Pros Scale

						95% CI	
Area	Critical t-value (2-tailed)	Sample t-value (2-tailed)	df	Sig. (2-tailed)	Mean Difference	Lower	Upper
Regression reduction	2.0484	-1.162	28	0.255	-0.13793	-0.3812	0.1053
Code design	2.0484	1.361	28	0.184	0.20690	-0.1044	0.5182
Development cost	2.0484	1.154	28	0.258	0.17241	-0.1337	0.4786
Maintenance cost	2.0484	3.279	28	0.003	0.44828	0.1683	0.7283
Development productivity	2.0484	4.218	28	0.000	0.44828	0.2306	0.6660
Development efficiency	2.0484	3.550	28	0.001	0.41379	0.1750	0.6526
Code complexity	2.0484	-1.864	28	0.073	-0.27586	-0.5791	0.0274
Maintainability	2.0484	1.978	28	0.058	0.27586	-0.0098	0.5615
Integration	2.0484	4.627	28	0.000	0.58621	0.3267	0.8457
Adaptability	2.0484	10.524	28	0.000	0.86207	0.6943	1.0299
Documentation/ Examples	2.0484	1.722	28	0.096	0.17241	-0.0327	0.3775
Encapsulation	2.0484	3.638	28	0.001	0.37931	0.1657	0.5929
Bug detection	2.0484	0.000	28	1.000	0.00000	-0.3050	0.3050
Code quality	2.0484	2.268	28	0.031	0.20690	0.0200	0.3938
Code robustness	2.0484	2.816	28	0.009	0.27586	0.0752	0.4765
Best practices	2.0484	3.839	28	0.001	0.34483	0.1608	0.5288
Schedule	2.0484	0.297	28	0.769	0.03448	-0.2036	0.2725

(continued on next page)

Table 4.24. *Network* 2.3 — Network Benefits on System-aspects Means, One-sample Test Results, Test Value 2, 1 to 3 Cons-Neutral-Pros Scale (continued)

Area	Critical t-value (2-tailed)	Sample t-value (2-tailed)	df	Sig. (2-tailed)	Mean Difference	95% CI	
						Lower	Upper
Code or algorithm optimization/efficiency	2.0484	0.254	28	0.801	0.03448	-0.2436	0.3126
Uniformity of coding style	2.0484	1.140	28	0.264	0.10345	-0.0825	0.2894
Domain knowledge	2.0484	2.289	28	0.030	0.27586	0.0290	0.5227
Code readability	2.0484	1.361	28	0.184	0.10345	-0.0522	0.2591
Security	2.0484	-4.396	28	0.000	-0.51724	-0.7583	-0.2762
I/0 efficiency	2.0484	-1.154	28	0.258	-0.17241	-0.4786	0.1337
Radiation hardness	2.0484	1.797	28	0.083	0.10345	-0.0144	0.2213
Fault tolerance	2.0518	1.888	27	0.070	0.25000	-0.0216	0.5216
Hardware complexity	2.0484	-1.978	28	0.058	-0.27586	-0.5615	0.0098
Latency	2.0484	-7.392	28	0.000	-0.72414	-0.9248	-0.5235
Determinism	2.0518	-5.667	27	0.000	-0.60714	-0.8270	-0.3873
Interoperability	2.0484	7.995	28	0.000	0.75862	0.5643	0.9530
Portability	2.0484	4.627	28	0.000	0.58621	0.3267	0.8457
Testing	2.0484	1.684	28	0.103	0.27586	-0.0597	0.6114
Reusability	2.0484	6.860	28	0.000	0.68966	0.4837	0.8956
Software upgradability	2.0484	3.822	28	0.001	0.44828	0.2080	0.6886
Hardware changes/flexibility	2.0484	4.627	28	0.000	0.58621	0.3267	0.8457
Adoption rates/software proliferation	2.0518	2.553	27	0.017	0.25000	0.0491	0.4509

(continued on next page)

Table 4.24. *Network* 2.3 — Network Benefits on System-aspects Means, One-sample Test Results, Test Value 2, 1 to 3 Cons-Neutral-Pros Scale (continued)

						95% CI	
Area	Critical t-value (2-tailed)	Sample t-value (2-tailed)	df	Sig. (2-tailed)	Mean Difference	Lower	Upper
Ease of use	2.0484	2.117	28	0.043	0.27586	0.0089	0.5428
Mission/Project requirement changes	2.0518	2.819	27	0.009	0.39286	0.1070	0.6788
Information Assurance	2.0484	-1.072	28	0.293	-0.13793	-0.4015	0.1257
Mission Assurance	2.0518	2.553	27	0.017	0.25000	0.0491	0.4509
Overall	2.0484	4.635	28	0.000	0.18989	0.1060	0.2768

Most aspects were seen as benefiting or not being impacted by open-network space systems. This is shown by the strong positive t-value for the *Overall* mean for all the aspects together. The participants only saw three aspects as being negatively affected: security, latency, and determinism. This research looks to address the security aspect so that this system type, that they generally perceive to beneficial and is specifically suited to reduce cost can be utilized even when security is a significant concern. Aspects like latency and determinism also need to be addressed in order for open-network space system to become a universally applicable solution as there do exist applications where timing is so critical that latency or determinism can rule out a open-network solution. That being said, a solution like SSSM addresses one of the major perceived pitfalls of open-network space system.

4.5.2 OSA and MONA Trends

This section analyzes the responses to *OSAM* Survey, Questions 2.4 and 2.5. Analysis of Question 2.4 shows that participants perceive there to be strong organization-level usage of MONAs and OSAs, and that this usage is increasing. Section 4.5.2.1 presents the analysis of Question 2.4. Analysis of Question 2.5 shows that the participants perceive that a majority of systems are closed proprietary systems even though Section 4.5.2.1 shows very strong organization-level use of OSA and MONA. The high organization-level use likely stems from the perceived benefits of using networking in systems development, Section 4.5.1 describes this. The disparity between system-level and organization-level usage might stem from missions or programs where security is a strong concern and the perceived negative impact of networking on security comes into play, Section 4.6.2.2 covers this perception of the negative impact of networking on security. Ideally organizations would have a more unified approach that would help them realize the positive effects of networking and reuse, a secured open-network software development framework, like SSSM, would help realize that end by alleviating the security concern.

4.5.2.1 *OSAM* Survey, Question 2.4 Analysis

Test 1 shows that participants perceive OSAs and MONAs to have have strong organization-level usage in the past, currently, and going into the future. This is different than the mission/program-level usage that is analyzed in Section 4.5.2.2 where usage was not perceived to be as strong. Test 2 shows that participants perceive OSA and MONA proliferation stayed pretty steady from past to present, and that they see a rise in organization-level usage of OSA and MONA going into the future.

Test 1, a one-sample t-test, looks at the organization-level usage of OSAs and MONAs. Test 2, a matched-pairs t-test, compares the past, present, and future organization-level usage of OSAs and MONAs.

Test 1

Summary: Participants perceive that more than 50% of organizations have utilized MONA and OSA in the past, are currently using them, and will continue to use them in future.

Question: *OSAM* Survey, Question 2.4 asks each participant to indicate if their organization has used MONA or OSA in the past, does use them currently, and if the organization will use them in future. The past, current, and future category selections are not exclusive and MONA and OSA are indicated separately.

Analysis: One-sample t-test with test value of 1.5, which would mean that an equal number of participants selected "Yes" as selected "No".

Hypothesis: Organizational usage of OSAs and MONAs will tend to the not used or the used side of the spectrum.

H_0: μOSA/MONAUsage $= 1.5$

H_A: μOSA/MONAUsage $\neq 1.5$

Table 4.25 shows that the mean for organization-level use of MONAs and OSAs is above the test value for past, current, and future categories. This puts the mean on the "Yes" side, or more than 50% organizational use for all categories.

Table 4.25. *OSAM* 2.4 — Organization-level Proliferation of MONAs and OSAs, One-sample Test Statistics, Test Value 1.5, 1 to 2 No-Yes Scale

Aspect	N	Mean	Std. Deviation	Std. Error Mean
Past OSA	44	1.75	0.438	0.066
Past MONA	44	1.66	0.479	0.072
Current OSA	44	1.77	0.424	0.064
Current MONA	44	1.68	0.471	0.071
Future OSA	44	1.91	0.291	0.044
Future MONA	44	1.86	0.347	0.052

Table 4.26 shows that more than 50% of the participants believe their organizations have used, are using, and will continue to use MONAs and OSAs; this suggests good organization-level usage. One of the weaknesses in the question is that it does not address what the participants think about other organizations. They might see their organization as the out-lier, however, because the trend is pretty strong it should still carry a good deal of weight, the exception being if the participants all work for the same organization which seems unlikely. The averages for past and current seem to be relatively flat while future use looks to be significantly higher. This trend will be analyzed in Test 2. The strong and

consistent perception of MONA and OSA use support the need for a open-network software development framework with security provisions, like the one this research implements.

Table 4.26. *OSAM* 2.4 — Organization-level Proliferation of MONAs and OSAs, One-sample Test Results, Test Value 1.5, 1 to 2 No-Yes Scale

Area	Critical t-value (2-tailed)	Sample t-value (2-tailed)	df	Sig. (2-tailed)	Mean Difference	95% CI Lower	Upper
Past OSA	2.0167	3.786	43	0.000	0.250	0.12	0.38
Past MONA	2.0167	2.201	43	0.033	0.159	0.01	0.30
Current OSA	2.0167	4.268	43	0.000	0.273	0.14	0.40
Current MONA	2.0167	2.560	43	0.014	0.182	0.04	0.33
Future OSA	2.0167	9.331	43	0.000	0.409	0.32	0.50
Future MONA	2.0167	6.948	43	0.000	0.364	0.26	0.47

Test 2

Summary: Participants perceive organization-level usage of MONA and OSA to have have stayed relatively flat from past to current. They perceive an increase in usage going from past to future.

Question: Same question as Test 1.

Analysis: Matched-pairs t-test to compare mean of current to past, future to current, and future to past organization-level usage of MONA and OSA.

Analysis: MONA and OSA organization-level usage will not stay the same across the past, current, and future categories.

H_0: μxUsage $= \mu$yUsage

H_A: μxUsage $\neq \mu$yUsage

Table 4.27 shows that the past and current usage is pretty consistent. The table shows that current to future usage shows a border-line increase for OSA and a significant increase for

MONA. Finally, the table shows a significant increase in mean for both going from past to future use.

Table 4.27. *OSAM* 2.4 — Organization-level Proliferation of MONAs and OSAs, Matched-pairs Test Statistics, 1 to 2 No-Yes Scale

	Network Area	Mean	N	Std. Deviation	Std. Error Mean
Pair 1	Current ORG OSA	1.77	44	0.424	0.064
	Past ORG OSA	1.75	44	0.438	0.066
Pair 2	Current ORG MONA	1.68	44	0.471	0.071
	Past ORG MONA	1.66	44	0.479	0.072
Pair 3	Future ORG OSA	1.91	44	0.291	0.044
	Current ORG OSA	1.77	44	0.424	0.064
Pair 4	Future ORG MONA	1.86	44	0.347	0.052
	Current ORG MONA	1.68	44	0.471	0.071
Pair 5	Future ORG OSA	1.91	44	0.291	0.044
	Past ORG OSA	1.75	44	0.438	0.066
Pair 6	Future ORG MONA	1.86	44	0.347	0.052
	Past ORG MONA	1.66	44	0.479	0.072

Table 4.28 shows that *Pair 1*, *Pair 2*, and *Pair 3* give weak evidence against the null hypothesis. This means that the difference in the averages is not significant. Past to current stayed statistically the same, and that usage stayed statistically the same from current to future for OSA.

Table 4.28 gives strong evidence against the null for *Pair 4*, *Pair 5*, and *Pair 6*. This all suggests that developers perceive there to be an steady increase in organization-level usage of MONAs as the results show significant positive difference between past and current, current and future, and past and future. There is also evidence of a trend towards increased usage for OSAs. OSAs did not show a significant difference between the time-adjacent categorizes, but there is a significant rise from past to future anticipated use.

MONA and OSA organization-level usage is trending upwards, the trend for MONAs is a bit stronger. This is interesting because a MONA is essentially a OSA that is network-focused suggesting that while OSAs are generally increasing the main manifestation of OSAs looks to be MONAs. This enforces an increasing need for secured open-network software development frameworks as the usage of MONAs increase, but this need becomes even stronger with the negative perception participants have of the effect of open-network space systems on security. This highlights a growing need for open-network software development frameworks like the one implemented for this research and described in Chapter 5.

Table 4.28.
acOSAM 2.4 — Organization-level Proliferation of MONAs and OSAs, Matched-pairs Test Results, 1 to 2 No-Yes Scale

				95% CI					
Network Area	Mean	Std. Dev.	Std. Error	Lower	Upper	Critical *t*-value (2-tailed)	Sample *t*-value (2-tailed)	*df*	Sig. (2-tailed)
Pair 1	0.023	0.263	0.040	-0.057	0.103	2.0167	0.573	43	0.570
Pair 2	0.023	0.340	0.051	-0.081	0.126	2.0167	0.443	43	0.660
Pair 3	0.136	0.462	0.070	-0.004	0.277	2.0167	1.957	43	0.057
Pair 4	0.182	0.446	0.067	0.046	0.317	2.0167	2.705	43	0.010
Pair 5	0.159	0.479	0.072	0.013	0.305	2.0167	2.201	43	0.033
Pair 6	0.205	0.462	0.070	0.064	0.345	2.0167	2.940	43	0.005

4.5.2.2 *OSAM* Survey, Question 2.5 Analysis

This section shows that participants perceive the current system-level usage of OSAs and MONAs to lag significantly behind the usage of closed proprietary and that OSAs and MONAs usage is perceived to be the same.

Summary: Participants perceive closed proprietary systems to far exceed the number of systems employing MONAs and OSAs. They perceive MONAs and OSAs to have a equal share of systems.

Question: *OSAM* Survey, Question 2.5 asks each participant indicated the percentage of systems they think use MONA, OSA, and closed proprietary infrastructure.

Analysis: Matched-pairs *t*-test to compare means of MONA to closed, OSA to closed, and MONA to OSA system-level usage.

Hypothesis: acMONA, OSA, and closed proprietary system-level usage will not be equal.

H_0: μSysTypeX $=$ μSysTypeY

H_A: μSysTypeX $\neq$ μSysTypeY

Table 4.29. *OSAM* 2.5 — System-level Usage of MONA, OSA, and Close Proprietary, Matched-pairs Test Statistics, Percentage Scale

	System Type	Mean	N	Std. Deviation	Std. Error Mean
Pair 1	OSA	30.4318	44	25.94663	3.91160
	Closed proprietary	64.5909	44	29.63681	4.46792
Pair 2	MONA	25.6136	44	23.46815	3.53796
	Closed proprietary	64.5909	44	29.63681	4.46792
Pair 3	MONA	25.6136	44	23.46815	3.53796
	OSA	30.4318	44	25.94663	3.91160

Table 4.30 gives very strong evidence against the null for *Pair 1* and *Pair 2*. This means there was a significant difference between closed and MONA, and closed and OSA. Suggesting that the perception is that many systems still use closed proprietary systems even though Section 4.5.2.1 shows very strong organization-level use. It is possible that this discrepancy between organizational-level and system-level usage stems from the point made in Section 4.5.2.1 about the weakness of the question, but it might also stem from system-level being a subset of organization-level usage. This organization-level versus system-level disparity might actually stem from an issue where organizations do not have a unified approach to their systems engineering and system software. This is understandable given the differences that might exists between programs or missions, however, not having a unified approach makes resource sharing and reuse difficult.

Table 4.30. *OSAM* 2.5 — System-level Usage of MONA, OSA, and Close Proprietary, Matched-pairs Test Results, Percentage Scale

				95% CI					
Networ Area	Mean	Std. Dev.	Std. Error	Lower	Upper	Critical t-value (2-tailed)	Sample t-value (2-tailed)	*df*	Sig. (2-tailed)
Pair 1	-34.159	36.320	5.475	-45.201	-23.116	2.0167	-6.238	43	0.000
Pair 2	-38.977	35.822	5.400	-49.868	-28.086	2.0167	-7.217	43	0.000
Pair 3	-4.818	17.576	2.649	-10.162	0.525	2.0167	-1.818	43	0.076

Section 4.5.1 illustrates that participants believe that open-network systems, like systems using a MONA, were seen as having an overall beneficial effect on a range of system aspects, e.g. reuse and development efficiency. Section 4.6.2.2 shows that participants perceive open-network systems to negatively impact security. Coupling those findings with the ones here suggests that a secure open-network software development framework, like the one in this research, could help organizations arrive at a more unified approach to systems development by reducing this negative effect on security and maximizing the benefits like reuse and development efficiency.

This analysis also shows that MONA and OSA usage was statistically the same, this suggests that most of the systems employing OSAs are using the MONA variant further emphasizing the need for a secure open-network software development frameworks.

4.6 Software Security

This section presents the findings relevant cybersecurity in space systems. These findings stem from analysis of answers to questions in the *CC*, *OSAM*, *Security*, and *Network* surveys. This section presents these findings in terms of the perception about security among participating space systems developers with regard to the difficulties, negative aspects, overall importance of security, and the importance of specific security provisions. These findings also cover the lack of perception among participating space systems developers that internal security is beneficial.

Section 4.6.1 shows that most of the *Security* Survey participants had some type of space systems cybersecurity experience which gives more context to their responses to questions on the *Security* Survey that dealt with security. However, they were just as likely as not to have experience in SFSYSs and STSYSs, which shows a potential need for tools and secured open-network software development frameworks for flight, test, and ground space systems can help to address the experience gap by allowing developers to develop software with a certain amount of security provisions baked into them.

Section 4.6.2 shows there is a general consensus among survey participants that security concerns are a significant hurdle to adopting open systems approaches and that networking has a negative effect on security in space systems and space systems development. SSSM adds security to an existing software MONA development framework lowering the hurdle of security in adopting open systems approaches while also addressing the perceived need for additional security for open-network space systems.

Section 4.6.3 shows that the perception among participating developers is that security is important to provide for space systems. This finding illustrates the need for a software development framework for space systems that includes security provisions. Further analysis shows consensus that security is more important for open-network space systems than it is for traditional point-to-point space systems. This finding gives importance to SSSM as an example of a more secure modular open-network software development framework. This finding also relates to the consensus that there is a trend in space systems development towards OSAs and MONAs, Section 4.5.2 covers this trend. Analysis shows that there is consensus that security it more important for SFS and SGS than it is for other software. Analysis also shows a lack of consensus on the difficulty of providing security provisions for space systems, suggesting there may be a need to better understand how these security provisions can be provided for space systems.

Section 4.6.4 shows that there is a consensus among the participating space systems developers as to the most important security provisions and features to provide for space systems and space systems development. This section shows provisions provided for by

SSSM as being perceived as some of the most important space systems security provisions.

Section 4.6.5 covers a lack of consensus among the participating space systems developers that internal security is beneficial to space systems development while showing that certain aspects of space systems were perceived as negatively affected by internal security provisions. Some of the aspects shown as being most negatively affected by internal security are code complexity, development costs, maintenance, and code design. Adding security to a open-network software development framework and demonstrating a minimal increase in code complexity allows a contribution like SSSM to address developer concerns about adding security because it will address issues like development cost via reuse and provide built-in security features.

4.6.1 Security Experience

This section presents findings related to developer security experience. These findings stem from analysis of results for a question in the *Security* Survey. These findings are presented in terms of the percentage of participating space systems developers that had security experience with SFSYSs, SGSYSs, STSYSs, penetration testing, and other.

Basic analysis also shows that only 16.13% of participants had no security experience, those that had penetration testing or other experience also had at least one other area of security experience that related to space systems of some kind. This shows that the majority of participants in the *Security* Survey claimed to have some direct experience with security. This gives strength or weight to their responses to questions on the *Security* Survey that dealt with security.

The results indicate that developers were more likely than not to have had security experience with SGSYSs. They were just as likely as not to have experience in SFSYSs and STSYSs, and much less likely to have experience with penetration testing or other areas. This shows a need for growth in security experience and/or development tools that incorporate security provisions. Having tools and secured open-network software development frameworks for flight, test, and ground space systems can help to address the experience gap by allowing developers to develop software with a certain amount of security provisions baked into them. This is part of the driver behind adding security provisions to SSM as proposed by this research.

This section analyzes the responses to *Security* Survey, Questions 2.2; as it relates to security experience. Analysis of Question 2.2 shows that participants generally had more security experience with space systems than with penetration testing or other areas, and that overall they had the most security experience with SGSYSs. Section 4.6.1.1 presents the analysis of Question 2.2.

4.6.1.1 ***Security*** **Survey, Question 2.2 Analysis**

This section shows that more than 50% of the participants had security experience with SGSYSs, about 50% had security experience with SFSYSs and STSYSs, and less than 50% had security experience with penetration testing or other.

Summary: More than 50% of the participants had security experience with SGSYSs, about 50% had security experience with SFSYSs and STSYSs, and less than 50% had security experience with with penetration testing or other.

Question: Question 2.2 asks each participant to indicate if they have direct experience with security in each of the areas listed in Table 4.31.

Analysis: One-sample t-test with test value of 1.5, which represents an equal number of "Yes" and "No" responses.

Hypothesis: The number of developers with security experience in the given area will be above or below 50%, meaning the average will be significantly above or below 1.5.

H_0: μS*SysSecurityExperience $= 1.5$

H_A: μS*SysSecurityExperience $\neq 1.5$

or

H_0: μOtherSoftwareExperience $= 1.5$

H_A: μOtherSoftwareExperience $\neq 1.5$

Table 4.31 shows that the means for SFSYSs and SGSYSs are above the neutral or 50% mark. The SGSYS area had the only strong evidence that developers are more likely than not to have security experience with SGSYSs.

SFSYSs and STSYSs showed very weak evidence against the null, and so it should be concluded in this case. This means that that likelihood of encountering a developer having security experience in SFSYSs or STSYSs is roughly the equivalent of flipping a coin if chosen at random. A brief analysis was done that shows little evidence that additional years of experience had any correlation with this average, but more in depth analysis or another survey could look at this in more detail to see if there is a correlation or causation.

Table 4.31. *Security* 2.2 — Security Experience Mean, One-sample Test Statistics, Test Value 1.5, 1 to 2 No-Yes Scale

Security Experience Area	N	Mean	Std. Deviation	Std. Error Mean
SFSYSs	31	1.5806	0.50161	0.09009
SGSYSs	31	1.7742	0.42502	0.07634
STSYSs	31	1.4194	0.50161	0.09009
Penetration testing	31	1.1290	0.34078	0.06121
Other	21	1.1905	0.40237	0.08781

Both penetration testing and other had strong evidence that the developers are more likely to not have security experience with penetration testing or in other areas. The "area" might be a bit too open ended, the low experience mark could be driven by a developer's general lack of experience outside space systems or could be that their experience outside space systems did not deal with security; this pushes against the concept that any software that is not used in total isolation should consider security. While this is a good concept, security is often not considered for general applications; this does fall in with the developer perception found in other parts of this analysis that security was more important for space software than for other fields.

Results show that 50% of the developers or greater were likely to have security experience in each of the space systems areas. This suggests that while they do tend to have space related security experience they may not have experience with testing and protecting against actual attacks and may be dealing more with information assurance, encryption, and static analysis.

The developers and designers of space systems might not have fully accepted or addressed that networked nature of ground systems or the trend towards open-network software development frameworks in space systems. For SFSYSs this might mean that they have not considered securing systems behind the COMSEC boundary as a truly interconnected system. This suggests a need for security provisions like those in SSSM that control access within a system in the case where the perimeter is penetrated to help address this under- or un-accounted for issue.

Table 4.32. *Security* 2.2 — Security Experience Mean, One-sample Test Results, Test Value 1.5, 1 to 2 No-Yes Scale

						95% CI	
Experience Area	Critical *t*-value (2-tailed)	Sample *t*-value (2-tailed)	df	Sig. (2-tailed)	Mean Difference	Lower	Upper
SFSYSs	2.0423	0.895	30	0.378	0.08065	-0.1033	0.2646
SGSYSs	2.0423	3.592	30	0.001	0.27419	0.1183	0.4301
STSYSs	2.0423	-0.895	30	0.378	-0.08065	-0.2646	0.1033
Penetration testing	2.0423	-6.061	30	0.000	-0.37097	-0.4960	-0.2460
Other	2.0860	-3.525	20	0.002	-0.30952	-0.4927	-0.1264

Because of the higher rates of security experience with SGSYSs it would seem like penetration testing should also have a higher experience rate. This lack of correlation needs more analysis, but it might stem from these systems being closed off from the larger Internet. However, what happens if an attacker gets in via physical or back-door access? What happens as more commercial providers are used or as ground segments trend to a conglomeration of different providers where network connectivity becomes more important and more unsecured networks are used?

Having tools and secured open-network software development frameworks for flight, test, and ground space systems can help to address the experience gap by allowing developers to develop software with a certain amount of security provisions baked into them. This is part of the driver behind adding security provisions to SSM as proposed by this research.

4.6.2 Networking and Security

This section analyzes the responses to *OSAM* Survey, Question 2.6, and to *Network* Survey, Question 2.3. Initial analysis of Question 2.6 shows that participants did not perceive security to be an above average factor in prohibiting the adoption of open-systems approaches, but additional analysis shows that the security factor was still in line with the other factors that might limit adoption. Section 4.6.2.1 presents the analysis of *OSAM* Survey, Question 2.6. Analysis of Question 2.3 shows that the participants perceive open-network space systems to have a negative impact on security, but an overall positive impact on the other aspects when taken as an average. Analysis also shows that these participants perceive the impact to be more negative when compared to the perceived effect of networking on other system aspects like Adaptability or Reusability. Section 4.6.2.2 presents the analysis of *Network* Survey, Question 2.3. There is a consensus among the space systems developers who participated in the *OSAM* and *Network* Surveys that, while security is not the critical factor it is not an insignificant factor in prohibiting the adoption of open-systems approaches. There is also consensus that open-network space systems negatively affect security. These perceptions show the importance of a solution, like SSSM, that aims to add security to an existing open-system software development framework, addressing security as prohibitive factor in adopting open-systems approaches by addressing the additional security needs or risk of open-network systems.

4.6.2.1 ***OSAM*** Survey, Question 2.6 Analysis

Test 1 shows that participants do not perceive security to be a critical factor in prohibiting their organization from adopting open-systems approaches. Test 2 shows that participants do perceive security to be as limiting a factor in prohibiting open-systems adoption as the other factors when taken as an average of their aggregate.

Test 1, a one-sample *t*-test, looks at the impact of security on adopting open-systems approaches. Test 2, a matched-pairs t-test, looks at the impact of security in relation to other factors in adopting open-systems approaches.

Test 1

Summary: Participants do not perceive security to be a determining or prohibitive factor in their organization adopting open-systems approaches.

Question: *OSAM* Survey, Question 2.6 asks each participant to select all the factors that might prohibit the adoption of open-systems approaches by their organization. There are 32 factors, including security. Other was also an option, but was never selected, suggesting the list provided was comprehensive. Appendix A gives a full listing of factors.

Analysis: One-sample t-test with test value of 0.5, which would mean that an equal number of participants selected it as did not select it, or a neutral value.

Hypothesis: Security will not have a neutral effect on prohibiting the adoption of open-systems approaches.

H_0: μFactorProhibition $= 0.5$

H_A: μFactorProhibition $\neq 0.5$

Table 4.33 shows that the mean for security is below the test value, slightly on the No side. Table 4.34 shows that the participants did not reach a consensus on the affect of security on the adoption of open-systems approaches.

Table 4.33. *OSAM* 2.6 — Prohibitive Effect on Adoption of Open-System Approaches Mean, One-sample Test Statistics, Test Value 0.5, 0 to 1 No-Yes Scale

Aspect	N	Mean	Std. Deviation	Std. Error Mean
Security	43	0.37	0.489	0.075

Interestingly enough the only factor that did have a strong enough t-value to possibly be considered a critical or primary factor in preventing adoption was "Management buy-in". This could stem from a lack of ability by developers to convince management of the benefits of adopting a open-systems approach, a system that already incorporates this approach could be a critical linchpin in convincing management. A lot of other factors, like developer buy in, development cost, maintenance cost, development productivity and development efficiency, trended heavily in the other direction. This suggest they were of little consequence, or possibly understood to be benefits of a open-systems approach. This pattern for some of the other factors actually puts the net neutral effect of security as more of a determining factor. This reinforces the need for a secured open-network software development framework, like the one proposed by this research.

Table 4.34. *OSAM* 2.6 — Prohibitive Effect on Adoption of Open-systems Approaches Mean, One-sample Test Results, Test Value 0.5, 0 to 1 No-Yes Scale

						95% CI	
Factor	Critical *t*-value (2-tailed)	Sample *t*-value (2-tailed)	df	Sig. (2-tailed)	Mean Difference	Lower	Upper
Management buy-in	2.0181	2.789	42	0.008	0.198	0.05	0.34
Legacy software requirements	2.0181	1.715	42	0.094	0.128	-0.02	0.28
Investment vs. return in money	2.0181	-2.789	42	0.008	-0.198	-0.34	-0.05
Compatibility with current infrastructure	2.0181	1.715	42	0.094	0.128	-0.02	0.28
Ownership of existing and future software	2.0181	-2.412	42	0.020	-0.174	-0.32	-0.03
Current proprietary systems	2.0181	-0.151	42	0.881	-0.012	-0.17	0.14
Developer buy in	2.0181	-3.192	42	0.003	-0.221	-0.36	-0.08

(continued on next page)

Table 4.34. *OSAM* 2.6 — Prohibitive Effect on Adoption of Open-systems Approaches Mean, One-sample Test Results, Test Value 0.5, 0 to 1 No-Yes Scale (continued)

						95% CI	
Factor	Critical t-value (2-tailed)	Sample t-value (2-tailed)	df	Sig. (2-tailed)	Mean Difference	Lower	Upper
Development cost	2.0181	-2.789	42	0.008	-0.198	-0.34	-0.05
Maintenance cost	2.0181	-4.631	42	0.000	-0.291	-0.42	-0.16
Development productivity	2.0181	-5.920	42	0.000	-0.337	-0.45	-0.22
Development efficiency	2.0181	-4.103	42	0.000	-0.267	-0.40	-0.14
Complexity	2.0181	-2.055	42	0.046	-0.151	-0.30	0.00
Maintainability	2.0262	-2.024	37	0.050	-0.158	-0.32	0.00
Bug detection	2.0181	-5.920	42	0.000	-0.337	-0.45	-0.22
Best practices	2.0181	-5.920	42	0.000	-0.337	-0.45	-0.22
Schedule	2.0181	-2.789	42	0.008	-0.198	-0.34	-0.05
Domain knowledge	2.0181	-5.229	42	0.000	-0.314	-0.44	-0.19
Security	2.0181	-1.715	42	0.094	-0.128	-0.28	0.02
I/0 efficiency	2.0181	-5.229	42	0.000	-0.314	-0.44	-0.19
Fault tolerance	2.0181	-6.742	42	0.000	-0.360	-0.47	-0.25
Latency	2.0181	-6.742	42	0.000	-0.360	-0.47	-0.25
Determinism	2.0181	-5.920	42	0.000	-0.337	-0.45	-0.22
Interoperability	2.0181	-3.627	42	0.001	-0.244	-0.38	-0.11
Portability	2.0181	-7.758	42	0.000	-0.384	-0.48	-0.28
Testing	2.0181	-4.631	42	0.000	-0.291	-0.42	-0.16
Reusability	2.0181	-4.631	42	0.000	-0.291	-0.42	-0.16
Upgradability	2.0181	-4.103	42	0.000	-0.267	-0.40	-0.14
Flexibility	2.0181	-4.103	42	0.000	-0.267	-0.40	-0.14
Ease of use	2.0181	-5.229	42	0.000	-0.314	-0.44	-0.19

(continued on next page)

Table 4.34. *OSAM* 2.6 — Prohibitive Effect on Adoption of Open-systems Approaches Mean, One-sample Test Results, Test Value 0.5, 0 to 1 No-Yes Scale (continued)

						95% CI	
Factor	Critical t-value (2-tailed)	Sample t-value (2-tailed)	df	Sig. (2-tailed)	Mean Difference	Lower	Upper
Information Assurance	2.0196	-3.467	41	0.001	-0.238	-0.38	-0.10

Test 2

Summary: Participants perceived security to be as much as a driving factor in prohibiting the adoption of open-systems approaches as they did the other factors.

Question: *OSAM* Survey, Question 2.6 asks each participant to select all the factors that might prohibit the adoption of open-systems approaches by their organization. There are 32 factors, including security.

Analysis: Matched-pairs t-test to compare mean of security prohibition against the overall prohibition mean; security is excluded from the overall mean.

Hypothesis: Participants will not think security will be as prohibitive in the adoption of open-systems approaches as the other factors.

H_0: μSecurityProhibitRating = μOverallProhitbitRating

H_A: μSecurityProhibitRating $\neq$ μOverallProhibitRating

Table 4.35 shows that the security measurements actually has a mean slightly above the overall mean more towards the "Yes" side.

Table 4.35. *OSAM* 2.6 — Prohibitive Effect on Adoption of Open-System Approaches Mean, Matched-pairs Statistics, 0 to 1 No-Yes Scale

	Security Area	Mean	N	Std. Deviation	Std. Error Mean
Pair 1	Security	0.37	43	0.489	0.075
	Overall	0.2814	43	0.20990	0.03201

The results in Table 4.36 give very weak or no evidence against the null, this means that the prohibitive effect of security on the adoption of open-systems approaches is in line with the average effect of all the factors the participants had to choose from. This suggesting that security was consequential in the choice to adopt open-systems approaches. It was not as critical a factor as "Management buy-in", but it was significantly above other factors that had very negative *t*-values as mentioned in Test 1. This reinforces the need for a secured open-network software development framework, like the one proposed by this research, to aid in the adoption of open-systems approaches.

Table 4.36. *OSAM* 2.6 — Prohibitive Effect on Adoption of Open-systems Approaches Mean, Matched-pairs Test Results, 0 to 1 No-Yes Scale

				95% CI					
Area	Mean	Std. Dev.	Std. Error	Lower	Upper	Critical *t*-value (2-tailed)	Sample *t*-value (2-tailed)	*df*	Sig. (2-tailed)
Pair 1	0.09070	0.4738	0.0723	-0.0551	0.2365	2.0181	1.255	42	0.216

4.6.2.2 *Network* Survey, Question 2.3 Analysis, Part 2

Test 1 shows that participants perceive using a open-network space system will have a negative impact on the security of that system. Test 2 shows that participants perceive using a open-network space system will have a more negative effect on security than on the other aspects of a system that the participants were asked to consider. *Network* Survey, Question 2.3 is a Pros-neutral-cons rating question.

Test 1, a one-sample *t*-test, looks at the impact of using a open-network space system on the security of that system. Test 2, a matched-pairs *t*-test, looks at the impact of using a open-network space system on various aspects of space systems and space systems development as they relate to each other.

Test 1

Summary: Participants perceive networking to have a negative impact on security.

Question: *Network* Survey, Question 2.3 asks each participant to record the impact that a open-network space system has on a range of system aspects in terms of Pros, Neutral, and Cons. Pros is coded as a 3, Neutral is coded as a 2, and Cons is coded as a 1. The impact rating was solicited for a wide range of aspects including security, Section 4.5.1.1 gives a full listing.

Analysis: One-sample t-test with test value of 2, which represents a neutral value. This section focuses on the effect on security.

Hypothesis: There is some consensus among space systems developers that the impact of open-network space systems will be negative or positive, not neutral.

H_0: $\mu\text{SecurityImpactRating} = 2$

H_A: $\mu\text{SecurityImpactRating} \neq 2$

Table 4.37 shows that the security measurement has a mean below the test value, on the Cons side.

Table 4.37. *Network* 2.3 — Network Benefits on Security-aspect Mean, One-sample Test Statistics, Test Value 2, 1 to 3 Cons-Neutral-Pros Scale

Aspect	N	Mean	Std. Deviation	Std. Error Mean
Security	29	1.4828	0.63362	0.11766
Overall (Security Included)	29	2.1899	0.22063	0.04097

Table 4.38 shows that the group perceives security to be negatively impacted by the use of open-network space systems. Table 4.38 also shows that overall the participants perceived open-network space systems to be beneficial which is in contrast to the perceived effect on the security aspect of a system.

Test 2 will see if this finding or separation is significant. These results together suggest that the developers see value in adding security to the fabric of an open-network space system, much like the system proposed by this research. This allows space systems developers to achieve the overall benefits of open-network space systems while addressing their security concerns.

Table 4.38. *Network* 2.3 — Network Benefits on Security-aspect Mean, One-sample Test Results, Test Value 2, 1 to 3 Cons-Neutral-Pros Scale

						95% CI	
Area	Critical t-value (2-tailed)	Sample t-value (2-tailed)	df	Sig. (2-tailed)	Mean Difference	Lower	Upper
Security	2.0484	-4.396	28	0.000	-0.51724	-0.7583	-0.2762
Overall (Security Included)	2.0484	4.635	28	0.000	0.18989	0.1060	0.2768

Test 2

Summary: Participants perceive networking to have a more negative impact on security than for other system aspects.

Question: *Network* Survey, Question 2.3 asks each participant to record the impact that a open-network space system has on a range of system aspects in terms of Pros, Neutral, and Cons. Pros is coded as a 3, Neutral is coded as a 2, and Cons is coded as a 1. The impact rating was solicited for a wide range aspects including security, Section 4.5.1.1 covers the wider set of aspects.

Analysis: Matched-pairs t-test to compare mean of security against the overall aspect mean.

Hypothesis: Participants will not think the impact of open-network space systems is the same for security as it is for the rest of the list of system aspects.

H_0: μOverallImpactRating $=$ μSecurityImpactRating

H_A: μOverallImpactRating $\neq$ μSecurityImpactRating

Table 4.39 shows that the security mean is below the *Overall* mean, putting security on the Cons side of the *Overall* mean.

Table 4.39. *Network* 2.3 — Networking Impact on Security Mean. Matched-pairs Test Statistics, 1 to 3 Cons-Neutral-Pros Scale

	Security Area	Mean	N	Std. Deviation	Std. Error Mean
Pair 1	Security	1.4828	29	0.63362	0.11766
	Overall (Security Excluded)	2.2104	29	0.22876	0.04248

Table 4.40 shows that participants did not perceive open-network space systems to have the same effect on the other aspects of a system as it did on security. They perceived the effect to much more negative on security than the other systems aspects. This suggests a need to add additional security provisions to an open-network space system so that this perceived negative will be less of an issue and the perceived benefits can be realized.

Table 4.40. *Network* 2.3 — Networking Impact on Security Mean, Matched-pairs Test Results, 1 to 3 Cons-Neutral-Pros Scale

				95% CI					
Security Area	Mean	Std. Dev.	Std. Error	Lower	Upper	Critical *t*-value (2-tailed)	Sample *t*-value (2-tailed)	*df*	Sig. (2-tailed)
Pair 1	0.72759	0.6109	0.1134	0.49520	0.95999	2.0484	6.413	28	0.000

4.6.3 Security Importance and Difficulty

Analysis of *CC* Survey, Question 2.1 and *Security* Survey, Question 2.6 shows there is a consensus among the space systems developers who participated in the *CC* and *Security* Surveys that security is important to provide for space systems. This analysis goes on to show that there is consensus that security is more important for certain space-related software categories than others and that security is more important for open-network space systems than for traditional point-to-point space systems. Section 4.6.3.1 presents the results from *CC* Survey, Question 2.1 that related to security importance. These results show there is consensus among these developers that security is more important for SFS and SGS software than for other software fields or STS. Section 4.6.3.2 presents the results from *Security* Survey, Question 2.6, and *Security* Survey, Question 2.7. The results from Question 2.6 show there is consensus among these developers that software security is more important for open-network space systems than for traditional point-to-point space systems. The results from Question 2.7 show there is a lack consensus among participants that it is difficult to add the security provisions in the question. These results illustrate the need to provide security for space systems in general and specifically for open-network space systems. It follows that adding security to a open-network software development framework for space systems makes it easier to develop open-network space systems, which the developers that participated in the *CC* and *Security* Surveys perceive as important.

4.6.3.1 *CC* Survey, Question 2.1 Analysis, Part 3

This section shows that the participants perceive security to be important for SFS, SGS, and other software fields, but not specifically for STS. This section also shows that participants perceive security to be more important for SFS and SGS than for *Other Software* fields, to be more important for *Other Software* fields than for STS, and finally to be more important for SGS than for SFS. This section shows security to be generally more important for space systems than for *Other Software*. The section also shows that security has an equivalent importance to reuse and code complexity across space systems software and *Other software*.

This analysis covers two statistical tests. Test 1, a one-sample t-test, looks at security importance for software in general. Test 2, a matched-pairs t-test, compares security importance across different software areas.

Test 1

Summary: Participants perceive security to be important for SFS, SGS, and *Other software* fields, but not for STS. Participants do perceive security to be generally important for space systems software.

Question: Question 2.1 asks each participant to rate the importance of security for SFS, SGS, STS, and *Other Software* fields on a 5-point Likert scale.

Analysis: One-sample t-test with test value of 3, which represents a neutral value.

Hypothesis: Software developers will find security to be important. This hypothesis is driven by the rise to prominence of computer security in recent years, as attacks and compromises make major headlines, as well as the generally low risk posture assumed when developing for space systems.

H_0: μSecurityImportance ≤ 3

H_A: μSecurityImportance > 3

Table 4.41. *CC* 2.1, Part 1 — Security Importance Mean One-sample Test Statistics, Test Value 3, 5-point Likert Scale

Area	N	Mean	Std. Deviation	Std. Error Mean
SFS	91	3.97	1.178	0.123
SGS	90	4.29	0.939	0.099
STS	89	2.94	1.237	0.131
Other Software	79	3.49	1.061	0.119
Overall	93	3.73	0.864	0.090

Table 4.41 shows that, all the importance means, except STS come out above 3. Table 4.42 shows that all of the software areas have strong evidence against the null except STS. This means that there is a very strong evidence that the group perceives security to be important for SFS, SGS, and *Other Software* fields.

STS shows little difference between the test value of 3, or there is very weak evidence against null so it must be concluded. This means that the participants see security importance as neutral for STS. However, when SFS, STS, and SGS are considered together the t-value is very strong, suggesting security is important for space systems as a whole when considered as an aggregate.

Table 4.42. *CC* 2.1. Part 3 — Security Importance Mean, One-sample Test Results, Test Value 3, 5-point Likert Scale

							95% CI	
Area	Critical t-value (2-tailed)	Sample t-value (2-tailed)	df	Sig. (2-tailed)	Sig. (1-tailed)	Mean Difference	Lower	Upper
SFS	1.9867	7.831	90	0.000	0.0000	0.967	0.72	1.21
SGS	1.9870	13.020	89	0.000	0.0000	1.289	1.09	1.49
STS	1.9873	-0.428	88	0.669	0.6655	-0.056	-0.32	0.20
Other Software	1.9909	4.137	78	0.000	0.0000	0.494	0.26	0.73
S*S Overall	1.9861	8.204	92	0.000	0.0000	0.735	0.56	0.91

Test 2

Summary: Participants perceive security to be more important for SFS, SGS than for *Other Software* fields. Security was not perceived to more important for STS.

Question: Question 2.1 asks each participant to rate the importance of security for SFS, SGS, STS, and other software fields on a 5-point Likert scale.

Analysis: Matched-pairs t-test to compare means of SFS, SGS, and STS against *Other Software* fields. The means of SFS and SGS were also compared, they will have the same hypothesis as before except *Other Software* is replaced with SFS.

Hypothesis:

H_0: $\mu\text{S*S} = \mu\text{OtherSoftware}$

H_A: $\mu\text{S*S} \neq \mu\text{OtherSoftware}$

Table 4.43 shows that the means for SFS and SGS are above the mean for *Other Software* fields while the mean for STS is below. Table 4.43 also shows that the mean for SGS is above the mean for SFS.

Table 4.43. *CC* 2.1, Part 3 — Security Importance Mean, Matched-pairs Test Statistics, 5-point Likert Scale

	Area	Mean	N	Std. Deviation	Std. Error Mean
Pair C1	SFS	4.03	77	1.170	0.133
	Other Software	3.55	77	1.020	0.116
Pair C2	SGS	4.32	77	0.938	0.107
	Other Software	3.49	77	1.047	0.119
Pair C3	STS	3.05	76	1.243	0.143
	Other Software	3.54	76	1.051	0.121
Pair C4	SGS	4.28	88	0.946	0.101
	SFS	3.97	88	1.198	0.128
Pair C5	S*S	3.80	78	0.847	0.096
	Other Software	3.51	78	1.054	0.119

Table 4.44 shows that *Pair C3* gives strong evidence against the null, but in the negative direction. This means the group perceives security to be less important for STS than for *Other Software* fields and by extension than for SFS and SGS. STS is not typically thought of as production software, and is usually isolated from the outside world so that might be one of the reasons for its perceived lack of importance. It is not unheard of for test software to transition to production software, in part or in whole, so this is a bit concerning.

