

Towards Polymorphic Systems Engineering

by John TJ Mathieson

B.S. in Aerospace Engineering, May 2004, University of Notre Dame
M.S. in Astronautical Engineering, May 2008, University of Southern California

A Dissertation submitted to

The Faculty of
The School of Engineering and Applied Science
of The George Washington University
in partial fulfillment of the requirements
for the degree of Doctor of Philosophy

January 8, 2021

Dissertation directed by

Thomas Mazzuchi
Professor of Engineering Management and Systems Engineering

Shahram Sarkani
Professor of Engineering Management and Systems Engineering & of Decision Sciences

ProQuest Number:28257912

All rights reserved

INFORMATION TO ALL USERS

The quality of this reproduction is dependent on the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if material had to be removed, a note will indicate the deletion.



ProQuest 28257912

Published by ProQuest LLC (2020). Copyright of the Dissertation is held by the Author.

All Rights Reserved.

This work is protected against unauthorized copying under Title 17, United States Code
Microform Edition © ProQuest LLC.

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 - 1346

The School of Engineering and Applied Science of The George Washington University certifies that John TJ Mathieson has passed the Final Examination for the degree of Doctor of Philosophy as of 30 October 2020. This is the final and approved form of the dissertation.

Towards Polymorphic Systems Engineering

John TJ Mathieson

Dissertation Research Committee:

Thomas A. Mazzuchi, Professor of Engineering Management and Systems Engineering, Dissertation Co-Director

Shahram Sarkani, Professor of Engineering Management and Systems Engineering & of Decision Sciences, Dissertation Co-Director

Amir Etemadi, Associate Professor of Electrical Engineering, Committee Member

Thomas Holzer, Professional Lecturer of Engineering Management and Systems Engineering, Committee Member

Joseph P. Blackford, Professional Lecturer of Engineering Management and Systems Engineering, Committee Member

© Copyright 2020 by John TJ Mathieson
All rights reserved

Dedication

First and foremost, I would like to dedicate this work and the countless hours spent toiling away on it to my wonderful wife, best friend, teammate, critic, and advocate, Michelle. Without your willingness to embark on my wild endeavors with me, none of this would have ever been possible. Your measured balance of patience with purpose is an attribute to which we should all aspire and one that kept me on track throughout this journey. Second, to my boys, Jack, Bowen, and particularly Rhett, considering your timely arrival into this world three days into my doctoral journey, for always bearing with me as I attempted to continually balance being both the student and the teacher, the friend and the disciplinarian and hopefully, if nothing more, a motivator for the three of you to explore your own future adventures. Lastly, to my parents, my sister, my in-laws, and my friends, for understanding an ambition to turn dreams into reality and never questioning my prioritization throughout this undertaking. You have all given me the strength and backing to persevere, and I cannot thank you all enough.

Acknowledgement

I also wish to thank my colleagues and counterparts at Northrop Grumman (both current and former), in particular Mike Violet, Brian Marland, Mike McMeekin, Todd Wantuch, Dr. Brian Barden, and Cassidy Chan. Our thought provoking conversations and desire to solve challenging problems continues to fuel the yearning to not only learn, but to chart new paths. Thank you for your continued inspiration and the support to make this journey happen.

Additionally, my gratitude to my advisors, Dr. Thomas Mazzuchi and Dr. Shahram Sarkani. Without your direction, advice, and continued counseling, none of this would have been possible. Thank you as well to Dr. Thomas Holzer for your thorough review, patience, and advisement in the finalization of this manuscript. Lastly, I would like to thank the professors and staff in The George Washington University, Engineering Management & Systems Engineering program. You are the ones who make all of this possible on a daily basis. Thank you.

Abstract of Dissertation
Towards Polymorphic Systems Engineering

Systems engineering is widely regarded as a full life cycle discipline and provides methodologies and processes to support the design, development, verification, sustainment, and disposal of systems. While this cradle-to-grave concept is well documented throughout literature, there has been recent and ever-increasing emphasis on evolving and digitally transforming systems engineering methodologies, practices, and tools to a model-based discipline, not only for advancing system development, but perhaps more importantly for extending agility and adaptability through the later stages of system life cycles – through system operations and sustainment.