Table 4.44. *CC* 2.1, Part 3 — Security Importance Mean, Matched-pairs Test Results, 5-point Likert Scale

				95% CI					
Area	Mean	Std. Dev.	Std. Error	Lower	Upper	Critical *t*-value (2-tailed)	Sample *t*-value (2-tailed)	*df*	Sig. (2-tailed)
Pair C1: SFS vs. Other Software	0.481	1.420	0.162	0.158	0.803	1.9917	2.970	76	0.004

(continued on next page)

Table 4.44. *CC* 2.1, Part 3 — Security Importance Mean, Matched-pairs Test Results, 5-point Likert Scale (continued)

				95% CI					
Area	Mean	Std. Dev.	Std. Error	Lower	Upper	Critical *t*-value (2-tailed)	Sample *t*-value (2-tailed)	*df*	Sig. (2-tailed)
Pair C2: SGS vs. Other Software	0.831	1.152	0.131	0.570	1.093	1.9917	6.333	76	0.000
Pair C3: STS vs. Other Software	-0.487	1.291	0.148	-0.782	-0.192	1.9921	-3.288	75	0.002
Pair C4: SGS vs. SFS	0.318	1.089	0.116	0.088	0.549	1.9876	2.742	87	0.007
Pair C5: S*S vs. Other Software	0.278	1.071	0.121	0.036	0.519	1.9913	2.291	77	0.025

Table 4.44 that the results for *Pair C1* and *Pair C2* give strong evidence that the group perceives security to be more important for SFS and SGS than for *Other Software* fields. This is likely due to generally lower risk postures for space assets and the inherent difficulties of developing for space applications.

Pair C4 shows that security for SGS is perceived to be more important than for SFS. This may be due to the physical isolation of objects in space and the ability of an attacker to go through a ground system to get to a space asset. This is one of the paradigms that is changing though, and cross-links and commercial satellite options are opening up multiple pathways in to space flight systems. It is concerning that the community does not see SFS security to be as important as SGS security, hopefully education and information sharing will change this perception.

Pair C5 shows that the participants generally perceive security to be more important for the space-related software in question than for *Other Software* fields.

4.6.3.2 *Security* Survey, Questions 2.6 and 2.7 Analysis

This section shows that the participants perceive the security provisions in the question to be important for traditional point-to-point and open-network space systems. This section also shows a lack of consensus around the difficulty of adding these provisions to traditional point-to-point or open-network space systems. This section shows that participants perceive the security provisions to be more important for open-network space systems than for traditional point-to-point space systems.

Test 1, a one-sample t-test, looks at security importance and difficulty for traditional and open-network space systems. Test 2, a matched-pairs t-test, compares security importance and difficulty across traditional and open-network space systems.

Test 1

Summary: Participants perceive the security provisions to be important for both open-network and traditional point-to-point space systems architectures. Participants perceive the security provisions to be neither difficult or easy to provide for both open-network and traditional point-to-point space systems architectures.

Question: Question 2.6 asks each participant to rate the importance of different security provisions on a 5-point Likert scale. Question 2.7 asks each participant to rate the difficulty of providing different security provisions on a 5-point Likert scale. The participants are asked to rate these provisions on both open-network and traditional point-to-point space systems architectures.

Analysis: One-sample t-tests with test value of 2, which represents a neutral value on a compressed 3-point Likert scale.

Hypothesis: The μ is considered as the mean importance or difficult for security provisions for open-network space systems and traditional point-to-point systems when considered separately.

H_0: μSecuritySurveyParticipants $= 2$

H_A: μSecuritySurveyParticipants $\neq 2$

Table 4.45 shows that the importance means are above the neutral test value of 2. Table 4.45 shows that the difficulty means are roughly equal to the neutral value of 2.

Table 4.45. *Security* 2.6 and 2.7 — Security Importance and Difficulty Mean, One-sample Test Statistics, Test Value 2, 3-point Likert Scale

Security Area	N	Mean	Std. Deviation	Std. Error Mean
Traditional Security Importance	29	2.3852	0.47539	0.08828
Open-network Security Importance	28	2.7340	0.35599	0.06728
Traditional Security Difficulty	26	1.9410	0.55665	0.10917
Open-network Security Difficulty	26	2.0377	0.55582	0.10900

Table 4.46 shows that the group perceives security to be important for traditional point-to-point space systems. Table 4.46 also shows that the group perceives security to be important for open-network space systems.

Table 4.46. *Security* 2.6 and 2.7 — Security Importance and Difficulty Mean, One-sample Test Results, Test Value 2, 3-point Likert Scale

						95% CI	
Security Area	Critical t-value (2-tailed)	Sample t-value (2-tailed)	df	Sig. (2-tailed)	Mean Difference	Lower	Upper
Traditional Security Importance	2.0484	4.364	28	0.000	0.38521	0.2044	0.5660
Open-network Security Importance	2.0518	10.911	27	0.000	0.73401	0.5960	0.8721
Traditional Security Difficulty	2.0596	-0.540	25	0.594	-0.05897	-0.2838	0.1659

(continued on next page)

Table 4.46. *Security* 2.6 and 2.7 — Security Importance and Difficulty Mean, One-sample Test Results, Test Value 2, 3-point Likert Scale (continued)

						95% CI	
Security Area	Critical t-value (2-tailed)	Sample t-value (2-tailed)	df	Sig. (2-tailed)	Mean Difference	Lower	Upper
Open-network Security Difficulty	2.0596	0.346	25	0.732	0.03773	-0.1868	0.2622

The difficulty mean for both traditional point-to-point and open-network space systems gives no evidence for refuting the null. This suggests that there is no consensus among developers that providing these security provisions is difficult or not difficult for space systems.

Test 2

Summary: Participants perceive the security provisions to be more important for open-network space systems than for for traditional point-to-point space systems.

Question: Question 2.6 asks each participant to rate the importance of different security provisions on a 5-point Likert scale. The participants are ask to rate these provisions on both open-network and traditional point-to-point space systems architectures.

Analysis: Matched-pairs t-test to compare means of security provisions for open-network space systems than for for traditional point-to-point space systems.

Hypothesis: The μ is considered as the mean importance for security provisions for open-network space systems and traditional point-to-point systems when considered separately.

H_0: μOpenSpaceSystems = μTradSpaceSystems

H_A: μOpenSpaceSystems $\neq$ μTradSpaceSystems

Table 4.47. *Security* 2.6 — Security Importance Mean, Matched-pairs Test Statistics, 3-point Likert Scale

Security Area	N	Mean	Std. Deviation	Std. Error Mean
Traditional Security Importance	28	2.4014	0.47594	0.08994
Open-network Security Importance	28	2.7340	0.35599	0.06728

Table 4.48 shows that the group perceives the mean for these security provisions to be more important for open-network space systems than for traditional point-to-point space systems. This result coupled with the results from Test 1 show that there is a consensus among developers that security is important for space systems and that it is generally more important for open-network space systems.

Table 4.48. *Security* 2.6 — Security Importance Mean, Matched-pairs Test Results, 3-point Likert Scale

				95% CI					
Security Area	Mean	Std. Dev.	Std. Error	Lower	Upper	Citical *t*-value (2-tailed)	Sample *t*-value (2-tailed)	df	Sig. (2-tailed)
Open-network vs. Traditional	0.3327	0.4859	0.0918	0.1442	0.5211	2.0518	3.622	27	0.001

4.6.3.3 Security Importance and Difficulty Summary

Section 4.6.3 presents results that show that there is a consensus among space systems developers that software security is more important for production space systems, namely SFS and SGS systems; and more important for these applications than it is for *Other Software* fields. These results also show that there is consensus that security is more important for open-network space systems than it is for traditional point-to-point space systems. This illustrates the need to provide and the importance of providing security for space systems in general and specifically for open-network space systems. It follows that adding security to a open-network software development framework for space systems, which is the aim of SSSM, makes it easier to develop secure open-network space systems, which the space systems development community perceives to be important.

4.6.4 Important Security Provisions

Analysis of *Security* Survey, Question 2.8 shows there to be a consensus among the space systems developers who participated in the *Security* Surveys that *authorization*, *integrity*, *identity management*, and *access control* are the most important security provisions for space systems. The survey also shows that *abstraction layers* is the least important security provision and that *mitigation* and *compliance* were also ranked below the other provisions. It should be noted that participants had the option to add their own provisions and rank them, but none were added; this suggests a good list. This consensus suggests that the space systems development community see the value in a secured space system that provides for *authorization*, *integrity*, *identity management*, and *access control*. SSSM is a secured open-network software development framework that provides for *authorization*, *integrity*, *identity management*, and *access control*.

4.6.4.1 *Security* Survey, Questions 2.8 Analysis

This section shows that participants perceive authorization, integrity, identity management, and access control to be the most important security provisions, and abstraction layers, mitigation, and compliance to be the least important.

> **Summary**: Participants perceive *authorization*, *integrity*, *identity management*, and *access control* as the most important security provisions for space systems.
> **Question**: *Security* Survey, Question 2.8 asks each participant to rank the provided security provisions from 1 to 15, 1 being the highest ranking. Participants also had the option to write in up to three “other” provisions and rank them, but no one wrote anything in, so other is not being consider as part of the ranking.

Analysis: One-sample t-test with test value of 7.5, which represents a middle-of-the-pack ranking.

Hypothesis: Certain security provisions will have a mean ranking below or above the middle ranking suggesting they are perceived as more or less important than other provisions that do tend towards the middle ranking of 7.5.

H_0: μProvisionRanking $= 8$

H_A: μProvisionRanking $\neq 8$

Table 4.49 shows that 4 of that security provision importance ranking means are decidedly below 7.5; namely *authorization*, *integrity*, *identity management*, and *access control.* In this case a lower mean indicates a better or "higher" ranking. Three provisions are decidedly above the test value of 7.5, namely *abstraction layers*, *mitigation*, and *compliance.*

Table 4.49. *Security* 2.8 — Security Provision Importance Ranking Mean, One-sample Test Statistics, Test Value 7.5, 1 to 15 Ranking Scale

Security Area	N	Mean	Std. Deviation	Std. Error Mean
Identity management	27	5.70	3.950	0.760
Mutual authentication	27	6.44	4.060	0.781
Authorization	27	4.78	3.693	0.711
Auditing	27	7.11	2.651	0.510
Confidentiality	27	7.56	2.722	0.524
Integrity	27	5.37	3.040	0.585
Availability	27	7.07	3.802	0.732
Well-defined interfaces	27	7.78	3.866	0.744
Abstraction layers	27	11.04	2.696	0.519
Access control	27	5.74	3.938	0.758
Compliance	27	9.44	4.032	0.776
Testing	27	8.59	4.822	0.928
Recovery	27	8.89	3.523	0.678
Mitigation	27	9.67	4.368	0.841

Table 4.50 shows that *authorization*, *integrity*, *identity management*, and *access control* have strong negative evidence against the null. This means there was a consensus amongst

participants that *authorization*, *integrity*, *identity management*, and *access control* were the most important, or were consistently ranked higher than the other provisions.

Table 4.50 shows that *abstraction layers*, *mitigation*, and *compliance* have strong positive evidence against the null. This means these provisions had a "lower" ranking than the average, or that the participants consistently ranked these provisions lower than the other provisions.

Table 4.50. *Security* 2.8 — Security Provision Importance Ranking Mean, One-sample Test Results, Test Value 7.5, 1 to 15 Ranking Scale

						95% CI	
Security Area	Critical *t*-value (2-tailed)	Sample *t*-value (2-tailed)	df	Sig. (2-tailed)	Mean Difference	Lower	Upper
Identity management	2.0555	-2.363	26	0.026	-1.796	-3.36	-0.23
Mutual authentication	2.0555	-1.351	26	0.188	-1.056	-2.66	0.55
Authorization	2.0555	-3.830	26	0.001	-2.722	-4.18	-1.26
Auditing	2.0555	-0.762	26	0.453	-0.389	-1.44	0.66
Confidentiality	2.0555	0.106	26	0.916	0.056	-1.02	1.13
Integrity	2.0555	-3.640	26	0.001	-2.130	-3.33	-0.93
Availability	2.0555	-0.582	26	0.566	-0.426	-1.93	1.08
Well-defined interfaces	2.0555	0.373	26	0.712	0.278	-1.25	1.81
Abstraction layers	2.0555	6.817	26	0.000	3.537	2.47	4.60
Access control	2.0555	-2.321	26	0.028	-1.759	-3.32	-0.20
Compliance	2.0555	2.506	26	0.019	1.944	0.35	3.54
Testing	2.0555	1.177	26	0.250	1.093	-0.81	3.00
Recovery	2.0555	2.049	26	0.051	1.389	0.00	2.78
Mitigation	2.0555	2.578	26	0.016	2.167	0.44	3.89

Table 4.50 also shows that *mutual authentication*, *auditing*, *confidentiality*, *availability*, *well-defined interfaces*, *testing*, and *recovery* had weak *t*-values. This means that these provisions had an average ranking.

The list of security provisions that SSSM was designed to address covers all the important provisions from Chapter 2, some of the average provisions, and none of the unimportant provisions. SSSM focused on *identity management*, *mutual authentication*, *authorization*, *confidentiality*, and *integrity*. *Confidentiality* and *mutual authentication* were both only seen as having average importance, and the cut-off does have to be somewhere. Average importance still suggest a need as well. The lower ranking for *confidentiality* and *mutual authentication* may be tied to a certain amount of trust developers are giving to the internals of their space system, going back to the typical COMSEC boundary model were everything behind the gateway is trusted. This assumption does not address what happens if the boundary is compromised or the fact that components might come from various untrusted or less trusted sources that are all now networked together. This is changing as more and more organizations trend towards the use of MONAs, a trend perceived by participants as Section 4.5.2 shows. There may be a disconnect or lack of understanding here though, that may explain the findings presented in Section 4.6.5 were the participant do not come to a consensus on the benefit of internal network security. SSSM addresses all of the provisions currently seen as important by participants and looks to address some provisions that will likely become more important in the future as the reality of fully networked system because more apparently.

Auditing is also addressed to a lesser extent and *availability* was left to future work; both of these provisions are considered to be of average importance by the participants. Their slightly lower ranking suggests that the priorities of the SSSM design are correct.

4.6.5 Internal Security Benefits

There is a lack of consensus among the space systems developers that participated in the *Security* Survey that internal security is beneficial to or even that it negatively impacted space systems and space systems development. There is consensus that certain aspects where positively affected and certain ones were negatively affected, but for larger aggregate set of aspects the perception is that the effect is neutral. Specially 11 aspects show a positive effect, 17 show a neutral effect, and 11 show a negative effect. This section analyzes the responses to *Security* Survey, Questions 2.1. Section 4.6.5.1 presents the analysis of Question 2.1. These perceptions of internal security's affect on system-aspects show that there is a need for a secured open-network software development framework that allow developers to address those aspects they deem to be negatively affected, improve on those that are neutrally affected, and realize the benefits of aspects that are perceived to be positively affected.

4.6.5.1 *Security* Survey, Questions 2.1 Analysis

This section shows that the perception of positive effect is the strongest for security, best practices, code quality, information assurance, and mission assurance; the participants perceive 11 total aspects to be positively affected. The perception of negative effect is the strongest for code complexity, latency, and development cost; the participants perceive 11 total aspects to be negatively affected. *Security* Survey, Question 2.1 is a Pros-neutral-cons rating question.

Summary: Participants perceive internal space systems security to have an overall "neutral" affect on the system and system development aspects that they rated.

Question: *Security* Survey, Question 2.1 asks each participant to record the impact that internal security in space systems has on a range of system aspects in terms of Pros, Neutral, and Cons. Pros is coded as a 3, Neutral is coded as a 2, and Cons is coded as a 1.

Analysis: One-sample t-test with test value of 2, which represents a neutral value.

Hypothesis: Participants will perceive internal security to have a non-neutral effect or impact on system-aspects.

H_0: μInternalSecurityImpact $= 2$

H_A: μInternalSecurityImpact $\neq 2$

Table 4.52 shows that there are 11 aspects that participants perceive to be positively impacted. There are 17 that are either neutrally affected or on which participants did not reach a consensus. Table 4.52 shows that there are 11 aspects that participants perceive to be negatively impacted.

Table 4.51. *Security* 2.1 — Internal Security Benefits on System-aspects Means, One-sample Test Statistics, Test Value 2, 1 to 3 Cons-Neutral-Pros Scale

Aspect	N	Mean	Std. Deviation	Std. Error Mean
Regression reduction	29	1.9655	0.77840	0.14455
Code design	29	1.6897	0.80638	0.14974
Development cost	29	1.4138	0.68229	0.12670
Maintenance cost	29	1.5517	0.63168	0.11730
Development productivity	29	1.9655	0.73108	0.13576
Development efficiency	29	1.6207	0.67685	0.12569
Code complexity	29	1.3448	0.61388	0.11399
Maintainability	29	1.9655	0.77840	0.14455
Integration	29	2.1034	0.77205	0.14337
Adaptability	29	1.8276	0.75918	0.14098
Documentation/Examples	29	2.2414	0.57664	0.10708
Encapsulation	29	2.2414	0.51096	0.09488
Bug detection	29	2.3448	0.66953	0.12433
Code quality	29	2.4483	0.50612	0.09398
Code robustness	29	2.3448	0.66953	0.12433
Best practices	29	2.6897	0.47082	0.08743
Schedule	29	1.5517	0.63168	0.11730
Code or algorithm optimization/efficiency	29	1.6897	0.66027	0.12261
Uniformity of coding style	29	2.2759	0.45486	0.08447
Domain knowledge	29	1.8966	0.67320	0.12501
Code readability	29	2.2759	0.52757	0.09797
Security	29	2.8966	0.30993	0.05755
I/0 efficiency	29	1.7586	0.57664	0.10708
Radiation hardness	29	2.1379	0.44111	0.08191
Fault tolerance	29	2.2414	0.63556	0.11802
Hardware complexity	29	1.7241	0.59140	0.10982
Latency	29	1.4828	0.50855	0.09443

(continued on next page)

Table 4.51. *Security* 2.1 — Internal Security Benefits on System-aspects Means, One-sample Test Statistics, Test Value 2, 1 to 3 Cons-Neutral-Pros Scale (continued)

Aspect	N	Mean	Std. Deviation	Std. Error Mean
Determinism	29	2.1724	0.60172	0.11174
Interoperability	29	1.8966	0.67320	0.12501
Portability	29	1.9310	0.65088	0.12087
Testing	29	2.0000	0.75593	0.14037
Reusability	29	2.1724	0.60172	0.11174
Software upgradability	29	2.0000	0.65465	0.12157
Hardware changes/flexibility	29	1.6897	0.54139	0.10053
Adoption rates/software proliferation	29	2.0000	0.65465	0.12157
Ease of use	29	1.8621	0.74278	0.13793
Mission/Project requirement changes	29	1.8966	0.61788	0.11474
Information assurance	29	2.5517	0.63168	0.11730
Mission assurance	29	2.5517	0.50612	0.09398
Overall	29	2.0106	0.30597	0.05682

A break down of the negatively, positively, and neutrally or inconclusively affected aspects by development cost, development quality, assurance/risk, and resource efficiency is shown below. For this categorization development costs are aspects that increase developer or development resource utilization, aspects that drive up schedule or make it harder to develop space systems. Development quality refers to the quality of software or hardware that is produced. Assurance/risk covers systems or development aspects that deal with the level of surety developers have that the system will handle faults, environmental stressors, or attacks and still be able to meet mission objectives. Resource efficiency covers how well a piece of software or hardware is able to do its job; for software that might refer to CPU utilization, for software and hardware this might refer to maximum data throughput. These categorization are based on researcher experience and background.

Negatively Affected

- **Development cost:** development cost, maintenance cost, development efficiency, schedule
- **Development quality:** code design, code complexity
- **Assurance/risk:**
- **Resource efficiency:** code or algorithm optimization/efficiency, I/O efficiency, latency, hardware change flexibility, hardware complexity

Positively Affected

- **Development cost:** bug detection
- **Development quality:** encapsulation, code robustness, best practices, uniformity of coding style, code readability, documentation/examples
- **Assurance/ris:** security, information assurance, mission assurance
- **Resource efficiency:**

Not Affected or no Consensus

- **Development cost:** ease of use, mission/project requirement changes, software upgradability, reusability, portability, interoperability, adaptability, integration, maintainability, development productivity, regression reduction
- **Development quality:** adoption rates/software proliferation
- **Assurance/risk:** testing, fault tolerance, radiation hardness
- **Resource efficiency:** determinism

Table 4.52. *Security* 2.1 — Internal Security Benefits on System-aspects Means, One-sample Test Results, Test Value 2, 1 to 3 Cons-Neutral-Pros Scale

						95% CI	
Area	Critical *t*-value (2-tailed)	Sample *t*-value (2-tailed)	df	Sig. (2-tailed)	Mean Difference	Lower	Upper
Regression reduction	2.0484	-0.239	28	0.813	-0.0345	-0.3306	0.2616
Code design	2.0484	-2.073	28	0.048	-0.3103	-0.6171	-0.0036
Development cost	2.0484	-4.627	28	0.000	-0.5862	-0.8457	-0.3267
Maintenance cost	2.0484	-3.822	28	0.001	-0.4483	-0.6886	-0.2080
Development productivity	2.0484	-0.254	28	0.801	-0.0345	-0.3126	0.2436
Development efficiency	2.0484	-3.018	28	0.005	-0.3793	-0.6368	-0.1218
Code complexity	2.0484	-5.747	28	0.000	-0.6552	-0.8887	-0.4217

(continued on next page)

Table 4.52. *Security* 2.1 — Internal Security Benefits on System-aspects Means, One-sample Test Results, Test Value 2, 1 to 3 Cons-Neutral-Pros Scale (continued)

						95% CI	
Area	Critical *t*-value (2-tailed)	Sample *t*-value (2-tailed)	df	Sig. (2-tailed)	Mean Difference	Lower	Upper
Maintainability	2.0484	-0.239	28	0.813	-0.0345	-0.3306	0.2616
Integration	2.0484	0.722	28	0.477	0.1035	-0.1902	0.3971
Adaptability	2.0484	-1.223	28	0.232	-0.1724	-0.4612	0.1164
Documentation/ Examples	2.0484	2.254	28	0.032	0.2414	0.0220	0.4607
Encapsulation	2.0484	2.544	28	0.017	0.2414	0.0470	0.4357
Bug detection	2.0484	2.774	28	0.010	0.3448	0.0902	0.5995
Code quality	2.0484	4.770	28	0.000	0.4483	0.2558	0.6408
Code robustness	2.0484	2.774	28	0.010	0.3448	0.0902	0.5995
Best practices	2.0484	7.888	28	0.000	0.6897	0.5106	0.8687
Schedule	2.0484	-3.822	28	0.001	-0.4483	-0.6886	-0.2080
Code or algorithm optimization/ efficiency	2.0484	-2.531	28	0.017	-0.3103	-0.5615	-0.0592
Uniformity of coding style	2.0484	3.266	28	0.003	0.2759	0.1028	0.4489
Domain knowledge	2.0484	-0.828	28	0.415	-0.1035	-0.3595	0.1526
Code readability	2.0484	2.816	28	0.009	0.2759	0.0752	0.4765
Security	2.0484	15.578	28	0.000	0.8966	0.7787	1.0144
I/0 efficiency	2.0484	-2.254	28	0.032	-0.2414	-0.4607	-0.0220
Radiation hardness	2.0484	1.684	28	0.103	0.1380	-0.0299	0.3057
Fault tolerance	2.0484	2.045	28	0.050	0.2414	-0.0004	0.4831
Hardware complexity	2.0484	-2.512	28	0.018	-0.2759	-0.5008	-0.0509
Latency	2.0484	-5.477	28	0.000	-0.5172	-0.7107	-0.3238
Determinism	2.0484	1.543	28	0.134	0.1724	-0.0565	0.4013

(continued on next page)

Table 4.52. *Security* 2.1 — Internal Security Benefits on System-aspects Means, One-sample Test Results, Test Value 2, 1 to 3 Cons-Neutral-Pros Scale (continued)

						95% CI	
Area	Critical t-value (2-tailed)	Sample t-value (2-tailed)	df	Sig. (2-tailed)	Mean Difference	Lower	Upper
Interoperability	2.0484	-0.828	28	0.415	-0.1035	-0.3595	0.1526
Portability	2.0484	-0.571	28	0.573	-0.0690	-0.3165	0.1786
Testing	2.0484	0.000	28	1.000	0.0000	-0.2875	0.2875
Reusability	2.0484	1.543	28	0.134	0.1724	-0.0565	0.4013
Software upgradability	2.0484	0.000	28	1.000	0.0000	-0.2490	0.2490
Hardware changes/ flexibility	2.0484	-3.087	28	0.005	-0.3103	-0.5163	-0.1044
Adoption rates/ software proliferation	2.0484	0.000	28	1.000	0.0000	-0.2490	0.2490
Ease of use	2.0484	-1.000	28	0.326	-0.1379	-0.4205	0.1446
Mission/Project requirement changes	2.0484	-0.902	28	0.375	-0.1035	-0.3385	0.1316
Information assurance	2.0484	4.704	28	0.000	0.5517	0.3114	0.7920
Mission assurance	2.0484	5.870	28	0.000	0.5517	0.3592	0.7442
Overall	2.0484	0.187	28	0.853	0.0106	-0.1058	0.1270

Participants did not perceive any aspects of assurance/risk aspects to be negatively affected by adding internal security, and felt that it positively affected overall security and assurance status of a space system while having a neutral affect on testing, fault tolerance and radiation hardness. This suggests that adding an addition layer of security is a acceptable way to increase a systems overall assurance and security. Development quality as a category also showed a generally positive trend, while development cost showed a more negative/neutral affect which might mean that while software development might take longer or cost more it would be of higher quality with internal security as part of the system.

The participants did not perceive any resource efficiency aspects to be positively affected by adding internal security. Specifically latency and algorithm optimization/efficiency were negatively affected, with latency having one of the strongest negative t-values. Section 4.4 shows these to be the only aspects affected negatively by reusability. Reusability had a neutral effect on security and the reverse was also true. There is commonality in what has to be done to a piece of software or module to make it reusable and to make it secure. Reusability typically drives the need for an interface at which other pieces of software or hardware can interact with a software module or unit, as mentioned previously, this level of encapsulation and decoupling make it easier to reuse the module and make changes to the module without affecting the rest of the system. Adding internal security to a modular system would naturally sit at some of these interaction or interface boundaries allowing these larger reusable pieces to also be securable units. This is something that the work carried out for SSSM has done, the modular open-network software development framework of SSM has interface points where security was added, i.e. the networking component of the IPC. The idea being that the reusability of modules will be maintained and that the impact on reusability and security will not be paid for twice.

As mentioned participants perceive development costs to generally be negatively or neutrally affected. The negatively affected aspects were development cost, maintenance cost, development efficiency, and schedule. The additional complexity in terms of design and development time to add security are perceived to negatively impact development lifecycle as well increase the resource burden on a system. Table 4.24 shows these aspects in red and orange. SSSM adds internal security to a reusable open-network software development framework that helps to buy down some of these development and maintenance costs. SSSM also tries to minimize the impact on code complexity for the end users which should also help with code design, development costs, and maintenance costs.

Some aspects appear to be at odds, like a neutral rating on maintainability but a negative effect on maintenance cost. Development quality appears to show a discrepancy between negative and positive effects, normally one might expect that code design would be tied to code robustness, encapsulation, and best practices. However, developer perception is that code design is negatively affected while these aspects are positively affected. One possibility here is that the participants think of this as more of the time taken to do the design than the quality of the resultant design. It is also possible that the developers have a different understanding of the interrelationships of the different aspects. These seemingly contradictory perceptions suggest a need to better understand the development process of and developed product for space systems. A open-network software development framework that adds security can be used to better understand the real effect of adding internal security and may in turn alleviate some developer concern about adding internal security to a space system.

Many aspects were seen as not being impacted by internal security provisions. This is shown by the weak t-value of the *Overall* mean for all the aspects together as well as the large number of aspects that showed a neutral affect. These low magnitude t-values are shown in gray, for these the null hypothesis cannot be refuted. The development cost category had the largest set of aspects, development cost and schedule are often the constraining factors for a development effort. As space systems development schedules shrink it is important that overall assurance can increase by adding internal security provision without too much perceived impact on development cost.

4.7 Conclusions of Survey Analysis

Appendix B contains a more detailed analysis of the survey participants. It shows that the surveys were taken by a diverse group of space systems developers with good overall experience in software systems development, management, hardware systems development, and procurement. Section 4.3 showed that the participants perceive minimal code complexity to be important and beneficial generally and specifically for SFS. The participants felt that lack of cohesion and cyclomatic complexity metrics were the most important complexity metrics. Section 4.4 demonstrated that the developers believe reuse is important and beneficial for software developed for space systems. Section 4.5 showed a general consensus among participants that networking is beneficial to space systems and space systems development. This section also showed that space systems design and implementation is trending towards OSAs and MONAs at an organizational level. Section 4.6 showed that security concerns are a significant hurdle to adopting open systems approaches and that networking has a negative effect on security in space systems and space systems development. This section also showed that security is important to provide for space systems and security is more important for open-network space systems than it is for traditional point-to-point space systems.

The overall consensus among participating space systems developers is that there would be value in a reusable open-network software development framework that implements MONA and does not substantially increase code complexity.

CHAPTER 5
SSSM DESIGN

5.1 Introduction

The primary goal of the SSSM design is providing security while maintaining the reusability that is present in SSM. To do this the design keeps as many changes as possible behind the SSM's original API. The design and implementation of SSSM demonstrates a way to address both the *Development Problem* and *Security Problem*. Chapter 4 shows that developers feel that SSM addresses issues that relate to reuse, modular open-network system approaches, and networking, but there is room for improvement. Chapter 4 also shows that space systems developers believe there is value in securing a reusable open-network software development framework that implements MONA while trying to minimize the burden on the developer.

The security research augments SSM so that the benefits of a reusable, modular open-network software development framework can be realized while minimizing the negative impact on security. This research achieves this by adding security provisions to SSM, this modified version of SSM is called SSSM. This helps to address both the *Development* and *Security* Problems by providing the capability developers need and expect in modern development environments with additional easy-to-use security provisions. Section 5.2 covers some of the security problems that SSSM addresses for SSM. Section 5.3 covers the changes to SSM and protocol definitions used to provide the additional security provisions for SSM. Chapter 6 will discuss the difference, from the developer-perspective, between implementing a secured producer-consumer paring and an unsecured producer-consumer pairing. Chapter 6 will also present a performance evaluation that compares some resource metrics between a secured producer-consumer paring and a unsecured producer-consumer pairing.

5.2 SSM Security Problem

The design of SSM does not consider security. There are no provisions in place to restrict the flow of data or protect it while in transit, the same is true for commands. Figure 5.1 shows an example of a space system utilizing SSM for its Command and Data Handling (CDH) system. Traffic traverses the radio-link from the ground, this could also be space-to-space radio-link; new communication paradigms and interconnected space systems pose additional security issues over traditional ground-based communication. Multiple points of egress to a space system allow for increased access and availability for legitimate space systems users, but also increase space system exposure to exploitation.

Figure 5.1 shows a *Radio Manager* that integrates with the radio hardware on one side to send and receive communication and on the other side it relays this communication to the CDH stack making use of the open-network interface provided by SSM. Applications that schedule commands and handle various hardware subsystems make up the CDH stack, e.g. a file-management service, a ADCS management component, or a GPS receiver management component. In the event that a malicious ground station is able to interrogate the space system or a *malicious application (MA)* is able to infiltrate the system, specifically the perimeter is compromised, then there are no protections in place. If the *payload* shown in Figure 5.1 is a high-value target then numerous entities with considerable resources might be trying to gain access to it or otherwise compromise its mission. Figure 5.1 shows a MA, that in conjunction with a malicious ground station, or on its own, can compromise or spoof *payload* data by interjecting traffic. Such an application could compromise the functionality of the underlying SSM infrastructure, CDH systems, and the *payload* in order to make the resource unavailable, task the resource, or relay manipulated data.

A secure system should provide a set of key features, including identity management, mutual authentication, authorization, auditing, confidentiality, integrity, availability, management, and compliance. Figure 5.1 illustrates that the vanilla version of SSM does not provide significant coverage for any of these features; SSM was designed to be open in order to facilitate communication across different subnets and network types without consideration for security. SSSM addresses this security concern by providing a security layer that allows applications and components to confidentially communicate, as well as mutually authenticate. A modular open networked security solution needs to retain as many

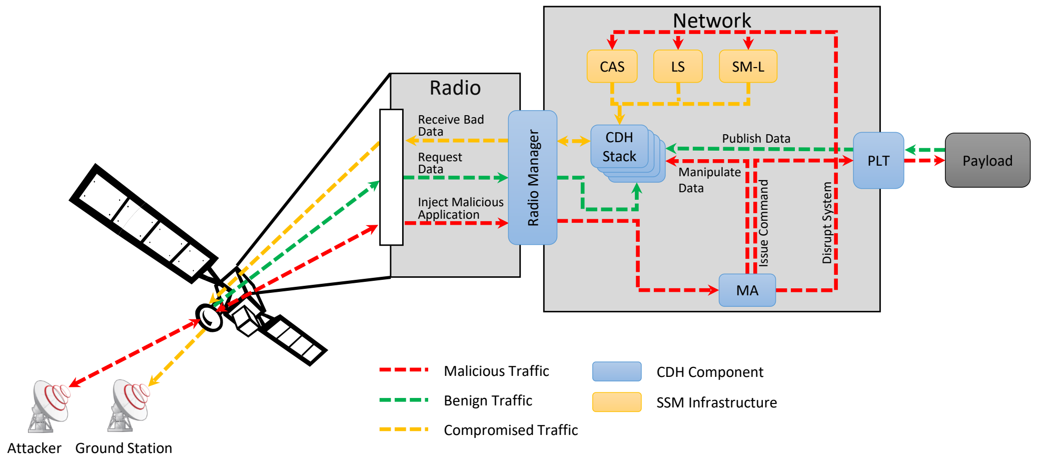


Fig. 5.1. Attack Scenario. Potential attack scenarios on a space system utilizing SSM for communication with a hosted payload and a malicious application.

of the benefits gained by using a modular open-network software development framework, e.g. the system needs to stay open, modular, ideally networked, and reusable. The design and implementation of SSSM demonstrates a way to add security to SSM and address the security problems and principles, that this research has discussed, while allowing developers to realize all the benefits they see in modular, reusable open-network software development frameworks. The design discussed in the next section seeks to enable the development of space systems that are more secure and that can have a level of compartmentalized and layered security. The design should provide this security while minimizing the impact on space systems developer and ideally on system resources.

5.3 SSSM Architectural and Protocol Design

The SSSM design protects traffic in transit, identifies components, authenticates components, and controls access to components. SSSM accomplishes this by borrowing concepts from Kerberos, adding a Permission Table, and utilizing AES-Galois/Counter Mode (GCM), as implemented in OpenSSL [49], for encryption and integrity verification. SSSM supports compilation with or without Federal Information Processing Standards (FIPS) 140–2 compliance [50]. SSM provides a LS which in turn provides a mechanism for applications and components to discover each other, share data or telemetry, and issue commands. SSSM augments the capabilities of the LS by adding functionality that allows for the authentication of applications and components using pre-shared keys. SSSM combines the functionality of the Kerberos Ticket Granting Server (TGS) and Authentication Server (AS), or theKey Distribution Center (KDC), into the SSM LS and uses pre-shared keys in place of passwords to authenticate each component that needs to participate in secure communication. Section C.2 of Appendix C explains the protocols Kerberos utilizes in more detail. The LS provides session keys for components to use to encrypt communication, as well as manage access between components at the interface-message level. The LS contains a Permission Table that it loads from a configuration file as depicted in Figure 5.2 and Sample 5.1 that allow the LS to control which entities can communicate securely on an interface-message level.

Figure 5.2 illustrates a basic example of SSSM capability and provides insight into the architecture of SSM and subsequently SSSM. SSSM architecture is explained in more detail in Section 5.3.1. Figure 5.2 shows three processing nodes running SSSM and communicating over SPA ethernet (SPA-E) (Ethernet); they could, in theory, be communicating over any other subnet combinations supported by SSM. The SPA-E subnet-manager (SM-E) components are the subnet-managers for Ethernet-based communication. The application level of the SSSM endpoints do not have any knowledge of the networks that a message traverses to reach its destination. This example is simple and the *Producer* would not typically provide the same data via a secured and unsecured interface, the example is shown this way for simplicity.

SSSM addresses each of the key security features, from Section 2.3, by adding the functionality listed below:

- **Identity management** — Identities of components are tied to universally unique identifiers and pre-shared keys
- **Mutual authentication** — Kerberos-like protocol using pre-shared keys[1]
- **Authorization** — Permission Table, session creation, and management by LS
- **Auditing** — SSM logging system to log pertinent events, e.g. failed authentication attempts
- **Confidentiality** — Industry standard AES-256 encryption
- **Integrity** — AES-256-GCM provides verification that the data was not tampered with
- **Availability** — Not addressed due to issues intrinsic to the current SPA networking standard and the scope of the problem[2]

Section 5.3.1 highlights the architectural changes and additions made to SSM. Section 5.3.2 explains the protocol exchanges that facilitate authentication, authorization, and secure communication between authenticated components.

5.3.1 Architectural Overview

SSSM adds the ability to house pre-shared keys for components, and permissions for controlling access and setting up sessions for secure communication to the LS. SSM provides the CAS, the LS, subnet-managers, and the SPA API for developing SPA applications in C++. SSSM updates these components so that a developer using the SPA API to write their applications can use the new security provisions with very few changes over non-secured versions of the same applications. The space systems software developer does not need to have any special security background when configuring their applications to use the security provisions. The developer needs to use secured versions of some classes in the API and needs to provide a permission table file. Sample 5.1 shows an example of a permission table. The

[1] Pre-shared keys could be changed to, or augmented by, public-key or certificate encryption schemes in future work.

[2] Availability is left to future work and requires changes to the underlying protocols of SSM.

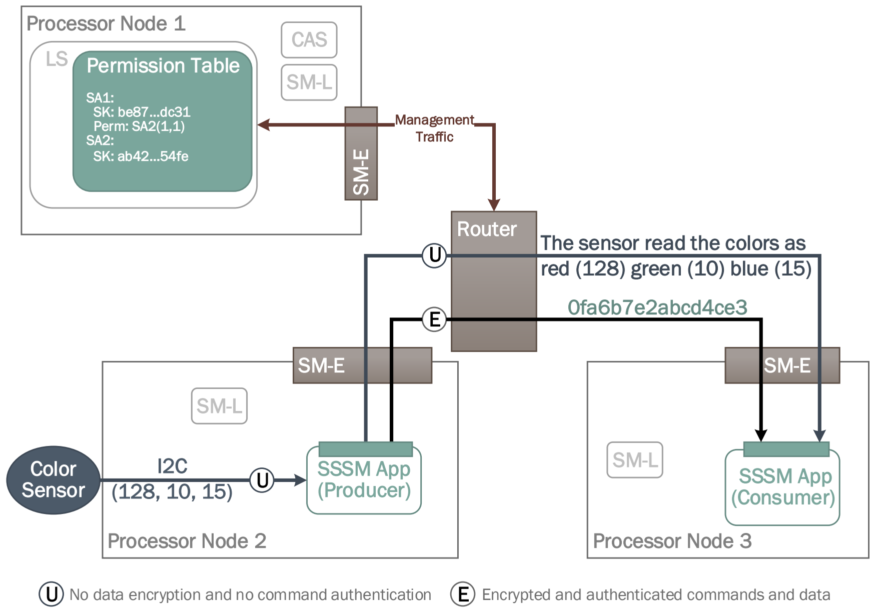


Fig. 5.2. SSSM Usage Example. An illustration of a potential network setup that utilizes features provided by SSSM. SSSM allows secured and unsecured traffic to share the same network.

rest of the changes are largely transparent to the developer. Chapter 6, Section 6.2 covers these developer-level deltas in more detail.

The CAS and LS are two of the most important services provided by SSM, these services do not need to reside on the same processing node or even the same subnet. The CAS gives logical address blocks out to each of the subnet-managers. The subnet-managers use these logical address blocks to give addresses to each of the components on their subnet. Together the LS and subnet-managers facilitate component discovery, data sharing, and commanding. This gives components the ability to publish their capabilities as shown in Sample 5.2. The permission table configuration file in Sample 5.1 relates to the example SSSM setup that Figure 5.2 depicts. Section 5.3.2 explains the protocol for setting up the secure exchange of information. The Permission Table configuration file houses the permission rules and pre-shared keys. The LS ingests this file on startup to configure the permission rules and pre-shared keys. In Sample 5.1, the application with the Universally Unique Identifier (UUID) ending in *aa33* on line 2 has permission to interact with the interface-message pairing of interface identifier 1 and message identifier 1, denoted as *(1,1)*, provided by the application with the UUID ending in *aa44* on line 10. This permission is shown on lines 5 and 6 where *aa44* is referred to as the *TargetComponent*. Sample 5.2 shows the xTEDS for *aa44*. The data provided by *(1,1)* is defined on line 9.

Both applications have a key to communicate with the LS, these symmetric keys are defined on lines 3 and 11; these lines also indicate that these applications require 32-byte or 256-bit keys for authentication and communication. These keys are only used during the authentication phase, another key is generated as part of a ticket to be used to communicate with the LS after authentication, and new keys will be generated for each session with other components or interface-message pairings. The lifetime on line 5 indicates that sessions with this interface-message pairing last 600 seconds or 10 minutes. Additional components and permissions can be defined following the same format. This file centralizes the configuration, allowing the components to only have knowledge of their own key. The LS now manages all of this. The LS is modified to handle key creation and management, ticket

```
 1 <PermissionTable>
 2   <SubjectComponent uuid='fccccb2a-30dc-604a-d5a8-2fbcccc3aa33'>
 3     <LookupServiceSymmetricKey key='be87d78d7f81e81a18aa1818569610874b12fefeb78eab389b73347891fddc31'
           preferredKeySize='32' />
 4     <TargetPermissionList>
 5       <TargetComponent uuid='faaace1a-30dc-604a-d5a8-2fbaaaa3aa44' lifetime='600' >
 6         <Permission interfaceId='1' messageId='1' />
 7       </TargetComponent>
 8     </TargetPermissionList>
 9   </SubjectComponent>
10   <SubjectComponent uuid='faaace1a-30dc-604a-d5a8-2fbaaaa3aa44'>
11     <LookupServiceSymmetricKey key='ab425cf198df81b981cbd6babb139847ab789bcdf78ebd78f548a865bcdd54fe'
           preferredKeySize='32' />
12     <TargetPermissionList />
13   </SubjectComponent>
14 </PermissionTable>
```

Sample 5.1. Permission Table for Color Producer Configuration. Permission Table that configures the LS and allows the component with UUID ending in *aa33* to communicate with *aa44* on *(1,1)*.

creation and management, session management, encryption, permission management, and authentication.

The two SSSM Apps depicted in Figure 5.2 illustrate a producer-consumer relationship that is very common, specifically in the context of SSM and CDH systems, but also generally. This is a generic example meant to illustrate a capability, as a color sensor is not generally a sensor in use in a CDH system. A developer builds these applications using the SPA API that provides a `SpaApplication`. In order to produce an application that provides the xTEDS interface depicted in Sample 5.2 the `SpaApplication` class is coupled with the xTEDS in question. The `SpaApplication` sets up and performs all the network side functionality needed to participate on a SSM network. It registers with its subnet-manager, registers its xTEDS with the LS, and provides callbacks to respond to registered commands and requests for notifications. The example xTEDS shows two interfaces: *(1,1)* and *(2,1)*. These are both notifications, meaning that data is produced and a consumer may subscribe to the data. Lines 9 and 16 of Sample 5.2 indicate that data is published periodically and that the data is a byte array. The *(1,1)* interface on line 9 is secured and the *(2,1)* interface on line 16 is not secured.

In this xTEDS example both interfaces produce the same data, this is purely for illustrative purposes. The xTEDS does not indicate if a given interface is secured or not secured, the names here are for illustrative purposes, nor does it indicate who is allowed to use a given interface. The designation of security and permission is made in the Permission Table configuration file, enforced by the LS, and in the code of the Producer or owner of the interface. In this case the developer uses the secured `SecureNotificationMsg` class for interface *(1,1)* and the standard `NotificationMsg` class for *(2,1)*. The secured notification on *(1,1)* requires the use of the protocols described in Section 5.3.2 and the notification on *(2,1)* works the same as it always has in SSM. Both interfaces require setup. The producer registers the notification via xTEDS with the LS and the consumer must issue a query for the notification. After the setup is complete the use of and production of the data is largely identical for both the secured and unsecured versions. The updated SPA API makes the authentication and encryption transparent to the user.

SSSM provides encryption functions and wrappers for developers as utilities, but developers do not need to use them directly. SSSM provides secured versions of various message classes, e.g. `SecureCommandMsg` and `SecureRequestMsg` to securely support commands without a response and commands that expect a response, respectively. SSM and the SPA

API also make use of various communicator classes that have been updated to handle the secured variants of SSM communication. In short, the SPA API and any associated communicators have been updated to provide for authentication with the LS and other components, requesting sessions, handling sessions, and encryption.

```xml
<?xml version='1.0' encoding='utf-8' ?>
<xTEDS xmlns='https://pnpsoftware.sdl.usu.edu/redmine/projects/xtedsschema'
       xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
       xsi:schemaLocation= 'https://pnpsoftware.sdl.usu.edu/redmine/projects/xtedsschema
            https://pnpsoftware.sdl.usu.edu/spa/xteds/current.xsd'
       name='SecureProducerXteds' version='1.0'>
  <Application name='SecureProducer' kind='application' programMemoryRequired='1' dataMemoryRequired='1'
        componentKey='ProducerComponentKey'/>
  <Interface name='SecureData' id='1' >
    <Notification>
      <DataMsg name='SecureData' id='1' msgArrival='PERIODIC' msgRate='0.2'>
        <DynamicArray name='secureString' kind='color' dataType='UINT8' maxArrayElements='1000'
             units='unitless' description='Some color data' />
      </DataMsg>
    </Notification>
  </Interface>
  <Interface name='PlaintextData' id='2' >
    <Notification>
      <DataMsg name='PlaintextData' id='1' msgArrival='PERIODIC' msgRate='0.2'>
        <DynamicArray name='plaintextString' kind='color' dataType='UINT8' maxArrayElements='1000'
             units='unitless' description='Some color data' />
      </DataMsg>
    </Notification>
  </Interface>
</xTEDS>
```

Sample 5.2. Secured Color Producer xTEDS. An xTEDS with an unsecured and secured nofication for the same data showing that the xTEDS definition is the same for both.

The subnet-managers, e.g. SM-E and SPA-L subnet-manager (SM-L), facilitate communication on the given subnet and between different subnets. Figure 5.2 shows the SM-E and SM-L subnet-managers passing both secured and unsecured communication. Updates have been made to these subnet managers to facilitate secured communication, e.g. handling of a secure header.

Common secured message classes have been added with SSSM that make use of the extended header capability of the SSM messaging system. The extended header capability of SSM allows an arbitrary number of headers to be added to any message, SSSM makes use of this functionality to designate security functionality. Messages classes for secure commanding and data sharing have been added as well as message classes for authentication, permission exchange, and session creation. Other utility classes like a ticket class have been added to encapsulate that functionality, as well as an encryption class for wrapping access to the OpenSSL libraries. These modifications and additions make it easy for a developer writing a SPA Application to add the ability to communicate securely on specific interface-message pairings. Section 5.3.2 details the protocols used for secure communication. From an architectural standpoint no new services or components were added, but existing services like the LS were augmented. Existing classes were augmented and new classes were added to support secure communication. Section 5.3.2 gives an overview of how the secured components authenticate themselves with the LS and subsequently communicate securely with other components.

5.3.2 Protocol Overview

SSSM models its authentication process after Kerberos. Kerberos can use symmetric keys, passwords, or even public-key encryption to establish identity and subsequently grant tickets that can be used to open sessions with services. SSSM makes use of pre-shared symmetric keys, these keys are used only during initial communication with the LS or if a component's authentication expires and the component needs to re-authenticate; the idea being to limit the usage of this authenticating and identifying key. This identifying key can be updated out-of-band, but currently there is no method for an in-band or online method for changing this key.

Figure 5.3 continues the example used in previous sections and shown in Figure 5.2 where the *Consumer*, or SSSM *App (SA1)*, wants to subscribe to data from the *Producer*, or (*SA2*). The messages that Figure 5.3 depicts do not include all the `SpaMessage` specific header and footer message components that wrap the secured portions of the messages, certain parts have been left off for brevity and clarity. The diagram starts after the components have been discovered on the network, i.e. contacted their subnet-managers and received a logical address and started the registration process with the LS, but before they have finished the registration process. This process works the same whether a components has secured interfaces or not. The *Authentication* block shows the authentication process for *SA1*, but the process is the same for *SA2*. *SA1* authenticates with the LS via a handshake and receives a key that it uses for requesting access to other services or components.

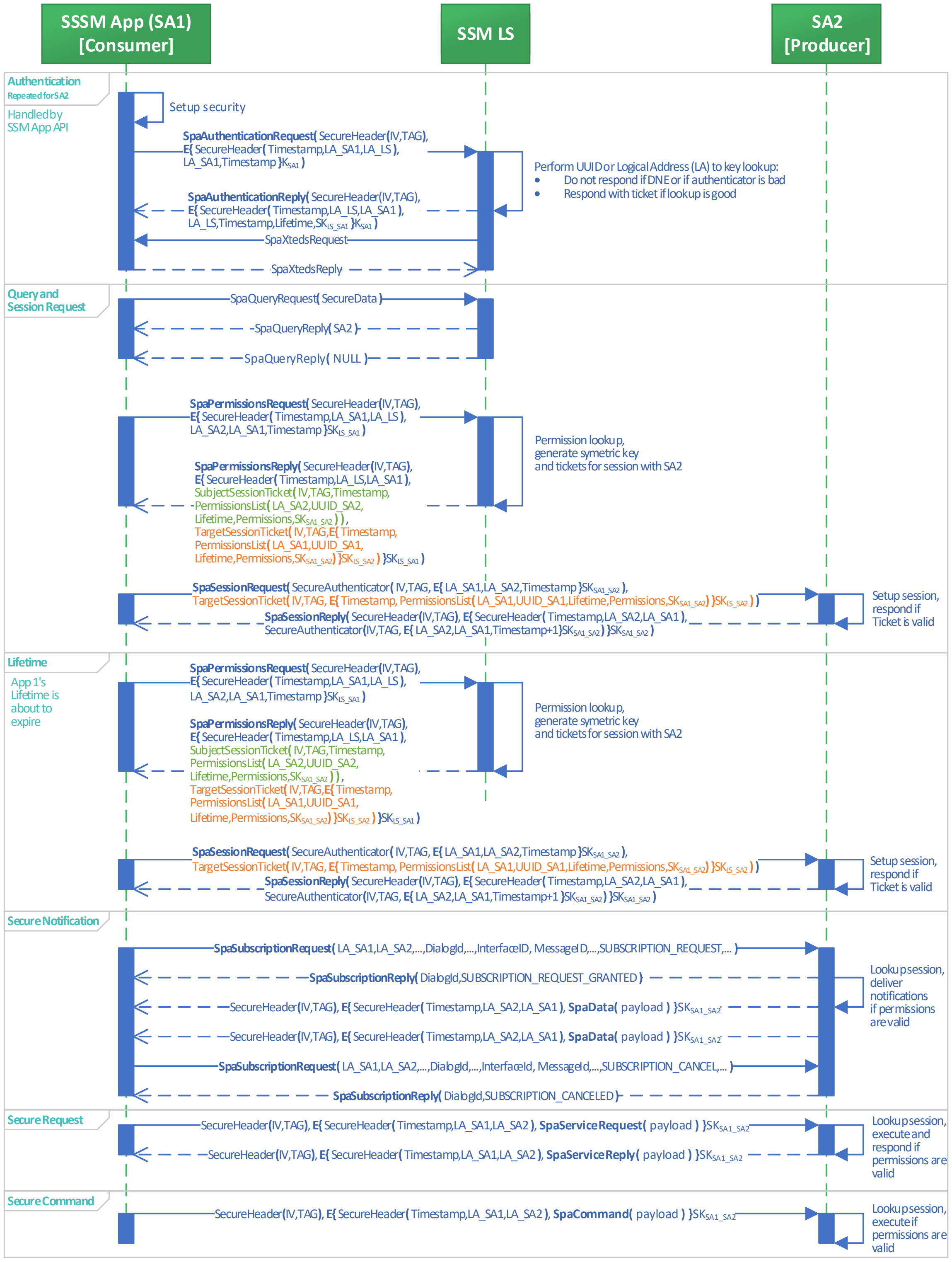

SSSM App (SA1) [Consumer]
SSM LS
SA2 [Producer]
Authentication
Repeated for SA2
Handled by SSM App API
Setup security
SpaAuthenticationRequest(SecureHeader(IV,TAG), E{ SecureHeader(Timestamp,LA_SA1,LA_LS), LA_SA1,Timestamp }K_{SA1})
Perform UUID or Logical Address (LA) to key lookup:
• Do not respond if DNE or if authenticator is bad
• Respond with ticket if lookup is good
SpaAuthenticationReply(SecureHeader(IV,TAG), E{ SecureHeader(Timestamp,LA_LS,LA_SA1), LA_LS,Timestamp,Lifetime,SK_{LS_SA1} }K_{SA1})
SpaXtedsRequest
SpaXtedsReply
Query and Session Request
SpaQueryRequest(SecureData)
SpaQueryReply(SA2)
SpaQueryReply(NULL)
SpaPermissionsRequest(SecureHeader(IV,TAG), E{ SecureHeader(Timestamp,LA_SA1,LA_LS), LA_SA2,LA_SA1,Timestamp }SK_{LS_SA1})
Permission lookup, generate symetric key and tickets for session with SA2
SpaPermissionsReply(SecureHeader(IV,TAG), E{ SecureHeader(Timestamp,LA_LS,LA_SA1), SubjectSessionTicket(IV,TAG,Timestamp, PermissionsList(LA_SA2,UUID_SA2, Lifetime,Permissions,SK_{SA1_SA2})), TargetSessionTicket(IV,TAG,E{ Timestamp, PermissionsList(LA_SA1,UUID_SA1, Lifetime,Permissions,SK_{SA1_SA2}) }SK_{LS_SA2}) }SK_{LS_SA1})
SpaSessionRequest(SecureAuthenticator(IV,TAG, E{ LA_SA1,LA_SA2,Timestamp }SK_{SA1_SA2}), TargetSessionTicket(IV,TAG, E{ Timestamp, PermissionsList(LA_SA1,UUID_SA1,Lifetime,Permissions,SK_{SA1_SA2}) }SK_{LS_SA2}))
Setup session, respond if Ticket is valid
SpaSessionReply(SecureHeader(IV,TAG), E{ SecureHeader(Timestamp,LA_SA2,LA_SA1), SecureAuthenticator(IV,TAG, E{ LA_SA2,LA_SA1,Timestamp+1}SK_{SA1_SA2}) }SK_{SA1_SA2})
Lifetime
App 1's Lifetime is about to expire
SpaPermissionsRequest(SecureHeader(IV,TAG), E{ SecureHeader(Timestamp,LA_SA1,LA_LS), LA_SA2,LA_SA1,Timestamp }SK_{LS_SA1})
Permission lookup, generate symetric key and tickets for session with SA2
SpaPermissionsReply(SecureHeader(IV,TAG), E{ SecureHeader(Timestamp,LA_LS,LA_SA1), SubjectSessionTicket(IV,TAG,Timestamp, PermissionsList(LA_SA2,UUID_SA2, Lifetime,Permissions,SK_{SA1_SA2})), TargetSessionTicket(IV,TAG,E{ Timestamp, PermissionsList(LA_SA1,UUID_SA1, Lifetime,Permissions,SK_{SA1_SA2}) }SK_{LS_SA2}) }SK_{LS_SA1})
SpaSessionRequest(SecureAuthenticator(IV,TAG, E{ LA_SA1,LA_SA2,Timestamp }SK_{SA1_SA2}), TargetSessionTicket(IV,TAG, E{ Timestamp, PermissionsList(LA_SA1,UUID_SA1,Lifetime,Permissions,SK_{SA1_SA2}) }SK_{LS_SA2}))
Setup session, respond if Ticket is valid
SpaSessionReply(SecureHeader(IV,TAG), E{ SecureHeader(Timestamp,LA_SA2,LA_SA1), SecureAuthenticator(IV,TAG, E{ LA_SA2,LA_SA1,Timestamp+1 }SK_{SA1_SA2}) }SK_{SA1_SA2})
Secure Notification
SpaSubscriptionRequest(LA_SA1,LA_SA2,...,DialogId,...,InterfaceID, MessageID,...,SUBSCRIPTION_REQUEST,...)
Lookup session, deliver notifications if permissions are valid
SpaSubscriptionReply(DialogId,SUBSCRIPTION_REQUEST_GRANTED)
SecureHeader(IV,TAG), E{ SecureHeader(Timestamp,LA_SA2,LA_SA1), SpaData(payload) }SK_{SA1_SA2}
SecureHeader(IV,TAG), E{ SecureHeader(Timestamp,LA_SA2,LA_SA1), SpaData(payload) }SK_{SA1_SA2}
SpaSubscriptionRequest(LA_SA1,LA_SA2,...,DialogId,...,InterfaceId, MessageId,...,SUBSCRIPTION_CANCEL,...)
SpaSubscriptionReply(DialogId,SUBSCRIPTION_CANCELED)
Secure Request
SecureHeader(IV,TAG), E{ SecureHeader(Timestamp,LA_SA1,LA_SA2), SpaServiceRequest(payload) }SK_{SA1_SA2}
Lookup session, execute and respond if permissions are valid
SecureHeader(IV,TAG), E{ SecureHeader(Timestamp,LA_SA1,LA_SA2), SpaServiceReply(payload) }SK_{SA1_SA2}
Secure Command
SecureHeader(IV,TAG), E{ SecureHeader(Timestamp,LA_SA1,LA_SA2), SpaCommand(payload) }SK_{SA1_SA2}
Lookup session, execute if permissions are valid

Fig. 5.3. SSSM Protocol Diagram. An illustration of the protocols employed to allow components or applications to authenticate with the LS and subsequently establish sessions to receive tickets and communicate with other components.

The next block shows the *Query and Session Request* protocol, *SA1* queries for the data or command interfaces it wants using the LS, this querying process is already part of SSM. In this case *SA1* wants to communicate with a secure interface on *SA2*. *SA1* asks the LS for a ticket it can use to communicate with *SA2*.

The *Lifetime* block depicts the exchange that occurs when a session is expiring and a component wishes to get another session to maintain communication. Figure 5.2 only use a secure notification but the protocol diagram depicts the exchange for secure notifications, secure requests, and secure commands in the *Secure Notification*, *Secure Request*, and *Secure Command* blocks of the protocol diagram in Figure 5.3.

5.3.2.1 Authentication

A secure component participates on the SSM by authenticating itself with the LS, all components already need to register with the LS so this is a natural extension of that process. If the LA of a component is already known then a malicious component could try to subvert the process, but they will fail to negotiate a session with the component without a valid session ticket from the LS. In this example *SA1* needs to authenticate with the LS. Figure 5.3 depicts the process that a component uses to authenticate with the LS in the *Authentication* block. *SA1* sends a `SpaAuthenticationRequest` to the LS, the `SpaAuthenticationRequest` is an extension of the `SpaMessage` that is part of SSM. The `SecureHeader` employed by SSSM bridges the encryption boundary, the Initialization Vector (IV) and tag are transmitted in-the-clear while the timestamp, source address (*LA_SA1*), and destination address (LA_LS) are encrypted along with the rest of the `SpaAuthenticationRequest` payload. Encrypting the timestamp and addresses reduces the feasibility of spoofing and replay attacks as well as facilitates authentication. The timestamp is carried out to the nanosecond and doubles as a nonce for replay attacks. While the source and destination addresses make it more difficult for one component to meaningfully masquerade as another component. These addresses have to match the addresses in the regular `SpaMessage` header that are used for routing.

The authenticator portion of the SpaAuthenticationRequest message follows the encrypted portion of the SecureHeader borrowing the authenticator concept from Kerberos. *SA1* encrypts the secured portion of the message with the K_{SA1} key, this is the pre-shared key that only the LS and *SA1*. *SA1* sends the message to the LS, at that point it is the job of the LS to use the LA in the SpaMessage header to look up the component and its pre-shared key, it can fall back to the component UUID if the component has not registered with a LA yet, but this should have already been matched during the probing process which is not depicted here. The LS simply does not reply if the lookup fails, this reduces the load if there is an attack, and makes it harder for a malicious entity to gain any information.

If the LS finds the component in its lookup table, then it uses the K_{SA1} to decrypt the encrypted portion of the SpaAuthenticationRequest, if the message passes validation then the LS crafts a SpaAuthenticationReply to send back to *SA1*. The SpaAuthenticationReply is basically like the SpaAuthenticationRequest except for the authenticator portion of the message at the end. The LS uses its LA, a new timestamp, a lifetime for which the new key is good, and a new key. The new key, denoted as K_{LS_SA1}, is generated by the LS. This key is used for the duration of the lifetime specified, at which point the component has to re-authenticate with the LS and get a new key. This is all encrypted with K_{SA1}; both parties have to know K_{SA1} for this exchange to work. Successful decryption on both sides coupled with the authenticator portion of the messages allows for mutual authentication, i.e. *SA1* can also be sure that it is talking to the real LS. This completes the authentication phase at which point the LS submits a SpaXtedsRequest to *SA1* asking for its xTEDS, and *SA1* sends it using a SpaXtedsReply. Both secured and unsecured components participate in the xTEDS exchange. The same process is carried out for *SA2*. Once a component has authenticated itself with the LS, it can request permissions for communicating with other secured components.

5.3.2.2 Query and Session Request

In this phase *SA1* issues a query for the data it wishes to consume. *SA1* queries for the *SecureData* interface using a SpaQueryRequest, this process works basically the same as it does in vanilla SSM. Figure 5.3 illustrates a specific case of a query for the *SecureData* interface as described in Sample 5.2. Queries work the same for other types of interfaces, i.e. commanding or requesting. The LS responds with as many `SpaQueryReply` messages as there are components that match the query, in this example there is only one. It is up to the application developer to decide if they want to purely go with the first one or use other criteria to decide which one to use if multiple providers exist, there is also a query for all option which allows a developer to source all the providers.

SA1 chooses a producer or service provider, in this case *SA2*, and then it must request the set of permissions it has for *SA2*. *SA1* makes this request to the LS using a `SpaPermissionsRequest` message. The `SpaPermissionsRequest` contains a `SecureHeader` like before, but instead of including the *LA_LS* in the rest of the `SpaPermissionsRequest` it contains *LA_SA2*, the LA of the component to which *SA1* wishes to subscribe, also known as the target. *SA1* encrypts this request with SK_{LS_SA1} to protect it in transit and contains the source address and timestamp to authenticate the sender as *SA1*.

The LS processes the `SpaPermissionsRequest` by decrypting it via LA to encryption key lookup, the LS then uses the addresses in the request and looks up *LA_SA1* as the subject and *LA_SA2* as the target to determine what permissions *SA1* has with regard to *SA2*. The LS packages up two permissions list in the reply to *SA1*. One list is for *SA1* and this list has all the permissions *SA1* has for *SA2*, this means that *SA1* does not have to issue separate permission requests for each *SA2* interface it wishes to use. The other permission list is for *SA2* and tells *SA2* what access it should grant to *SA1*. These list become part of the session tickets that the LS builds for the `SpaPermissionsReply`, *SA1* and *SA2* each get a session ticket.

The LS constructs a `SubjectSessionTicket` for *SA1* and a `TargetSessionTicket` for *SA2*. These tickets are based off of the ticket employed by Kerberos and are signed by the LS. These tickets allow *SA1* to show it has permission to communicate with *SA2* on specific interfaces as controlled by the LS. At the same time the tickets allow *SA1* to authenticate itself with *SA2* and give *SA2* the encryption key they use to communicate over the duration of their session.

The SubjectSessionTicket is for *SA1*; the PermissionList returned is from the perspective of *SA1* and includes information about the permissions that *SA1* has with the *SA2*. This ticket is not encrypted separately, but is encrypted in transit by $\mathrm{SK}_{\mathrm{LS\_SA1}}$ because the entire message, sans portions of the SecureHeader, is wrapped in encryption by the LS. *SA1* can decrypt the encrypted portion of the SpaPermissionsRequest message and retrieve the SubjectSessionTicket. This gives *SA1* the permissions it has with *SA2*, but the SubjectSessionTicket also provides two other critical pieces of information. It provides the encryption key that *SA1* and *SA2* use to communicate as well as the *Lifetime* over which the session key is viable. When the lifetime expires *SA1* has to make a new SpaPermissionsRequest and get a new SubjectSessionTicket from the LS and essentially repeat the session setup again; the *Lifetime* block of Figure 5.3 shows this process.

The TargetSessionTicket is for *SA2*; the PermissionList returned is from the perspective of *SA2* and lets *SA2* know which permissions it should grant to *SA1*. This ticket is encrypted with $\mathrm{SK}_{\mathrm{LS\_SA2}}$, but is also wrapped in encryption by the whole message encryption, this means that only *SA1* can remove the outer layer and then get the TargetSessionTicket that it needs to send to *SA2* as part of the SpaSessionRequest sent below. Only *SA2* can decrypt the TargetSessionTicket because it is encrypted with $\mathrm{SK}_{\mathrm{LS\_SA2}}$, *SA1* cannot decrypt this, but merely relays it when making a session request to *SA2*.

The last exchange of the *Query and Session Request* protocol is the setup of a session between *SA1* and *SA2*. They use the tickets provided by the LS to setup the session. *SA1* sends a SpaSessionRequest to *SA2*, this request contains a SecureAuthenticator and the TargetSessionTicket. SpaSessionRequest does not use a SecureHeader because *SA2* does not have a key with which to decrypt it until after the message is decrypted. The SecureAuthenticator can be used to validate the sender's identity and ties it to *LA_SA1*, and it is encrypted with $\mathrm{SK}_{\mathrm{SA1\_SA2}}$. Using $\mathrm{SK}_{\mathrm{SA1\_SA2}}$ means that only *SA1* or the LS could have created the SecureAuthenticator. *SA2* cannot get $\mathrm{SK}_{\mathrm{SA1\_SA2}}$ until it successfully uses $\text{SK}_{\text{LS\_SA2}}$ to decrypt the `TargetSessionTicket`. At this point *SA2* can verify the `SecureAuthenticator`, i.e. that it decrypts with $\text{SK}_{\text{SA1\_SA2}}$, that the source and destination address match, and that the timestamp is within allowance. Once *SA2* decrypts `TargetSessionTicket` it has the `PermissionList` for *SA1*, $\text{SK}_{\text{SA1\_SA2}}$, and the *Lifetime* of the session. *SA2* is now able to decrypt the `SecureAuthenticator`, this allows *SA2* to authenticate *SA1* as the sender of the message and tie this identity to *LA_SA1*, it also gives *SA2* the timestamp it uses in its response message so that *SA1* can authenticate *SA2* and tie its identity to *LA_SA2*.

SA2 uses the timestamp from the `SecureAuthenticator` and $\text{SK}_{\text{SA1\_SA2}}$ to craft its response in the form of a `SpaSessionReply`. *SA2* takes the timestamp from the `SecureAuthenticator` and adds one to it and returns it in its `SecureAuthenticator`. The `SpaSessionReply` message allows *SA1* to authenticate *SA2* when it receives the `SpaSessionReply` and decrypts it using $\text{SK}_{\text{SA1\_SA2}}$. Now both parties have authenticated each other and exchanged permissions, *SA1* can request data from and issue commands to *SA2*. This is a one way session, i.e. *SA1* can request secured data from and issue secured commands to *SA2*, but it does not allow for communication going the other way. *SA2* cannot request secured data from or issue secured commands to *SA1* after this session setup. *SA2* would need to repeat the same *Query and Session Request* protocol exchange for *SA1* in order to establish a session going the other way. Both parties can participate in unsecured communication, these restrictions only deal with secured messaging. *SA1* can proceed with secured notifications, requests, or commands as shown in the *Secure Notification*, *Secure Request*, and *Secure Command* blocks of Figure 5.3. This is provided that the permissions exchanged allow for these types of communication on *SA2* interfaces. Figure 5.2 shows that *SA1* is interested in secure notifications from *SA2*, specifically color data on (1,1) from the color sensor.

5.3.2.3 Lifetime

The *Lifetime* block of Figure 5.3 shows the protocol exchange that takes place when a session between two secure components has expired. It is basically the same as the *Query* *and Session Request* block, except that the queries do not have to be issued again. Both parties have knowledge of when the *Lifetime* expires because it is delivered in the session tickets from the LS. This does require a certain level of synchronization in the time used by the components, but *SA2* drops the ticket and *SA1* requests a new one.

5.3.2.4 Secure Notification

The *Secure Notification* block of Figure 5.3 shows the secured notification exchange that can take place after a successful session establishment. The *Query and Session Request* block depicts a query for *SecureData*, this is the name of the notification interface depicted on line 7 of Sample 5.2 and used in Figure 5.2. The developer of *SA1* makes the query using a `SecureNotificationMsg` to be populated upon a successful query result; the developer of *SA2* needs to make use of the `SecureNotificationMsg` so that the `SpaApplication` and supporting objects know to publish the notification securely. Using a `SecureNotificationMsg` allows the `SpaApplication` and supporting objects to track the subscription and handle the encrypted `SpaData` messages while the subscription is active. This sequence diagram shows the subscription being negotiated directly between *SA1* and *SA2*, but it can also be issued and brokered by way of the LS.

After *SA1* completes the *Query and Session Request* exchange successfully it makes a request to *SA2* to subscribe to data that *SA2* provides, assuming *SA1* has the necessarily permissions. To do this *SA1* checks for the existence of a valid session, otherwise it requests one, and then sends a regular `SpaSubscriptionRequest` completely in the clear to *SA2*. This message could be encrypted and contain a `SecureHeader`, but initial analysis suggested that this would be invasive and difficult given the knowledge that each layer has along the communication path. The main downside here is that a malicious party could essentially carry out denial-of-service (DoS) attacks by either creating more subscription requests than a component can handle, or by canceling other components active subscriptions. Subscribing on behalf of other components is allowed by the SPA Logical Interface Standard [51] and is used by some subnet-managers; this makes it a difficult problem to address without changing existing protocol definitions. The current plan is to leave this problem for future work as it may require consensus from the SSM community. The malicious party still cannot decipher that data because the resulting notification in the form of an encrypted `SpaData` message still requires the $\mathrm{SK}_{\mathrm{SA1\_SA2}}$ key for decryption, it would also be difficult to actually have the resultant message routed to them because the sessions are tied to a LA, and it also requires that a valid session exists.

Figure 5.3 shows *SA2* receiving the `SpaSubscriptionRequest` directly and handling it much the same way any subscription request is handled in SSM, i.e. *SA2* does not perform a permission check when determining how to respond to the request from *SA1*. In general, the `SpaSubscriptionRequest` results in a `SUBSCRIPTION_REQUEST_GRANTED` `SpaSubscriptionReply`, but a negative response could be returned if resources are maxed out or if *SA2* was shutting down; *SA1* can also end the subscription at any time for similar reasons. The `SpaSubscriptionRequest` contains a dialog identifier that is used in the `SpaSubscriptionReply` so that *SA1* can track which subscription has been granted. The `SpaSubscriptionRequest` contains other information used to setup the subscription, but does not carry any security specific data. It does carry the subscriber or consumer address, *LA_SA1*, which is used to perform a permission lookup before data is actually published. *SA2* sends off the `SpaSubscriptionReply` and starts flowing notifications according to the notification parameters, e.g. publish frequency and delivery divisor. In the case of a secured notification this also entails a permissions lookup. If the permission is valid then the `SpaData` message has a `SecureHeader` added and is encrypted using $\mathrm{SK}_{\mathrm{SA1\_SA2}}$ before being sent off to *SA2* as depicted in the *Secure Notification* block of Figure 5.3. If the permission is not valid then no data flows. Figure 5.3 shows a couple of message but this could continue until the lease expires (SSM construct), the session expires (SSSM construct), or until either party chooses to terminate the subscription for whatever reason.

The *Secure Notification* block of Figure 5.3 shows *SA1* canceling the subscription at the end after two `SpaData` messages have been passed. The process to request a subscription and to cancel a subscription are largely the same except that the `SpaSubscriptionRequest` message is initiated with a `SUBSCRIPTION_CANCEL` flag by *SA1* and the `SpaSubscriptionReply` contains a `SUBSCRIPTION_CANCELED` enumeration to relay that the subscription has been canceled. No additional authentication or validation is performed before a subscription is canceled.

5.3.2.5 Secure Request

SA1 issues a query for an interface that provides a SPA request interface, this is in place of or in addition to the *SecureData* query made in the *Query and Session Request* block of Figure 5.3. The application developer needs to make and register the query using a `SecureRequestMsg` as the underlying message type so that the `SpaApplication` object and underlying communicators can properly handle the message. In keeping with with the Figure 5.2 example, the SPA request interface is provided by *SA2*; note that our scenario only shows a SPA notification interface, but the concept is largely the same.

In a SPA request a command is issued in the form of a `SpaServiceRequest` and a response is received in the form of a `SpaServiceReply`. Both of these messages allow for the passing of parameters, if specified. *SA1* has to setup permissions before a secure `SpaServiceRequest` can be successfully issued to *SA2*.

As mentioned, the developer uses `SecureRequestMsg`, but this message is converted to a standard `SpaServiceRequest` message in transit. The *Secure Request* block illustrates a *Service Request* exchange between *SA1* and *SA2*, and assumes that proper query and permission setup has already occurred. The protocol makes use of a `SecureHeader`, this header caries the IV and tag used by AES-GCM in the clear and the rest of the header and the `SpaServiceRequest` payload is encrypted using SK_{SA1_SA2}. It should be noted that there is a primary `SpaMessage` header that comes first for all messages, which carries the source and destination address as well, the source address is used to perform the encryption key lookup and both source and destination addresses must match the ones in the `SecureHeader` after decryption.

SA1 sends an encrypted `SpaServiceRequest` to *SA2*. *SA2* looks up the encryption key and decrypts the `SecureHeader` and message. If the header passes validation then *SA2* acts on the message and provides a reply. This reply is in the form of an encrypted

`SpaServiceReply` with a `SecureHeader`. This protects the reply in transit and allows *SA1* to validate and authenticate the reply. However, if the header fails validation then no reply is issued and *SA2* simply drops the `SpaServiceRequest`. Likewise, if the `SecureHeader` fails validation in the `SpaServiceReply` then *SA1* discards the reply.

5.3.2.6 Secure Command

Issuing a secured command works much like issuing a secure request except that a `SecureCommandMsg` is used and no reply is provided or expected. In order to initiate a secured command *SA1* creates its query with a `SecureCommandMsg` to be populated upon a successful query result as described in the query section of the *Query and Session Request* block of Figure 5.3. Using a `SecureCommandMsg` allows the `SpaApplication` object and underlying communicators to correctly handle and encrypt the message before it is transmitted. The API converts the `SecureCommandMsg` to a standard `SpaCommand` message, adds a `SecureHeader`, and encrypts the message as shown in Figure 5.3.

The *Secure Command* block illustrates the command exchange between *SA1* and *SA2*, and assumes that a proper query and permission setup has occurred. *SA1* sends an encrypted `SpaCommand` with `SecureHeader` to *SA2*. *SA2* looks up the encryption key using the source in the `SpaMessage` header and decrypts the `SecureHeader` and message. If the header passes validation then *SA2* acts on the message. Pass or fail no response is given to *SA2*, this is the same for unsecured commands.

CHAPTER 6
SSSM EVALUATION

6.1 Introduction

This research evaluates the change in developer and system resource burden when developing unsecured versus secured versions of the same producer and consumer software using SSSM. An increase in developer burden reduces the reusability of SSM as it makes it harder for developers to use the software which in turn increases the cost in terms of dollars and schedule to develop software; all this makes reuse of the software less attractive. An increase in system resource utilization reduces the set of hardware, typically thought of in terms of a single board computer (SBC) or set of processing nodes, that have enough margin to accommodate the software. This chapter discusses both of these burdens in terms of SSM versus SSSM or unsecured versus secured to better understand the effects of adding security to a reusable modular open-network software development framework.

Section 6.2 outlines the differences in developing secured interfaces in SSSM versus unsecured interfaces in SSM. This comparison shows that the difference is minimal in terms of actual application development with the main differences being a configuration file that must be created by developers for the LS and the provisioning of keys for any secured end-points. Section 6.3 presents the results of the side-by-side performance of SSM and SSSM. These results show an increase in CPU and memory utilization and decreased payload byte throughput particularly at higher message throughputs. This means that reusability may be reduced because performance requirements and system resources need to be more carefully considered for SSSM than they do for SSM. These results also show that there may be some ways to address this without changes to the API that might increase developer burden and/or that future work may need to fully address this divergence. Section 6.4 concludes the chapter by summarizing the findings of this evaluation.

6.2 Unsecured versus Secured Development

This section explains the difference, from the perspective of a developer, between developing an interface that is unsecured and one that is secured in SSSM. This shows the relatively low impact that adding security to SSM has on the application developer. Developer cost is a significant driver of a space system development budget and minimizing impact here is critical to maintaining the reusability of SSSM. This section uses an example to cover three pieces of the development process. They are developing a producer; developing a consumer; and, in the case of the secured variant, creating the permissions table that the LS uses to authenticate secured components, lookup permissions, and issue session tickets.

6.2.1 Producer Development

This comparison uses the xTEDSs defined in Sample 6.1 and Sample 6.2. Sample 6.1 is implemented by the unsecured *API* producer, named *ApiProducer*, and Sample 6.2 is implemented by the secured API producer, named *ApiSecureProducer*.

The xTEDSs are functionally the same. The only differences between the two are the names of the application, the interfaces, and the messages. *Secure* has been prefixed to each of the interfaces and messages, and inserted into the application name in the secured version. This example uses these differing names for illustrative purposes, the *Secure* prefix is not a requirement for using SSSM. There is no functional difference in the xTEDS definition for unsecured versus secured development. The *ApiProducer* and *ApiSecureProducer* applications will incorporate and implement these xTEDSs. They are included as part of the binary so the API can be provide them to the LS during registration. The development framework includes a Python script that takes the xTEDS file as input and creates a c-string and a Extensible Transducer Electronic Data Sheet Universally Unique Identifier (XUUID) that can be used in the application. This process works the same for the secured and unsecured versions.

These xTEDSs both show two interfaces. The command, requests, and notifications do not have to be separated this way, but in this example the **CounterManagement* interface,

```
<?xml version='1.0' encoding='utf-8' ?>
<xTEDS xmlns='https://pnpsoftware.sdl.usu.edu/redmine/projects/xtedsschema'
       xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
       xsi:schemaLocation= 'https://pnpsoftware.sdl.usu.edu/redmine/projects/xtedsschema
             https://pnpsoftware.sdl.usu.edu/spa/xteds/current.xsd'
       name='ApiProducerXteds' version='1.0'>
  <Application name='ApiProducer' kind='application' programMemoryRequired='1' dataMemoryRequired='1'
        componentKey='apiProducerComponentKey'/>
  <Interface name='CounterData' id='1' >
    <Notification>
      <DataMsg name='CounterData' id='1' msgArrival='PERIODIC' msgRate='1.0' >
        <Variable name='counter' kind='count' dataType='FLOAT32' units='count' />
        <Variable name='currentIncrement' kind='count' dataType='FLOAT32' units='count' />
      </DataMsg>
    </Notification>
    <Request>
      <CommandMsg name='GetCurrentIncrement' id='2' />
      <DataReplyMsg name='CurrentIncrementReply' id='3' >
        <Variable name='curIncrement' kind='count' dataType='FLOAT32' units='count'/>
      </DataReplyMsg>
    </Request>
  </Interface>
  <Interface name='CounterManagement' id='2' >
    <Command>
      <CommandMsg name='changeIncrement' id='1' >
        <Variable name='newIncrement' kind='count' dataType='FLOAT32' units='count' />
      </CommandMsg>
    </Command>
  </Interface>
</xTEDS>
```

Sample 6.1. API Producer xTEDS. The xTEDS that the *ApiProducer* will implement and provide.

interface *2*, is for commanding or modifying the behavior of the producer. In this example the *Api*Consumers*, described later, can use this interface to change the value of a variable that the *Api*Producer* maps to *newIncrement*. This section describes this process shortly. The **CounterData* interface groups together data a consumer or user might request or to which they might subscribe. This example is a bit contrived, but this interface lets the consumer get the current *counter* value every time it is published, or request the current value of the increment being used to update the counter before the producers publishes it.

SSSM adds to the SPA API so the process of implementing the *SpaApplication* class is basically the same in the unsecured and secured cases. Handling and producing messages for the SSM network all works the same from the developer's perspective. The developer registers callback functions that the API can trigger when a command or request is received or when a notification is about to go out. Within the function callbacks for commands and requests the messages have already been decrypted, assuming they pass authentication and validation, so the developer does not need to handle them any differently. In the case of notifications the API will encrypt these messages after they leave the callback so the

```
<?xml version='1.0' encoding='utf-8' ?>
<xTEDS xmlns='https://pnpsoftware.sdl.usu.edu/redmine/projects/xtedsschema'
       xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
       xsi:schemaLocation= 'https://pnpsoftware.sdl.usu.edu/redmine/projects/xtedsschema
            https://pnpsoftware.sdl.usu.edu/spa/xteds/current.xsd'
       name='ApiSecureProducerXteds' version='1.0'>
  <Application name='ApiSecureProducer' kind='application' programMemoryRequired='1' dataMemoryRequired='1'
       componentKey='apiProducerComponentKey'/>
  <Interface name='SecureCounterData' id='1' >
    <Notification>
      <DataMsg name='SecureCounterData' id='1' msgArrival='PERIODIC' msgRate='1.0' >
        <Variable name='counter' kind='count' dataType='FLOAT32' units='count' />
        <Variable name='currentIncrement' kind='count' dataType='FLOAT32' units='count' />
      </DataMsg>
    </Notification>
    <Request>
      <CommandMsg name='SecureGetCurrentIncrement' id='2' />
      <DataReplyMsg name='SecureCurrentIncrementReply' id='3' >
        <Variable name='curIncrement' kind='count' dataType='FLOAT32' units='count'/>
      </DataReplyMsg>
    </Request>
  </Interface>
  <Interface name='SecureCounterManagement' id='2' >
    <Command>
      <CommandMsg name='SecureChangeIncrement' id='1' >
        <Variable name='newIncrement' kind='count' dataType='FLOAT32' units='count' />
      </CommandMsg>
    </Command>
  </Interface>
</xTEDS>
```

Sample 6.2. Secured API Producer xTEDS. The xTEDS that the *ApiSecureProducer* will implement and provide.

developer never has to deal with encryption or access control.

There are more complicated cases than this example, but at basic level the only difference for the developer between an unsecured and secured producer is in the declaration of the message. Sample 6.3 shows the header file for the *ApiProducer* class. Line 25 shows the declaration for the *CommandMsg* that the *ApiProducer* will use to implement the command on line 22 of the *ApiProducer* xTEDS in Sample 6.1 This allows the *ApiConsumer* programs to change the value of the increment via command. Line 26 of the header file shows the declaration for the *RequestMsg* that implements the request on line 14 of the *ApiProducer* xTEDS. This allows the *ApiConsumer* program to retrieve the current increment via a request message. Finally, line 27 of the header file shows the declaration of the *NotifcationMsg* that implements the notification on line 8 of the *ApiProducer* xTEDS. This allows the *ApiConsumer* to subscribe to and receive the current value of the counter every time it is published by the *ApiProducer*.

These three lines where *m_myChangeIncrementCmd*, *m_myCurrentIncrementRequest*, and *m_myCounterNotification* are delcared are the main difference between the *ApiProducer*

```cpp
#ifndef _API_PRODUCER_HPP
#define _API_PRODUCER_HPP

#include "ssm/api/SpaApplication.hpp"

/**
 * @class ApiProducer
 *
 * @brief A simple producer application which publishes a notification message and
 * accepts a command which alters the data being published
 */
class ApiProducer : public SpaApplication
{
  public:
    ApiProducer(void);
    void appInit(Int32 argc, char** pArgv);

    // Callbacks
    void preCounterMsgPublish(void);
    void onChangeCounterIncrementCmd(void);
    void onCurrentIncrementRequest(void);

  private:
    // Messages from xTEDS
    CommandMsg m_myChangeIncrementCmd;         //!< Command message to change the increment
    RequestMsg m_myCurrentIncrementRequest;    //!< Request message to get the current increment
    NotificationMsg m_myCounterNotification;   //!< Notification message for the counter

    // Mapped member variables
    Float32 m_counter;                         //!< A counter which updates each time data is published
    Float32 m_curIncrement;                    //!< Size of each increase
    Float32 m_newIncrement;                    //!< Contains the new increment value set through a CommandMsg

    CALLBACK_CONFIG(ApiProducer);    // Required in classes deriving from SpaApplication to properly setup
         callbacks
};

#endif // _API_PRODUCER_HPP
```

Sample 6.3. *ApiProducer* Declaration. The header file for the *ApiProducer*.

and the *ApiSecureProducer*. These same three lines in Sample 6.4 show the difference. The *ApiSecureProducer* uses *SecureCommandMsg* instead of *CommandMsg*, *SecureRequestMsg* instead of *RequestMsg*, and *SecureNotifcationMsg* instead of *NotifcationMsg*. Review of the two header files shows that both are the same except for the declaration of those the messages, and the name of the class. The producers use the other member variable to track and modify the counter, and the current increment being used for the counter. This shows that setting up the xTEDS and the header file for the a SPA API application is very similar for SSM and SSSM, this is by design.

Next this section looks at the implementation file for the example producers. This is the source file shown in Sample 6.5 and the implementation is the same for the unsecured and secured versions, except for the class name, so only one code sample is used to review the implementation of both the *ApiProducer* and the *ApiSecureProducer*. The default constructor instantiates an instance of the class and sets up each of the messages so they

```
#ifndef _API_SECURE_PRODUCER_HPP
#define _API_SECURE_PRODUCER_HPP

#include "ssm/api/SpaApplication.hpp"

/**
 * @class ApiSecureProducer
 *
 * @brief A simple secure producer application which publishes a
 * notification message and accepts a command which alters the
 * data being published
 */
class ApiSecureProducer : public SpaApplication
{
  public:
    ApiSecureProducer(void);
    void appInit(Int32 argc, char** pArgv);

    // Callbacks
    void preCounterMsgPublish(void);
    void onChangeCounterIncrementCmd(void);
    void onCurrentIncrementRequest(void);

  private:
    // Messages from xTEDS
    SecureCommandMsg m_myChangeIncrementCmd;        //!< Command message to change the increment
    SecureRequestMsg m_myCurrentIncrementRequest;   //!< Request message to get the current increment
    SecureNotificationMsg m_myCounterNotification;  //!< Notification message for the counter

    // Mapped member variables
    Float32 m_counter;         //!< A counter which updates each time data is published
    Float32 m_curIncrement;    //!< Size of each increase
    Float32 m_newIncrement;    //!< Contains the new increment value set through a SecureCommandMsg

    CALLBACK_CONFIG(ApiSecureProducer);    // Required in classes deriving from SpaApplication to properly
         setup callbacks
};
#endif // _API_SECURE_PRODUCER_HPP
```

Sample 6.4. *ApiSecureProducer* Declaration. The header source file for the *ApiSecureProducer*.

match the their *XTEDS* entries. The interface identifiers and message identifiers must match for LS and API to properly handle messages, and in the secure version they are also used for permissions and session creation, so they must match the entries in the permission table configuration file as well. The consumer side does not need to know these identifiers in advance as LS provides them in the query reply. The *appInit* function that follows is called by the API and allows the developer to setup the messages that they need the API to handle for them; the developers setup callbacks that the API uses to pass off the messages to developer code when the messages are received or when they are about to be published. The rest of this section walks though how a developer sets up each of the messages so they can use them.