This research adopts principles from the software engineering domain DevOps concept (a collaborative merger of system development and system operations) into a Systems Engineering DevOps Lemniscate life cycle model. This progression on traditional life cycle models lays a foundation for the continuum of model-based systems engineering artifacts during the life of a system and promotes the coexistence and symbiosis of variants throughout. This is done by facilitating a merger of model-based systems engineering processes, tools, and products into a surrogate and common modeling environment in which the operations and sustainment of a system is tied closely to the curation of a descriptive system model. This model-based approach using descriptive system models, traditionally leveraged for system development, is now expanded to include the operational support elements necessary to operate and sustain the system (i.e. executable procedures, command scripts, maintenance manuals, etc. modeled as part of the core system). This evolution on traditional systems engineering implementation, focused on digitally

transforming and enhancing system operations and sustainment, capitalizes on the ability of model-based systems engineering to embrace change to improve agility in the later life cycle stages and emphasizes the existence of polymorphic systems engineering (performing a variety of systems engineering roles in simultaneously occurring life cycle stages to increase system agility).

A model-based framework for applying the Systems Engineering DevOps life cycle model is introduced as a new Systems Modeling Language profile. A use-case leveraging this “Model-Based System Operations” framework demonstrates how merging operational support elements into a spacecraft system model improves adaptability of support elements in response to faults, failures, and evolving environments during system operations, exemplifying elements of a DevOps approach to cyber-physical system sustainment.

Table of Contents

Dedication	iv
Acknowledgement.....	v
Abstract of Dissertation.....	vi
List of Figures	xiii
List of Tables.....	xvi
List of Acronyms.....	xvii
Chapter 1: Introduction	1
1.1 A Brief Taxonomy.....	1
1.2 Problem Identification & Description	4
1.2.1 Background on Systems Engineering Evolution.....	5
1.2.2 Full Life Cycle Need for Model-Based Systems Engineering.....	6
1.2.3 A Related Problem-Space – Lessons from Software Engineering.....	8
1.3 Thesis Statement.....	9
1.4 Research Questions & Objectives	10
1.4.1 Research Questions	10
1.4.2 Objectives	10
1.5 Hypothesis.....	11
1.6 Proposed Solution to Enable Polymorphic Systems Engineering: The SEDevOps Lemniscate & MBSO	12
1.7 Significance, Findings, and Conclusions.....	13

1.8	Summary of Dissertation	14
Chapter 2: Literature Review		16
2.1	Literature Review Map	18
2.1.1	Primary Disciplines	19
2.1.2	Secondary & Intersecting Disciplines	19
2.1.3	Method of Literature Presentation.....	20
2.2	Systems Engineering	20
2.2.1	Systems Engineering Definition.....	20
2.2.2	Recent Developments in Systems Engineering.....	23
2.2.2.1	Model-Based Systems Engineering.....	23
2.2.2.2	Agile Systems Engineering.....	25
2.3	Software Engineering	26
2.3.1	Software Engineering - Why Systems Engineers Care	26
2.3.2	Recent Advancements in Software Engineering.....	27
2.3.2.1	Microservices	27
2.3.2.2	DevOps	29
2.3.3	Opportunities for Systems Engineering Advancement through Software Engineering Advancements	30
2.4	System Life Cycles.....	32
2.4.1	Life Cycle Definition.....	32
2.4.2	Sequential & Plan-Driven Life Cycle Models	35
2.4.3	Evolutionary & Concurrent Life Cycle Models.....	37
2.4.4	Interpersonal & Emergent Life Cycle Models	40

2.5	Digital Engineering	43
2.5.1	Definition of Digital Engineering.....	43
2.5.2	Recent Developments in Digital Engineering.....	44
2.6	Cross-Cutting Topics	45
2.6.1	Formal Methods	46
2.6.2	Safety Critical Systems.....	47
2.7	Bringing It All Together – Summarizing the State of the Art with a Call to Action	47
Chapter 3: Research Methods, Resulting Life Cycle Methodology & Framework		49
3.1	Research Methodology Overview	49
3.1.1	The Inventor’s Paradox.....	49
3.1.2	Architecture, Methodology, Framework Development.....	50
3.1.3	Methodology Map	50
3.2	Introducing the Systems Engineering DevOps Lemniscate.....	51
3.2.1	Putting It Together into SEDevOps.....	51
3.2.2	Description of Parts from Existing Life Cycle Models	54
3.2.3	Focus on Test to Enable Continuous Integration, Test, & Deployment.....	57
3.2.4	Addressing Decommissioning	59
3.2.5	Continuity & Collaboration Throughout the Life Cycle	60
3.2.6	Evolution & Adaptation.....	62
3.2.7	SEDevOps Summary	63
3.3	A Framework for Applied Methodology: Model-Based System Operations .	64
3.3.1	Summary.....	64

3.3.2	MBSO Domain Specific Language Components.....	66
3.3.3	Life-In-A-Day (LIAD) Testing	69
3.3.4	MBSO Modeling Methodology	71
3.3.5	MBSO Extension with Formal Methods	76
Chapter 4: Data Collection & Analysis.....		78
4.1	Data Source: FireSat-II Modified through MBSO	81
4.1.1	FireSat-II Base Model.....	81
4.1.2	MBSO Extension to FireSat-II Model.....	85
4.2	Use Case: Life-In-A-Day Simulation & Response	92
4.2.1	Impact to Critical Unit Failure & Simplified Ops Product Roll-Out	92
4.2.2	Discussion on Cost & Implications	97
4.2.3	Use Case Summary.....	100
4.2.4	Discussion on Additional Use Cases	100
4.2.4.1	Managing Recurring Unit Toggles Use Case – A Candidate for Formal Methods Application	101
4.2.4.2	A Mission Expansion Use Case – A Return to the Development Cycle	102
4.3	Summarizing the Insight Gained Through Use Cases and Addressing Hypothesis.....	103
Chapter 5: Conclusions		106
5.1	What was Accomplished: Contributions to the Field.....	106
5.2	What was Not Accomplished and Limitations of Findings	108
5.3	Areas of Future Research.....	109
5.3.1	Formal Methods & Safety Critical System Operations	109

5.3.2	Autonomic System Operations	109
5.3.3	Prognostics, Diagnostics, & Data Trending.....	110
5.4	Research Benefits & Potential Implementation Challenges	111
5.4.1	Benefits of SEDevOps.....	111
5.4.2	Benefits of MBSO	112
5.4.3	Potential Implementation Challenges.....	113
5.5	Final Remarks	114
Chapter 6: Bibliography.....		116

List of Figures

Figure 1 - Literature Map.....	19
Figure 2 - Generic System Life Cycle (Blanchard and Blyer 2016).	32
Figure 3 - INCOSE Generic System Life Cycle Stages (adapted from INCOSE 2015)..	33
Figure 4 - Vee Life Cycle Model (adapted from Douglass 2016).....	36
Figure 5 - Iterative and Incremental Development Life Cycle Model (Forsberg, Mooz and Cotterman 2005)	38
Figure 6 - Spiral Life Cycle Model (adapted from Douglass 2016).....	39
Figure 7 - Agile Methodology (Boehm and Turner 2004).....	41
Figure 8 - Representative DevOps Life Cycle Process (Compuware 2019).....	42
Figure 9 - Research Methodology Map	51
Figure 10 - The SEDevOps Life Cycle Model.....	52
Figure 11 - Feature Sources in the SEDevOps Life Cycle Model.....	53
Figure 12 - Notional Expansion of System Boundaries in SEDevOps.....	54
Figure 13 - System Adaptation & Evolution via the SEDevOps Life Cycle Model	63
Figure 14 - MBSO Ontology Represented in SysML Profile Diagram.....	65
Figure 15 - MBSE and MBSO Focus in the SEDevOps Life Cycle Model	70
Figure 16 - View of MBSO Profile Organization	72
Figure 17 - MBSO Operational Element Stereotype Details in SysML Profile Diagram	73
Figure 18 - MBSO Operational Element Stereotype Applied to Physical Architecture in a SysML bdd (adapted from Friedenthal 2017)	74
Figure 19 - Applying Operational Status Attributes in a SysML ibd (adapted from Friedenthal 2017)	75