The command from the xTEDS is setup using *m_myChangeIncrementCmd*, this is constructed on line 11 with interface identifier '2' and message identifier '1'. This pairing, *(2,1)*, ties this message to the command in the xTEDS, or at least sets if up so it can be tied

```
#include "ApiProducer.hpp"
#include "ApiProducerXteds.hpp"

/**
 * @brief Constructor which sets the location of the xTEDS, the name of the application, and the xTEDS UUID
 */
ApiProducer::ApiProducer(void)
  : SpaApplication("ApiSecureProducer", API_PRODUCER_XTEDS, API_PRODUCER_XUUID),
    m_myCounterNotification(1, 1, true, 1.0), // Set interface id and message id from xTEDS
    m_myCurrentIncrementRequest(1, 2, 3), // Set interface and message ids for both command and data reply
         portions
    m_myChangeIncrementCmd(2, 1),
    m_counter(0.0),
    m_curIncrement(1.1f),
    m_newIncrement(0)
{
  // empty
}

/**
 * @brief Function called by API to allow user to issue queries and register messages from their xTEDS
 *
 * @param argc Number of command line arguments
 * @param pArgv Array of character string command line arguments
 */
void ApiProducer::appInit(Int32 argc, char** pArgv)
{
  // Register provided command message
  m_myChangeIncrementCmd.addVariable(VariableType(FLOAT32_TYPE), &m_newIncrement);
  m_myChangeIncrementCmd.onCommandReceived = callback(&ApiProducer::onChangeCounterIncrementCmd);
  registerCommand(m_myChangeIncrementCmd);

  // Register provided request message
  m_myCurrentIncrementRequest.addDataReplyVariable(VariableType(FLOAT32_TYPE), &m_curIncrement);
  m_myCurrentIncrementRequest.onRequestReceived = callback(&ApiProducer::onCurrentIncrementRequest);
  registerRequest(m_myCurrentIncrementRequest);

  // Register provided notification messages
  m_myCounterNotification.addVariable(VariableType(FLOAT32_TYPE), &m_counter); // Setup message per xTEDS
       definition
  m_myCounterNotification.addVariable(VariableType(FLOAT32_TYPE), &m_curIncrement);
  m_myCounterNotification.onPublish = callback(&ApiProducer::preCounterMsgPublish);
  registerNotification(m_myCounterNotification); // Register the notification message with the framework
}

/** @brief Will be called periodically to allow app to perform any needed processing */
void ApiProducer::myActiveProcessing(void)
{
  LOG_INFO("Hello from my periodic callback!");
  // Do any needed processing here
}

/** @brief Called when a change counter increment command is received */
void ApiProducer::onChangeCounterIncrementCmd(void)
{
  LOG_INFO("Received change counter increment command, now incrementing by %f", m_newIncrement);
  m_curIncrement = m_newIncrement;
}

/** @brief Called when a counter increment request message is received */
void ApiProducer::onCurrentIncrementRequest(void)
{
  LOG_INFO("Received counter increment request, now replying with %f", m_curIncrement);
  // Request's DataReply sent after returning from this function with the variable we mapped to it
}

/** @brief Will be called before publishing the counter data message to subscribers */
void ApiProducer::preCounterMsgPublish(void)
{
  // Update member variables before publish
  m_counter = m_counter + m_curIncrement;
  LOG_INFO("Published current counter value of %f", m_counter);
  publish(m_myCounterNotification);
}
```

Sample 6.5. *Api*Producer* Definition. The implemetation source file for the *ApiProducer*, the implementation file for the *ApiSecureProducer* is the same except for the class name.

to the command in the xTEDS in the *appInit* starting on line 28. Line 28 maps the variable so that marshalling and unmarshalling methods used in the API know what variables are included in the message. The value in this variable will be updated by the API when this message received. The next line sets up the callback that the API should call when this message is received, and last line in that block finishes the registration process. Line 52 of Sample 6.5 shows the implementation of the *onChangeCounterIncrementCmd* function. In this function the developer can simply use *m_newIncrement* because the API updates the variable when the message is received. This process works the same for the secure version, the developer does not to deal with encryption or permissions directly, this is all handled by the API. There is no change in process here from SSM to SSSM and using data is as simple as mapping the variable and then using them in the message callback. The process is the same for the request and notification messages. In the case of the variables mapped to the notification the developer can change the value of *m_counter* in the *preCounterMsgPublish* callback before the notification message is sent out.

Creating a producer-type application using the API to create a secured or unsecured messages is essentially the same, except for the message classes that the developer uses.

6.2.2 Consumer Development

Developing an unsecured versus secured consumer-type application is, again, basically the same for the application developer. The developer again uses the secured versions of the command, request, and notification messages, the same messages that are shown in Sample 6.4 are used on the consumer side as well. The header files are very similar to the ones shown for the producer example so they are not shown. Sample 6.6 shows the source file for the *Api*Consumer* examples, the files are the same except for the class name so only one code sample is used to describe the implementation of both the *ApiConsumer* and the *ApiSecureConsumer*. The constructors for the message classes do not need to pass in the interface and message identifiers consumer-side, as those will be provided by the LS when it replies to the queries that are issued by the consumer on lines 27, 33, and 40. These queries and their associated messages will be in an unsatisfied state until the LS replies to the queries.

The process that the API goes through to handle the xTEDS registration and queries is different for the unsecured versus secured versions, but at the application developer level there is no noticeable difference aside from using different message classes. Once the *m_newIncrement* variable is mapped to *m_setIncrementCommand*, on line 26 its value can be manipulated in developer-side code and then the command can be sent, the function starting on line 60 shows an example of this. The API takes care of marshalling the updated value into the message before it is sent in both the secured and unsecured cases, and the API takes care of encrypting the variable as part of the message payload before it it sent in the secured case. This is all achieved by the API and the developer simply calls *sendMsg* as shown on line 68.

A similar process is used to map variables for the *m_getCurIncrementRequest* and *m_counterNotification* messages. Sample 6.6 shows how the different messages can be used in consumer-side code, but the interaction at the developer-level is the same for SSM and SSSM. SSSM supports both cases, secured and unsecured versions can be developed against the SSSM version of the API by using either the unsecured or secured versions of the message classes.

6.2.3 Policy Development

The basic vehicle for setting up security policies is explained in Section 5.3.1 of Chapter 5 and it entails setting up a policy table for the LS and providing each secured component with a unique encryption key. Sample 6.7 shows a possible configuration file for the *ApiSecureProducer* and *ApiSecureConsumer*. The example permission table has comments to help explain the relevant lines of Extensible Markup Language (XML). The table provides the encryption keys, preferred key lengths, and permissions. Sample 6.7 shows that the *ApiSecureProducer* has no permissions with regard to *ApiSecureConsumer*, and in our example the *ApiSecureConsumer* does not provide any interfaces. Typically applications are not pure consumer or pure producer, but this example was kept this way for simplicity.

```cpp
#include "ApiConsumer.hpp"

/** @brief ApiConsumer constructor */
ApiConsumer::ApiConsumer()
  : SpaApplication("ApiConsumer"),
    m_setIncrementCommand(),
    m_getCurIncrementRequest(),
    m_counterNotification(),
    m_counter(0),
    m_increment(0),
    m_newIncrement(0),
    m_curIncrement(0),
    m_recvCount(0)
{ /* do nothing */ }

/**
 * @brief Space provided to issue queries and setup any processing needs
 * @param argc Number of command line arguments
 * @param pArgv Array of character string command line arguments
 */
void ApiConsumer::appInit(Int32 argc, char** pArgv)
{
  Query setNewIncrementQuery(Query::COMMAND,
        Query::QUERY_CURRENT|Query::QUERY_CANCELLATIONS|Query::QUERY_FUTURE);
  setNewIncrementQuery.message.addAttribute("name", QueryOperand(EQUAL), "ChangeIncrement", 0);
  setNewIncrementQuery.interface.addAttribute("name", QueryOperand(EQUAL), "CounterManagement", 0);
  m_setIncrementCommand.addVariable(VariableType(FLOAT32_TYPE), &m_newIncrement);  // Map variable
  issueQuery(setNewIncrementQuery, &m_setIncrementCommand);  // Issue query

  Query getCurIncrementRequestQuery(Query::REQUEST, Query::QUERY_CURRENT | Query::QUERY_CANCELLATIONS |
        Query::QUERY_FUTURE);
  getCurIncrementRequestQuery.message.addAttribute("name", QueryOperand(EQUAL), "GetCurrentIncrement", 0);
  m_getCurIncrementRequest.addDataReplyVariable(VariableType(FLOAT32_TYPE), &m_curIncrement);  // Map variables
  m_getCurIncrementRequest.onDataReplyReceived = callback(&ApiConsumer::onCurrentIncrementDataReply);
  issueQuery(getCurIncrementRequestQuery, &m_getCurIncrementRequest);  // Issue query

  Query counterQuery(Query::NOTIFICATION, Query::QUERY_CURRENT | Query::QUERY_CANCELLATIONS |
        Query::QUERY_FUTURE);
  counterQuery.message.addAttribute("name", QueryOperand(EQUAL), "CounterData", 0);
  m_counterNotification.addVariable(VariableType(FLOAT32_TYPE), &m_counter);  // Map message variables
  m_counterNotification.addVariable(VariableType(FLOAT32_TYPE), &m_increment);
  m_counterNotification.onDataReceived = callback(&ApiConsumer::onCounterDataReceived);  // Setup events
  issueQuery(counterQuery, &m_counterNotification); // Issue query

  // Setup pro-active processing
  setupPeriodicCallback(callback(&ApiConsumer::activeProcessing), 0.5);
}

/** @brief Called after receiving a CounterData data message and populating mapped member variables */
void ApiConsumer::onCounterDataReceived(void)
{
  LOG_INFO("Received data msg - counter: %f curIncrement: %f", m_counter, m_increment);
  m_recvCount++;
}

/** @brief Called after receiving a CounterData data message and populating mapped member variables */
void ApiConsumer::onCurrentIncrementDataReply(void)
{
  LOG_INFO("Received data reply containing current increment: %f", m_curIncrement);
}

/** @brief Called at the rate specified in appInit to do any active processing */
void ApiConsumer::activeProcessing(void)
{
  if (m_setIncrementCommand.isSatisfied() && (m_recvCount % 5) == 0 && m_recvCount > 0)
  {
    if (m_increment > 1000) m_newIncrement = 1.1f;
    else m_newIncrement = m_increment * 2;

    LOG_INFO("Sending command to change increment increase increment 2x");
    sendMsg(&m_setIncrementCommand);
  }

  if (m_getCurIncrementRequest.isSatisfied() && (m_recvCount % 3) == 0)
  {
    LOG_INFO("Sending request for current increment ");
    sendMsg(&m_getCurIncrementRequest);
  }
}
```

Sample 6.6. *Api*Consumer* Definition. The implemetation source file for the *ApiConsumer*, the implementation file for the *ApiSecureConsumer* is the same except for the class name.

Line 14 starts the permissions that the *ApiSecureConsumer* has with the *ApiSecureProducer* target. The LS ingests the file and uses the keys and permissions to authenticate the producer and consumer, and create a session ticket that allows the producer and consumer applications to communicate securely on the producers *(1,1)*, *(1,2)*, *(1,3)*, and *(2,1)* interfaces. The session ticket and session management is handled by the LS and API without affecting application-level code, or burdening the developer.

```
 1 <PermissionTable>
 2   <!-- ApiSecureProducer -->
 3   <SubjectComponent uuid='fccccb2a-30dc-604a-d5a8-2fbcccc3aa33' preferredKeySize='32'>
 4     <!-- ApiSecureProducer to LS Key -->
 5     <LookupServiceSymmetricKey key='00112233445566778899aabbccddeeff00112233445566778899aabbccddeeff' />
 6     <TargetPermissionList /> <!-- No permissions -->
 7   </SubjectComponent>
 8   <!-- ApiSecureConsumer -->
 9   <SubjectComponent uuid='faaace1a-30dc-604a-d5a8-2fbaaaa3aa44'>
10     <!-- ApiSecureConsumer to LS Key -->
11     <LookupServiceSymmetricKey key='ffeeddccbbaa99887766554433221100ffeeddccbbaa99887766554433221100' />
12     <TargetPermissionList>
13       <!-- ApiSecureConsumer to ApiSecureProducer permissions -->
14       <TargetComponent uuid='fccccb2a-30dc-604a-d5a8-2fbcccc3aa33'  >
15         <Permission interfaceId='1' messageId='1' />
16         <Permission interfaceId='1' messageId='2' />
17         <Permission interfaceId='1' messageId='3' />
18         <Permission interfaceId='2' messageId='1' />
19       </TargetComponent>
20     </TargetPermissionList>
21   </SubjectComponent>
22 </PermissionTable>
```

Sample 6.7. LS Permission Table for *ApiSecureProducer* and *ApiSecureConsumer*. Permission Table that configures the LS and allows the component with UUID ending in *aa33* to communicate with *aa44* on *(1,1)*.

There is one more piece to the puzzle that the application developer needs to provide, and that is a unique key for the secured producer and one for the consumer. These keys must match the ones in the permission table that the developer uses to configure the LS. The keys are provided to each application through command-line parameters, and would typically be included as part of a script or service that manages the startup of the application. The preferred key size is also passed to the application the same way and also needs to match the permission table. Sample 6.8 shows how this would be setup for the *ApiSecureProducer*. The key is provided as a American Standard Code for Information Interchange (ASCII) hexadecimal-string to represent its binary value.

Setting up the permissions table and passing the keys to applications with secured

```
#!/bin/bash

# Start the API Producuer
$ ./ApiSecureProducer -k=00112233445566778899aabbccddeeff00112233445566778899aabbccddeeff
     --preferred-key-size=32 &
```

Sample 6.8. Startup Script for *ApiSecureProducer*. Simple script that starts the *ApiSecureProducer* with the specified encryption key and preferred key size for use with the LS.

interfaces is the primary, and really only, additional developer-effort required to develop secured message interfaces instead of unsecured interfaces.

6.3 Performance Evaluation

This section discusses the delta in payload byte throughput, CPU utilization, and memory usage when security is added to a representative reusable modular open-network software development framework. It is important to evaluate this delta in order to get a basic understanding of how adding security has affected reusability. For example, lower byte throughput and higher CPU utilization can reduce the scenarios where SSSM can be used and therefore affect the reusability of software systems that use it as a core.

To this end this section presents a high-level comparison of unsecured versus secured while providing the same functionality. This section reviews a series of test that evaluate the delta from unsecured to secured messaging in terms of resource utilization and message or byte throughput if a developer creates a simple set of publish/subscribe applications, namely a secure producer/consumer set of applications versus a unsecured producer/consumer set of applications. There is a delta from unsecured to secured in term of resource utilization and payload byte throughput. This section shows that performance requirements and system resources need to be more carefully considered for SSSM than they do for SSM; reusability has taken a hit but future performance improvements and careful API usage can help to address this.

6.3.1 Setup

This evaluation conducts a simple apples to apples comparison of a notification interface that is unsecured to a notification interface that is secured. This requires the creation of two sets of producer/consumer applications, a unsecured producer/consumer and a secured producer/consumer pair.

Each producer/consumer pair was exercised on a Linux Virtual Machine Ware (VMware) virtual machine (VM) instance with 2 gigabyte (GB) of random access memory (RAM). This VM is hosted on a Windows 10 machine with an i7-4700MQ CPU running at 2.40 gigahertz (GHz), 4 Cores and 8 Logical Processors. The host has 32 GB of RAM. The host machine and VM are not representative of the resources that a typical small space system might have and so this is not meant as a representative benchmark of performance and resource utilization, but as a comparison between the unsecured and secured versions. This illustrates the difference in resource usage and throughput for the two variants.

Both producer applications make use of the notification shown in Sample 6.9. The notifications have a few bytes of management variables, i.e. msgNumber, startSeconds, and startNanoseconds. The rest of the message is a variable size payload, this size can be varied during execution, but for these tests the size is passed in as a command-line parameter and fixed during execution. The producer applications will attempt to publish the notifications at a constant rate for the duration of a test, for these tests a rate of 1000 hertz (Hz) is always passed into the producer application and the size of the *DynamicArray* is changed between runs to change the amount of message data moved per second.

Both consumer applications make use of the *appInit* function shown in Sample 6.10. In this function variables are initiated, and mapped to the notification the consumers will receive. Lines 24 and 25 show where the consumers build the query for the "MeasureData" notification, this query is then issued on line 32 after the variables are mapped and a callback function is setup. The variables and callback need to be setup before the query is issued so the API can appropriately map the variables and callback before a reply to the query is received from the LS. Lines 27 through 30 in Sample 6.10 correspond with lines 11 through 14 in the producer xTEDS. The API will populate these variables just prior to calling *onCounterDataReceived*, this allows the developer to easily access data in

```
<?xml version='1.0' encoding='utf-8' ?>
<xTEDS
  xmlns='https://pnpsoftware.sdl.usu.edu/redmine/projects/xtedsschema'
  xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
  xsi:schemaLocation='https://pnpsoftware.sdl.usu.edu/redmine/projects/xtedsschema
        https://pnpsoftware.sdl.usu.edu/spa/xteds/current.xsd'
  name='Api*MeasureProducerXteds' version='1.0' >
  <Application name='Api*MeasureProducer' kind='application' programMemoryRequired='1' dataMemoryRequired='1'
        componentKey='apiProducerComponentKey'/>
  <Interface name='MainInterface' id='1' >
    <Notification>
      <DataMsg name='MeasureData' id='1' msgArrival='EVENT' >
        <Variable name='msgNumber' kind='count' dataType='UINT16' units='count'/>
        <Variable name='startSeconds' kind='count' dataType='UINT32' units='seconds'/>
        <Variable name='startNanoseconds' kind='count' dataType='UINT32' units='nanoseconds'/>
        <DynamicArray name='positionOne' kind='count' dataType='UINT08' units='count'
              maxArrayElements='64000'/>
      </DataMsg>
    </Notification>
  </Interface>
</xTEDS>
```

Sample 6.9. Measure API Producer xTEDS. The xTEDS that both the unsecured and secured producers use for this evaluation.

the message payload without having to index into into the message buffer. The evaluation that follows shortly covers some of the price that comes with this ease-of-use. Once the LS returns a the query response the applications can begin exchanging this notification.

As previously mentioned the rate at which the message are sent is held constant, but the size of the messages is increased for each set of ten test runs. The message sizes used are: 4,000, 8,000, 16,000, 22,000, 26,000, 30,000, 34,000, 38,000, 44,000, 48,000, 54,000, 60,000, and 64,000 bytes. These rates correspond to 4, 8, 16, 22, 26, 30, 34, 38, 44, 48, 54, 60, and 64 megabytes per second (MBps) if the message rate of 1000 Hz can be maintained; these rates are the desired rates for each set of tests. Each producer/consumer pair is run 10 times at each of these message sizes and the average CPU utilization, memory usage, and payload byte throughput are computed. This average is used to combat other background process that might compete for system resources during the test runs. The producer and consumer receive the message rate and size via command-line from a script that runs all the tests. The consumer multiplies the rate by a fixed test duration to know how many messages it should receive for the given test run over the test duration if the desired message rate can be maintained. This duration is a constant in the code base that cannot be changed without recompiling. The tests were intended to be approximately 30 seconds in duration, but would run longer when the consumer/producer trailed behind the desired rate or dropped

```
void Api__MeasureConsumer::appInit(Int32 argc, char** pArgv)
{
  if (m_pLsKey != NULL)
  {
    m_keySize = static_cast<Int32>(m_preferredKeySize);
  }

  m_pNotificationData = new (g_pPoolController) UInt8[ Api__MeasureConsumer::MAX_NOTIFICATION_SIZE];
  m_pMsgNumberData = new (g_pPoolController) UInt16[m_sampleSize];
  m_pStartSecondsData = new (g_pPoolController) UInt32[m_sampleSize];
  m_pStartNanosData = new (g_pPoolController) UInt32[m_sampleSize];
  m_pEndSecondsData = new (g_pPoolController) UInt32[m_sampleSize];
  m_pEndNanosData = new (g_pPoolController) UInt32[m_sampleSize];
  m_pRcvRateData = new (g_pPoolController) UInt16[ Api__MeasureConsumer::DURATION_SECONDS];

  memset(m_pNotificationData, 0,  Api__MeasureConsumer::MAX_NOTIFICATION_SIZE);
  memset(m_pMsgNumberData, 0, m_sampleSize * sizeof(UInt16));
  memset(m_pStartSecondsData, 0, m_sampleSize * sizeof(UInt32));
  memset(m_pStartNanosData, 0, m_sampleSize * sizeof(UInt32));
  memset(m_pEndSecondsData, 0, m_sampleSize * sizeof(UInt32));
  memset(m_pEndNanosData, 0, m_sampleSize * sizeof(UInt32));
  memset(m_pRcvRateData, 0,  Api__MeasureConsumer::DURATION_SECONDS * sizeof(UInt16));

  Query dataNotifQuery(Query::NOTIFICATION, Query::QUERY_CURRENT | Query::QUERY_CANCELLATIONS |
        Query::QUERY_FUTURE);
  dataNotifQuery.message.addAttribute("name", QueryOperand(EQUAL), "MeasureData", 0);

  m_dataNotification.addVariable(VariableType(UINT16_TYPE), &m_msgNumber);
  m_dataNotification.addVariable(VariableType(UINT32_TYPE), &m_startSeconds);
  m_dataNotification.addVariable(VariableType(UINT32_TYPE), &m_startNanos);
  m_dataNotification.addDynamicArray(VariableType(UINT8_TYPE), m_pNotificationData, &m_notificationSize,
        Api__MeasureConsumer::MAX_NOTIFICATION_SIZE);
  m_dataNotification.onDataReceived = callback(&Api__MeasureConsumer::onCounterDataReceived);
  issueQuery(dataNotifQuery, &m_dataNotification);
}
```

Sample 6.10. ApiMeasureConsumer and ApiSecureMeasureConsumer *appInit* Function. The same *appInit* function is used for the unsecured and secured consumers. This function shows how the variables are setup for tracking message rates, mapping variables, and linking the variable size data array to the xTEDS notification used for testing.

messages. The command-line arguments allowed the message rate to be varied, but that was kept constant for these tests and only the message size was changed. Future work can evaluate the performance of keeping the message size constant and increasing the message rate.

In summary, the publish rate of the test notification message was held constant while the message size is increased. Each test was run ten times at each speed with both the unsecured and secured variants, this means there are a total of 130 test runs for unsecured and 130 for secured. The specific concepts are outlined below:

- **Publish rate** — The publishing rates for the notification is held constant at 1000 Hz.
- **Message payload size** — Message payload for the notification was increased with each set of test runs by 4,000 bytes, from 4000 up to 64,000 bytes. A test run consisted

of 10 test iterations at the same payload size.

- **Network throughput** — Throughput only considers the size of the dynamic array. Throughput is calculated by tracking the number of messages moved per second. The actual number of bytes moved is higher, but this test is for comparison and not designed to measure capability so the total number is not as important. The overhead of the secured heard will in theory reduce the number of payload bytes that can be moved. These tests were conducted on a local host only network with all processes running on the same host.
- **CPU utilization** — CPU utilization was tracked for the producer and consumer applications during each test run. The process are competing with other SSM processes as well as other, and user and system processes.
- **Memory utilization** — Memory utilization was tracked for the producer and consumer applications during each test run.

6.3.2 Evaluation Results

The tests and this review of the test results is not a formal analysis of the unsecured and secured message handling performance. These test and this evaluation seek to look at the delta between unsecured and secured message handling. Future work can perform a in-depth performance analysis and look at ways to optimize the secured, and even the unsecured variants. This review is an evaluation of the secured variant against the the unsecured version to quantify at a basic level of some of the performance-related effects of adding security provisions to SSM. It is important to be aware of this moving forward as significant deltas from SSM to SSSM negatively impact of the reusability of this open-network software development framework.

First this evaluation reviews the payload throughput in bytes of the unsecured and secured producer/consumer pairs. Figure 6.1 shows the desired payload byte throughput in blue, this is the MBps that would be achieved if the producer/consumer pairs exchange the messages at 1000 Hz. Figure 6.1 shows that both the unsecured and secured variants match this desired rate until 32 MBps. They both start to deviate from the desired rate at around 36 MBps and still hold similar rates of around 38 MBps while attempting 40 MBps. After 40 MBps the rates start to diverge, the secured variant tops out at about 38 MBps. Trying to run at higher rates than that actually decreases throughput, this is likely due to the queues holding messages getting backed up; this will be discussed more shortly in terms of the memory utilization of the different applications as the test increase the attempted payload byte throughput. CPU time is being split between queuing incoming messages and trying to decrypt and reassemble them and empty the queue; beyond this threshold it appears that the system starts to thrash a bit. The same thing happens for the unsecured variant, just with a higher ceiling of about 46 MBps. This gives a 17% in maximum throughput difference between the two, this is not an insignificant difference and affects the reuse of SSSM in systems where higher rates are needed, or in systems that are generally more resource limited. Future work will look at ways to optimize how the API is used or optimizations that can made to SSM and SSSM to increase throughput. Below 36 MBps there is no difference in throughput, this brings up the next point of comparison: what is the CPU utilization and memory load of these applications during test?

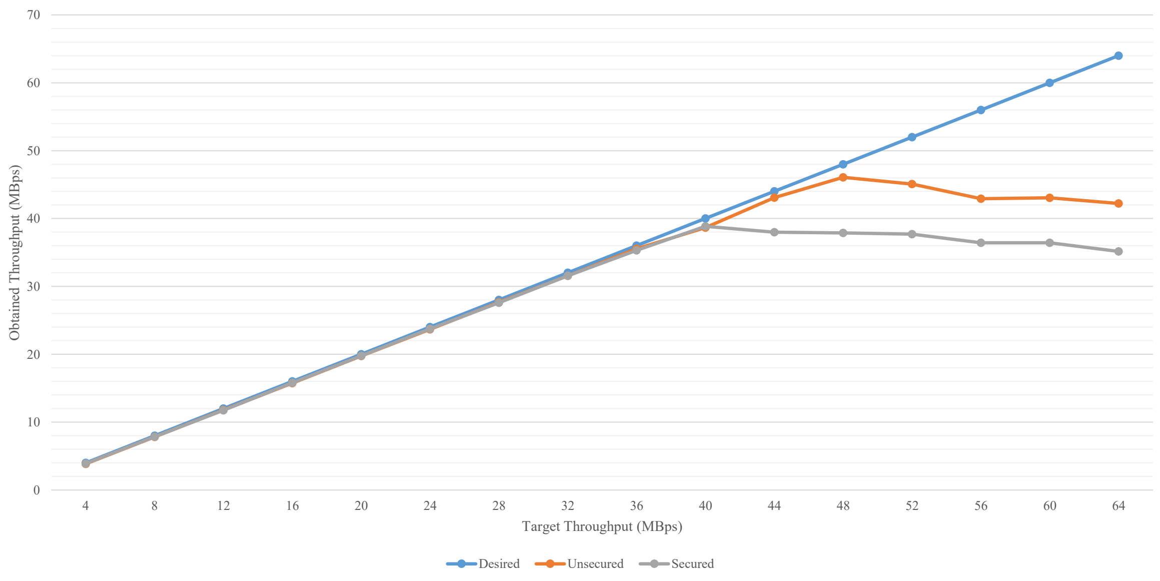


Fig. 6.1. Unsecured versus Secured Payload Byte Throughput. Comparison of attempted payload byte throughput for unsecured messaging and secured messaging.

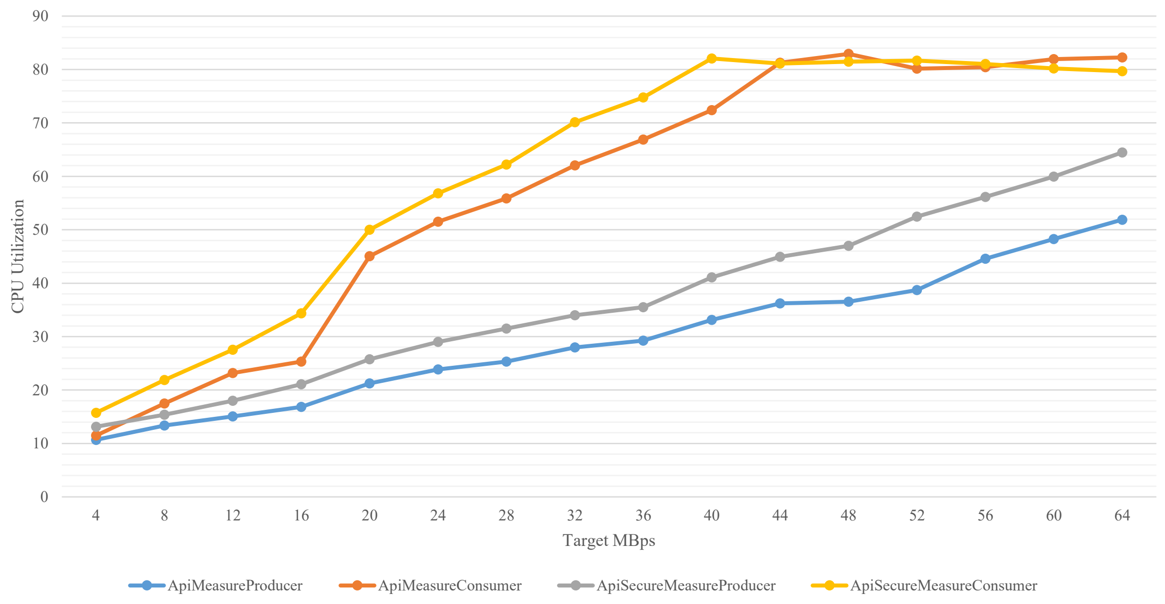


Fig. 6.2. Unsecured versus Secured CPU Utilization. Comparison of CPU utilization for unsecured messaging and secured messaging.

Figure 6.2 shows the CPU profile at the various payload data rates and Figure 6.3 shows the memory usage at these same data rates. All of the applications start off with a very similar CPU utilization for the 4 MBPS payload throughput. All of the applications are using between 10% to 16% of the CPU with the *ApiMeasureProducer* using close to 10% and the *ApiSecureMeasureConsumer* using close to 16%. The *ApiMeasureProducer* and the *ApiMeasureConsumer* make up the unsecured messaging pair and the *ApiSecure-MeasureProducer* and *ApiSecureMeasureConsumer* make up the secured message pair. The *ApiMeasureProducer* and *ApiSecureMeasureConsumer* generally frame the low and high for CPU usage for all the tests. The producers in both cases use less CPU, suggesting that the factor limiting throughput is on the receiving or consumer end of the the message exchange. In the case of the unsecured consumer application the API handles the reception and unmarshalling of the messages from the producer. This works the same for the secured consumer except the message must be decrypted before this happens. The API also copies the values from the message into mapped variables, typically setup in the *appInit* function, and triggers the callback shown in Sample 6.11.

Sample 6.11 shows the callback triggered every time a new *m_dataNotification* message

```
1 /**
2  * @brief Called after receiving a CounterData data message and populating mapped member variables,
3  * also show how access is provided to received SpaMessage, xTEDS section, and message definition.
4  *
5  * @param pMessage Pointer to actual received SpaMessage object (of subtype SpaData, SpaCommand, or
       SpaDataReply)
6  * @param xtedsSection String containing the xTEDS section associated with pMessage
7  * @param msgDef String containing the message definition of pMessage
8  */
9 void Api__MeasureConsumer::onCounterDataReceived(SpaMessage* pMessage, std::string xtedsSection, std::string
      msgDef)
10 {
11   getSpaTime(m_endSeconds, m_endNanos);
12
13   if (m_index < m_sampleSize)
14   {
15     m_pMsgNumberData[m_index] = m_msgNumber;
16     m_pStartSecondsData[m_index] = m_startSeconds;
17     m_pStartNanosData[m_index] = m_startNanos;
18     m_pEndSecondsData[m_index] = m_endSeconds;
19     m_pEndNanosData[m_index] = m_endNanos;
20     m_index++;
21   }
22
23   if (m_msgNumber >= m_sampleSize)
24   {
25     unsubscribe(m_dataNotification);
26     handleReport();
27     quit();
28   }
29 }
```

Sample 6.11. ApiMeasureConsumer and ApiSecureMeasureConsumer *onCounterDataReceived* Function. The same *onCounterDataReceived* function is used for the unsecured and secured consumers. This function shows the work being done to track throughput and message transit duration.

is received. It is important not to spend much time in this function because the next message cannot be dequeued until this function returns; for this reason trackers are quickly updated on lines 15 through 19 without performing any processing. Post-processing is performed in the *handleReport* function once enough messages have been received. Notice that *m_pRcvRateData* is also not used in the function, but that is was mapped back in *appInit* on line 30 of Sample 6.10. The entire array is copied into the *m_pRcvRateData* every time the notification is received even though it is not used, this takes time, and the data in the buffer is actually available as part of *pMessage* passed into the callback, this mapping to *m_pRcvRateData* just makes it easier for a developer to use. In this consumer nothing is done with the buffer because the intent is to measure the throughput and simply receiving, and the secured case decrypting, the message accomplishes this. However, in this case the mere fact the variable is mapped is causing extra work for the consumer in the name of making the data easier to work with, at the end of this section throughput that can be achieved without this mapping will be reviewed; keep in mind that the data is still available, but must be accessed through *pMessage*.

Coming back to the CPU usage there is a pretty linear increase in utilization for the producers with the secure producer using more CPU, the difference grows steadily from about 4% at the slowest speed to about 15% at the fastest speed. At lower speed this difference is likely acceptable, but at higher speeds the separation becomes significant and might reduce the choice of platforms that would handle the load. The CPU usage for the consumers is much more interesting. Figure 6.2 shows jump in usage when throughput goes from 16 MBps to 20 MBps, both show a similar jump. SSM has a maximum transmission unit (MTU) which determines how big a message can be before it is segmented, these applications were compiled with an MTU of 20,000 bytes; this limit includes headers and footers. When the dynamic array size went from 16,000 bytes to 20,000 bytes to attempt the throughput of 20 MBps the messages start being segmented. This means the message must be reassembled on the other end, the jump in CPU appears to correlate with this new additional work. After this the next point of interest occurs around 40 to 44 MBps, this roughly corresponds to the payload byte throughputs where the producer/consumer pairs are maxing out. In this window both the consumers hit their CPU usage ceiling, the secured version hits the ceiling first and also maxes out on throughput first. The CPU ceiling is the same for both at around 81%, the difference being that the unsecured version is moving about 6 – 8 more MBps. It is likely that between the VM overhead, CPU clock speed, host processes, guest processes, and consumer verse producer contention the maximum CPU has been reach. Note that ps was used to get these CPU measurements which means consumer was executing on the CPU 80% of the time over its process life in the maximum throughput cases. Future work can look at optimizing SSMs handling of messages on the receiving end as this appears to be the bottleneck for both the unsecured and secured versions.

Next this evaluation considers memory usage. Figure 6.3 shows that memory usage stays almost constant for all the applications until they hit the throughput ceiling. Both of the producers still show about constant memory usage after this point, but the consumer memory usage explodes. The *ApiSecureMeasureConsumer* starts to balloon first and this

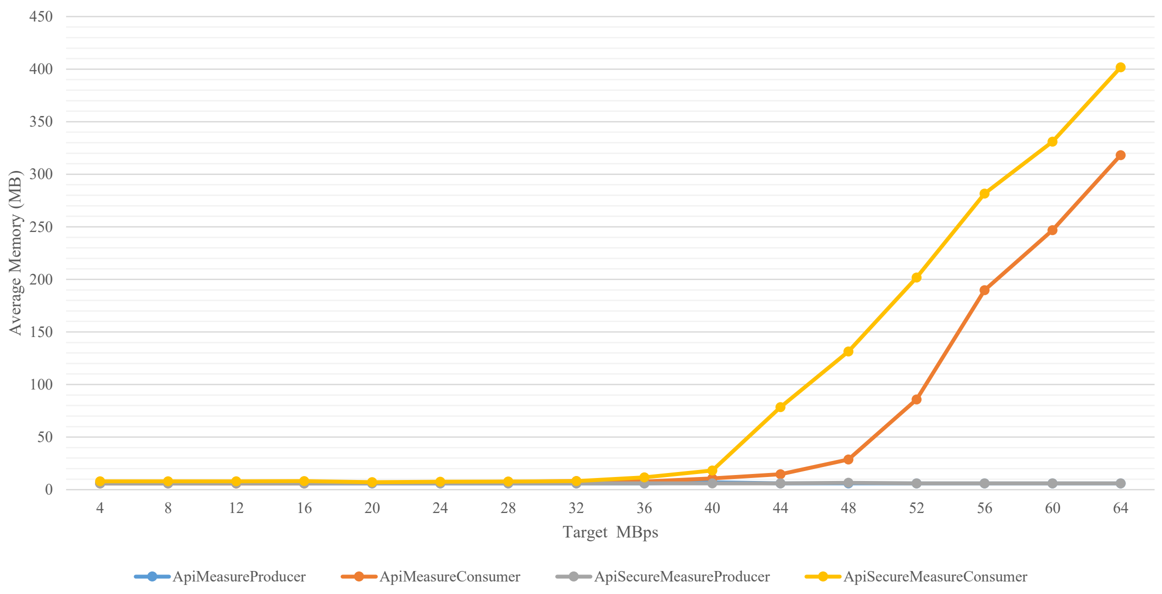


Fig. 6.3. Unsecured versus Secured Memory Usage. Comparison of memory usage for unsecured messaging and secured messaging.

correlates with the secured throughput ceiling, as the rates increases past this point it follows that the message queue that is holding the messages received from the producer start getting backed up. The implementation of the API actually allows the message queue to grow without bound and the queue of messages hits over 400 megabyte (MB)s for about a 30 second run; this extenuates the importance of benchmarking software on the hardware that will host it with a representative load as early as possible in order to determine margins and feasibility. This also suggests that future might consider bounding the amount of memory these queues are allowed to use so that excess messages are simply dropped in order to avoid a scenario where all system memory is allocated and new allocations fail as this would result in an unstable system. This is also a problem for the unsecured variant as it climbs to about 320 MBs in 30 seconds. This shows about an 80 MB difference between their averages at the maximum rate tested, and this actually down from an maximum difference of about 110 MB. If this test was carried out it is possible that the buffer overrun would converge between the two as system resources were exhausted, this might even happen if the test were allowed to run longer than 30 seconds. This level of testing is sufficient to show the payload byte throughput, CPU utilization, and memory usage are only marginally different at lower throughputs, show strong divergence at higher throughput and start to come back together once throughput ceilings are reached. This also shows that SSM may need updates to better handle higher throughputs.

There may be tweaks and optimizations that can be made to increase throughput at the cost of ease-of-use, this is a balancing act. One example of this is the use of mapped variables, this increases ease-of-use, but it comes at a cost. Figure 6.4 shows the payload byte throughput for the same set of tests run with line 30 in Sample 6.10 commented out so that *m_pNotificationData* is no longer being mapped. The data is still accessible via *pMessage*, but it is no longer nicely put into *m_pNotificationData*. This graph shows a dramatic change in maximum throughput, and actually this set of tests does not appear to reach a maximum. This is an example of a way to improve throughput without changes

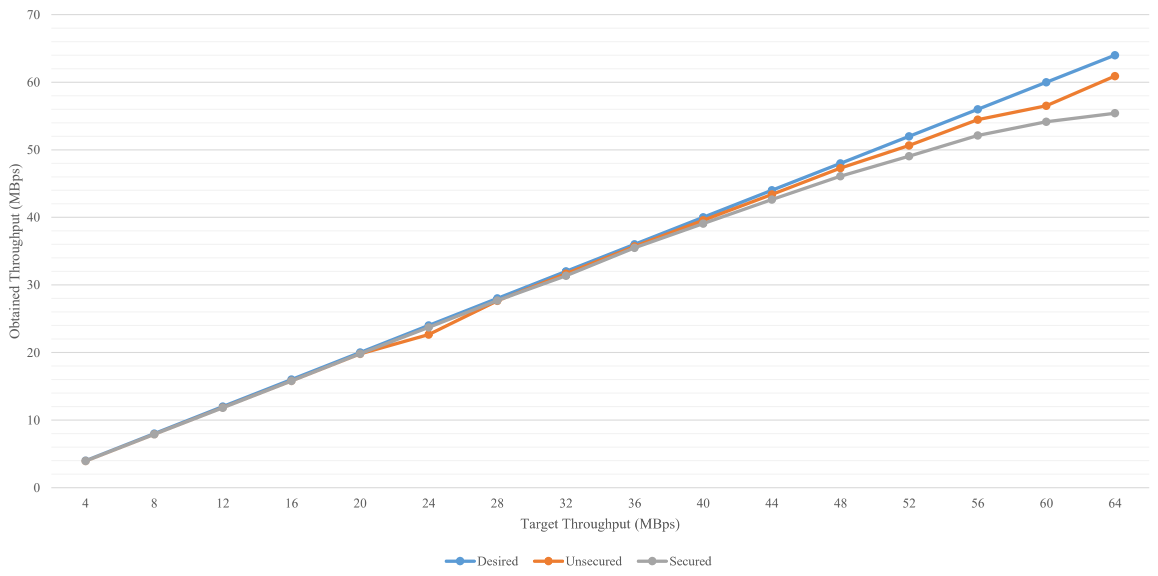


Fig. 6.4. Unsecured versus Secured Payload Byte Throughput. Comparison of attempted payload byte throughput for unsecured messaging and secured messaging.

to the API, just how a developer uses the API. This might increase the complexity of the developer's code as they have to access data more directly and might need to use something like direct memory access (DMA) to efficiently move the incoming data without slowing the messages down. This access is usually hardware and/or OS dependent but might be needed in scenarios that require high performance; future updates to SSM could look at adding this to the SSM portability utilities as a hardware abstraction library. This would mean that new hardware would require new updates to the library, but could provide this performance without affecting user-level code.

6.4 Conclusions of SSSM Evaluation

Section 6.2 shows that additional developer hardship is minimal. The developer does have to provide a configuration file for the LS and encryption keys for any secured endpoints; beyond that they need to use secured versions of the message classes, but the parameters for these are the same for the unsecured versions and the API handles the rest. This shows that the change in developer burden is minimal, the developer has not had to change their approach or add additional code. Keeping this change to a minimum in turn keeps any negative effects on reuse to a minimum. From this perspective reusing SSSM-based software should be just as advantageous and easy as reusing SSM-based software.

Section 6.3 shows that there is a performance hit in simply going from unsecured to secured messaging, it also shows there may be some easy ways around this issue. At higher rates resource limited system may not be able to handle the demands of the secured version; this does impact the reusability of SSSM and needs to be considered for future work. The payload byte throughput is very comparable at lower rates, even up to 36 MBps, note that is bytes not bits. Section 6.3 also showed that CPU and memory usage were higher for the secured producer/consumer pair, and had a large separation after the secured version reached is maximum throughput first and message processing backed up on the consumer side. This disparity started to converge again once the unsecured variant reaches its maximum throughput and may fully converge if the high throughput tests were run longer or higher payload throughput was attempted. This suggests that some optimization to SSMs message handling would be beneficial and should help the performance of SSSM as well.

Another set of tests without variable mapping showed that much higher rates could be achieved for both secured and unsecured variants, this was a sort of side test and was not discussed in terms of memory and CPU. In this scenario both variants had much higher throughput ceilings, achieving throughput in the 50 and 60 MBps range. The maximum attempted rate of 64 MBps may not have realize their full maximum rates as they were just starting to diverge from each other and the attempted rate.

Performance requirements and system resources need to be more carefully considered for SSSM than they do for SSM; reusability has taken a hit but future performance improvements and careful API usage can help to address this.

CHAPTER 7
CONCLUSION

7.1 Introduction

This research addresses two interrelated problems for Small Space, namely the *Development Problem* and the *Security Problem.* The crux of the *Development Problem* is developing capable resilient space systems that will deploy into harsh, isolated, and contested environments under the constraint of shrinking budgets and schedules. The results of the SISDPA surveys and the on-going trends in Small Space development show a path-forward where space systems are comprised of multi-vendor hardware and software systems; heterogeneous systems of systems. The results of the SISDPA surveys show that developers see reuse and open-network systems as the best way to wrangle these heterogeneous systems of systems with good abstraction, modularity, and encapsulation. The results of SISDPA surveys as well as our daily reliance on space-backed infrastructure highlight the importance of cybersecurity for space systems generally and that developers are specifically more concerned when these space systems are networked systems of systems. Pulling these threads together shows that SSSM as a secure modular reusable open-network software development framework address both the *Development Problem* and the *Security Problem.*

7.2 Contribution

The primary contributions of this research are the crystallization of a viable path forward for Small Space in the form of a secure modular reusable open-network software development framework that addresses the *Development* and *Security* Problems, and a working easy-to-use prototype of such a framework.

The SISDPA surveys and their analysis illustrate the need for a secure modular reusable open-networking software development framework. The surveys did this by showing that developers believe reuse and networking is important and beneficial for software developed for space systems, and that there was consensus amongst developers that networking has a negative effect on security and is even a hurdle in the adoption of OSAs and MONAs. These results show that providing better cybersecurity alleviates this security concern clearing the path for modular reusable open-network systems that can easily combine multi-vendor systems to provide capable resilient cost-effective space systems.

The SSSM work shows that a modular reusable open-network software development framework can have access control, encryption, and basic policy enforcement added to it with minimal increase in developer hardship thereby maintaining the same level of reusability from a developer perspective. There was a non-trivial increase in resource load at higher data throughput which will be addressed in future work. This sets up SSSM as one of the building blocks of a secure development platform the could integrate some of the process, tools, and additional solutions from Chapter 2.

7.3 Future Work

The problem space is always changing and growing; a software development framework, like SSSM, can be combined with other cybersecurity processes, tools, and solutions to create a more complete security platform. There are many different avenues for future work, here the focus will be on a few of them.

The performance of SSM is in need of improvement and so SSSM suffers from the same bottlenecks and adds additional burden. Performance analysis both in terms of more directed and controlled testing, and in terms dynamic and static performance profiling will determine were SSM and SSSM need optimization. More directed and controlled testing will isolate the SSM core services, producer, and consumer all on separate nodes instead of sharing resources to better characterize the overall performance of each piece. This work will also look at holding message size constant and increasing messaging rates to understand how handling more messages versus larger messages effects performance. Google PerfTools will be used to better understand performance bottlenecks and potential optimizations that might benefit SSM and SSSM thereby increasing the reusability of both software development frameworks. This testing could also target more representative hardware, again allowing for better characterization, with an eye towards understanding how additional security is affecting performance and therefor reusability.

The inherent vulnerabilities present in SSM and SSSM will be evaluated using tools like Fortify, OpenVAS, and Metasploit. SSSM could also be used as a testbed for determining the feasibility of using Sophia to provide continuous monitoring or using an other intrusion detection system (IDS) and intrusion prevention system (IPS) solutions. SSSM is a candidate for layering on-top of other security provisions, like a secure kernel or hypervisor that could provide a secure base and separation between process. Process separation is currently not provided by SSSM if all the software components reside on the same processor. Combination with other tools like this helps to address the data-at-rest problem that SSSM does not directly solve when untrusted processes or software share the same processor.

Another avenue of interest is the exploration of the applicability of SSM and SSSM to other problem domains like a smart hub for IOT, ICSs without deterministic requirements, and the smart grid. SSSM might be too heavy to run in full on a IOT sensor or endpoint, but the full stack could be run on a smart-hub, and different end-point versions of the software could be developed for different hardware against the open-network interface that SSM and SSSM provide.

SSSM would benefit from user-interface tools that made is easier to manage and validate policy, and look at complex rules, e.g. conditionally allow certain components to communicate only if the vehicle as in contact mode.

Future work will look at more formal analysis of reusability, using metrics like code complexity and additional surveys targeted at developer perception reusability. More directed surveys and additional analysis of the existing data set can still be used to better understand the problem space and keep track of it is changing over time. There are still some questions that relate to this area in the survey series that can be further analyzed as well as more detailed work with cross-tabulation.

The viability of space systems using reusable development frameworks should tracked in terms of the schedule costs to design, develop, test, and fly. Additional work can pull together measurements of cost and schedule for projects using reusable development frameworks and contrast that with more traditional development approaches. It will be difficult to normalize this against product requirements and capability as well as development personal, but would be very valuable in understanding if developer sentiment about reusable software actually proves to be economically beneficial. The longevity of the space systems should be tracked so that the effectiveness of this reusable modular open-network path can be quantified with the goal of understanding if these systems are really cheaper, more resilient, and more secure.

REFERENCES

[1] V. Pisacane, *Fundamentals of Space Systems*, ser. Applied Physics Laboratory series in science and engineering / The Johns Hopkins University. Oxford University Press, 2005. [Online]. Available: https://books.google.com/books?id=jf1TAAAAMAAJ

[2] L. R. Messeri and M. G. Richards, Eds., *Standards in the space industry: Looking back, looking forward*, ser. Management & organizational history. - Abingdon : Taylor & Francis, ISSN 1744-9359, ZDB-ID 22120002. - Vol. 4.2009, 3, p. 281-298. Abingdon: Taylor & Francis, 2009, vol. 4, no. 3.

[3] D. Leone, "NASA: James Webb Telescope Expected To Cost $8.7 Billion," http://spacenews.com/nasa-james-webb-telescope-expected-cost-87-billion/, 2011, [Online; accessed: 09-May-2016].

[4] M. Hicks, M. Enoch, L. Capots *et al.*, "Hexpak – a flexible, scalable architecture for responsive spacecraft," in *Proceedings of the 3rd Responsive Space Conference, Paper No. RS3-2005-3006*. Los Angeles, California: AIAA, April 2005.

[5] S. Clyde and J. E. Lascano, "Unifying definitions for modularization, abstraction, and encpasulation as a step toward foundational multi-paradigm software engineering principles," in *Proceedings of the Twelfth International Conference on Software Engineering Advances*, IARIA. Athens, Greece: International Academy, Research, and Industry Association (IARIA), October 2017, pp. 105–113. [Online]. Available: https://www.thinkmind.org/index.php?view=article&articleid=icsea_2017_5_20_10076

[6] J. H. Christensen, D. B. Anderson, M. E. Greenman, and B. D. Hansen, "Scalable network approach for the space plug-and-play architecture," in *Aerospace Conference, 2012 IEEE*, March 2012, pp. 1–10.

[7] J. H. Christensen, "Space plug-and-play architecture networking: A self-configuring heterogeneous network architecture," Ph.D. dissertation, Utah State University, December 2012. [Online]. Available: http://digitalcommons.usu.edu/etd/1422

[8] J. Cheng, "Northrop's Modular Space Vehicle gives Air Force faster satellite capability," https://defensesystems.com/articles/2014/03/03/msv-plug-and-play-satellite-northrop.aspx?admgarea=TC_DefenseIT, 2014, [Online; accessed: 01-June-2016].

[9] G. Ellis, "SNAP: MONA's Foundation at SMC," http://gsaw.org/wp-content/uploads/2014/03/2014s11e_ellis.pdf, 2014, [Online; accessed: 03-June-2016].

[10] C. J. Kief, B. Zufelt, S. R. Cannon, J. Lyke, and J. K. Mee, "The Advent of the PnP Cube Satellite," in *Aerospace Conference, 2012 IEEE*, March 2012, pp. 1–5.

[11] R. O. Bartlett, "Nasa standard multimission modular spacecraft for future space exploration," in *16th American Astronautical Society and Deutsche Gesellschaft fuer Luft- und Raumfahrt, Goddard Memorial Symposium*, Washington, D.C., March 1978.

[12] E. Falkenhayn, JR, "Multimission modular spacecraft (MMS)," in *Space Programs and Technologies Conference*. AIAA, June 1988.

[13] M. Bajracharya, M. W. Maimone, and D. Helmick, "Autonomy for mars rovers: Past, present, and future," *Computer*, vol. 41, no. 12, pp. 44–50, 2008.

[14] Department of Defense Directive 5000.01, *The Defense Acquisition System*. Department of Defense, May 2003.

[15] Office of the Under Secretary of Defense, *Acquisition, Technology and Logistics Memorandum: Better Buying Power 2.0: Continuing the Pursuit for Greater Efficiency and Productivity in Defense Spending*. Department of Defence, November 2012.

[16] Department of Defense, *Open Systems Architecture Contract Guidebook for Program Managers*. Department of Defense, June 2013, vol. 1.1.

[17] Department of Defense Instruction, *Operation of the Defense Acquisition System*. Department of Defense, January 2015.

[18] U.S. Government Accountability Office, *DOD Efforts to Adopt Open Systems for Its Unmanned Aircraft Systems Have Progressed Slowly*. Government Accountability Office, July 2013, GA0-13-651.

[19] J. Wilmot, "A core plug and play architecture for reusable flight software systems," in *Proceedings of the Space Mission Challenges for Information Technology, IEEE International Conference on*, vol. 0. Los Alamitos, CA, USA: IEEE Computer Society, 2006, pp. 443–447.

[20] ——, "A core flight software system," in *Proceedings of the 3rd IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, ser. CODES+ISSS '05. New York, NY, USA: ACM, 2005, pp. 13–14. [Online]. Available: http://doi.acm.org/10.1145/1084834.1084842

[21] ——, "Implications of responsive space on the flight software architecture, paper no. rs4-2006-6003," in *Proceedings of the AIAA 4th Responsive Space Conference*, NASA Goddard Space Flight Center. AIAA, Jan 2006.

[22] ——, "A reusable and adaptable software architecture for embedded space flight system: The core flight software system (cfs)," in *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis*, Jersey City, NJ, Sep 2005, pp. 18–21.

[23] "Driving down mission costs - new flight software package delivered to lunar mission," http://gsfctechnology.gsfc.nasa.gov/MissionCost.html, January 2010, [Online; accessed: 20-July-2017].

[24] D. L. Dvorak, "Nasa study on flight software complexity," *Final Report*, March 2009.

[25] SPA Committee, *Space Plug-and-Play Architecture Standard: Networking (AIAA S-133-2-2013)*. American Institute of Aeronautics and Astronautics, Inc., January 2013.

[26] J. Lyke, D. Fronterhouse, D. Lanza, and T. Byers, "A plug-and-play concept for spacecraft," *Military and Aerospace Programmable Logic Devices (MAPLD)*, September 2005.

[27] D. Lanza, J. Lyke, P. Zetocha, D. Fronterhouse, and D. Melanson, "Responsive space through adaptive avionics," in *Proceedings of the AIAA 2nd Responsive Space Conference*, vol. 6002. Los Angeles, CA: AIAA, April 2004.

[28] T. Morphopoulos, J. Hansen, J. Pollack, J. Lyke, and S. Cannon, "Plug-and-play – an enabling capability for responsive space missions," in *Proceedings of the AIAA 2nd Responsive Space Conference*, vol. 5002. Los Angeles, CA: AIAA, April 2004.

[29] J. Lyke, D. Fronterhouse, S. Cannon, D. Lanza, and T. Bryers, "Space plug-and-play avionics," in *Proceedings of the AIAA 3rd Responsive Space Conference*, AIAA. Los Angeles, California: AIAA, April 2005.

[30] J. Summers, "Plug and play testbed to enable responsive space missions," in *2005 IEEE Aerospace Conference*, March 2005, pp. 557–563.

[31] J. Lyke, S. Cannon, D. Fronterhouse, D. Lanza, and T. Byers, "A plug-and-play system for spacecraft components based on the usb standard," in *19th Annual AIAA/USU Conference on Small Satellites*. Logan, Utah: AIAA/USU, 2005. [Online]. Available: http://digitalcommons.usu.edu/smallsat/2005/all2005/9/

[32] G. Falco, *Cybersecurity Principles for Space Systems*. AIAA, December 2018, vol. 16, no. 2.

[33] D. Livingstone and P. Lewis, "Space, the final frontier for cybersecurity?" *Chatham House, September*, 2016.

[34] D. P. Fidler, "Cybersecurity and the new era of space activities," *Digital and Cyberspace Policy Program, April 2018*, 2018.

[35] U.S.-China Economic and Security Review Commission, *2011 Report to Congress*. U.S. Government Printing Office, 2011, [Online; accessed: 10-June-2016]. [Online]. Available: https://www.uscc.gov/sites/default/files/annual_reports/annual_report_full_11.pdf

[36] P. K. Martin and I. General, "Nasa cybersecurity: An examination of the agencys information security," *US House of Representatives*, Feb 2012.

[37] National Institute of Standards and Technology, U.S. Department of Commerce, *Guidelines for Smart Grid Cyber Security*. NIST, August 2010, NISTIR 7268.

[38] C.-C. Sun, A. Hahn, and C.-C. Liu, "Cyber security of a power grid: State-of-the-art," *International Journal of Electrical Power & Energy Systems*, vol. 99, pp. 45 – 56, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0142061517328946

[39] L. Coppolino, S. DAntonio, and L. Romano, "Exposing vulnerabilities in electric power grids: An experimental approach," *International Journal of Critical Infrastructure Protection*, vol. 7, no. 1, pp. 51 – 60, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1874548214000043

[40] B. Shirley, Q. Young, P. Wegner, J. Christensen, and J. Janicik, "Multi-layered security approaches for a modular open network architecture-based satellite," in *Proceedings of the AIAA/USU Conference on Small Satellites, Intelligent Software Systems, SSC14-II-3*. Logan, Utah: AIAA/USU, 2014. [Online]. Available: https://digitalcommons.usu.edu/smallsat/2014/IntellSoftware/3/

[41] D. Lane, E. Leon, D. Solio, D. Cunningham, D. Obukhov, and F. Tacliad, "High-assurance cyber space systems for small satellite mission integrity," in *Proceedings of the AIAA/USU Conference on Small Satellites, Ground, SSC17-V-01*. Logan, Utah: AIAA/USU, 2017. [Online]. Available: https://digitalcommons.usu.edu/smallsat/2017/all2017/95/

[42] K. Ingols, M. Zhivich, J. Brandon, and E. Koziel, "Cyber in a world of plenty: Secure high-performance on-orbit processing," in *Proceedings of the AIAA/USU Conference on Small Satellites, Advanced Technologies 3, SSC17-XII-08*. Logan, Utah: AIAA/USU, 2017. [Online]. Available: https://digitalcommons.usu.edu/smallsat/2017/all2017/161/

[43] D. Cunningham, G. P. Jr., and J. Romero-Mariona, "Towards effective cybersecurity for modular, open architecture satellite systems," in *Proceedings of the AIAA/USU Conference on Small Satellites, Advanced Technologies 1, SSC16-IV-6*. Logan, Utah: AIAA/USU, 2016. [Online]. Available: https://digitalcommons.usu.edu/smallsat/2016/TS4AdvTech1/6/

[44] S. Nazir, S. Patel, and D. Patel, "Assessing and augmenting scada cyber security: A survey of techniques," *Computers & Security*, vol. 70, pp. 436 – 454, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167404817301293

[45] S. Jackson, J. Straub, and S. Kerlin, "Exploring a novel cryptographic solution for securing small satellite communications." *IJ Network Security*, vol. 20, no. 5, pp. 988–997, 2018.

[46] J. T. Kohl, B. C. Neuman, and T. Y. Tso, "The evolution of the kerberos authentication service," in *Distributed Open Systems*. IEEE Computer Society Press, 1994, pp. 78–94.

[47] J. Arfman and P. Roden, "Project athena: Supporting distributed computing at mit," *IBM Systems Journal*, vol. 31, no. 3, pp. 550–563, 1992.

[48] R. Nugent, R. Munakata, A. Chin, R. Coelho, and J. Puig-Suari, "The cubesat: The picosatellite standard for research and education," in *SPACE Conference Conference and Exposition*. San Diego, California: AIAA, September 2008, pp. 756–766.

[49] J. John Viega, M. Messier, and P. Chandra, *Network Security with OpenSSL: Cryptography for Secure Communications*. O'Reilly Media, Inc., 2002.

[50] National Institute of Standards and Technology, U.S. Department of Commerce, *Security requirements for cryptographic modules, Federal Information Processing Standards Publication (FIPS PUB) 140-2*. NIST, 2002, NISTIR 7268.

[51] SPA Committee, *Space Plug-and-Play Architecture Standard: Logical Interface (AIAA S-133-3-2013)*. American Institute of Aeronautics and Astronautics, Inc., January 2013.

[52] A. V. Konstantinou, D. Florissi, and Y. Yemini, "Towards self-configuring networks," in *DARPA Active Networks Conference and Exposition, 2002. Proceedings*. IEEE, May 2002, pp. 143–156.

[53] IEEE Standards Association, "Ieee standard for heterogeneous interconnect (hic) (low-cost, low-latency scalable serial interconnect for parallel system construction)," *IEEE Standard 1355-1995*, 1996.

[54] B. M. Cook and C. P. H. Walker, "Spacewire and ieee 1355 revisited," in *International Spacewire Conference*, vol. 17, 2007, p. 19.

[55] European Cooperations for Space Standardization, "Spacewire – links, nodes, routers and networks," *ECSS-E-ST-50-12C*, pp. 1–129, 2008.

[56] "ISO/IEC 7498-1:1994(E), Information technology Open Systems Interconnection Basic Reference Model: The Basic Model. ISO," http://standards.iso.org/ittf/PubliclyAvailableStandards/s020269ISOIEC7498-11994(E).zip, 2005, [Online; accessed: 03-June-2016].

[57] J. Christensen, S. Cannon, B. Hansen, and J. Lyke, "Automatic generation of sdm application source code from xteds," in *Proceedings of the AIAA/USU Conference on Small Satellites, Connecting the Dots: Bringing Visionaries, System Implementers & Mission Sponsors Together, SSC10-X-5*. Logan, Utah: AIAA/USU, 2010. [Online]. Available: http://digitalcommons.usu.edu/smallsat/2010/all2010/57

[58] "Xtce vs xteds: A brief comparison," https://cwe.ccsds.org/sois/docs/SOIS-APP/Meeting%20Materials/2010/Fall/PnP%20Background/XTCE_vs_xTEDS_V0.2.docx, January 2010, [Online; accessed: 24-March-2017].

[59] IEEE Standards Association, "Standard for a smart transducer interface for sensors and actuators-mixed-mode communication protocols and transducer electronic data sheet (teds) formats," *IEEE Standard 1451.4-2004*, pp. 1–430, 2004.

[60] S. A. McDermott and D. J. Goldstein, "The Bitsy™ spacecraft kernel: reducing mission cost with modular architecture and miniature technology," in *Aerospace Conference Proceedings, 2000 IEEE*, vol. 4. IEEE, March 2000, pp. 1–6.

[61] P. Peti, R. Obermaisser, W. Elmenreich, and T. Losert, "An architecture supporting monitoring and configuration in real-time smart transducer networks," in *Sensors, 2002. Proceedings of IEEE*, vol. 2. IEEE, 2002, pp. 1479–1484.

APPENDICES

APPENDIX A
SISDPA Survey Series Instruments

A.1 Introduction

This appendix contains the material used to obtain approval from the USU's IRB and the material used to administer the survey, i.e. the recruitment materials and printouts for each survey. It also includes definitions for the items used in the *Pro-neutral-con* questions that was available to participants while they took the survey.

Section A.2 contains the Letter of Information submitted to USU's IRB. Section A.3 contains the renewed approval letter from USU's IRB. Section A.4 contains the language used in the survey recruitment emails. Sections A.5 through A.9 contains the survey printouts for each survey. Section A.10 contains the *Pro-neutral-con* question term definitions.

A.2 LOI

Department of Computer Science
4205 Old Main Hill
Logan UT 84322-4205
Telephone: (435) 797-2451

Page 1 of 3
USU IRB Exemption Granted: 7/28/2015
USU IRB Exemption Expires: 7/27/2016
Amendment #1 Approved: 12/29/2015
Protocol #6484
IRB Password Protected per IRB Director

LETTER OF INFORMATION

A Survey of Space Industry Software Development Practices and Attitudes

Introduction/ Purpose

Dr. Stephen Clyde (PI) & Brandon Shirley (Graduate student researcher) in the Department of Computer Science at Utah State University are conducting a set of surveys to find out more about software development practices and attitudes in the space industry, specifically flight, ground, and testing software. We are asking you to take part in this set of surveys because of your affiliation with or involvement in the space industry. We are soliciting responses from approximately 3000 professionals, educators, and students with backgrounds and connections to the space industry. We greatly need your participation and your responses to the surveys will make a meaningful contribution to this research.

Procedures

Multiple surveys compromise this survey set. Currently there are six surveys (listed below) and each should take approximately 10 minutes or less to complete. Each survey has a brief introduction section that explains the overall survey set, i.e. *Space Industry Software Development Practices and Attitudes* (SISDPA), and the specific survey you are taking, e.g. *Open Systems Architecture and Modularity*.

1) SISDPA : Core Concepts
2) SISDPA : Development Preferences
3) SISDPA : Open Systems Architecture and Modularity
4) SISDPA : Security
5) SISDPA : Reuse, Interoperability, Portability, Code Complexity
6) SISDPA : Network

Each survey will also have a common background section, this section is the same across all the surveys and the hope is that if you take more than one survey you will give identical responses across each survey. This will allow the researchers to consider your experience with regard to the responses you provide.

If you agree to participate, then you may use the provided email links to access and complete the surveys. The links provided are anonymous. We plan to distribute the surveys over the space of a few months. We expect to give you two to three weeks to complete the survey, i.e. from the time you receive the original distribution email you will have two to three weeks to complete the survey.

You can opt to only take some of them. If you start a survey but do not finish, then your partial result may be included in the results. You may contact Dr. Stephen Clyde or Brandon Shirley (see contact information provide in the "Explanation & offer to answer questions" section) if you have any questions with regard to participation.

At the end of each survey you will be redirected to another webpage where you may provide your email address if you want to enter the optional drawing. Your email address will not be connected to the survey responses. See the Payments/Compensation Section for more information on the drawing.

Page 2 of 3
USU IRB Exemption Granted: 7/28/2015
USU IRB Exemption Expires: 7/27/2016
Amendment #1 Approved: 12/29/2015
Protocol #6484
IRB Password Protected per IRB Director

UtahState University
Department of Computer Science
4205 Old Main Hill
Logan UT 84322-4205
Telephone: (435) 797-2451

LETTER OF INFORMATION

A Survey of Space Industry Software Development Practices and Attitudes

<u>New Findings</u>
During the course of this research study, you will be informed of any significant changes in the procedures, risks or benefits resulting from participation in the research, or new alternatives to participation that might cause you to change your mind about continuing in the study. If necessary, this Letter of Information may be amended to reflect said changes.

<u>Risks</u>
There is minimal risk involved in participating in any of the surveys in the set. The only potential stress factor that we foresee is the duration of taking the entire survey set. We spread out the distribution of the various surveys to minimize this potential stress.

The surveys collect some information about your background and experience so we can better understand your answers and opinions, but we do not collect or store any other personal identifying information. There is a virtually no risk of loss of confidentiality relative to the data collective. We ensure this to be the case by doing the following:

- The anonymous nature of the link used to take the survey automatically disassociates your identity and contact information from your responses
- We will keep the raw survey data private and properly secured
- We will only publish aggregated data – no individual responses will be made public

<u>Benefits</u>
The main benefit of this study is an increase in the body of knowledge that relates to software development practices and attitudes in the space industry, and to direct future research. Taking the survey should provide you with an opportunity to reflect on your current software development practices and how they affect the projects on which you have and will work.

Once the survey is closed, we will analyze the data, and generate results. We will seek publication of these results in a conference or journal.

<u>Explanation & offer to answer questions</u>
The above Procedures section explains this research study. If you have other questions or research-related problems, you may reach Dr. Stephen Clyde at (435) 797- 2307 or Stephen.Clyde@usu.edu, or Brandon Shirley at (435) 994-9165 or b.l.s@aggiemail.usu.edu.

<u>Voluntary nature of participation and right to withdraw without consequence</u>
Participation in these surveys is voluntary. You may refuse to participate or withdraw at any time without consequence or loss of benefits. You may opt-out at any time by either not starting any of the surveys or only taking some of the surveys. The surveys you have started and/or completed will be include in the results; this is due the anonymous links used to administer the survey. We have no way to identify your responses and so we cannot remove them. Starting a survey does not obligate you to complete it, nor are you obligated to complete all the surveys just because you completed one.

Page 3 of 3
USU IRB Exemption Granted: 7/28/2015
USU IRB Exemption Expires: 7/27/2016
Amendment #1 Approved: 12/29/2015
Protocol #6484
IRB Password Protected per IRB Director

Department of Computer Science
4205 Old Main Hill
Logan UT 84322-4205
Telephone: (435) 797-2451

LETTER OF INFORMATION

A Survey of Space Industry Software Development Practices and Attitudes

Payment/Compensation

There will be seven drawings for gift cards: one for each survey and one overall drawing for the survey set. For each survey-specific drawing, two participants in that survey will be selected at random to receive a $25 gift card. For the overall drawing, two participants from the pool of all participants of any survey will be selected to receive a $200 gift card, such that if you have completed all of the surveys, you will have six entries into that drawing. At the end of each of the surveys, you will be redirected to a webpage that asks for an email address. A participant must enter a valid email address to be considered for that survey's drawing or the overall drawing.

Confidentiality

There is virtually no risk of a privacy or confidentiality breach. The researchers will not be observing or collecting data on any behavior outside of what would normally occur in the work place. Research records will be kept confidential, consistent with federal and state regulations. Only the investigator and the graduate student researcher will have access to the raw data. No identifying information is associated with the survey, the links provided in the distribution email are anonymous.

IRB Approval Statement

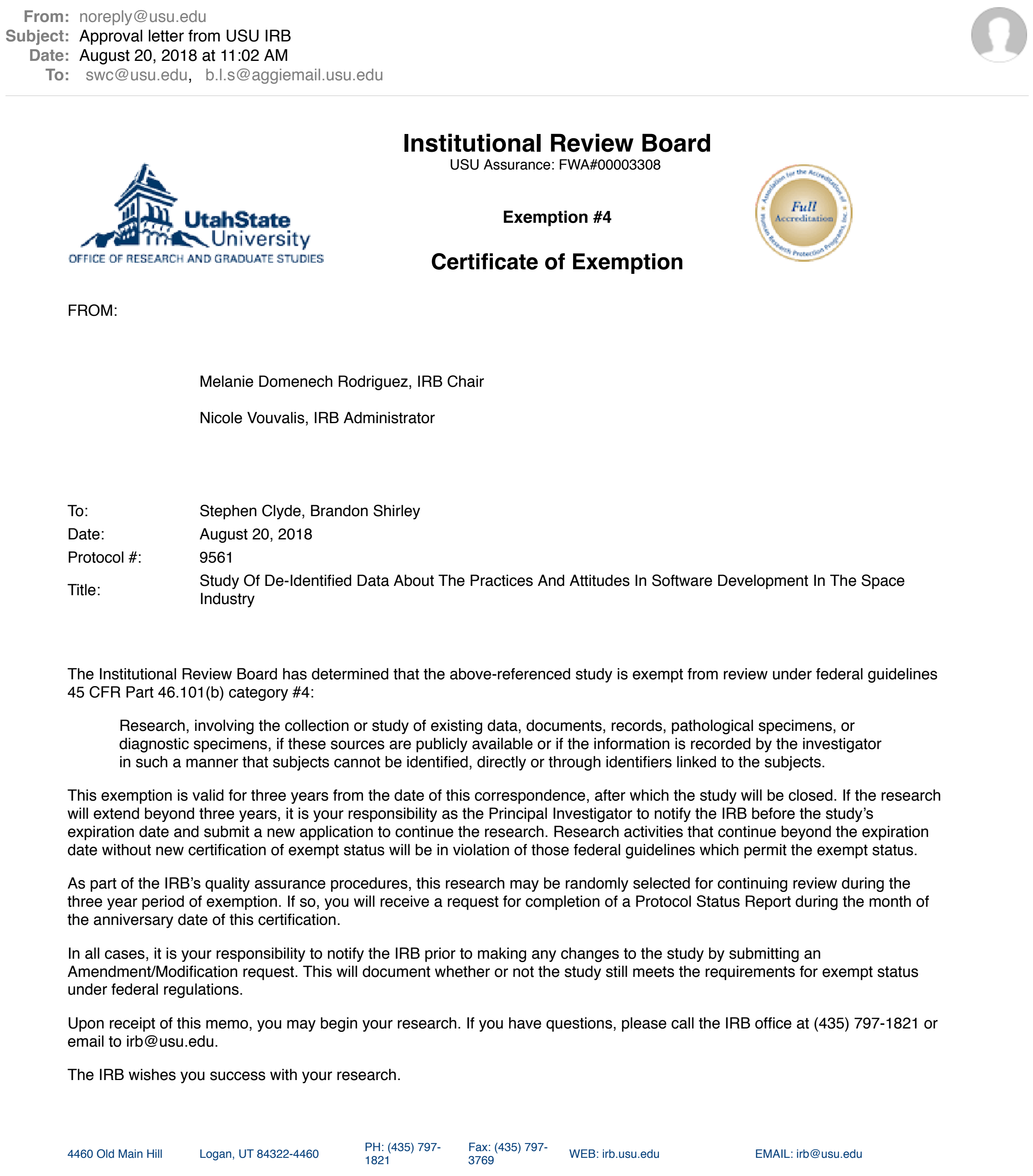
The Institutional Review Board for the protection of human participants at Utah State University has approved this research study. If you have any questions or concerns about your rights or a research-related injury and would like to contact someone other than the research team, you may contact the IRB Director at (435) 797-0567 or email irb@usu.edu to obtain information or to offer input.

Investigator Statement

"I certify that the research study has been explained to the individual, by me or my research staff, and that the individual understands the nature and purpose, the possible risks and benefits associated with taking part in this research study. Any questions that have been raised have been answered."

Dr. Stephen Clyde	Brandon Shirley
Principal Investigator	Student Researcher
(435) 797-2307	(435) 994-9165
Stephen.Clyde@usu.edu	b.l.s@aggiemail.usu.edu

A.3 Extension Approval Letter



From: noreply@usu.edu
Subject: Approval letter from USU IRB
Date: August 20, 2018 at 11:02 AM
To: swc@usu.edu, b.l.s@aggiemail.usu.edu

Institutional Review Board
USU Assurance: FWA#00003308

UtahState University
OFFICE OF RESEARCH AND GRADUATE STUDIES

Exemption #4

Certificate of Exemption

FROM:

Melanie Domenech Rodriguez, IRB Chair

Nicole Vouvalis, IRB Administrator

To:	Stephen Clyde, Brandon Shirley
Date:	August 20, 2018
Protocol #:	9561
Title:	Study Of De-Identified Data About The Practices And Attitudes In Software Development In The Space Industry

The Institutional Review Board has determined that the above-referenced study is exempt from review under federal guidelines 45 CFR Part 46.101(b) category #4:

> Research, involving the collection or study of existing data, documents, records, pathological specimens, or diagnostic specimens, if these sources are publicly available or if the information is recorded by the investigator in such a manner that subjects cannot be identified, directly or through identifiers linked to the subjects.

This exemption is valid for three years from the date of this correspondence, after which the study will be closed. If the research will extend beyond three years, it is your responsibility as the Principal Investigator to notify the IRB before the study's expiration date and submit a new application to continue the research. Research activities that continue beyond the expiration date without new certification of exempt status will be in violation of those federal guidelines which permit the exempt status.

As part of the IRB's quality assurance procedures, this research may be randomly selected for continuing review during the three year period of exemption. If so, you will receive a request for completion of a Protocol Status Report during the month of the anniversary date of this certification.

In all cases, it is your responsibility to notify the IRB prior to making any changes to the study by submitting an Amendment/Modification request. This will document whether or not the study still meets the requirements for exempt status under federal regulations.

Upon receipt of this memo, you may begin your research. If you have questions, please call the IRB office at (435) 797-1821 or email to irb@usu.edu.

The IRB wishes you success with your research.

4460 Old Main Hill   Logan, UT 84322-4460   PH: (435) 797-1821   Fax: (435) 797-3769   WEB: irb.usu.edu   EMAIL: irb@usu.edu

A.4 Survey Recruitment Letter

Department of Computer Science
4205 Old Main Hill
Logan UT 84322-4205
Telephone: (435) 797-2451

Dr. Stephen Clyde
Principal Investigator
Stephen.Clyde@usu.edu
(435) 797-2307

Brandon Shirley
Student Researcher
b.l.s@aggiemail.usu.edu
(435) 994-9165

Space Industry Software Development Practices and Attitudes (SISDPA)

Recruitment Letter

1 INTRODUCTION

Dr. Stephen Clyde and Brandon Shirley plan to conduct a set of surveys on Space Industry Software Development Practices and Attitudes; specifically in the areas of flight, ground, and test software. The letter below will accompany the link for the appropriate survey. Distribution may occur via letter, email, or website link.

2 RECRUITMENT CONTENT

The emails will vary slightly based on the distribution method, e.g. for Qualtrics generated and distributed emails we will have the name of the person to whom we are distributing the email.

2.1 QUALTRICS

From: b.l.s@aggiemail.usu.edu

Dear firstName,

2.2 MAILING LISTS

From: Brandon.Shirley@sdl.usu.edu

To whom it may concern,

2.3 COMMON

Subject: New Distribution of **Current Survey** or Reminder Email for **Current Survey**

Dr. Stephen Clyde & Brandon Shirley in the Department of Computer Science at Utah State University are conducting a set of surveys as part of Brandon's PhD research to find out more about software development practices and attitudes in the space industry.

You have received this email because of your interest in or involvement with the space industry. We greatly need your participation. We are asking that anyone that has insight into in software development that relates to space industry participate in this survey. The beginning of the survey has background questions that will give context to your responses.

The survey set is currently made of six surveys, listed below, that will be distributed over the course of a few months. You have a chance at receiving a gift card for participating in this survey as well as a chance

Department of Computer Science
4205 Old Main Hill
Logan UT 84322-4205
Telephone: (435) 797-2451

Dr. Stephen Clyde
Principal Investigator
Stephen.Clyde@usu.edu
(435) 797-2307

Brandon Shirley
Student Researcher
b.l.s@aggiemail.usu.edu
(435) 994-9165

at receiving a gift card for your overall participation in the entire survey set. There will be seven drawings for gift cards: one drawing for each survey and one overall drawing for the survey set. For each survey-specific drawing, two participants in that survey will be selected at random to receive a $25 gift card. For the overall drawing, two participants from the pool of all participants of any survey will be selected to receive a $200 gift card. At the end of this survey, you will be redirected to a webpage that asks for an email address. You must enter a valid email address to be considered for this surveys drawing or the overall survey set drawing.

Each survey stands on its own: opting to not participate in one survey does not exclude you in any way from participating in any subsequent surveys. Likewise, participating in one survey does not obligate you to participate in any subsequent surveys. The surveys are as follows and will be distributed in the following order:

1) SISDPA : Core Concepts
2) SISDPA : Development Preferences
3) SISDPA : Open Systems Architecture and Modularity
4) SISDPA : Security
5) SISDPA : Reuse, Interoperability, Portability, Code Complexity
6) SISDPA : Network

You can you use the link below to access the __________ survey.

We greatly need your participation and your responses to the surveys will make a meaningful contribution to this research while allowing you an opportunity to reflect on you current software development practices and how these practices affect the projects on which you have and will work. This survey set will be used to determine the path of future research, as well as increase the body of knowledge that relates to software development practices and attitudes in the space industry.

You can access the _______ survey of the Space Industry Software Development Practices and Attitudes survey set using the following link:

http://usu.qualtrics.link.com.

You can also copy and paste the following URL into your address bar if you prefer not to use the provided link: http://usu.qualtrics.link.com

We plan to send out one reminder email a week until the survey closes. The duration of the survey will be two to three weeks. You will receive the reminder email regardless of where or not you have participated in a survey. The subject of the email will read “Reminder Email…” for these weekly reminders, and “New Distribution…” at the start of a new survey.

See the http://brandon.bluezone.usu.edu/Files/LOISpaceSoftwareAttitudes_Final.pdf for the Letter of Intent (LOI) that explains your role as a participant should you choose to participate. If you have questions please direct them to Brandon Shirley, via email b.l.s@aggiemail.usu.edu, the LOI lists additional contact information.

Once the survey set is closed, we will analyze the data, and generate results. We will seek publication of these results in a conference or journal.

Department of Computer Science
4205 Old Main Hill
Logan UT 84322-4205
Telephone: (435) 797-2451

Dr. Stephen Clyde
Principal Investigator
Stephen.Clyde@usu.edu
(435) 797-2307

Brandon Shirley
Student Researcher
b.l.s@aggiemail.usu.edu
(435) 994-9165

This is a legitimate request for you participation, if you have any questions about the validity of this email you may refer to the Letter of Intent, contact Brandon Shirley, via email b.l.s@aggiemail.usu.edu, or Utah State University's Internal Review Board administrator at (435) 797 – 0567 or email irb@usu.edu.

2.4 Special Note for Core Concepts Survey

Note that the Core Concepts Survey was previously distributed. If you participated in the survey already then please visit the survey link and you will be given the opportunity to provide your email for entry into the Core Concepts Survey drawing as well as the overall survey set drawing.

3 Thank You Content

We will send out a thank you email to thank everyone, whether they have participated or not, that thanks them for participating, lets them know the current survey is closed, and notifies them of the timeframe within which the drawing winners will be notified.

3.1 Content

Subject: **Current Survey** has ended

Dr. Stephen Clyde & Brandon Shirley in the Department of Computer Science at Utah State University would like to thank everyone who participated in the ______ survey; the ______ survey has now closed. We will notify the winners of this survey's drawing within __ week(s). The winners of the overall survey set will be notified after the entire survey set closes.

4 Conclusion

We may need to distribute some additional emails, but they we will be in line with what we have here and will be strictly for survey management or clarification. I may also make grammatical corrections if needed.

A.5 *CC* Survey

SISDPA : Core Concepts - Survey

Start of Block: Introduction

Intro – Dr. Stephen Clyde & Brandon Shirley in the Department of Computer Science at Utah State University are conducting a set of surveys as part of Brandon's PhD research to find out more about software development practices and attitudes in the space industry.

You are being asked to take part in this set of surveys because of your affiliation with or involvement in the space industry. Your participation and your responses to the surveys are greatly needed and will make a meaningful contribution to this research.

This survey comprises the **Core Concepts** portion of a survey set that makes up a survey on Space Industry Software Development Practices and Attitudes (SISDPA). Each question will ask for your input and explain how you should answer the question.

There are two main benefits for participating in this study. One benefit is an increase in the body of knowledge that relates to software development practices and attitudes in the space industry while informing future research. The other benefit is the chance to receive a gift card for participating in this survey as well as a chance to receive a gift card for your overall participation in the entire survey set. There will be seven drawings for gift cards: one for each survey and one overall drawing for the survey set. For each survey-specific drawing, two participants in that survey will be selected at random to receive a $25 gift card. For the overall drawing, two participants from the pool of all participants of any survey will be selected at random to receive a $200 gift card. Taking the survey should provide you with an opportunity to reflect on your current software development practices and how they affect the projects on which you have and will work.

At the end of this survey you will be redirected to another webpage where you may provide your email address if you want to enter the optional drawings. Your email address will not be connected to the survey responses. You must enter a valid email address to be considered for this survey's drawing or the overall drawing.

If you start this survey but do not finish, then your partial result may be included in the results. Once the survey set is closed, we will analyze the data for inclusion in a conference or journal paper.

If you have other questions or research-related problems, you may reach Dr. Stephen Clyde at (435) 797- 2307 or Stephen.Clyde@usu.edu, or Brandon Shirley at (435) 994-9165 or b.l.s@aggiemail.usu.edu.

Please note that this survey was previously distributed. For those of you who have already participated and would like the opportunity to win the drawing for this survey or for the overall survey set you can specify that you have already participated in the next section.

End of Block: Introduction

Start of Block: Block 3

Q15 – Have you already taken this survey?

- ○ Yes
- ○ No

End of Block: Block 3

Start of Block: Background

1.0 – Timing
First Click
Last Click
Page Submit
Click Count

1.1 – Please select your role(s) in space systems development. You must select at least one role.

Software engineer
- ☐ Electrical engineer
- ☐ Systems engineer
- ☐ Mechanical engineer
- ☐ Aerospace engineer
- ☐ Program Manager
- ☐ Technical Lead
- ☐ Thermal engineer
- ☐ Principle Investigator
- ☐ Other (please specify) ____

1.2 – Please categorize the entity for which you currently work using the options below. You must select at least one category.

- ☐ Industry
- ☐ SETA/UARC/FFRDC
- ☐ Government
- ☐ Other (please specify) ____

1.3 – How many years have you spent working in the following areas with regard to space systems? Please drag the sliders to indicate the number of years of experience you have for each area. You must click or move each slider even if you want your responses marked as zero, the slider will turn purple/blue upon input.

	Years 0 5 10 15 20 25 30 35 40 45 50
Software Systems Development	
Management	
Hardware Systems Development	
Procurement	
Other (please specify)	

1.4 – Based on you experience, i.e. the projects on which you have worked, what are the typical durations for the phases listed below? Please drag the sliders to indicate the number of months you think are typically spent on each phase. You must click or move each slider even if you want your responses marked as zero, the slider will turn purple/blue upon input. If you do not have direct experience, then go off of what you think is typical.

	Months 0 10 20 30 40 50 60 70 80 90 100110120
Planning	
Development	
Testing	
Operations	
Other (please specify)	

1.5 – About how many missions/projects have you worked on over the course of your career in the following areas? Please drag the sliders to indicate the number missions/projects on which you have worked. You must click or move each slider even if you want your responses marked as zero, the slider will turn purple/blue upon input.

	0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30
Space Systems	
Other fields	

1.6 – About how many missions/projects do you typically work on at the same time? Please drag the sliders to indicate the number of projects that you typically work on at the same time. You must click or move each slider even if you want your response marked as zero, the slider will turn purple/blue upon input.

	0 1 2 3 4 5 6 7 8 9 10
Projects/Missions	

End of Block: Background

Start of Block: Core Concepts

2.1 – For each software characteristic listed in a row below, rate its importance in each of the four domains: Space Flight Software, Space Ground Software, Space Test Software, and Other Software Fields. 1 means "not important" and 5 means "very important."

	Space Flight Software					Other Software Fields					Space Ground Software					Space Test Software				
	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5
Reuse																				
Portability																				
Interoperability																				
Minimal Code Complexity																				
Rapid Development																				
Cost of Ownership (e.g. maintenance, upgrades)																				
Security																				

2.2 – Please order the following software characteristics according to importance, by dragging them so that the most important (1) comes first and the least important (7) comes last. You must move at least one characteristic for the question to be marked as answered, you can move it back if you feel the initial ordering is correct.

______ Reuse
______ Portability
______ Interoperability
______ Minimal Code Complexity
______ Rapid Development
______ Cost of Ownership (e.g. maintenance, upgrades)
______ Security

2.3 – Timing
First Click
Last Click
Page Submit
Click Count

2.4 – What percentage of your time do you spend developing in the following areas? Please drag the bars so that the total adds up to 100.
 _______ Space Flight Software
 _______ Space Ground Software
 _______ Space Test Software
 _______ Other software fields (please specify)
 _______ Non-software (please specify)

2.5 – Timing
First Click
Last Click
Page Submit
Click Count

End of Block: Core Concepts

A.6 *OSAM* Survey

SISDPA : Open Systems Architecture and Modularity - Survey

Start of Block: Introduction

Intro – I'm Brandon Shirley, I am conducting these surveys for my PhD research at Utah State University. I really need your participation. There is a chance to win some gift cards.

You are being asked to take part in this set of surveys because of your affiliation with or involvement in the space industry. Your participation and your responses to the surveys are greatly needed and will make a meaningful contribution to this research.

This survey comprises the **Open Systems Architecture and Modularity** portion of a survey set that makes up a survey on Space Industry Software Development Practices and Attitudes (SISDPA). Each question will ask for your input and explain how you should answer the question.

Answer as many of the questions as you want, partial surveys may still be very helpful. At the end of this survey, you will be redirected to a webpage that asks for an email address. You must enter a valid email address to be considered for survey drawings or the overall survey set drawing.

You have a chance at receiving a gift card for participating in this survey as well as a chance at receiving a gift card for your overall participation in the entire survey set. There will be 2 winners of $25 gift cards for each survey and 2 winners of $200 gift cards for the survey set.

At the end of this survey you will be redirected to another webpage where you may provide your email address if you want to enter the optional drawings. Your email address will not be connected to the survey responses. You must enter a valid email address to be considered for this survey's drawing or the overall drawing.

If you start this survey but do not finish, then your partial result may be included in the results.

Once the survey set is closed, we will analyze the data for inclusion in a conference or journal paper.

If you have other questions or research-related problems, you may reach Dr. Stephen Clyde at (435) 797- 2307 or Stephen.Clyde@usu.edu, or Brandon Shirley at (435) 994-9165 or b.l.s@aggiemail.usu.edu.