Figure 20 - FireSat-II Model Organization prior to MBSO Integration (Friedenthal 2017).....	82
Figure 21 - FireSat-II Spacecraft (Friedenthal 2017)	82
Figure 22 - FireSat-II Spacecraft Physical Decomposition, 2nd Level (Friedenthal 2017).....	83
Figure 23 - FireSat-II Spacecraft Subsystem Interconnection (Friedenthal 2017).....	84
Figure 24 - MBSO Operational Element Stereotype Applied to Existing Physical Architecture in SysML bdd and ibd (adapted from Friedenthal 2017)	86
Figure 25 - Notional FireSat-II Command Database created with MBSO Profile Elements, in a Customized SysML Table Diagram	88
Figure 26 - Notional FireSat-II Telemetry Database created with MBSO Profile Elements, in a Customized SysML Table Diagram	89
Figure 27 - FireSat-II Notional Command & Telemetry Element Interrelation in a SysML Dependency Matrix	90
Figure 28 - Command & Telemetry Interrelationship to Inertial Measurement Unit in SysML Block Definition Diagram	91
Figure 29 - Command & Telemetry Interrelationship to High Rate (HR) Transmit Amplifier Unit in SysML Block Definition Diagram	91
Figure 30 - FireSat-II Model Organization Following MBSO Integration (adapted from Friedenthal 2017).....	92
Figure 31 - SysML Activity Diagram representing the Flow of Actions for Command Procedure to Reconfigure Power Loads for Failed Converter Unit	93

Figure 32 - Customized SysML Table Diagram representing the Command Procedure
to Reconfigure Power Loads for Failed Converter Unit..... 94

Figure 33 - FireSat-II Procedural Step Dependency Matrix in SysML..... 95

List of Tables

Table 1 - Life Cycle Model Comparison	34
Table 2 - SEDevOps Characteristics	56
Table 3 - Customized Elements in the MBSO Ontology	67
Table 4 - Research Questions Answered	104

List of Acronyms

bdd	Block Definition Diagram (SysML)
CONOPS	Concept of Operations
DE	Digital Engineering
DevOps	Development & Operations
DITL	Day In The Life
DoD	Department of Defense
DSL	Domain-Specific Language
ibd	Internal Block Diagram (SysML)
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IID	Incremental and Iterative Development
INCOSE	International Council on Systems Engineering
ISO	International Organization for Standardization
LCC	Life Cycle Cost
LIAD	Life In A Day
M&S	Modeling & Simulation
MBE	Model-Based Engineering
MBSE	Model-Based Systems Engineering
MBSO	Model-Based System Operations
NASA	National Aeronautics and Space Administration
NDIA	National Defense Industrial Association

SEBoK	Systems Engineering Book of Knowledge
SEDevOps	Systems Engineering Development & Operations
SLC	Signature, Limitation, & Constraint
SOI	System of Interest
SW	Software
SysML	Systems Modeling Language
UML	Unified Modeling Language

Chapter 1: Introduction

1.1 A Brief Taxonomy

In order to properly frame the concepts and applications in this body of work, it is imperative to describe in a brief taxonomy the specific connotation of foundational terms introduced in the title and abstract and used throughout this manuscript.

The term *polymorph* can be traced back to Greek roots and quite literally means “of many (poly) forms (morphé)” (Dictionary.com 2020). *Polymorphic* therefore describes something occurring in many different forms and is traditionally applied in a biological sense to entities which have *evolved* into multiple simultaneously occurring variants. The software engineering realm has adopted the concept of polymorphic functions applied in a way as to take on different variable types and behaviors at different times. The application of this in the software domain is to establish the ability “to satisfy dynamic reconfiguration, plug-n-play, extensibility, and system redundancy requirements” (Bryson 2010). Common to polymorphic elements, and necessary to make polymorphic software functions work, is the existence of a single interface capable of representing all variants taken on by that entity (Bryson 2010).

As described in detail in Chapters 2 & 3, systems engineering, and therefore Model-Based Systems Engineering (MBSE), is a discipline required to take on many forms throughout the life cycle of a system in order to ensure system success. What is currently missing from the intrinsic elements of MBSE is 1) this concept of a common MBSE interface throughout a full system life cycle, and 2) the implementation of MBSE in different forms at *each* stage of a system’s existence, including at times simultaneously in

different stages such as in system operations and system development (i.e. supporting and performing continual system development while in the system operations stage).

Another homage to Greek roots is the use of the term *lemniscate* and its application here to engineering methodology, introduced and applied in the concepts to follow. In Greek, lemniscate can be translated as “laying of ribbon” and has, over many years, taken on a variety of forms most typically in a descriptive mathematical sense, including the infinity symbol (Erickson 2011). What is of most significance to the concepts introduced herein is the notion and importance of a continuum as applied to the stages of a system’s life cycle, discussed in greater detail and context in Section 3.2.

According to ISO/IEC/IEEE Standard 15288, typical generic system *life cycle stages* are broken out to concept, development, production, utilization, support, and retirement. For the purposes of life cycle discussions throughout this manuscript, the stages of concept, development, and production are grouped into the generalized “development phase,” or stage, while utilization, support, and retirement are grouped into the generalized “operations phase,” or stage. This distinction is discussed during the literature review on life cycles in Section 2.4 as well as throughout the concepts introduced in Sections 3.2 and 3.3.