End of Block: Introduction

Start of Block: Open Systems Architecture and Modularity

2.0 – Timing
First Click
Last Click
Page Submit
Click Count

2.1 – Consider the items on the left in the context of **open systems architecture** in space flight systems. In this instance open system architecture means vendor-independent, non-proprietary, software or device design and implementation based on official and/or popular standards. Place an item under **Pros** if positively impacted, under **Neutral** if not impacted or not applicable, and under **Cons** if negatively impacted by utilizing open systems architecture. Think of this impact in terms of multi-project, multi-mission, or multi-platform use. **All items should be placed for the answer to be considered complete by the system. In considering partial or incomplete answers, unplaced items will be considered neutral.**

Pros	Neutral	Cons
Regression reduction	Regression reduction	Regression reduction
Code design	Code design	Code design
Development cost	Development cost	Development cost
Maintenance cost	Maintenance cost	Maintenance cost
Development productivity	Development productivity	Development productivity
Development efficiency	Development efficiency	Development efficiency
Code Complexity	Code Complexity	Code Complexity
Maintainability	Maintainability	Maintainability
Integration	Integration	Integration
Adaptability	Adaptability	Adaptability
Documentation/Examples	Documentation/Examples	Documentation/Examples
Encapsulation	Encapsulation	Encapsulation
Bug detection	Bug detection	Bug detection
Code quality	Code quality	Code quality
Code robustness	Code robustness	Code robustness
Best practices	Best practices	Best practices
Schedule	Schedule	Schedule

Code or algorithm optimization/efficiency	Code or algorithm optimization/efficiency	Code or algorithm optimization/efficiency
Uniformity of coding style	Uniformity of coding style	Uniformity of coding style
Domain knowledge	Domain knowledge	Domain knowledge
Code readability	Code readability	Code readability
Security	Security	Security
I/0 efficiency	I/0 efficiency	I/0 efficiency
Radiation hardness	Radiation hardness	Radiation hardness
Fault tolerance	Fault tolerance	Fault tolerance
Hardware complexity	Hardware complexity	Hardware complexity
Latency	Latency	Latency
Determinism	Determinism	Determinism
Interoperability	Interoperability	Interoperability
Portability	Portability	Portability
Testing	Testing	Testing
Reusability	Reusability	Reusability
Software upgradability	Software upgradability	Software upgradability
Hardware changes/flexibility	Hardware changes/flexibility	Hardware changes/flexibility
Adoption rates/software proliferation	Adoption rates/software proliferation	Adoption rates/software proliferation
Ease of use	Ease of use	Ease of use
Mission/Project requirement changes	Mission/Project requirement changes	Mission/Project requirement changes
Information Assurance	Information Assurance	Information Assurance
Mission Assurance	Mission Assurance	Mission Assurance

2.2 – Consider the items on the left in the context of **modular open network architecture** in space flight systems. Modular open network architecture implies that the architecture is designed natively for network-centric data transfer. Place an item under **Pros** if positively impacted, under **Neutral** if not impacted or not applicable, and under **Cons** if negatively impacted by utilizing modular open network architecture. Think of this impact in terms of multi-project, multi-mission, or multi-platform use. **All items should be placed for the answer to be considered complete by the system. In considering partial or incomplete answers, unplaced items will be considered neutral.**

Pros	Neutral	Cons
Regression reduction	Regression reduction	Regression reduction
Code design	Code design	Code design
Development cost	Development cost	Development cost
Maintenance cost	Maintenance cost	Maintenance cost
Development productivity	Development productivity	Development productivity
Development efficiency	Development efficiency	Development efficiency
Code Complexity	Code Complexity	Code Complexity
Maintainability	Maintainability	Maintainability
Integration	Integration	Integration
Adaptability	Adaptability	Adaptability
Documentation/Examples	Documentation/Examples	Documentation/Examples
Encapsulation	Encapsulation	Encapsulation
Bug detection	Bug detection	Bug detection
Code quality	Code quality	Code quality
Code robustness	Code robustness	Code robustness
Best practices	Best practices	Best practices
Schedule	Schedule	Schedule
Code or algorithm optimization/efficiency	Code or algorithm optimization/efficiency	Code or algorithm optimization/efficiency
Uniformity of coding style	Uniformity of coding style	Uniformity of coding style
Domain knowledge	Domain knowledge	Domain knowledge
Code readability	Code readability	Code readability
Security	Security	Security
I/0 efficiency	I/0 efficiency	I/0 efficiency
Radiation hardness	Radiation hardness	Radiation hardness

Fault tolerance	Fault tolerance	Fault tolerance
Hardware complexity	Hardware complexity	Hardware complexity
Latency	Latency	Latency
Determinism	Determinism	Determinism
Interoperability	Interoperability	Interoperability
Portability	Portability	Portability
Testing	Testing	Testing
Reusability	Reusability	Reusability
Software upgradability	Software upgradability	Software upgradability
Hardware changes/flexibility	Hardware changes/flexibility	Hardware changes/flexibility
Adoption rates/software proliferation	Adoption rates/software proliferation	Adoption rates/software proliferation
Ease of use	Ease of use	Ease of use
Mission/Project requirement changes	Mission/Project requirement changes	Mission/Project requirement changes
Information Assurance	Information Assurance	Information Assurance
Mission Assurance	Mission Assurance	Mission Assurance

2.3 – Consider the items on the left in the context of **software modularity** in space flight systems, specifically the degree to which software is divided into functional modules. Place an item under **Pros** if positively impacted, under **Neutral** if not impacted or not applicable, and under **Cons** if negatively impacted. Think of this impact in terms of multi-project, multi-mission, or multi-platform use. **All items should be placed for the answer to be considered complete by the system. In considering partial or incomplete answers, unplaced items will be considered neutral.**

Pros	Neutral	Cons
Regression reduction	Regression reduction	Regression reduction
Code design	Code design	Code design
Development cost	Development cost	Development cost
Maintenance cost	Maintenance cost	Maintenance cost
Development productivity	Development productivity	Development productivity
Development efficiency	Development efficiency	Development efficiency

Code Complexity	Code Complexity	Code Complexity
Maintainability	Maintainability	Maintainability
Integration	Integration	Integration
Adaptability	Adaptability	Adaptability
Documentation/Examples	Documentation/Examples	Documentation/Examples
Encapsulation	Encapsulation	Encapsulation
Bug detection	Bug detection	Bug detection
Code quality	Code quality	Code quality
Code robustness	Code robustness	Code robustness
Best practices	Best practices	Best practices
Schedule	Schedule	Schedule
Code or algorithm optimization/efficiency	Code or algorithm optimization/efficiency	Code or algorithm optimization/efficiency
Uniformity of coding style	Uniformity of coding style	Uniformity of coding style
Domain knowledge	Domain knowledge	Domain knowledge
Code readability	Code readability	Code readability
Security	Security	Security
I/0 efficiency	I/0 efficiency	I/0 efficiency
Radiation hardness	Radiation hardness	Radiation hardness
Fault tolerance	Fault tolerance	Fault tolerance
Hardware complexity	Hardware complexity	Hardware complexity
Latency	Latency	Latency
Determinism	Determinism	Determinism
Interoperability	Interoperability	Interoperability
Portability	Portability	Portability
Testing	Testing	Testing
Reusability	Reusability	Reusability
Software upgradability	Software upgradability	Software upgradability
Hardware changes/flexibility	Hardware changes/flexibility	Hardware changes/flexibility
Adoption rates/software proliferation	Adoption rates/software proliferation	Adoption rates/software proliferation
Ease of use	Ease of use	Ease of use

Mission/Project requirement changes	Mission/Project requirement changes	Mission/Project requirement changes
Information Assurance	Information Assurance	Information Assurance
Mission Assurance	Mission Assurance	Mission Assurance

2.4 – Does or has your organization:

	Yes	No
Employed open systems architecture in the past	○	○
Employed modular open network architecture in the past	○	○
Currently utilize open systems architecture	○	○
Currently utilize modular open network architecture	○	○
Plan on utilizing open system architecture in the future	○	○
Plan on utilizing modular open network architecture in the future	○	○

2.5 – Based on direct experience, or indirect perception, what percentage of projects/missions do organizations that develop space system utilize the following: (Please drag the bars to the appropriate percentage, the percentages do not have to total 100 as these provisions are not mutually exclusive)

________ Open system architecture
________ Modular open network architecture
________ Closed proprietary systems
________ Other

2.6 – If your organization is considering, or was to consider, adopting a open systems approach what factors might prohibit its adoption? (check all that apply)

- ☐ Management buy in
- ☐ Legacy software requirements
- ☐ Investment vs. return in time
- ☐ Investment vs. return in money
- ☐ Compatibility with current infrastructure
- ☐ Ownership of existing and future software
- ☐ Current proprietary systems
- ☐ Developer buy in
- ☐ Development cost
- ☐ Maintenance cost
- ☐ Development productivity
- ☐ Development efficiency
- ☐ Complexity
- ☐ Maintainability
- ☐ Bug detection
- ☐ Best practices
- ☐ Schedule
- ☐ Domain knowledge
- ☐ Security
- ☐ I/0 efficiency
- ☐ Fault tolerance
- ☐ Latency
- ☐ Determinism
- ☐ Interoperability
- ☐ Portability
- ☐ Testing
- ☐ Reusability
- ☐ Upgradability
- ☐ Flexibility
- ☐ Ease of use

- ☐ Information Assurance
- ☐ Other ____

2.7 – Consider the statements below with regard to software reuse from mission to mission. Please select the statement that is most indicative of your experience with or perception of software reuse.

- ○ Target some number of missions with a certain level of reuse in mind
- ○ Work mission to mission while reusing if possible
- ○ Work mission to mission with no or minimal reuse

End of Block: Open Systems Architecture and Modularity

Start of Block: Background

1.0 – Timing
First Click
Last Click
Page Submit
Click Count

1.1 – Please select your role(s) in space systems development. You must select at least one role.

- ☐ Software engineer
- ☐ Electrical engineer
- ☐ Systems engineer
- ☐ Mechanical engineer
- ☐ Aerospace engineer
- ☐ Program Manager
- ☐ Technical Lead
- ☐ Thermal engineer
- ☐ Principle Investigator
- ☐ Other (please specify) ____

1.2 – Please categorize the entity for which you currently work using the options below. You must select at least one category.

- ☐ Industry
- ☐ SETA/UARC/FFRDC
- ☐ Government
- ☐ Other (please specify) ____

1.3 – How many years have you spent working in the following areas with regard to space systems? Please drag the sliders to indicate the number of years of experience you have for each area. You must click or move each slider even if you want your responses marked as zero, the slider will turn purple/blue upon input.

	Years 0 5 10 15 20 25 30 35 40 45 50
Software Systems Development	
Management	
Hardware Systems Development	
Procurement	
Other (please specify)	

1.4 – Based on you experience, i.e. the projects on which you have worked, what are the typical durations for the phases listed below? Please drag the sliders to indicate the number of months you think are typically spent on each phase. You must click or move each slider even if you want your responses marked as zero, the slider will turn purple/blue upon input. If you do not have direct experience, then go off of what you think is typical.

	Months 0 10 20 30 40 50 60 70 80 90 100110120
Planning	
Development	
Testing	
Operations	
Other (please specify)	

1.5 – About how many missions/projects have you worked on over the course of your career in the following areas? Please drag the sliders to indicate the number missions/projects on which you have worked. You must click or move each slider even if you want your responses marked as zero, the slider will turn purple/blue upon input.

	0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30
Space Systems	
Other fields	

1.6 – About how many missions/projects do you typically work on at the same time? Please drag the sliders to indicate the number of projects that you typically work on at the same time. You must click or move each slider even if you want your response marked as zero, the slider will turn purple/blue upon input.

	0	1	2	3	4	5	6	7	8	9	10
Projects/Missions											

End of Block: Background

A.7 *Security* Survey

SISDPA : Security - Survey

Start of Block: Introduction

Intro – I'm Brandon Shirley, I am conducting these surveys for my PhD research at Utah State University. I really need your participation. There is a chance to win some gift cards.

You are being asked to take part in this set of surveys because of your affiliation with or involvement in the space industry. Your participation and your responses to the surveys are greatly needed and will make a meaningful contribution to this research.

This survey comprises the **Security** portion of a survey set that makes up a survey on Space Industry Software Development Practices and Attitudes (SISDPA). Each question will ask for your input and explain how you should answer the question.

Answer as many of the questions as you want or as much of a question as you want, partial surveys may still be very helpful. At the end of this survey, you will be redirected to a webpage that asks for an email address. You must enter a valid email address to be considered for survey drawings or the overall survey set drawing.

You have a chance at receiving a gift card for participating in this survey as well as a chance at receiving a gift card for your overall participation in the entire survey set. There will be 2 winners of $25 gift cards for each survey and 2 winners of $200 gift cards for the survey set.

At the end of this survey you will be redirected to another webpage where you may provide your email address if you want to enter the optional drawings. Your email address will not be connected to the survey responses. You must enter a valid email address to be considered for this survey's drawing or the overall drawing.

If you start this survey but do not finish, then your partial result may be included in the results.

Once the survey set is closed, we will analyze the data for inclusion in a conference or journal paper.

If you have other questions or research-related problems, you may reach Dr. Stephen Clyde at (435) 797- 2307 or Stephen.Clyde@usu.edu, or Brandon Shirley at (435) 994-9165 or b.l.s@aggiemail.usu.edu.

End of Block: Introduction

Start of Block: Security

2.0 – Timing
First Click
Last Click
Page Submit
Click Count

2.1 – Consider the items on the left in the context of **internal security** in space systems. This would be security mechanisms in addition to the traditional gateway COMSEC, that add additional security provisions, e.g. auditing, component authentication, and access control. Place an item under **Pros** if positively impacted; under **Neutral** if not impacted, not applicable, or if you are not familiar with the term; or under **Cons** if negatively impacted by using internal security provisions. Keep in mind that some of these items have overlap with other items. **All items should be placed for the answer to be considered "complete" by the system. In considering partial or incomplete answers, unplaced items will be considered neutral.**

Pros	Neutral	Cons
Regression reduction	Regression reduction	Regression reduction
Code design	Code design	Code design
Development cost	Development cost	Development cost
Maintenance cost	Maintenance cost	Maintenance cost
Development productivity	Development productivity	Development productivity
Development efficiency	Development efficiency	Development efficiency
Code Complexity	Code Complexity	Code Complexity
Maintainability	Maintainability	Maintainability
Integration	Integration	Integration
Adaptability	Adaptability	Adaptability
Documentation/Examples	Documentation/Examples	Documentation/Examples
Encapsulation	Encapsulation	Encapsulation
Bug detection	Bug detection	Bug detection
Code quality	Code quality	Code quality
Code robustness	Code robustness	Code robustness
Best practices	Best practices	Best practices

Schedule	Schedule	Schedule
Code or algorithm optimization/efficiency	Code or algorithm optimization/efficiency	Code or algorithm optimization/efficiency
Uniformity of coding style	Uniformity of coding style	Uniformity of coding style
Domain knowledge	Domain knowledge	Domain knowledge
Code readability	Code readability	Code readability
Security	Security	Security
I/0 efficiency	I/0 efficiency	I/0 efficiency
Radiation hardness	Radiation hardness	Radiation hardness
Fault tolerance	Fault tolerance	Fault tolerance
Hardware complexity	Hardware complexity	Hardware complexity
Latency	Latency	Latency
Determinism	Determinism	Determinism
Interoperability	Interoperability	Interoperability
Portability	Portability	Portability
Testing	Testing	Testing
Reusability	Reusability	Reusability
Software upgradability	Software upgradability	Software upgradability
Hardware changes/flexibility	Hardware changes/flexibility	Hardware changes/flexibility
Adoption rates/software proliferation	Adoption rates/software proliferation	Adoption rates/software proliferation
Ease of use	Ease of use	Ease of use
Mission/Project requirement changes	Mission/Project requirement changes	Mission/Project requirement changes
Information Assurance	Information Assurance	Information Assurance
Mission Assurance	Mission Assurance	Mission Assurance

2.2 – Do you have direct experience with security in the following areas:

	Yes	No
Space flight systems	○	○
Space ground systems	○	○
Space test systems	○	○
Penetration testing	○	○
Other (please specify)	○	○

2.3 – Based on direct experience, or indirect perception, what percentage of space flight systems do you think use the following security provisions for internal protection, i.e. within space vehicle systems? Please drag the bars to the appropriate percentage, the percentages do not have to total 100 as these provisions are not mutually exclusive.

_______ **Identity Management** Ability to establish identity of components
_______ **Mutual Authentication** Method for components to authenticate each others identities
_______ **Authorization (includes access control)** Method for determining permissions of each component
_______ **Auditing** Tracking pertinent system events
_______ **Encryption** Ability to protect data in transit and at rest
_______ **Network Segmentation** Separating networks physically or virtually
_______ **Recovery** Ability to recover from attacks
_______ **Mitigation** Ability to prevent attacks
_______ **None** No internal provisions
_______ Other (please specify)

2.4 – Based on direct experience, or indirect perception, what percentage of space ground systems do you think use the following security provisions for internal protection, i.e. within the ground system? Please drag the bars to the appropriate percentage, the percentages do not have to total 100 as these provisions are not mutually exclusive.
_______ **Identity Management** Ability to establish identity of components
_______ **Mutual Authentication** Method for components to authenticate each others identities
_______ **Authorization (includes access control)** Method for determining permissions of each component
_______ **Auditing** Tracking pertinent system events
_______ **Encryption** Ability to protect data in transit and at rest
_______ **Network Segmentation** Separating networks physically or virtually
_______ **Recovery** Ability to recover from attacks
_______ **Mitigation** Ability to prevent attacks
_______ **None** No internal provisions
_______ Other (please specify)

2.5 – Based on direct experience, or indirect perception, what percentage of space test systems do you think use the following security provisions for internal protection, i.e. within the test system? Please drag the bars to the appropriate percentage, the percentages do not have to total 100 as these provisions are not mutually exclusive.
_______ **Mutual Authentication** Method for components to authenticate each others identities
_______ **Identity Management** Ability to establish identity of components
_______ **Authorization (includes access control)** Method for determining permissions of each component
_______ **Auditing** Tracking pertinent system events
_______ **Encryption** Ability to protect data in transit and at rest
_______ **Network Segmentation** Separating networks physically or virtually
_______ **Recovery** Ability to recover from attacks
_______ **Mitigation** Ability to prevent attacks
_______ **None** No internal provisions
_______ Other (please specify)

2.6 – For each security feature listed below, rate its importance in each of the two domains: Open Networked Space Flight Software Systems and Traditional (Point-to-point) Flight Software Systems. 1 means "not important" and 5 means "very important." Use the "Other" items to represent provisions or features that you feel are missing from this list.

	Open Networked Space Flight Software Systems					Traditional Space Flight Software Systems				
	1	2	3	4	5	1	2	3	4	5

Identity Management - Ability to establish identity of components	○	○	○	○	○	○	○	○	○	○
Mutual Authentication - Method for components to authenticate each others identities	○	○	○	○	○	○	○	○	○	○
Authorization - Method for determining permissions of each component	○	○	○	○	○	○	○	○	○	○
Auditing - Tracking pertinent system events	○	○	○	○	○	○	○	○	○	○
Confidentiality - Ability to ensure data is private	○	○	○	○	○	○	○	○	○	○
Integrity - Ability to ensure data is has not been tampered with	○	○	○	○	○	○	○	○	○	○
Availability - Ensure that components are available when expected	○	○	○	○	○	○	○	○	○	○
Well-defined interfaces - Clearly defined hardware and software interfaces; Idea being to limit potential misuse	○	○	○	○	○	○	○	○	○	○
Abstraction layers - The separation of concerns to facilitate interoperability and platform independence	○	○	○	○	○	○	○	○	○	○

Access control - Ability to ensure permissions are enforced	○	○	○	○	○	○	○	○	○	○
Network Segmentation - Separating networks physically or virtually	○	○	○	○	○	○	○	○	○	○
Compliance - Ensure protocols are correctly implemented	○	○	○	○	○	○	○	○	○	○
Testing - Static and dynamic analysis from a security perspective	○	○	○	○	○	○	○	○	○	○
Recovery - Ability to recover from attacks	○	○	○	○	○	○	○	○	○	○
Mitigation - Ability to prevent attacks	○	○	○	○	○	○	○	○	○	○
Other (please specify)	○	○	○	○	○	○	○	○	○	○
Other (please specify)	○	○	○	○	○	○	○	○	○	○
Other (please specify)	○	○	○	○	○	○	○	○	○	○

2.7 – For each security feature listed below, rate the difficulty of providing for it in each of the two domains: Open Networked Space Flight Software Systems and Traditional (Point-to-point) Flight Software Systems. 1 means "not difficult" and 5 means "very difficult." Use the "Other" items to represent provisions or features that you feel are missing from this list.

	Open Networked Space Flight Software Systems					Traditional Space Flight Software Systems				
	1	2	3	4	5	1	2	3	4	5

Identity Management	○	○	○	○	○	○	○	○	○	○
Mutual Authentication	○	○	○	○	○	○	○	○	○	○
Authorization	○	○	○	○	○	○	○	○	○	○
Auditing	○	○	○	○	○	○	○	○	○	○
Confidentiality	○	○	○	○	○	○	○	○	○	○
Integrity	○	○	○	○	○	○	○	○	○	○
Availability	○	○	○	○	○	○	○	○	○	○
Well-defined interfaces	○	○	○	○	○	○	○	○	○	○
Abstraction layers	○	○	○	○	○	○	○	○	○	○
Access control	○	○	○	○	○	○	○	○	○	○
Network Segmentation	○	○	○	○	○	○	○	○	○	○
Compliance	○	○	○	○	○	○	○	○	○	○
Testing	○	○	○	○	○	○	○	○	○	○
Recovery	○	○	○	○	○	○	○	○	○	○
Mitigation	○	○	○	○	○	○	○	○	○	○
Other	○	○	○	○	○	○	○	○	○	○
Other	○	○	○	○	○	○	○	○	○	○
Other	○	○	○	○	○	○	○	○	○	○

2.8 – Please order the following software security provisions and features, in the context space flight software systems (C&DH), according to importance. Drag them so that the most important (1) comes first and the least import (17) comes last. You must move at least one characteristic for the question to marked as answered, you can move it back if you feel the initial ordering is correct. Use the "Other" items to represent provisions or features that you feel are missing from this list, if you do not feel anything is missing then simply leave the text entry boxes empty and put them last.

______ **Identity Management** - Ability to establish identity of components
______ **Mutual Authentication** - Method for components to authenticate each others identity
______ **Authorization** - Method for determining permissions of each component
______ **Auditing** - Tracking pertinent system events
______ **Confidentiality** - Ability to ensure data is private
______ **Integrity** - Ability to ensure data is has not been tampered with
______ **Availability** - Ensure that components are available when expected
______ **Well-defined interfaces** - Clearly defined hardware and software interfaces; idea being to limit potential misuse
______ **Abstraction layers** - The separation of concerns to facilitate interoperability and platform independence
______ **Access control** - Ability to ensure permissions are enforced
______ **Compliance** - Ensure protocols are correctly implemented
______ **Testing** - Static and dynamic analysis from a security perspective
______ **Recovery** - Ability to recover from attacks
______ **Mitigation** - Ability to prevent attacks
______ Other (please specify)
______ Other (please specify)
______ Other (please specify)

2.9 – Please designate your opinion of the effect of publicly releasing software of its security.

○ Negatively effects security

○ No effect on security

○ Positively effects security

2.10 – Please designate your opinion of the effect of open sourcing software on system security.

- ○ Negatively effects security
- ○ No effect on security
- ○ Positively effects security

End of Block: Security

Start of Block: Background

1.0 – Timing
First Click
Last Click
Page Submit
Click Count

1.1 – Please select your role(s) in space systems development. You must select at least one role.

- ▢ Software engineer
- ▢ Electrical engineer
- ▢ Systems engineer
- ▢ Mechanical engineer
- ▢ Aerospace engineer
- ▢ Program Manager
- ▢ Technical Lead
- ▢ Thermal engineer
- ▢ Principle Investigator
- ▢ Other (please specify) ____

1.2 – Please categorize the entity for which you currently work using the options below. You must select at least one category.

- ☐ Industry
- ☐ SETA/UARC/FFRDC
- ☐ Government
- ☐ Other (please specify) ______________________________

1.3 – How many years have you spent working in the following areas with regard to space systems? Please drag the sliders to indicate the number of years of experience you have for each area. You must click or move each slider even if you want your responses marked as zero, the slider will turn purple/blue upon input.

	Years
	0 5 10 15 20 25 30 35 40 45 50
Software Systems Development	
Management	
Hardware Systems Development	
Procurement	
Other (please specify)	

1.4 – Based on you experience, i.e. the projects on which you have worked, what are the typical durations for the phases listed below? Please drag the sliders to indicate the number of months you think are typically spent on each phase. You must click or move each slider even if you want your responses marked as zero, the slider will turn purple/blue upon input. If you do not have direct experience, then go off of what you think is typical.

Months

0 10 20 30 40 50 60 70 80 90 100110120

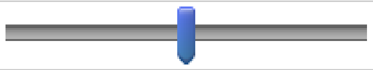
Planning	
Development	
Testing	
Operations	
Other (please specify)	

1.5 – About how many missions/projects have you worked on over the course of your career in the following areas? Please drag the sliders to indicate the number missions/projects on which you have worked. You must click or move each slider even if you want your responses marked as zero, the slider will turn purple/blue upon input.

0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30

Space Systems	
Other fields	

1.6 – About how many missions/projects do you typically work on at the same time? Please drag the sliders to indicate the number of projects that you typically work on at the same time. You must click or move each slider even if you want your response marked as zero, the slider will turn purple/blue upon input.

	0 1 2 3 4 5 6 7 8 9 10
Projects/Missions	

End of Block: Background

A.8 *RIPCC* Survey

SISDPA : Reuse, Interoperability, Portability, Code Complexity - Survey

Start of Block: Introduction

Intro – I'm Brandon Shirley, I am conducting these surveys for my PhD research at Utah State University. I really need your participation. There is a chance to win some gift cards.

You are being asked to take part in this set of surveys because of your affiliation with or involvement in the space industry. Your participation and your responses to the surveys are greatly needed and will make a meaningful contribution to this research.

This survey comprises the **Reuse, Interoperability, Portability, Code Complexity** portion of a survey set that makes up a survey on Space Industry Software Development Practices and Attitudes (SISDPA). Each question will ask for your input and explain how you should answer the question.

Answer as many of the questions as you want and as much of a question as you want, partial surveys may still be very helpful. At the end of this survey, you will be redirected to a webpage that asks for an email address. You must enter a valid email address to be considered for survey drawings or the overall survey set drawing.

You have a chance at receiving a gift card for participating in this survey as well as a chance at receiving a gift card for your overall participation in the entire survey set. There will be 2 winners of $25 gift cards for each survey and 2 winners of $200 gift cards for the survey set.

At the end of this survey you will be redirected to another webpage where you may provide your email address if you want to enter the optional drawings. Your email address will not be connected to the survey responses. You must enter a valid email address to be considered for this survey's drawing or the overall drawing.

If you start this survey but do not finish, then your partial result may be included in the results.

Once the survey set is closed, we will analyze the data for inclusion in a conference or journal paper. Please see the Letter of Intent for additional information about the surveys.

If you have other questions or research-related problems, you may reach Dr. Stephen Clyde at (435) 797- 2307 or Stephen.Clyde@usu.edu, or Brandon Shirley at (435) 994-9165 or b.l.s@aggiemail.usu.edu.

End of Block: Introduction

Start of Block: Reuse, Interoperability, Portability, Code Complexity

2.0 – Timing
First Click
Last Click
Page Submit
Click Count

2.1 – Please select what you think are the current, desirable, and obtainable percentages of reuse from mission to mission or project to project in relation to **space flight software**.

	0 10 20 30 40 50 60 70 80 90 100
Current	
Desirable	
Obtainable	

2.2 – Please select what you think are the current, desirable, and obtainable percentages of reuse from mission to mission or project to project in relation to **space ground software**.

	0 10 20 30 40 50 60 70 80 90 100
Current	
Desirable	
Obtainable	

2.3 – Please select what you think are the current, desirable, and obtainable percentages of reuse from mission to mission or project to project in relation to **space test software**.

	0 10 20 30 40 50 60 70 80 90 100
Current	
Desirable	
Obtainable	

2.4 – Please select what you think are the current, desirable, and obtainable percentages of reuse from mission to mission or project to project in relation to **other software fields**.

	0 10 20 30 40 50 60 70 80 90 100
Current	
Desirable	
Obtainable	

2.5 – Please select what you think are the current, desirable, and obtainable percentages of reuse from mission to mission or project to project in relation to **space flight hardware**.

	0 10 20 30 40 50 60 70 80 90 100
Current	
Desirable	
Obtainable	

2.6 – Please select what you think are the current, desirable, and obtainable percentages of reuse from mission to mission or project to project in relation to **space ground hardware**.

	0 10 20 30 40 50 60 70 80 90 100
Current	
Desirable	
Obtainable	

2.7 – Please select what you think are the current, desirable, and obtainable percentages of reuse from mission to mission or project to project in relation to **space test hardware**.

	0 10 20 30 40 50 60 70 80 90 100
Current	
Desirable	
Obtainable	

2.8 – Please select what you think are the current, desirable, and obtainable percentages of reuse from mission to mission or project to project in relation to **space test hardware**.

	0 10 20 30 40 50 60 70 80 90 100
Current	
Desirable	
Obtainable	

2.9 – Please select what you think are the current, desirable, and obtainable percentages of reuse from mission to mission or project to project in relation to **hardware in other fields**.

	0	10	20	30	40	50	60	70	80	90	100
Current						●					
Desirable						●					
Obtainable						●					

2.10 – Consider the items on the left in the context of **software reuse** across space systems, where software reuse means the ability to leverage the code or designs for any module from on system on another system. Place an item under **Pros** if positively affected, under **Neutral** if not affected or not applicable, or under **Cons** if negatively affected by attempting to maximize reuse. Think of this impact in terms of multi-project, multi-mission, or multi-platform use. Keep in mind that some of these items have overlap with other items. **All items should be placed for the answer to be considered complete by the system. In considering partial or incomplete answers, unplaced items will be considered neutral.** See Term definitions for specification of the items. If you are using a mouse then you can also hover over the items for a definition.

Pros	Neutral	Cons
Regression reduction	Regression reduction	Regression reduction
Code design	Code design	Code design
Development cost	Development cost	Development cost
Maintenance cost	Maintenance cost	Maintenance cost
Development productivity	Development productivity	Development productivity
Development efficiency	Development efficiency	Development efficiency
Code Complexity	Code Complexity	Code Complexity
Maintainability	Maintainability	Maintainability
Integration	Integration	Integration
Adaptability	Adaptability	Adaptability
Documentation/Examples	Documentation/Examples	Documentation/Examples
Encapsulation	Encapsulation	Encapsulation
Bug detection	Bug detection	Bug detection

Code quality	Code quality	Code quality
Code robustness	Code robustness	Code robustness
Best practices	Best practices	Best practices
Schedule	Schedule	Schedule
Code or algorithm optimization/efficiency	Code or algorithm optimization/efficiency	Code or algorithm optimization/efficiency
Uniformity of coding style	Uniformity of coding style	Uniformity of coding style
Domain knowledge	Domain knowledge	Domain knowledge
Code readability	Code readability	Code readability
Security	Security	Security
I/0 efficiency	I/0 efficiency	I/0 efficiency
Radiation hardness	Radiation hardness	Radiation hardness
Fault tolerance	Fault tolerance	Fault tolerance
Hardware complexity	Hardware complexity	Hardware complexity
Latency	Latency	Latency
Determinism	Determinism	Determinism
Interoperability	Interoperability	Interoperability
Portability	Portability	Portability
Testing	Testing	Testing
Reusability	Reusability	Reusability
Software upgradability	Software upgradability	Software upgradability
Hardware changes/flexibility	Hardware changes/flexibility	Hardware changes/flexibility
Adoption rates/software proliferation	Adoption rates/software proliferation	Adoption rates/software proliferation
Ease of use	Ease of use	Ease of use
Mission/Project requirement changes	Mission/Project requirement changes	Mission/Project requirement changes
Information Assurance	Information Assurance	Information Assurance
Mission Assurance	Mission Assurance	Mission Assurance

2.11 – Consider the items on the left in the context of **interoperability** in space systems, specifically semantic interoperability, i.e. having a common information exchange description that allows components to communicate in a meaningful way. Place an item under **Pros** if positively impacted, under **Neutral** if not impacted or not applicable, or under **Cons** if negatively impacted by attempting to maximize interoperability. Think of this impact in terms of multi-project, multi-mission, or multi-platform use. Keep in mind that some of these items have overlap with other items. **All items should be placed for the answer to be considered complete by the system. In considering partial or incomplete answers, unplaced items will be considered neutral.**

See Term definitions for specification of the items. If you are using a mouse then you can also hover over the items for a definition.

Pros	Neutral	Cons
Regression reduction	Regression reduction	Regression reduction
Code design	Code design	Code design
Development cost	Development cost	Development cost
Maintenance cost	Maintenance cost	Maintenance cost
Development productivity	Development productivity	Development productivity
Development efficiency	Development efficiency	Development efficiency
Code Complexity	Code Complexity	Code Complexity
Maintainability	Maintainability	Maintainability
Integration	Integration	Integration
Adaptability	Adaptability	Adaptability
Documentation/Examples	Documentation/Examples	Documentation/Examples
Encapsulation	Encapsulation	Encapsulation
Bug detection	Bug detection	Bug detection
Code quality	Code quality	Code quality
Code robustness	Code robustness	Code robustness
Best practices	Best practices	Best practices
Schedule	Schedule	Schedule
Code or algorithm optimization/efficiency	Code or algorithm optimization/efficiency	Code or algorithm optimization/efficiency
Uniformity of coding style	Uniformity of coding style	Uniformity of coding style
Domain knowledge	Domain knowledge	Domain knowledge
Code readability	Code readability	Code readability
Security	Security	Security

I/0 efficiency	I/0 efficiency	I/0 efficiency
Radiation hardness	Radiation hardness	Radiation hardness
Fault tolerance	Fault tolerance	Fault tolerance
Hardware complexity	Hardware complexity	Hardware complexity
Latency	Latency	Latency
Determinism	Determinism	Determinism
Interoperability	Interoperability	Interoperability
Portability	Portability	Portability
Testing	Testing	Testing
Reusability	Reusability	Reusability
Software upgradability	Software upgradability	Software upgradability
Hardware changes/flexibility	Hardware changes/flexibility	Hardware changes/flexibility
Adoption rates/software proliferation	Adoption rates/software proliferation	Adoption rates/software proliferation
Ease of use	Ease of use	Ease of use
Mission/Project requirement changes	Mission/Project requirement changes	Mission/Project requirement changes
Information Assurance	Information Assurance	Information Assurance
Mission Assurance	Mission Assurance	Mission Assurance

2.12 – Consider the items on the left in the context of software **portability** in space systems, specifically an increase in ability to move software from platform to platform with minimal or no changes. Place an item under **Pros** if positively impacted, under **Neutral** if not impacted or not applicable, or under **Cons** if negatively impacted. Think of this impact in terms of multi-project, multi-mission, or multi-platform use. Keep in mind that some of these items have overlap with other items. **All items should be placed for the answer to be considered complete by the system. In considering partial or incomplete answers, unplaced items will be considered neutral.**

See Term definitions for specification of the items. If you are using a mouse then you can also hover over the items for a definition.

Pros	Neutral	Cons
Regression reduction	Regression reduction	Regression reduction
Code design	Code design	Code design
Development cost	Development cost	Development cost
Maintenance cost	Maintenance cost	Maintenance cost
Development productivity	Development productivity	Development productivity
Development efficiency	Development efficiency	Development efficiency
Code Complexity	Code Complexity	Code Complexity
Maintainability	Maintainability	Maintainability
Integration	Integration	Integration
Adaptability	Adaptability	Adaptability
Documentation/Examples	Documentation/Examples	Documentation/Examples
Encapsulation	Encapsulation	Encapsulation
Bug detection	Bug detection	Bug detection
Code quality	Code quality	Code quality
Code robustness	Code robustness	Code robustness
Best practices	Best practices	Best practices
Schedule	Schedule	Schedule
Code or algorithm optimization/efficiency	Code or algorithm optimization/efficiency	Code or algorithm optimization/efficiency
Uniformity of coding style	Uniformity of coding style	Uniformity of coding style
Domain knowledge	Domain knowledge	Domain knowledge
Code readability	Code readability	Code readability
Security	Security	Security

I/0 efficiency	I/0 efficiency	I/0 efficiency
Radiation hardness	Radiation hardness	Radiation hardness
Fault tolerance	Fault tolerance	Fault tolerance
Hardware complexity	Hardware complexity	Hardware complexity
Latency	Latency	Latency
Determinism	Determinism	Determinism
Interoperability	Interoperability	Interoperability
Portability	Portability	Portability
Testing	Testing	Testing
Reusability	Reusability	Reusability
Software upgradability	Software upgradability	Software upgradability
Hardware changes/flexibility	Hardware changes/flexibility	Hardware changes/flexibility
Adoption rates/software proliferation	Adoption rates/software proliferation	Adoption rates/software proliferation
Ease of use	Ease of use	Ease of use
Mission/Project requirement changes	Mission/Project requirement changes	Mission/Project requirement changes
Information Assurance	Information Assurance	Information Assurance
Mission Assurance	Mission Assurance	Mission Assurance

2.13 – Consider the items on the left in the context of reduced or minimal software **code complexity** in space systems. Place an item under **Pros** if positively impacted, under **Neutral** if not impacted or not applicable, or under **Cons** if negatively impacted by attempting to minimize code complexity. Keep in mind that some of these items have overlap with other items. **All items should be placed for the answer to be considered complete by the system. In considering partial or incomplete answers, unplaced items will be considered neutral.** See Term definitions for specification of the items. If you are using a mouse then you can also hover over the items for a definition.

Pros	Neutral	Cons
Regression reduction	Regression reduction	Regression reduction
Code design	Code design	Code design
Development cost	Development cost	Development cost
Maintenance cost	Maintenance cost	Maintenance cost
Development productivity	Development productivity	Development productivity
Development efficiency	Development efficiency	Development efficiency
Code Complexity	Code Complexity	Code Complexity
Maintainability	Maintainability	Maintainability
Integration	Integration	Integration
Adaptability	Adaptability	Adaptability
Documentation/Examples	Documentation/Examples	Documentation/Examples
Encapsulation	Encapsulation	Encapsulation
Bug detection	Bug detection	Bug detection
Code quality	Code quality	Code quality
Code robustness	Code robustness	Code robustness
Best practices	Best practices	Best practices
Schedule	Schedule	Schedule
Code or algorithm optimization/efficiency	Code or algorithm optimization/efficiency	Code or algorithm optimization/efficiency
Uniformity of coding style	Uniformity of coding style	Uniformity of coding style
Domain knowledge	Domain knowledge	Domain knowledge
Code readability	Code readability	Code readability
Security	Security	Security
I/0 efficiency	I/0 efficiency	I/0 efficiency
Radiation hardness	Radiation hardness	Radiation hardness

Fault tolerance	Fault tolerance	Fault tolerance
Hardware complexity	Hardware complexity	Hardware complexity
Latency	Latency	Latency
Determinism	Determinism	Determinism
Interoperability	Interoperability	Interoperability
Portability	Portability	Portability
Testing	Testing	Testing
Reusability	Reusability	Reusability
Software upgradability	Software upgradability	Software upgradability
Hardware changes/flexibility	Hardware changes/flexibility	Hardware changes/flexibility
Adoption rates/software proliferation	Adoption rates/software proliferation	Adoption rates/software proliferation
Ease of use	Ease of use	Ease of use
Mission/Project requirement changes	Mission/Project requirement changes	Mission/Project requirement changes
Information Assurance	Information Assurance	Information Assurance
Mission Assurance	Mission Assurance	Mission Assurance

2.14 – Consider the items on the left in the context of plug-and-play components and associated software in space systems. A terrestrial example of plug-and-play is the Human Interface Device (HID) protocol and a celestial example is the Space Plug-and-Play Avionics (SPA) Standard. Place an item under **Pros** if positively impacted, under **Neutral** if not impacted or not applicable, or under **Cons** if negatively impacted by attempting to minimize code complexity. Keep in mind that some of these items have overlap with other items. **All items should be placed for the answer to be considered complete by the system. In considering partial or incomplete answers, unplaced items will be considered neutral.** See Term definitions for specification of the items. If you are using a mouse then you can also hover over the items for a definition.

Pros	Neutral	Cons
Regression reduction	Regression reduction	Regression reduction
Code design	Code design	Code design
Development cost	Development cost	Development cost
Maintenance cost	Maintenance cost	Maintenance cost
Development productivity	Development productivity	Development productivity
Development efficiency	Development efficiency	Development efficiency
Code Complexity	Code Complexity	Code Complexity
Maintainability	Maintainability	Maintainability
Integration	Integration	Integration
Adaptability	Adaptability	Adaptability
Documentation/Examples	Documentation/Examples	Documentation/Examples
Encapsulation	Encapsulation	Encapsulation
Bug detection	Bug detection	Bug detection
Code quality	Code quality	Code quality
Code robustness	Code robustness	Code robustness
Best practices	Best practices	Best practices
Schedule	Schedule	Schedule
Code or algorithm optimization/efficiency	Code or algorithm optimization/efficiency	Code or algorithm optimization/efficiency
Uniformity of coding style	Uniformity of coding style	Uniformity of coding style
Domain knowledge	Domain knowledge	Domain knowledge
Code readability	Code readability	Code readability
Security	Security	Security

I/0 efficiency	I/0 efficiency	I/0 efficiency
Radiation hardness	Radiation hardness	Radiation hardness
Fault tolerance	Fault tolerance	Fault tolerance
Hardware complexity	Hardware complexity	Hardware complexity
Latency	Latency	Latency
Determinism	Determinism	Determinism
Interoperability	Interoperability	Interoperability
Portability	Portability	Portability
Testing	Testing	Testing
Reusability	Reusability	Reusability
Software upgradability	Software upgradability	Software upgradability
Hardware changes/flexibility	Hardware changes/flexibility	Hardware changes/flexibility
Adoption rates/software proliferation	Adoption rates/software proliferation	Adoption rates/software proliferation
Ease of use	Ease of use	Ease of use
Mission/Project requirement changes	Mission/Project requirement changes	Mission/Project requirement changes
Information Assurance	Information Assurance	Information Assurance
Mission Assurance	Mission Assurance	Mission Assurance

2.15 – For each metric listed in a row below, rate its importance with regard to measuring code complexity.1 means "not important" and 5 means "very important." Note that some of these only related to object-oriented programming.

	1	2	3	4	5
Cyclomatic complexity (McCabe Metric)	○	○	○	○	○
Depth of inheritance	○	○	○	○	○
Class coupling	○	○	○	○	○
Methods per class	○	○	○	○	○
Lack of cohesion	○	○	○	○	○
Data complexity (Chapin Metric)	○	○	○	○	○
Data flow complexity (Elshof Metric)	○	○	○	○	○
Decisional complexity (Mcclure Metric)	○	○	○	○	○
Language complexity (Haltsead Metric)	○	○	○	○	○
Interface Complexity (Henry fan-in/fan-out Metric)	○	○	○	○	○
Lines of code	○	○	○	○	○
Other (please specify)	○	○	○	○	○

End of Block: Reuse, Interoperability, Portability, Code Complexity

Start of Block: Background

1.0 – Timing
First Click
Last Click
Page Submit
Click Count

1.1 – Please select your role(s) in space systems development. You must select at least one role.

- ☐ Software engineer
- ☐ Electrical engineer
- ☐ Systems engineer
- ☐ Mechanical engineer
- ☐ Aerospace engineer
- ☐ Program Manager
- ☐ Technical Lead
- ☐ Thermal engineer
- ☐ Principle Investigator
- ☐ Other (please specify) ____

1.2 – Please categorize the entity for which you currently work using the options below. You must select at least one category.

- ☐ Industry
- ☐ SETA/UARC/FFRDC
- ☐ Government
- ☐ Other (please specify) ____

1.3 How many years have you spent working in the following areas with regard to space systems? Please drag the sliders to indicate the number of years of experience you have for each area. You must click or move each slider even if you want your responses marked as zero, the slider will turn purple/blue upon input.

	Years 0 5 10 15 20 25 30 35 40 45 50
Software Systems Development	
Management	
Hardware Systems Development	
Procurement	
Other (please specify)	

1.4 – Based on you experience, i.e. the projects on which you have worked, what are the typical durations for the phases listed below? Please drag the sliders to indicate the number of months you think are typically spent on each phase. You must click or move each slider even if you want your responses marked as zero, the slider will turn purple/blue upon input. If you do not have direct experience, then go off of what you think is typical.

	Months 0 10 20 30 40 50 60 70 80 90 100110120
Planning	
Development	
Testing	
Operations	
Other (please specify)	

1.5 – About how many missions/projects have you worked on over the course of your career in the following areas? Please drag the sliders to indicate the number missions/projects on which you have worked. You must click or move each slider even if you want your responses marked as zero, the slider will turn purple/blue upon input.

	0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30
Space Systems	
Other fields	

1.6 – About how many missions/projects do you typically work on at the same time? Please drag the sliders to indicate the number of projects that you typically work on at the same time. You must click or move each slider even if you want your response marked as zero, the slider will turn purple/blue upon input.

	0 1 2 3 4 5 6 7 8 9 10
Projects/Missions	

End of Block: Background

A.9 *Network* Survey

SISDPA : Network - Survey

Start of Block: Introduction

Intro – I'm Brandon Shirley, I am conducting these surveys for my PhD research at Utah State University. I really need your participation. There is a chance to win some gift cards.

You are being asked to take part in this set of surveys because of your affiliation with or involvement in the space industry. Your participation and your responses to the surveys are greatly needed and will make a meaningful contribution to this research.

This survey comprises the **Network** portion of a survey set that makes up a survey on Space Industry Software Development Practices and Attitudes (SISDPA). Each question will ask for your input and explain how you should answer the question.

Answer as many of the questions as you want and as much of a question as you want, partial surveys may still be very helpful. At the end of this survey, you will be redirected to a webpage that asks for an email address. You must enter a valid email address to be considered for survey drawings or the overall survey set drawing.

You have a chance at receiving a gift card for participating in this survey as well as a chance at receiving a gift card for your overall participation in the entire survey set. There will be 2 winners of $25 gift cards for each survey and 2 winners of $200 gift cards for the survey set.

At the end of this survey you will be redirected to another webpage where you may provide your email address if you want to enter the optional drawings. Your email address will not be connected to the survey responses. You must enter a valid email address to be considered for this survey's drawing or the overall drawing.

If you start this survey but do not finish, then your partial result may be included in the results.

Once the survey set is closed, we will analyze the data for inclusion in a conference or journal paper. Please see the Letter of Intent for additional information about the surveys.

If you have other questions or research-related problems, you may reach Dr. Stephen Clyde at (435) 797- 2307 or Stephen.Clyde@usu.edu, or Brandon Shirley at (435) 994-9165 or b.l.s@aggiemail.usu.edu.

End of Block: Introduction

Start of Block: Network

2.0 – Timing
First Click
Last Click
Page Submit
Click Count

2.1 – Do you have direct network experience in the following areas? You must select "Yes" or "No" for each area for the answer to be complete.

	Yes	No
Purely Point-to-Point (serial, etc.)	○	○
Master-slave (includes networks like I2C where only one component can communicate at a time)	○	○
Purely open (Spacewire, Ethernet, etc.)	○	○
Hybrid Networks (combination of point-to-point and open network solutions)	○	○
Other	○	○

2.2 – Based on direct experience, or indirect perception, what percentage of space systems do you think use the following network types? Please drag the bars so that the total adds up to 100.
_______ **Purely Point-to-Point** (serial, etc.)
_______ **Master-slave** (includes networks like I2C where only one component can communicate at a time)
_______ **Purely Open** (SpaceWire, Ethernet, etc.)
_______ **Hybrid Networks** (combination of point-to-point and open network solutions)
_______ **Other**

2.3 – Consider the items on the left in the context of **networked** space systems, meaning a collection of three or more systems connected on an internal network instead of point-to-point links. Place an item under **Pros** if positively impacted, under **Neutral** if not impacted or not applicable, and under **Cons** if negatively impacted by using networked space systems. Think of this impact in terms of multi-project, multi-mission, or multi-platform use. **All items should be placed for the answer to be considered complete by the system. In considering partial or incomplete answers, unplaced items will be considered neutral.**
See Term definitions for specification of the items. If you are using a mouse then you can also hover over the items for a definition.

Pros	Neutral	Cons
Regression reduction	Regression reduction	Regression reduction
Code design	Code design	Code design
Development cost	Development cost	Development cost
Maintenance cost	Maintenance cost	Maintenance cost
Development productivity	Development productivity	Development productivity
Development efficiency	Development efficiency	Development efficiency
Code Complexity	Code Complexity	Code Complexity
Maintainability	Maintainability	Maintainability
Integration	Integration	Integration
Adaptability	Adaptability	Adaptability
Documentation/Examples	Documentation/Examples	Documentation/Examples
Encapsulation	Encapsulation	Encapsulation
Bug detection	Bug detection	Bug detection
Code quality	Code quality	Code quality
Code robustness	Code robustness	Code robustness
Best practices	Best practices	Best practices
Schedule	Schedule	Schedule
Code or algorithm optimization/efficiency	Code or algorithm optimization/efficiency	Code or algorithm optimization/efficiency
Uniformity of coding style	Uniformity of coding style	Uniformity of coding style
Domain knowledge	Domain knowledge	Domain knowledge
Code readability	Code readability	Code readability
Security	Security	Security
I/0 efficiency	I/0 efficiency	I/0 efficiency

Radiation hardness	Radiation hardness	Radiation hardness
Fault tolerance	Fault tolerance	Fault tolerance
Hardware complexity	Hardware complexity	Hardware complexity
Latency	Latency	Latency
Determinism	Determinism	Determinism
Interoperability	Interoperability	Interoperability
Portability	Portability	Portability
Testing	Testing	Testing
Reusability	Reusability	Reusability
Software upgradability	Software upgradability	Software upgradability
Hardware changes/flexibility	Hardware changes/flexibility	Hardware changes/flexibility
Adoption rates/software proliferation	Adoption rates/software proliferation	Adoption rates/software proliferation
Ease of use	Ease of use	Ease of use
Mission/Project requirement changes	Mission/Project requirement changes	Mission/Project requirement changes
Information Assurance	Information Assurance	Information Assurance
Mission Assurance	Mission Assurance	Mission Assurance

End of Block: Network

Start of Block: Background

1.0 – Timing
First Click
Last Click
Page Submit
Click Count

1.1 – Please select your role(s) in space systems development. You must select at least one role.

- ☐ Software engineer
- ☐ Electrical engineer
- ☐ Systems engineer
- ☐ Mechanical engineer
- ☐ Aerospace engineer
- ☐ Program Manager
- ☐ Technical Lead
- ☐ Thermal engineer
- ☐ Principle Investigator
- ☐ Other (please specify) ____

1.2 – Please categorize the entity for which you currently work using the options below. You must select at least one category.

- ☐ Industry
- ☐ SETA/UARC/FFRDC
- ☐ Government
- ☐ Other (please specify) ____

1.3 – How many years have you spent working in the following areas with regard to space systems? Please drag the sliders to indicate the number of years of experience you have for each area. You must click or move each slider even if you want your responses marked as zero, the slider will turn purple/blue upon input.

	Years 0 5 10 15 20 25 30 35 40 45 50
Software Systems Development	
Management	
Hardware Systems Development	
Procurement	
Other (please specify)	

1.4 – Based on you experience, i.e. the projects on which you have worked, what are the typical durations for the phases listed below? Please drag the sliders to indicate the number of months you think are typically spent on each phase. You must click or move each slider even if you want your responses marked as zero, the slider will turn purple/blue upon input. If you do not have direct experience, then go off of what you think is typical.

	Months 0 10 20 30 40 50 60 70 80 90 100110120
Planning	
Development	
Testing	
Operations	
Other (please specify)	

1.5 – About how many missions/projects have you worked on over the course of your career in the following areas? Please drag the sliders to indicate the number missions/projects on which you have worked. You must click or move each slider even if you want your responses marked as zero, the slider will turn purple/blue upon input.

	0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30
Space Systems	
Other fields	

1.6 – About how many missions/projects do you typically work on at the same time? Please drag the sliders to indicate the number of projects that you typically work on at the same time. You must click or move each slider even if you want your response marked as zero, the slider will turn purple/blue upon input.

	0 1 2 3 4 5 6 7 8 9 10
Projects/Missions	

End of Block: Background

A.10 Survey Pro-neutral-con Term Definitions

Term Definitions

A

- **Adaptability** - Ability of a system to adapt itself efficiently and quickly to changes in circumstance
- **Adoption rates/software proliferation** - Likelihood that software will be used and developed by other parties

B

- **Best practices** - Adherence to industry best practices standards
- **Bug detection** - Ease with which bugs and be detected and corrected

C

- **Code complexity** - Cyclomatic Complexity (structural complexity), Depth of Inheritance, Class Coupling, Lines of Code (LOC)
- **Code design** - The difficulty to architect software, e.g. amount of constraints and requirements placed on a software design
- **Code or algorithm optimization/efficiency** - Overhead incurred while completing operations
- **Code quality** - How well software complies with its design based on functional requirements
- **Code readability** - Ease with which a programmer can understand written code
- **Code robustness** - Codes ability to handle failure scenarios or misuse

D

- **Development cost** - Cost in terms of schedule and money to develop a system
- **Development efficiency** - Overhead incurred in hardware and software development, e.g. management time, design time
- **Determinism** - Ability of a system to always produce the same output given the same input
- **Development productivity** - Ability to produce software and hardware
- **Documentation/Examples** - Documentation of how a system works or how to use it
- **Domain knowledge** - Amount of knowledge or background need to use software or hardware

E

- **Ease of use** - Ease with which a non-developer can use the system, i.e. someone who was not involved in the implementation of the system
- **Encapsulation** - Bundling data with functionality (if desired)

F

- **Fault tolerance** - Ability of a system to remain operational in the event of failures

H

- **Hardware changes/flexibility** - Ease with which a system, including software, can handle changes to hardware components
- **Hardware complexity** - Metrics for hardware complexity are not well-defined, base this more on your perception

I

- **Information Assurance** - Process of getting the right information to the right place at the right time
- **Integration** - Ease with which system components can be integrated
- **Interoperability** - Specifically semantic interoperability (this generally encompasses Syntactic interoperability)
- **I/0 efficiency** - Overhead incurred during input and output operations, e.g. network transfer or write to nonvolatile storage

L

- **Latency** - Time delay experienced by a system

M

- **Maintainability** - Tipping point at which it becomes cheaper or less risky to rewrite code than to change it
- **Maintenance cost** - Cost in terms of schedule and money to update code with bug fixes and new features
- **Mission Assurance** - Ability to achieve success of design, development, testing, deployment, and operations
- **Mission/Project requirement changes** - Ease with which new requirements can be addressed

P

- **Portability** - The ability of software to be used on multiple platforms

R

- **Radiation hardness** - Resistance to damage or malfunctions caused by ionizing radiation and high-energy electromagnetic radiation
- **Regression reduction** - Introduction of new bugs or loss of features as new features are added or maintenance is performed
- **Reusability** - Ability to reuse existing assets (hardware or software) in new missions or projects

S

- **Schedule** - Ability to adhere to a given timeline
- **Security** - The basics: confidentiality, integrity, and availability
- **Software upgradability** - The ease with which new versions of software can be deployed to a system

T

- **Testing** - How easily the system can be tested

U

- **Uniformity of coding style** - Adherence to code style guidelines (if present)

APPENDIX B

SISDPA Survey Participant Backgrounds

B.1 Introduction

This appendix reviews the development roles with which participants identified as well as the years of experience that they have in different areas of space systems development, e.g. hardware systems or software systems development. It is important to understand who took the surveys in order to give context to the consensuses and perceptions that are reached on the relevant questions in each of the different surveys. These roles are not analyzed to a level where statistical significance is considered, rather this section presents more of a snapshot of who took each survey and the experience level mixture for each survey.

This analysis looks at two questions from the background section of each survey. These questions are the same for all the surveys. The first question is Question 1.1, this question asks participants to select their development roles. Unfortunately, this question is not specific enough in terms of current versus past, or duration of time in a role for it to be counted. Noting this shortcoming, this question still gives a window into the roles with which the participants identify and participate in at some level.

SE had the best representation in terms specialized and combination roles for all of the surveys; this may not be the best representation for the entire community on general issues, but it is a good balance for a set of surveys focused primarily on the software development aspects of space systems. systems engineer (SysE)s, technical lead (TL)s, program manager (PM)s, and principal investigator (PI)s all had good representation across all the surveys when considered as inclusive roles. Each of the sections that follow give an overview of the role breakdown.

The second question is Question 1.3, this question asks each participant to specify the years of experience they have in software systems development, management, hardware systems development, procurement, and other.

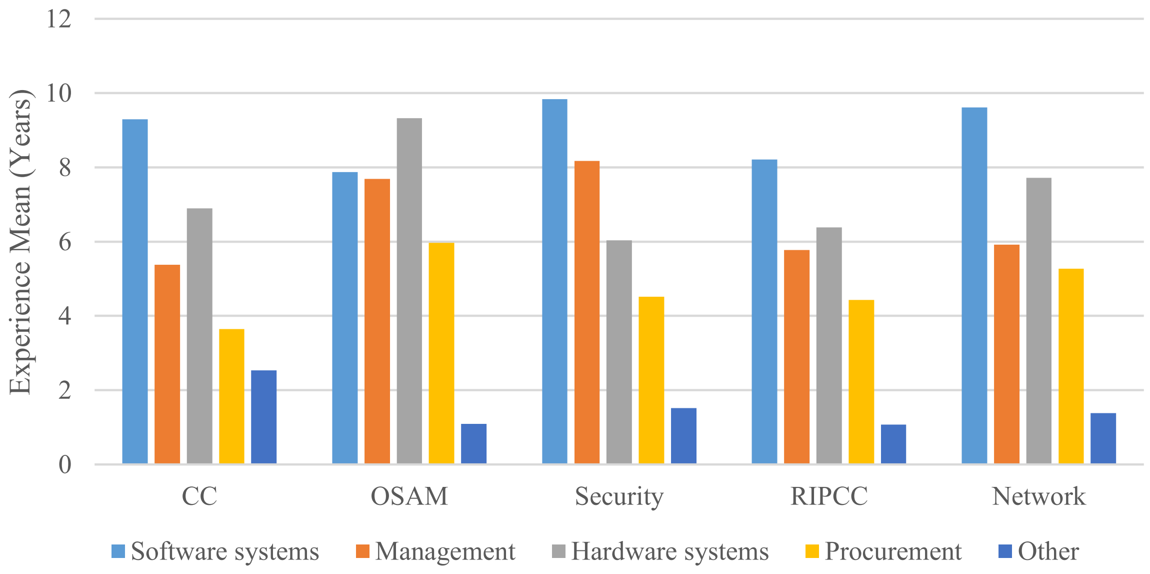


Fig. B.1. Survey Experience Means. The average years for experience for software systems development, management, hardware systems development, procurement, and other for each of the five surveys.

Figure B.1 shows the area experience averages across all the surveys. Figure B.1 shows that *software systems* development experience mean is fairly consistent across all of the surveys at between 8 and 10 years. *Management* shows greater variance across the surveys, but still averaged over 5 years for all the surveys. The *hardware systems* development experience average ranged between 6 and 10 years. The *procurement* average was a bit weaker dipping under 4 for the *CC* Survey, but generally holding in the 4 to 8 year range. Finally, *other* varied the most going as low as a year and as high as almost 8 years. This shows close to a 5 year plus experience average in all the surveys across all the development areas excluding *other*. This suggests that the participants have a good body of knowledge to draw upon when answering the questions for all of the surveys.

Sections B.2 through B.6 cover the role and experience breakdowns for each of the surveys. These roles and experience breakdowns show strong SE representation and good development experience mixtures.

B.2 *CC* Survey — Roles and Development Experience

Table B.1 shows that SEs had the strongest representation with 16 participants that identified strictly as an SE, or about 17%. These are participants that identified exclusively as SEs, this is the largest exclusive representation by far. Those that had an aggregate inclusive of SEs accounted for 47 of the participants or about 48%. This bodes well for a survey focused on software. It also shows a good representation of electrical engineer (EE)s and aerospace engineer (AE)s at close to 15% each, when considered as aggregate roles, showing that technical areas that might deal more with hardware are also represented.

SysEs were the next biggest exclusive group with 7 or about 7%. In considering roles that were inclusive of SysEs then the total is 40, or about 41%. The next highest aggregate inclusive groups was made up of TLs with 29 participants, or about 30%, even though TL exclusive participants only account for 3 of the total. Low exclusive counts for TLs and SysEs suggest that either these types of roles tend to engage in multiple roles at once or that they have a lot of experience in different roles before becoming TLs or SysEs, unfortunately the survey question did not adequately specify how this question should be answered. It would be ideal or at least better to break this out into at least two questions: one focusing on concurrent currently active roles and one focusing on cumulative or past roles in order to better understand this experience versus active roles relationship.

Table B.1. *CC* Survey, Question 1.1 — Role Breakdown

Role	Exclusive Aggregate Count	Exclusive Aggregate Share	Inclusive Count	Inclusive Share
SE	16	16.49%	47	48.45%
EE	1	1.03%	12	12.37%
SysE	7	7.22%	40	41.24%
ME	2	2.06%	8	8.25%
AE	6	6.19%	14	14.43%
PM	5	5.15%	16	16.49%
TL	2	2.06%	29	29.90%
TE	0	0.00%	4	4.12%

(continued on next page)

Table B.1. *CC* Survey, Question 1.1 — Role Breakdown

Role	Exclusive Aggregate Count	Exclusive Aggregate Share	Inclusive Count	Inclusive Share
PI	3	3.09%	8	8.25%
Other	6	6.19%	9	9.28%

Roles like TLs and SysEs tend to be thought of big pictures roles, meaning that they encompass larger portions of a program or mission than a more focused role like a TE. This is evidenced in Table B.1 where both the TL and SysE roles have relatively normal exclusive aggregate shares of 2.1% and 7.2%, but jump to 29.9% and 41.2% respectively when their inclusive share is considered. This result shows that participants that identified with these roles also tended to be involved with multiple other roles, as mentioned before it is hard to say if this is due to experience or concurrent roles. It is notable that the SEs had actually had the largest inclusive share, at 48.5%. This is likely driven by at least two factors. First, SEs where the largest exclusive aggregate by a factor of 2. Second, software tends to have to touch every segment of a mission in one way or another.

Figure B.2 gives the overall role affiliations of each of the respondents that took the survey. Figure B.3 shows the experience breakdown for each role for software systems development, management, hardware systems development, procurement, and other areas. Note these roles are not exclusive so there is overlap. Some of these groups are not really large enough to give a good average, but remember this is really just for illustrative purposes and some understanding of the group that took the survey. The PM and PI showed strong overall years of experience in all the roles, but also showed 10+ years of experience in management.

SEs showed the strongest skew towards their "discipline", but it is not clear if this relates to the higher number of SEs in general or a real difference in how focused SEs are with respect to other roles. The higher number of SEs might allow for a more representative experience ratio.

Once again these sections are more to illustrate the demographic, a lot of effort could have spent analyzing this area, but that was not the focus of this research and future work

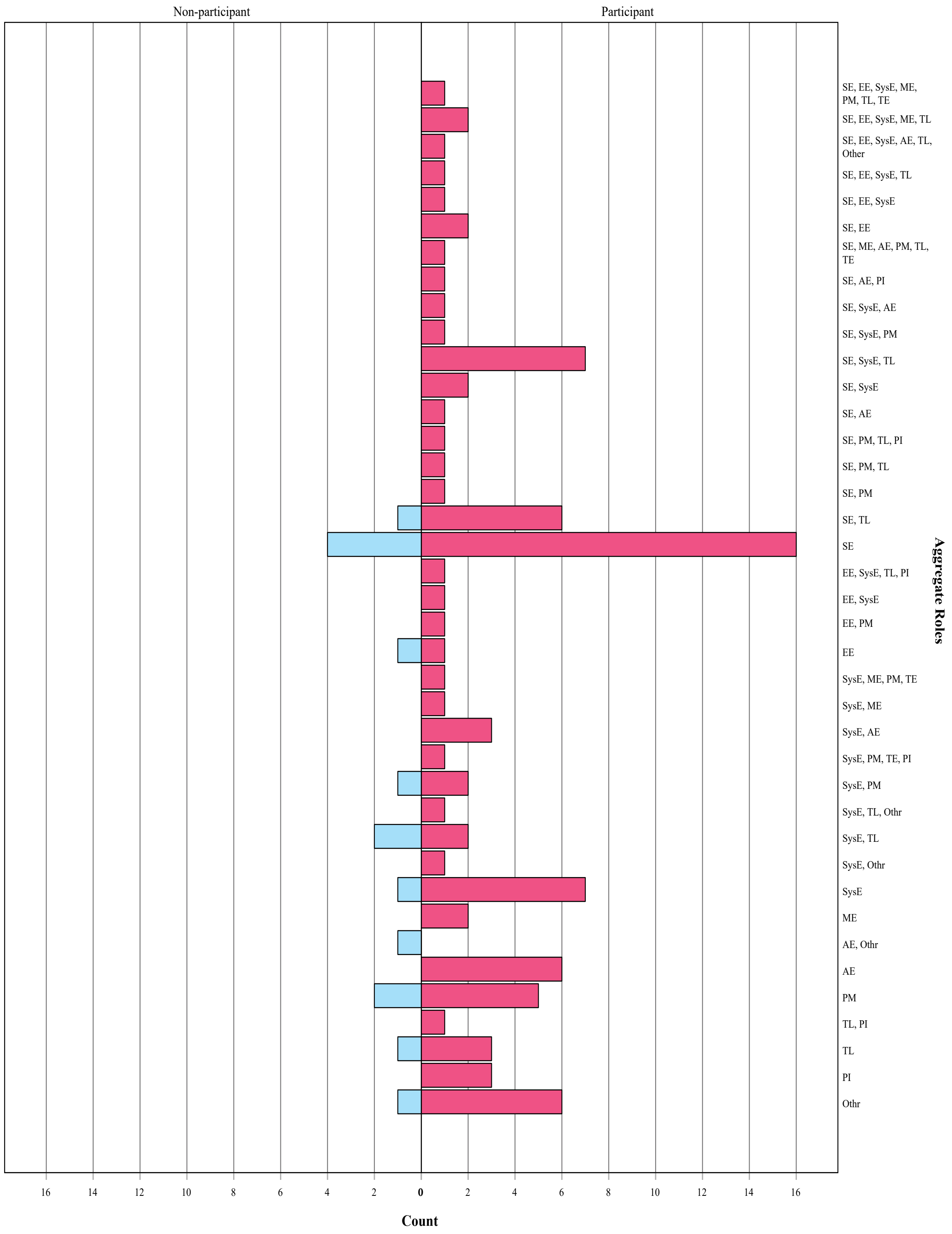


Fig. B.2. *CC* Survey, Question 1.1 — Aggregate Roles. The aggregate make-up of participants in terms of their self-identified roles for those that only participated in the background section (Non-participant) and those that participated in the full survey (Participant).

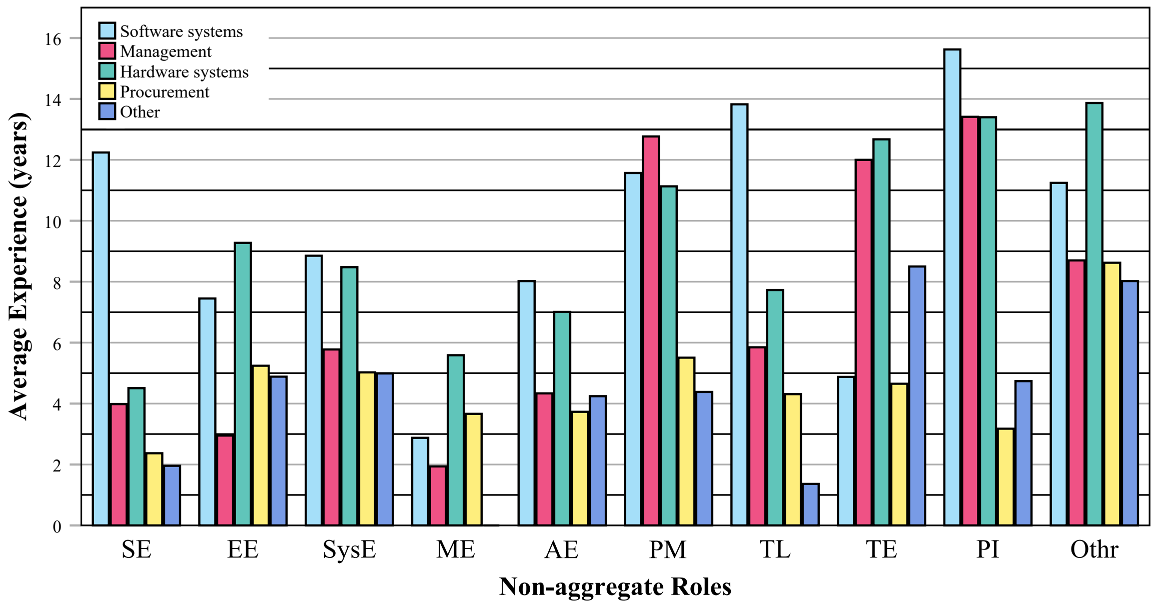


Fig. B.3. *CC* Survey — Non-aggregate Roles vs. Average Experience. The average years of experience in the different development disciplines for each of the non-exclusive roles.

might look at this either via detailed analysis of these surveys or a more role-experience targeted survey.

B.3 *OSAM* Survey — Roles and Development Experience

Table B.2 shows that SEs had the strongest representation with 9 participants that identified strictly as an SE, or about 20%. Roles inclusive of SEs totaled 17 participants or about 39% that were SE inclusive. The next closest dedicated role was made up of PMs, they accounted for 6 of the participants or about 14%. If PM inclusive roles sets are included then the total goes to 17 or about 39%. SysEs, TLs, and PMs continued their trend of belonging to multi-role aggregates.

Table B.2. *OSAM* Survey, Question 1.1 — Role Breakdown

Role	Exclusive Aggregate Count	Exclusive Aggregate Share	Inclusive Count	Inclusive Share
SE	9	20.45%	17	38.64%
EE	1	2.27%	7	15.91%

(continued on next page)

Table B.2. *OSAM* Survey, Question 1.1 — Role Breakdown

Role	Exclusive Aggregate Count	Exclusive Aggregate Share	Inclusive Count	Inclusive Share
SysE	2	4.55%	16	36.36%
ME	0	0.00%	3	6.82%
AE	0	0.00%	8	18.18%
PM	6	13.64%	17	38.64%
TL	0	0.00%	15	34.09%
TE	0	0.00%	3	6.82%
PI	1	2.27%	5	11.36%
Other	1	2.27%	4	9.09%

They can have low exclusive aggregate counts, and still have a high count when inclusive aggregates are considered; PMs were a bit of an exception because they had a relatively high exclusive aggregate count this time, but their inclusive coverage was still disproportionately high when compared to other roles like AEs or EEs. SysEs, TLs, and PMs account for about 36%, 39%, and, 34%, respectively, of the *OSAM* Survey participants when considered as inclusive aggregates.

Figure B.4 shows the participant roles for the *OSAM* Survey. Figure B.5 shows the experience breakdown for each role that participated in the *OSAM* Survey. All the roles in this survey show really good overall experience in all the areas, and generally the averages are higher in this survey than the are for the first two surveys. This time EEs show the strongest skew towards their discipline. PM and PIs show very strong overall experience, this is expected for those types of roles. AEs also show good experience across the board.

B.4 *Security* Survey — Roles and Development Experience

Table B.3 shows that the strongest pure role was again SEs with 5 participants, or about 16%. The SE inclusive aggregate totaled out at 13 participants or about 42%. Next came the SysE dedicated role, they accounted for 4 of the participants or about 13%. The SysE inclusive aggregate role accounted for 17 participants or about 55%. SysEs, TLs, and

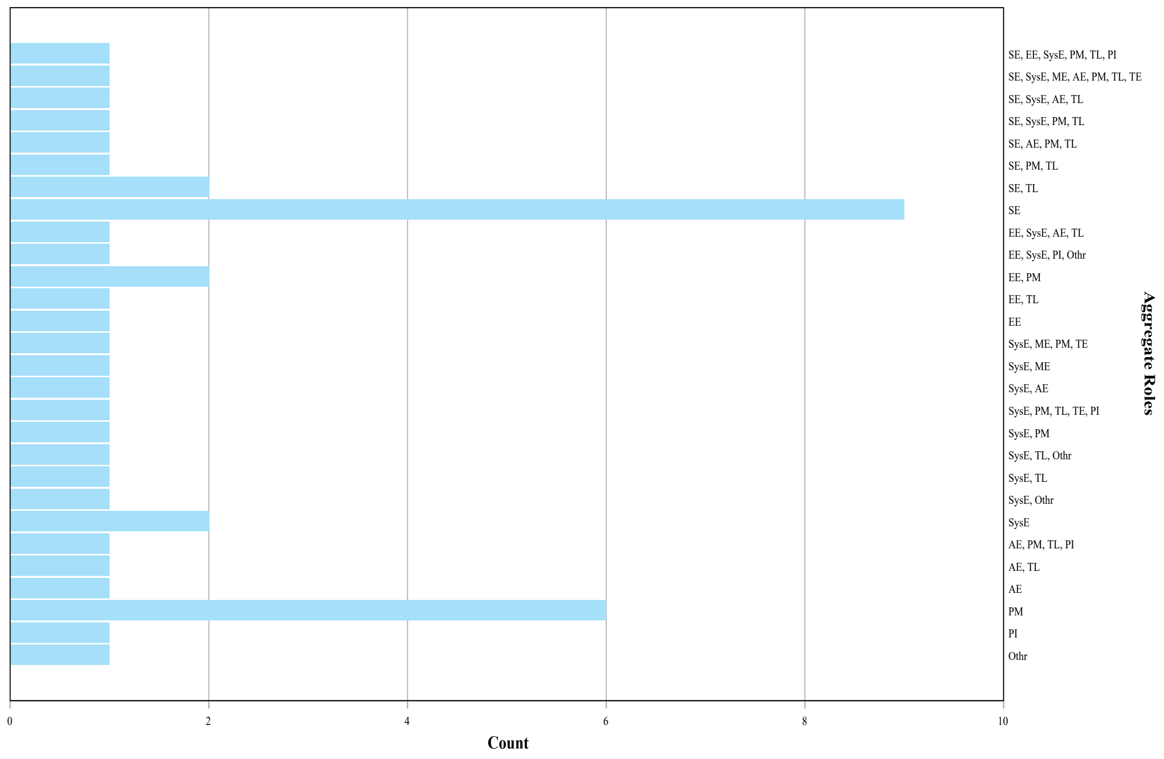


Fig. B.4. *OSAM* Survey, Question 1.1. — Aggregate Roles. The aggregate make-up of participants in terms of their self-identified roles.

PMs continued their trend of belonging to multi-role aggregates at higher rates then any of the other roles, aside from SEs, which was the dominant role overall.

Table B.3. *Security* Survey, Question 1.1 — Role Breakdown

Role	Exclusive Aggregate Count	Exclusive Aggregate Share	Inclusive Count	Inclusive Share
SE	5	16.13%	13	41.94%
EE	0	0.00%	1	3.23%
SysE	4	12.90%	17	54.84%
ME	0	0.00%	1	3.23%
AE	0	0.00%	4	12.90%
PM	0	0.00%	9	29.03%
TL	0	0.00%	10	32.26%
TE	0	0.00%	1	3.23%
PI	0	0.00%	4	12.90%

(continued on next page)

Table B.3. *Security* Survey, Question 1.1 — Role Breakdown

Role	Exclusive Aggregate Count	Exclusive Aggregate Share	Inclusive Count	Inclusive Share
Other	3	9.68%	4	12.90%

To that end TLs and PMs both accounted for 0 of the participants in terms of exclusive roles, but when counted in inclusive aggregate they accounted for 32% and 29% of the participants respectively. The other roles that had 0 participants when considered in exclusive aggregate, e.g. EE, AE, and ME, only accounted for between 3% and 12% when considered in inclusive aggregate and actually averaged around 6%.

Figure B.6 shows the participant roles for the *Security* Survey. Figure B.7 shows the experience breakdown for each roles that participated in the *Security* Survey. The roles did not have as balanced experience levels for this survey as they did for the previous surveys. For example, the EEs show a strong skew towards hardware systems and a little management but no other experience. There is only one EE in this case. SEs again show the strongest skew towards software systems development experience. As usual PM, TLs, and PIs show very strong overall experience.

B.5 ***RIPCC*** **Survey — Roles and Development Experience**

Table B.4 shows that the largest pure role was again SEs with 5 participants, or about 17%. The SE inclusive aggregate totaled out at 15 participants or about 56%. No other role had a significant exclusive share, the next closest being PMs and *Other* with 2 each or about 7% each. Even with SysEs, TLs, and PMs not having much in the way of exclusive share they still have a strong showing when considered in inclusive aggregate. SysEs, TLs, and PMs continued their trend of belonging to multi-role aggregates at higher rates then any of the other roles, aside from SEs, coming in at 48%, 35%, and 38%, respectively.

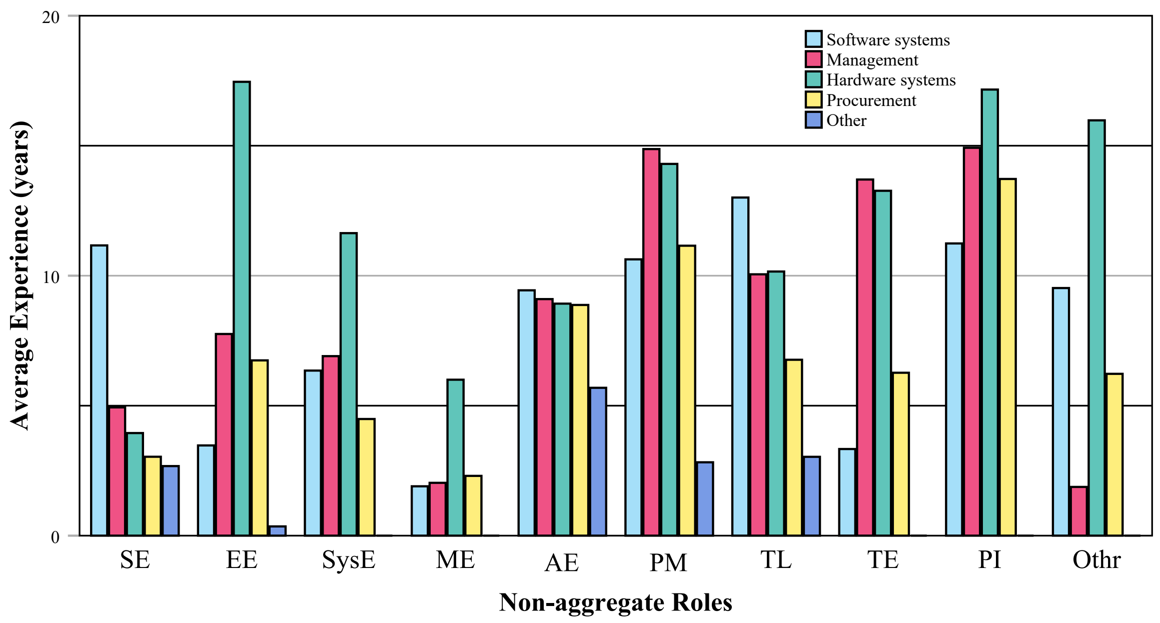


Fig. B.5. *OSAM* Survey — Non-aggregate Roles vs. Average Experience. The average years of experience in the different development disciplines for each of the non-exclusive roles.

Table B.4. *RIPCC* Survey, Question 1.1 — Role Breakdown

Role	Exclusive Aggregate Count	Exclusive Aggregate Share	Inclusive Count	Inclusive Share
SE	5	17.24%	15	51.72%
EE	1	3.45%	4	13.79%
SysE	1	3.45%	14	48.28%
ME	0	0.00%	1	3.45%
AE	0	0.00%	6	20.69%
PM	2	6.90%	11	37.93%
TL	1	3.45%	10	34.48%
TE	0	0.00%	1	3.45%
PI	0	0.00%	1	3.45%
Other	2	6.90%	2	6.90%

The other roles ranged from 3% to 21% when considered in inclusive aggregates, giving an average of about 9%. The high of 21% was driven by the AE role, and this actually a trend that was consistent across all the surveys, i.e. the AEs consistently came next in terms of inclusive aggregate after SEs, SysEs, TLs, and PMs.

Figure B.8 shows the participant roles for the RIPCC Survey. Figure B.9 shows the experience breakdown for each role that participated in the RIPCC Survey. The PI role had a pretty strong software systems and management experience skew, but also just a lot

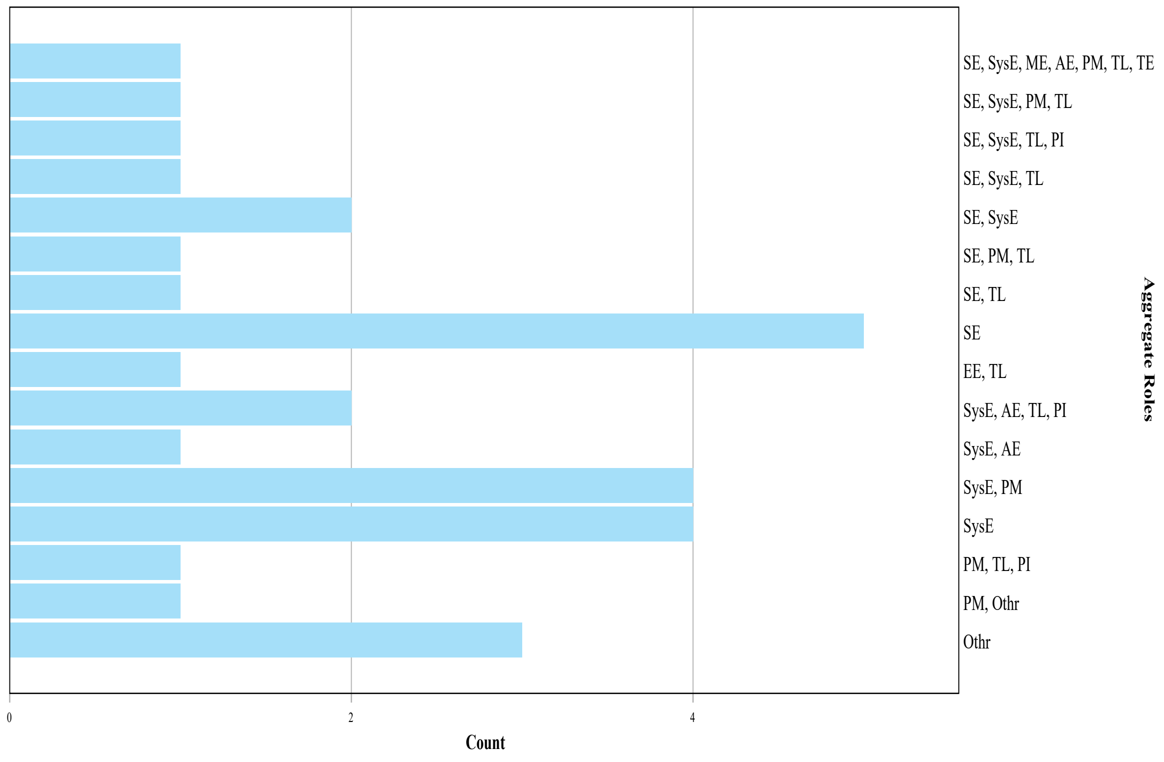


Fig. B.6. *Security* Survey, Question 1.1 — Aggregate Roles. The aggregate make-up of participants in terms of their self-identified roles.

about 6%. SysEs, TLs, and PMs continued their trend of belonging to multi-role aggregates at higher rates then any of the other roles, aside from SEs, coming in at 44%, 31%, and 40%, respectively. The other roles ranged from 3% to 22% when considered in inclusive aggregates, giving an average of about 11%. This time the EEs were the next highest mark at 22%, but the AE role was not far behind at 19%. No TEs participated in this survey, even when inclusive aggregates are considered.

Table B.5. *Network* Survey, Question 1.1 — Role Breakdown

Role	Exclusive Aggregate Count	Exclusive Aggregate Share	Inclusive Count	Inclusive Share
SE	5	15.63%	18	56.25%
EE	2	6.25%	7	21.88%
SysE	1	3.13%	14	43.75%
ME	0	0.00%	4	12.50%

(continued on next page)

Table B.5. *Network* Survey, Question 1.1 — Role Breakdown

Role	Exclusive Aggregate Count	Exclusive Aggregate Share	Inclusive Count	Inclusive Share
AE	0	0.00%	6	18.75%
PM	2	6.25%	10	31.25%
TL	1	3.13%	13	40.63%
TE	0	0.00%	0	0.00%
PI	1	3.13%	4	12.50%
Other	0	0.00%	1	3.13%

Figure B.10 shows the breakdown of participants by aggregate roles of those who participated in the *Network* survey. Figure B.11 shows the experience breakdown for each roles that participated in the *Network* Survey. The experience means here show similar trends as the other surveys that had good experience mixes. The exception being the *Other* role with 20 years of other experience, there is only one participant that identified this way. It is odd because they also identified as an SE, but do not appear to claim any software systems development experience. Might have been an error in how they filled out the survey.

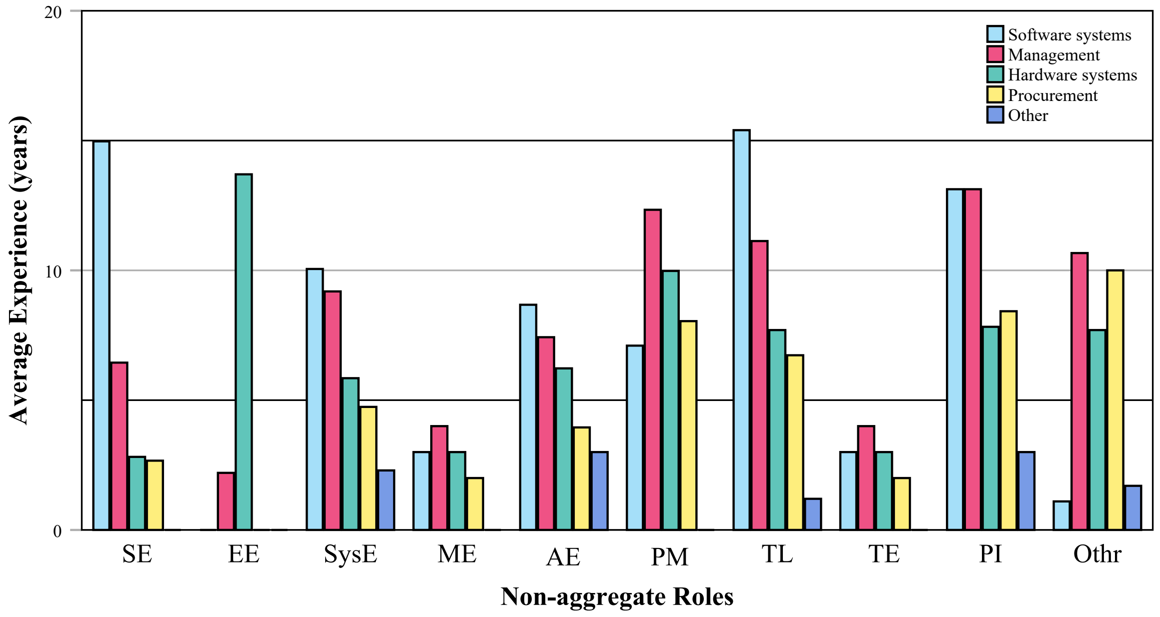


Fig. B.7. *Security* Survey — Non-aggregate Roles vs. Average Experience. The average years of experience in the different development disciplines for each of the non-exclusive roles.

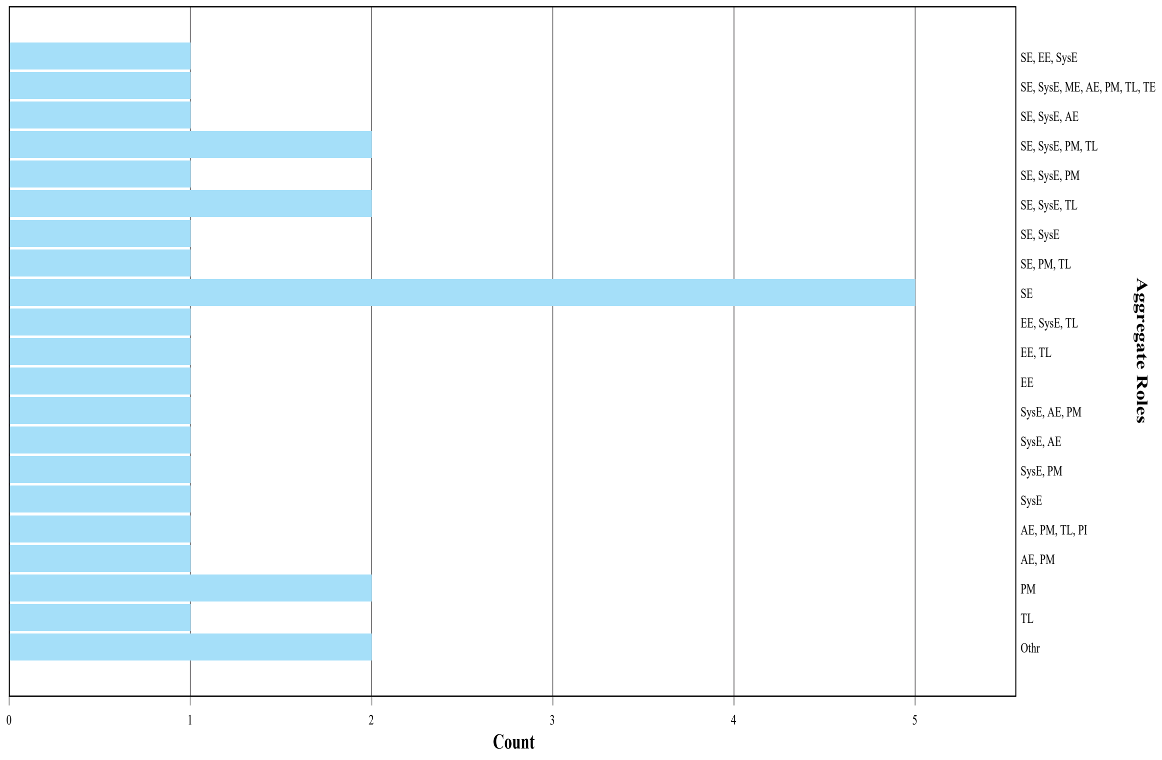


Fig. B.8. *RIPCC* Survey, Question 1.1 — Aggregate Roles. The aggregate make-up of participants in terms of their self-identified roles.

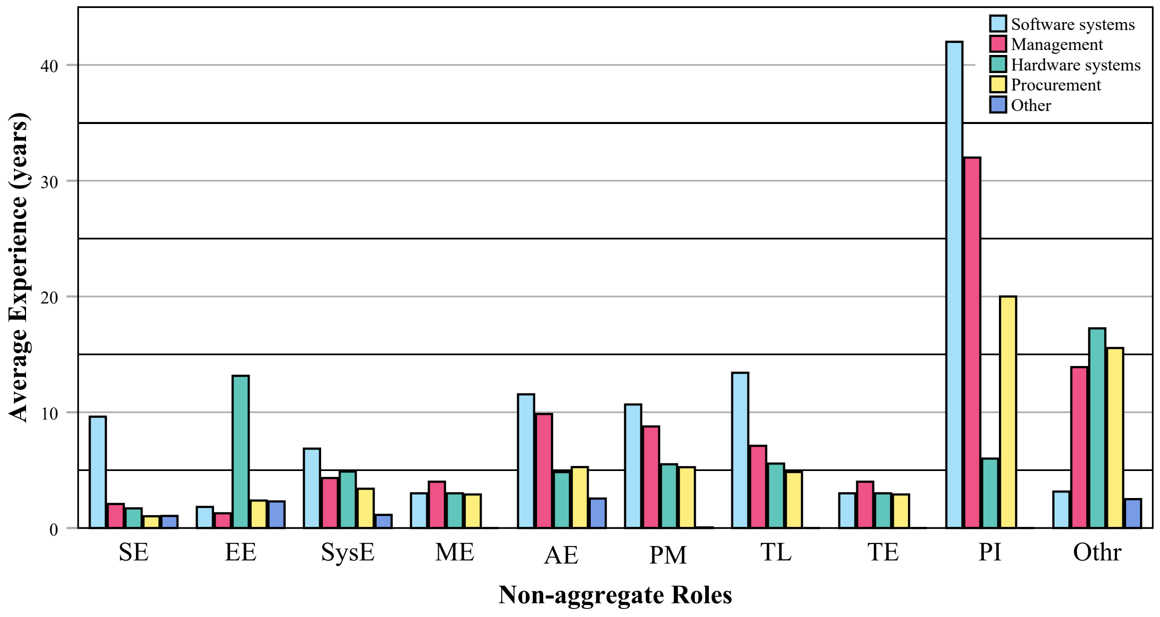


Fig. B.9. *RIPCC* Survey — Non-aggregate Roles vs. Average Experience. The average years of experience in the different development disciplines for each of the non-exclusive roles.

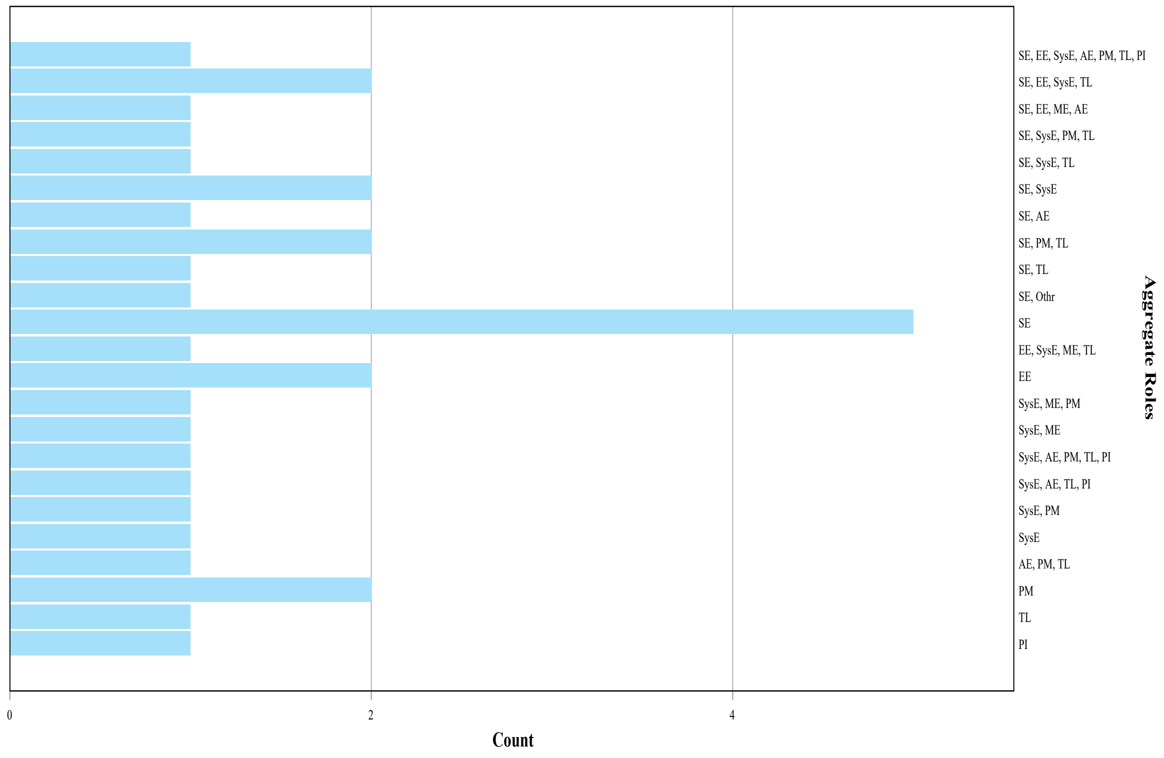


Fig. B.10. *Network* Survey, Question 1.1 — Aggregate Roles. The aggregate make-up of participants in terms of their self-identified roles.

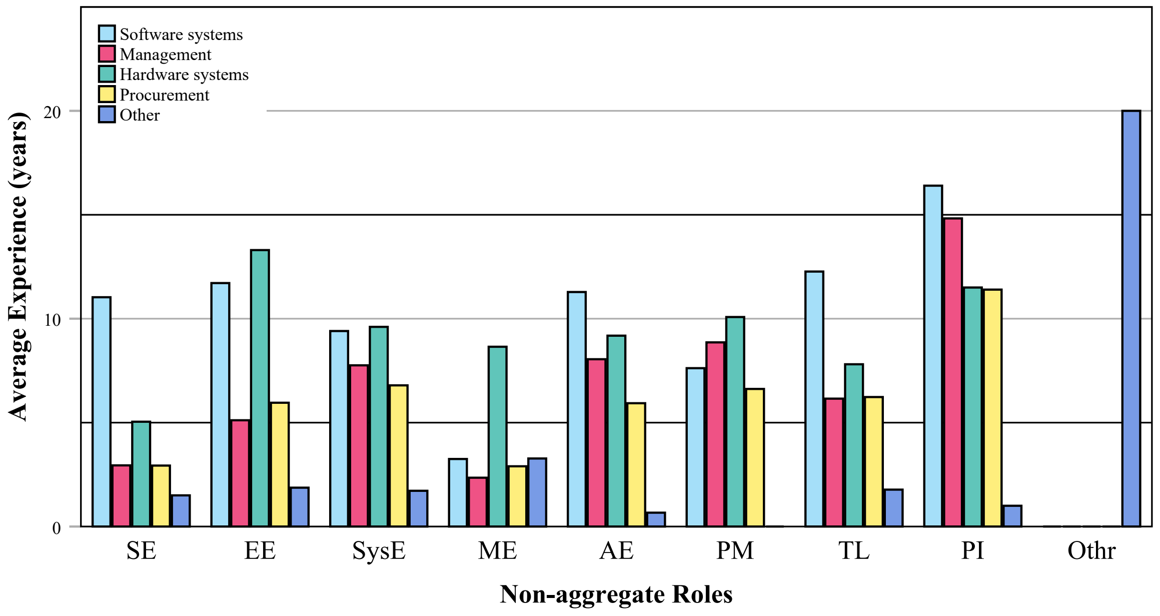


Fig. B.11. *Network* Survey — Non-aggregate Roles vs. Average Experience. The average years of experience in the different development disciplines for each of the non-exclusive roles.

APPENDIX C
Background

This appendix presents a more detailed description of the existing work used by SSSM. Specifically SSM and Kerberos, this appendix exists to provide context for a reader who is not familiar with SSM or Kerberos. Both are leveraged heavily to create SSSM and understand them is helpful in understanding what SSSM brings to the table. If the reader is already familiar with SSM and Kerberos then this appendix does not cover any new ground. This appendix is referenced in various other chapters were additional background might be helpful.

C.1 SSM

SSM is an open networked architecture that allows for communication amongst components without distinction between software and hardware entities [6]. SSM is essentially a self-configuring network geared towards space systems. There are many self configuring network examples in terrestrial systems, and while an expert may be needed to optimize them, they can largely configure themselves. SSM attempts to remove the need for expertise by replacing it with well-defined protocols and routing infrastructure [7, 52]; the intent being to make is easier to develop software systems for space applications. SSM supports Ethernet, I^{2}C, SpaceWire, Controller Area Network (CAN), and local subnets, with plans to support more, while allowing the communication across various subnets to be transparent to the communicating endpoints [6]. In the example SSM network depicted in Figure C.1 there is a heterogeneous network comprised of local subnets: a I^{2}C subnet, a Ethernet subnet, and a SpaceWire subnet. SpaceWire stems from the Institute of Electrical and Electronics Engineers (IEEE) 1355–1995 standard for Heterogeneous Interconnect publish in 1995 [53]. IEEE 1355 was later adapted in 2003 to apply to space systems, specifically SpaceWire [54, 55].

Consider a scenario where one of the SPA SpaceWire (SPA-S) Components wants to subscribe to data from one of the SPA I^{2}C (SPA-1) components. Setting up this relay requires querying for the data, setting up the subscription, and then receiving the data. The subscription traffic traverses an I^{2}C subnet, a local subnet, an Ethernet subnet, another local subnet, and a SpaceWire subnet. This is handled by the SPA Data Link Layer that is part of SSM and is realized by the SPA Subnet Managers [6]. In this case the subnet managers that are facilitating this communication are the SM-L, the SPA-1 subnet-manager (SM-1), the SM-E, and the SPA-S subnet-manager (SM-S).[3] SSM provides a set of services, namely the CAS and the LS that along with the subnet managers help to provision the network and facilitate communication amongst the various components.

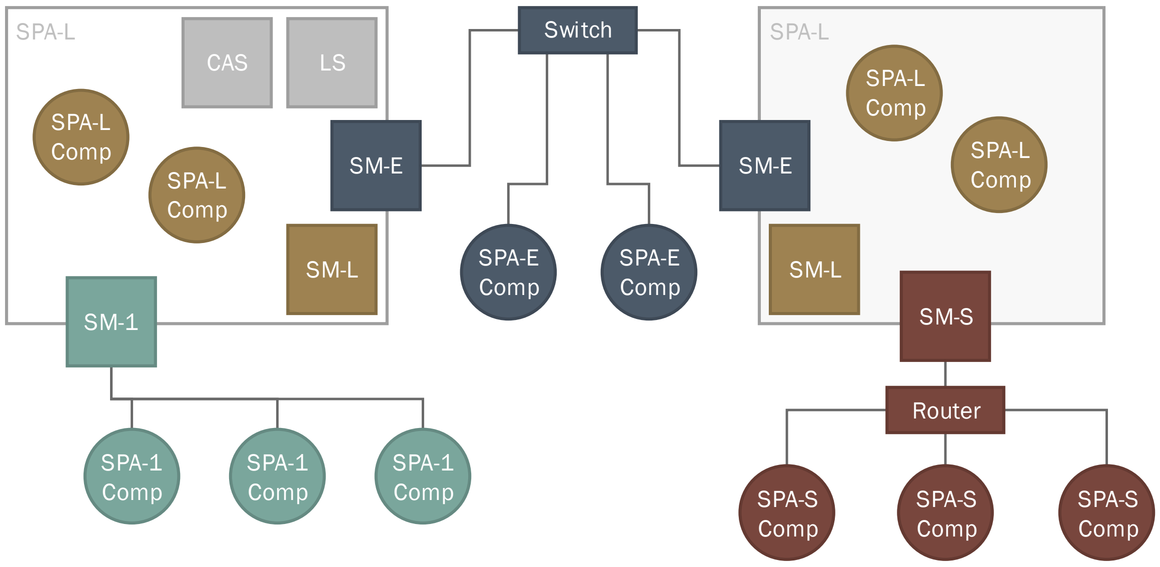


Fig. C.1. SSM Network Topology Example. An example heterogeneous SSM network comprised of local, SpaceWire, Ethernet, and I^{2}C subnets.[3]

C.1.1 The LS and xTEDS

The LS acts as a directory service, systems or applications register with the LS when they join the network by sending the LS some information about what services, commands,

[3]Figures and examples are adapted from "Scalable Network Approach for the Space Plug-and-Play Architecture" [6].

or interfaces they provide. Figure C.3 depicts the life-cycle of two components or applications on an SSM network, the diagram leaves off some detail. This life-cycle and the process of finding services and negotiating communication exists in the 'Applications and Devices' or SPA Application level of the the OSI-like [56] SPA Network Model in Figure C.2. This

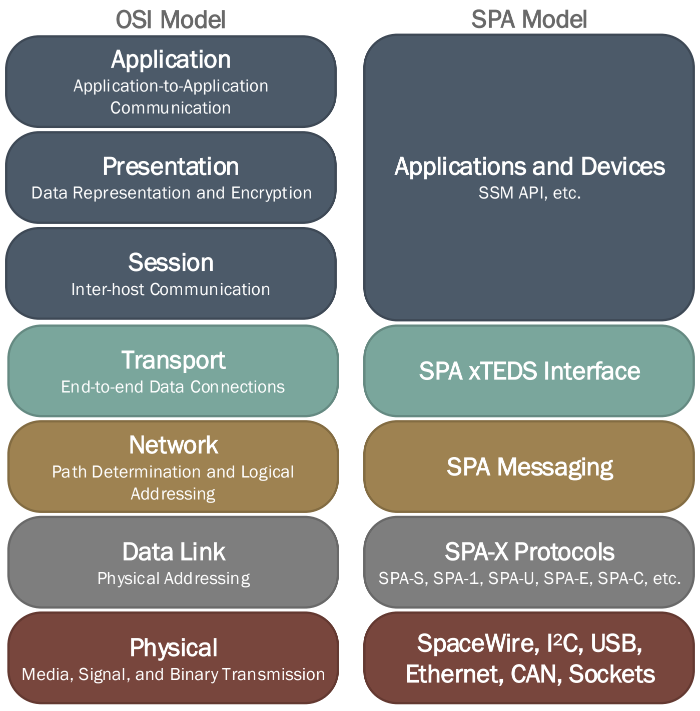

OSI Model
SPA Model
Application
Application-to-Application Communication
Presentation
Data Representation and Encryption
Session
Inter-host Communication
Transport
End-to-end Data Connections
Network
Path Determination and Logical Addressing
Data Link
Physical Addressing
Physical
Media, Signal, and Binary Transmission
Applications and Devices
SSM API, etc.
SPA xTEDS Interface
SPA Messaging
SPA-X Protocols
SPA-S, SPA-1, SPA-U, SPA-E, SPA-C, etc.
SpaceWire, I²C, USB, Ethernet, CAN, Sockets

Fig. C.2. OSI to SPA Network Stack Comparison. A comparison between the OSI and SPA Network layers.[3]

level or layer is charged with the transactions that occur in the Registration, Query, Subscription, and Data Exchange blocks shown in Figure C.3 and described below; it should be noted that actual data discovery occurs in the Transport or SPA xTEDS Interface layer and so there is some overlap between the two layers.

This example follows a Consumer or SSM App (*S1*) who wishes to subscribe to some data, at the start of this interaction *S1* does not need to know who provides the data, the subscription can be provider agnostic, or the query can be tailored to restrict or filter providers. Figure C.3 deals with a subscription which is a Notification in Sample C.2, Sample C.2 is explained in more detail later in this section, and the other message types are covered.

To begin each component or application on the network registers with the LS, this is shown in the Registration block of Figure C.3, and happens after probing that involves the subnet managers. During the registration process each component supplies information about what services, commands, or interfaces they provide, which the LS uses to build up a catalog of what is available and from whom. In this example both *S1* and *S2* register with the LS by providing an overview of their capabilities using markup as shown in Sample C.2. Once *S1* is registered it queries the LS for some data to which it wants to subscribe. In Figure C.3 this data is provide by *S2*, also referred to as a producer. It is typical when using SSM to describe these data relationships in terms of a producer and a consumer, so in this example *S1* is consuming the data produced by *S2*; this terminology also holds for commands or requests in SSM terminology.

```xml
<SpaQuery targetType='Notifcation'>
  <Interface>
    <Attribute name='name' operand='eq' value='ExampleInterface'/>
  <Interface>
  <Message>
    <Attribute name='name' operand='eq' value='ExampleData'/>
  </Message>
</SpaQuery>
```

Sample C.1. Notification Query. An example query that would be issued to the LS in order to find a provider of a notification with an interface named ‘ExampleInterface’ and message named ‘ExampleData’.

This query is shown in the Query block of Figure C.3 where *S1* sends a `SpaQueryRequest` to the LS. The basic syntax of the query is shown in Sample C.1, additional attributes could be added to make the query more restrictive. The LS check sa table of registered components and look for a match to the query from *S1*. This matches against the ‘registration information’ provided by *S2* shown in Sample C.2 and explained in more detail shortly. The LS returns a `SpaQueryReply` with information about *S2* to *S1* that *S1* can then use to make a subscription request as shown in the ‘Start Subscription’ block, if desired. This querying process works the same for the three different messaging types that are about to be explained.

This subscription request comes in the form of a `SpaSubscriptionRequest` message with some information about the data in which *S1* has interest. The message contains the producer and consumer logical addresses, a dialog identifier, the interface and message identifiers for the message in question, and a flag indicative of the subscription action or type. The producer logical address (LA_S2), and the interface (InterfaceId) and message (MessageId) identifiers, are provided by the LS in the `SpaQueryReply` in the Query block and uniquely identify a path to the message in question. The identifiers for the message are coded in the "registration information" provided by *S2* during registration, this "registration information" is explained in more detail shortly. The dialog identifier (DialogId) is used to correlate messages that relate to the subscription between *S1* and *S2*. The "Start Subscription" block shows *S2* replying with a `SpaSubscriptionReply` message containing the correlating `DialogId` and a flag indicating that the subscription was granted. The exchange now moves on to the next phase shown in the "Data Exchange" block of Figure C.3 where *S2* starts relaying data to *S1* in the form of a `SpaData` message which contains the corresponding `DialogId` and data. Now *S1* and *S2* can share data for some time as shown in the "Data Exchange" block. Either party can terminate the subscription at any time. The "Stop Subscription" block in Figure C.3 shows *S1* canceling the subscription by passing a `SpaSubscriptionRequest` that is almost identical to the one in the "Start Subscription" block except the subscription indicates a cancellation via the `SUBSCRIPTION_CANCEL` flag. Finally, *S2* responds with a `SpaSubscriptionReply` and the subscription has been terminated. *S1* can restart the subscription at any time.

The "registration information," i.e. information about services provided, that an application or component shares with the LS is standardized so that all parties engaged in communication can understand how messages are formatted and what messages are available. SSM makes use of an XML Transducer Electronic Data Sheet also referred to as xTEDS to encapsulate a XML specification of the data products and commands available from a component [57]. xTEDS is a schema to describe the characteristics of interfaces provided by each component and the messages used to accept and relay information via those interfaces. The main characteristics encapsulated in an xTEDS are: interface characteristics, messages by interface (data products and command messages), data types used in messages, message formats, timing constraints, one-dimensional arrays, and some range specification [58]. When systems register with the SPA Network implemented as SSM they send a copy of their xTEDS to the LS, these xTEDS can also be cached, and a XUUID can be used instead. This exchange of xTEDS information is part of data discovery and connections and so correlates with the Transport or SPA xTEDS Interface layers.

IEEE defined a Transducer Electronic Data Sheet (TEDS) standard as part of the IEEE 1451.4 standard [57,59]. Various work for configuring networks of transducers with relation to modular space systems started in the early of first decade of the new millennium [60,61]. Around the middle of the 2000's xTEDS was created by USU as an extension of the TEDS standard using XML. It was originally developed by Utah State University in conjunction with Satellite Data Model (SDM) to allow for the configuration of on-board subsystems. SDM is a predecessor to SSM. xTEDS saw continued updates under work sponsored by the ORS program office of the United States Air Force at Kirtland and Utah State University. Sample C.2 depicts a very basic xTEDS example.

Sample C.2 contains one `Interface` shown on line 7. This `Interface` contains a Notification, a Request, and a Command message. An xTEDS may contain multiple `Interfaces` and a `Interface` may contain multiple messages. Each `Interface` has a Interface identifier, shown on line 7, that is unique within a given xTEDS. Each `Message` has a Message identifier, shown on lines 8, 13, and 20, which are unique within a `Interface`; the unique Interface and Message identifier pairings allow a message to be defined uniquely within an xTEDS while allowing like functionality to be grouped together in an `Interface`. These unique `Interface` and `Message` identifier pairings also allow the security additions to SSM to specify permissions at a very granular level, Section 5.3 documents this granularity and security.

These messages are implemented in SSM as `SpaMessage` objects, specifically `SpaData`, for Notifications; `SpaServiceRequest` and `SpaServiceReply` for Requests; and `SpaCommand`

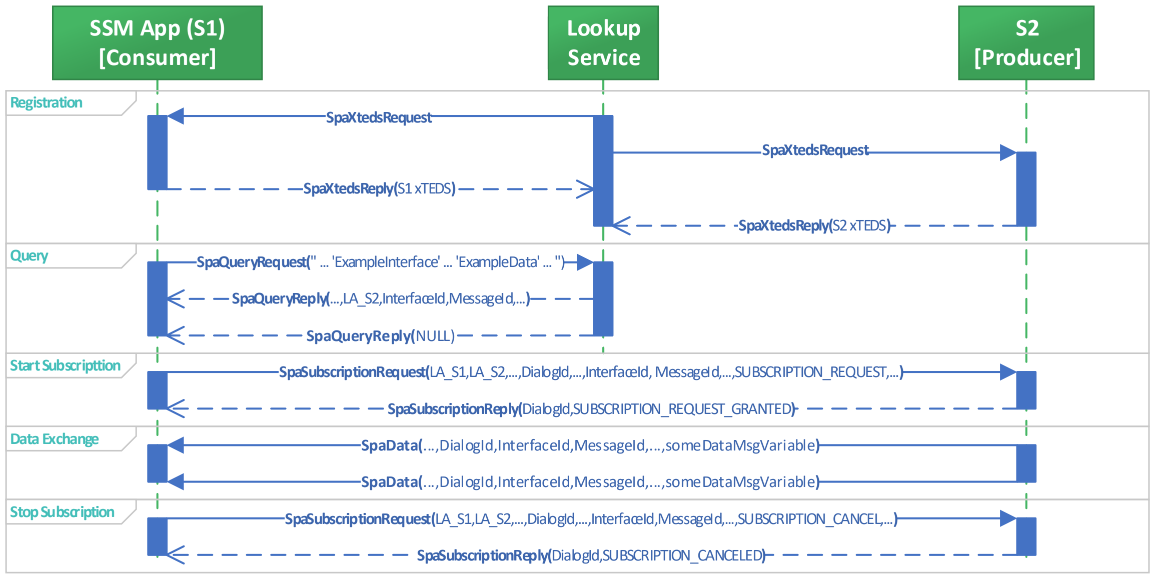

SSM App (S1) [Consumer]
Lookup Service
S2 [Producer]
Registration
SpaXtedsRequest
SpaXtedsRequest
SpaXtedsReply(S1 xTEDS)
SpaXtedsReply(S2 xTEDS)
Query
SpaQueryRequest(" ... 'ExampleInterface' ... 'ExampleData' ... ")
SpaQueryReply(...,LA_S2,InterfaceId,MessageId,...)
SpaQueryReply(NULL)
Start Subscripttion
SpaSubscriptionRequest(LA_S1,LA_S2,...,DialogId,...,InterfaceId, MessageId,...,SUBSCRIPTION_REQUEST,...)
SpaSubscriptionReply(DialogId,SUBSCRIPTION_REQUEST_GRANTED)
Data Exchange
SpaData(...,DialogId,InterfaceId,MessageId,...,someDataMsgVariable)
SpaData(...,DialogId,InterfaceId,MessageId,...,someDataMsgVariable)
Stop Subscription
SpaSubscriptionRequest(LA_S1,LA_S2,...,DialogId,...,InterfaceId,MessageId,...,SUBSCRIPTION_CANCEL,...)
SpaSubscriptionReply(DialogId,SUBSCRIPTION_CANCELED)

Fig. C.3. SSM Notification Lifecycle Example. Example of a SSM Application (*S1*) querying the LS for and subscribing to data produced by *S2* for some duration and then stopping the subscription.[3]

for Commands. These messages are also referenced in Figure 5.3 where Section 5.3.2 explains the additions to SSM that have been made for this research. The purpose and structure of each message type is explained in more detail below:

Notification

Notifications can be thought of as data that a consumer subscribes to from a producer. The consumer queries the LS to find the data it wants, shown in Figure C.3, and then subscribes to said data. In this example *S1* is querying for the 'ExampleData' message shown on line 9 in the Notification of the xTEDS example in Sample C.2. This data is encapsulated in a `SpaData` message that allows the data to traverse the various subnets and be understood on the receiving end by the consumer or *S1* in Figure C.3. The message header contains, among other items, the producer address, the interface identifier, and message identifier; these items allow the consumer to identify which message it is receiving and handles it appropriately as implemented by the developer. The hand-off of the `SpaData` to the developer is handled by the SSM API via callback function. The payload of the `SpaData` message contains the actual data described in the xTEDS on line 9 and placed there by the producer. In this example the payload contain a dynamically sized array of bytes that could contain up to 1000 elements, in this case bytes. The consumer receives these bytes from the producer and then decide what to do with them. The Notifications can be event driven or periodically produced at some interval. In this example the *ExampleData* notification is produced periodically at a rate of 1 Hz, this is designated on line 9 of the example xTEDS.

Request

Requests can be thought of as a command followed by a response. In SSM this Request is implemented as a `SpaServiceRequest`, or `CommandMsg` in the xTEDS, and a `SpaServiceReply`, or `DataReplyMsg` in the xTEDS. These are one-off exchanges where one command gets one response. In the example xTEDS in Sample C.2 the Request is on line 13. The next line shows a `CommandMsg` named *ExampleRequestCommand* that has no variables or parameters. The `DataReplyMsg` that follows contains a single variable of `UINT32` type or a 4-byte unsigned integer. This lets both parties understand the relayed data and how it should be interpreted.

Command

This is simply a message that *S1* sends to *S2* that *S2* handles without providing any response to *S1*, or a command without a response. At present anyone on the SSM network can locate this command and send the command without any type of verification, truly an open interconnect, being able to control this access, when desired, is when the research described in Chapter 5 comes into play. In the example xTEDS Sample C.2 the Command is on line 20. This command is named “ExampleCommand”, this name is something for which a consumer looking for the command could query from the LS; the consumer could also query for the variable name on line 22, which is *someCommandVariable*, or other attributes that are not present or specially called out in this simple example. In this case the xTEDS owner is expecting a consumer or command user to send a command with a `FLOAT32` in network byte order as part of the payload of the command. The receiver of this command could pass the handling of the `FLOAT32` to a callback so that a developer can perform some operations or logic determinant on the value of said variable.

```xml
<?xml version='1.0' encoding='utf-8' ?>
<xTEDS xmlns='https://pnpsoftware.sdl.usu.edu/redmine/projects/xtedsschema'
       xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
       xsi:schemaLocation= 'https://pnpsoftware.sdl.usu.edu/redmine/projects/xtedsschema
            https://pnpsoftware.sdl.usu.edu/spa/xteds/current.xsd'
       name='ExampleXteds' version='1.0'>
  <Application name='SsmApp2' kind='application' programMemoryRequired='1' dataMemoryRequired='1'/>
  <Interface name='ExampleInterface' id='1'>
    <Notification>
      <DataMsg name='ExampleData' id='1' msgArrival='PERIODIC' msgRate='1.0' >
        <DynamicArray name='someDataMsgVariable' kind='color' dataType='UINT8' maxArrayElements='1000'
              units='unitless' description='Example of dynamic byte array' />
      </DataMsg>
    </Notification>
    <Request>
      <CommandMsg name='ExampleRequestCommand' id='2'>
      </CommandMsg>
      <DataReplyMsg name='ExampleDataReply' id='3'>
        <Variable name='someDataReplyMsgVariable' kind='count' dataType='UINT32' units='count'/>
      </DataReplyMsg>
    </Request>
    <Command>
      <CommandMsg name='ExampleCommand' id='4'>
        <Variable name='someCommandVariable' kind='count' dataType='FLOAT32' units='count'/>
      </CommandMsg>
    </Command>
  </Interface>
</xTEDS>
```

Sample C.2. Basic xTEDS. An example xTEDS showing simple notifcation, request, and command message all under a single inferface.

C.1.2 CAS, Subnet Managers, and LAs

The previous section explained the LS, xTEDS, and basic messages. The question that still remains is: how do these messages make it from one component to another? This traversal requires an addressing and a routing scheme. The SDL implementation of SSM provides an implementation of the "Applications and Devices" level of the SPA Model, this maps to the Application, Presentation, and Session levels in the OSI Model. This is the level that most of the SSSM additions and modifications targeted as well as some changes at the SPA Messaging level to allow for proper message handling and setup as described in Chapter 5.

SSM makes use of logical addresses LAs that are translated to physical addresses as then traverse the various subnets. This requires that some entity be available to disperse unique logical addresses in addition to the physical addresses tied to each of the subnets.

It is important to first understand what an LA is and then what entities are charged with supplying and translating these LAs within an SSM network.

The LA is a hardware medium independent method for addressing components, this is what allows all the different subnet types to use a common addressing scheme. The LA exists at the SPA Messaging or Network Layer shown in Figure C.2. The LA does have to be mapped to a hardware specific address as messages traverse the actual physical network; this mapping is done at a lower level and information needed is in routing tables like the one depicted in Table C.1. This table belongs to one of the Ethernet subnet managers denoted as SM-E, the SM-E in question, is denoted with a LA of [2,0] in Figure C.4. This subnet manager is the first step for `SpaData` messages flowing from *S2* to *S1* as these messages traverse the local subnet, an Ethernet subnet, another local subnet, and then a SpaceWire subnet. The ability to flow in-to, out-of, and within subnets is part of this SPA Messaging or Network Layer and is explained in more detail shortly. LAs are 4-byte addresses. They have a 2-byte subnet identifier and a 2-byte component identifier; this allows for 65,536 component address per subnet and 65,536 subnets [6]. The notation for this is [2,0] where the '2' denotes the subnet identifier and the '0' denotes the component identifier or address.

Table C.1. Routing Table for Subnet Manager: Ethernet (SM-E)[3]

Logical Address	Physical Address Type	Physical Address (subnet route)
[0,1]	Local	3333
[1,0]	Local	7777
[1,3]	Local	2222
[2,0]	Self	Self
[2,1]	Ethernet	12.24.48.96:4444
[2,2]	Ethernet	12.24.48.120:5555
[3,0]	Local	9999
[4,0]	Ethernet	12.24.48.55:3333
[5,0]	Logical	[4,0]

(continued on next page)

Table C.1. Routing Table for Subnet Manager: Ethernet (SM-E) (continued)

Logical Address	Physical Address Type	Physical Address (subnet route)
[6,0]	Logical	[4,0]
[6,1]	Logical	[4,0]

Figure C.4 shows all the components and subnet managers with example LAs. This is where the CAS comes into play. The CAS takes a special static known logical address of [0,1], its subnet identifier is 0 and its component identifier is 1. A known address is used so that subnet managers can find the CAS. The subnet managers in Figure C.4 are two SM-Ls, two SM-Es, one SM-1, and one SM-S. Figure C.4 also has various components within each of the subnets: I^{2}C components (SPA-1), Ethernet components (SPA-E), Local components (SPA local (SPA-L)), and SpaceWire components (SPA-S). *S2* is a SPA-L node on the same Local subnet that houses the CAS, the LS, amongst other things, and *S1* sits on a SpaceWire subnet that is a few hops away.

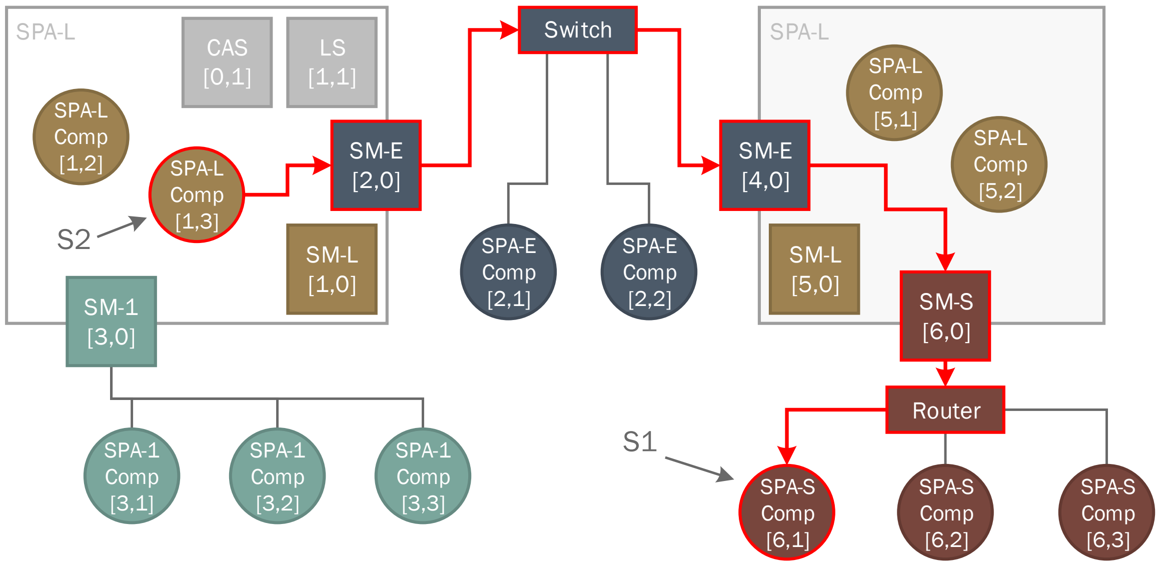


Fig. C.4. SSM Routing Example. Route that a message takes from *S2* to *S1*, e.g. when *S2* is sending Notifications or `SpaData` messages to *S1*. SM-E [2,0] uses the routing table in Table C.1 to identify where the message is routed to next in its trip to the ultimate destination of [6,1] or *S1*.[3]

Each subnet requires a certain degree of management to perform discovery on a physical or network topology level. This management is provided by SPA Subnet Managers and contained in the SPA Data Link or SPA-X Protocols layer as shown in Figure C.2. Each of the subnet managers issues and responds to probes so that the network can be mapped out continuously and so that each subnet manager can locate the CAS and the other subnets and populate its routing table. Each component or end-point also has a routing table, but for components that are not on its subnet the entries point to its subnet manager or another relevant manager if it bridges their subnet.

The CAS tells each subnet manager which subnet identifier or address block it gets, this gives the subnet manager a block of component identifiers or addresses it can give out to components on its subnet. For the example routing table in Table C.1 the SM-E received a subnet identifier of 2, it takes a component identifier of 0, and then it has 65,535 address it can give out to its subnet; whether this many components can even exist on a given subnet depends on the subnet type. Once the routing tables are built up the components can find the LS and register, the querying and other messaging described in Section C.1.1 can now take place as shown in Figure C.3.

Figure C.4 shows a message as it traverses various subnets on its trip from *S2*, a SPA-L component, to *S1*, a SPA-S Component. This is a path that a `SpaData` message might take in the 'Data Exchange' block of Figure C.3 after the query has finished and the components begin communicating. The first hop the `SpaData` message takes is to the SM-E at LA [2,0]. This assumes that *S2* already has an entry for *S1* in its routing table, otherwise it might have to hit its SM-L at [1,0] first. *S2* has an entry in its routing table for LA [6,1] with a LA of [2,0], much like the SM-E at [2,0] has an entry in its routing for LA [6,1] with an LA of [4,0]. *S2* then has to look up [2,0] to see how to get there in terms of a physical address.

Each entry in the table represents the next hop a message should take to reach its final destination. This means that *S2* looks up [6,1] and sees [2,0] and so it forwards its message. The example routing table in Table C.1 for the SM-E at [2,0] shows that next hop is to [4,0]. The SM-E at [2,0] then has to look up the [4,0] address to see how to get a message there. The table shows that this LA maps to a physical Ethernet address with with an Internet Protocol (IP) address of 12.24.48.55 and a port of 3333. The SM-E at [2,0] sends the message to the SM-E at [4,0] over Ethernet using the physical address it found in its routing table.

Now the message is at the SM-E at [4,0] which also has a routing table that it uses to look up the next hop to reach [6,1]. Figure C.4 shows that the SM-E found the next LA to be [6,0] as indicated by the next hop taken by the `SpaData` message shown in red. The SM-E at [4,0] has to first look up [6,1] which returns [6,0], and then send the message to [6,0]. The [6,0] needs to resolve the LA to a local address in the table or SM-E needs to probe the SM-L at [5,0] for the path. This is a port number or socket on the local host. This brings the message to the SM-S at [6,0], now the SM-S at [6,0] resolves the [6,1] address to a SpaceWire physical address and send it the SPA-S component at LA [6,1] and *S1* receives the `SpaData` message without understanding of the path of the message. Each component generally needs to understand the path to the next hop. SSM provides the management needed at each subnet to convert logical addressing to the appropriate network routing and properly relay messages to the appropriate endpoint or next subnet managers.[3] The part of the traversal where LAs are translated to physical address and transfer over physical medium exists in SPA Physical Layer where SpaceWire, I^2C, Universal Serial Bus (USB), Ethernet, CAN, or sockets are used to actually move the bits.

SSM is an example of a totally modular open network architecture. This means that any component can communicate with any other component; the other thing to note is that security was not considered in the design. This can be a problem if a malicious entity somehow gains access to any portion of the network; there is no assurance of availability, integrity, or confidentiality. Any component can masquerade as another component, issue any command, subscribe to any data, carry out a DoS or availability type attack, or various other types of attacks. This can exacerbate problems over a system more typical in a space system where everything is not interconnected, but SSM, and open networked systems in general, have a unique ability to address the concerns of availability, integrity, and confidentiality. SSM is modeled after a typical network stack with layered functionality, so it lends itself to security additions used in similar networked systems to add security.

C.2 Kerberos

Kerberos is a longstanding example of a centralized authentication system. Kerberos can be augmented with other services to provide authorization data or control permissions [46]. The rest of this section explains the basic protocols that are used to perform authentication of a Subject or client, and optionally the Target or service, and to establish a secure session between the two entities. This is discussed as a authentication platform that is intended to make service sharing easier, SSSM leverages these protocol concepts to bring the same ease of use to securing SSM. These protocols are described here to give context to the modifications to SSM described in Chapter 5.

Kerberos utilizes a centralized KDC that must be available, it should be noted this service should be secure and reliable since it poses a single point of failure. This issue can be minimized using replication, but other security provisions may be necessary to protect the over-all functionality of the network. One of the benefits of this type of centralized system is that the identity of who has sessions open with whom is known; this means that service utilization, and to some extent, failed attempts can be tracked on a per system basis.

The AS and the TGS comprise the KDC as shown in Figure C.5. These two services, together with secret information stored on the system that wishes to authenticate itself, allow said system to establish itself as its claimed identity and, with the help of some other protocols, get access to the services on the network to which it has the proper permissions. The KDC achieves this using symmetric key cryptography, or the secret information previously mentioned, this could also be thought of as a password. Kerberos now allows for the use of public key cryptography during certain phases of the authentication process, but for the purposes of this research, and resource utilization, symmetric keys are utilized. The clients use the secret or key that they share with the AS in conjunction with special protocols to authenticate themselves. Figure C.5 diagrams each of the steps outlined below and Table C.2 provides definitions for the terms that are being used.

Step 1 (Subject/AS exchange):

- Subject asks the AS for a Ticket Granting Ticket (TGT) to the TGS by sending Auth1
- AS looks up the Subject in its database, then generates a session key (SK_{S_TGS}) for use between the Subject and the TGS
- AS encrypts SK_{S_TGS} using the Subjects private key and sends to Subject
- AS uses the TGSs secret key (known only to the AS and the TGS) to create and send the Subject a TGT which also includes SK_{S_TGS}

Step 2 (TGS exchange):

- Subject decrypts the message and recovers SK_{S_TGS}, then uses it to create an Auth2
- Subject sends Auth2, along with the TGT, to the TGS, requesting access to the Target
- TGS decrypts the TGT and recovers SK_{S_TGS} which it then uses to decrypt Auth2
- TGS verifies information in the Auth2 and TGT, if everything matches then the Subject has authenticated to the TGS and the TGS lets the request proceed
- TGS creates SK_{S_T} for the Subject and Target to use and encrypts it using SK_{S_TGS} and sends it to the Subject
- TGS creates TK-TS which includes SK_{S_T} along with some identifying information that is then encrypted with the Targets key and sends it to the Subject

Step 3 (Subject/Target exchange):

- Subject decrypts the message and gets the SK_{S_T} (Subject cannot decrypt TK-TS)
- Subject creates Auth3, and sends Auth3 and TK-TS to the Target

- Target decrypts and checks the TK-TS and then decrypts Auth3 using SK_{S_T} found in TK-TS, and verifies that the information matches and that TK-TS is still valid
- (Optional) Target returns Auth4, which contains the previous timestamp + 1 from Auth3, encrypted with SK_{S_T} proving to the Subject that the Target actually knew its own secret key and could decrypt TK-TS and then Auth3

Step 4 (Secure communications):

- The Target knows that the Subject is who they claim to be, and the two now share SK_{S_T} for secure communications

Table C.2. Kerberos Terminology[4]

Term	Name	Description
Subject	Subject	The client/consumer/entity that wants to communicate with the Target
Target	Target	The server/producer/entity that provides a service (handles commands or produces data) with which a Subject wants to communicate
AS	Authorization Service	This service is responsible for authenticating clients/subjects. It also issues TGTs and creates SK_{S_TGS}
TGS	Ticket Granting Service	This service is responsible for handling a Subjects request to communicate with a given Target. It does this by creating TK-TS and SK_{S_T} upon request. It needs another service to decide if a Subject is allowed to communicate with a given Target before issuing a session ticket.
KDC	Key Distribution Center	Comprised of the AS and TGS, and so is responsible for authenticating subjects, creating tickets, and creating keys
Auth1	Subject to AS Auth	This is the authenticator used to authenticate the Subject to the AS, E{subjectName, subjectAddress, timeStamp}$K_{Subject}$
Auth2	Subject to TGS Auth	This is the authenticator used to authenticate the Subject to the TGS, E{subjectName, subjectAddress, timeStamp}SK_{S_TGS}

(continued on next page)

Table C.2. Kerberos Terminology (continued)

Term	Name	Description
Auth3	Subject to Target Auth	This is the authenticator used to authenticate the Subject to the Target, E{subjectName, subjectAddress, timeStamp}SK_{S_T}
Auth4	Target to Subject Auth	This is the authenticator used to authenticate the Target to the Subject, E{targetName, targetAddress, timestamp + 1}SK_{S_T}
TGT	Ticket Granting Ticket	This is a Ticket issued by the AS that allows the holder of the ticket to request access to Targets. E{subjectName, subjectAddress, validity, SK_{S_TGS}}K_{TGS}, where validity is the duration for which the TGT is valid.
SK_{S_TGS}	Session Key Subject/TGS	This is the key for communicating between the Subject and TGS, or Key(Subject, TGS)
SK_{S_T}	Session Key Subject/ Target	This is the key for communicating between the Subject and the Target, or Key(Subject, Target)
TK-TS	Session Ticket	This ticket is used to initiate a session between the subject and the target. It helps authenticate the Subject to the Target, E{subjectName, subjectAddress, validity, SK_{S_T}}K_{Target}.

Using Kerberos imposes some requirements on a system that utilizes it. For example, the ability to prevent replay attacks is predicated on some level of time synchronization between clients, services, and the KDC. Another restriction when using symmetric key encryption is that service and/or client on the network needs its own Kerberos key that is shares with the KDC.

[4]It should be noted that Kerberos realms have been dropped from the depiction of Kerberos as only a single realm is being described or considered.

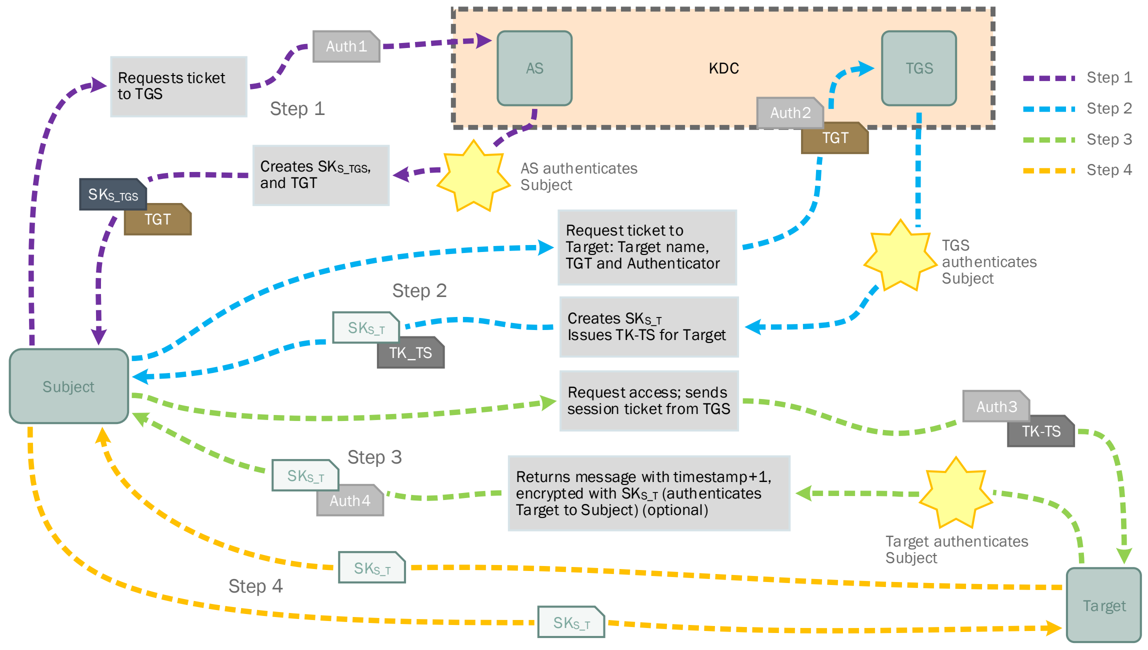

Requests ticket to TGS
Auth1
Step 1
AS
KDC
TGS
Auth2
TGT
Step 1
Step 2
Step 3
Step 4
Creates SKs_TGS, and TGT
AS authenticates Subject
SKs_TGS
TGT
Request ticket to Target: Target name, TGT and Authenticator
TGS authenticates Subject
Step 2
SKs_T
TK_TS
Creates SKs_T Issues TK-TS for Target
Subject
Request access; sends session ticket from TGS
Auth3
TK-TS
Step 3
SKs_T
Auth4
Returns message with timestamp+1, encrypted with SKs_T (authenticates Target to Subject) (optional)
Target authenticates Subject
SKs_T
Step 4
SKs_T
Target

Fig. C.5. Simplified Kerberos Authentication Exchange. Depiction of the exchange between a Subject, that wants to use a service provided by the Target, and the KDC to get a session ticket, and ultimately communicate with the Target.