Digital Engineering is defined “as an integrated digital approach that uses authoritative sources of system data and models as a continuum across disciplines to support life cycle activities from concept through disposal” (U.S. Department of Defense 2018). Throughout this manuscript, Digital Engineering is considered the discipline charged with advancing digital toolchains, ecosystems, and methods. **Digital toolchains** are considered throughout as the combination of several interrelated tools that facilitate data management,

visualization, development, and use. An example of a common digital toolchain is the Microsoft Office suite of applications which are capable of interrelating and broadly using data created and managed in any of the applications. A *digital ecosystem* is the full environment in which data resides throughout the life of that data. Using the Microsoft Office example again, the ecosystem would then include the machines and operating systems on which the applications are used, the data storage mechanisms (i.e. servers or drives), the network on which the data is accessed and maintained, etc.

A *descriptive model* as used throughout this work refers to an object-oriented model of elements with detailed, interrelated metadata properties. The Systems Modeling Language (SysML) is a common language for building descriptive models in the systems engineering domain.

Agility, used herein, can be defined as “the capability to successfully cope with changes in circumstances” (Alberts 2011). This is one of the foundational elements of the work to follow and is used throughout in the context as defined by Alberts.

The term *framework* is generally used to describe a basic structure on which additional details, uses, and systems can be built. It is used in the context of a modeling framework in this text on which to build cohesive, detailed, and coordinated descriptive models according to inherent guidance within the modeling profile.

While a concise and widely agreed upon definition of *cyber-physical system* does not yet exist (SEBoK Editorial Board 2020), the use in this manuscript is in the context of a system in which hardware is integrated with computing platforms to control the physical processes of the system. Cyber-physical systems therefore entail more than purely software applications.

Lastly, the concept of *Model-Based* suggests the creation and curation of a surrogate, or model, on which to closely base elements of reality. The specific application of the term throughout this manuscript denotes a digital version of a model. In concert with the application of life cycle stages across a lemniscate arrangement, a digital model-based existence enables a common interface for MBSE implementation and interaction throughout the life of any system. In other words, a common modeling language and source-of-truth data repository through which to develop, modify, and retrieve data products throughout the life of a system.

With the taxonomy introduced, it is now possible to dive into foundational elements of the problem addressed in the ensuing manuscript.

1.2 Problem Identification & Description

Since MBSE's 2007 inception, it has rarely been applied beyond a system's development stage resulting in lost opportunities for improved agility and cost reductions during the utilization, support, and retirement stages.

Systems Engineering is broadly acknowledged as a full life cycle discipline and the application of MBSE is therefore intended to support the entire life cycle (INCOSE 2015). Despite this, MBSE has primarily been applied to the *design of operations* and seldom leveraged to actively support change, improve agility, and reduce resultant costs *during the system operations and sustainment stages*, which historically account for >50% of overall life cycle costs (Madni and Sievers 2018), (INCOSE 2015). As noted by traditional technical authorities as well as progressive practitioners in the discipline, systems engineering is in the midst of a transformation to a more digital and agile discipline (Dove and LaBarge 2014), as evidenced by the continuing trend towards MBSE implementation

(Madni and Sievers 2018). However, as a full life cycle discipline, there is an opportunity for stronger focus on applications of MBSE practices and products during the operations and sustainment stages of a system's life cycle (INCOSE 2014), (U.S. Department of Defense 2018).

The INCOSE Systems Engineering Vision 2025 articulates the need for systems engineering evolution during later life cycle stages, stating “the systems engineering discipline will expand its applicability and recognition along several fronts” including “increased emphasis on downstream life cycle phases such as sustainment” (INCOSE 2014). Additionally, the 2018 Digital Engineering Strategy by the U.S. Department of Defense identified a focus of its Goal #1 on “the formalized application of modeling to support all the system life cycle phases from concept through disposal” (U.S. Department of Defense 2018).

The research provided in this manuscript addresses methods for increasing the MBSE emphasis during downstream life cycle stages by revisiting the generic life cycle model as defined by ISO 15288 to determine if a variation improves the emphasis and utility of a model-based support framework as a means to improve agility and reduce costs during system operations.

1.2.1 Background on Systems Engineering Evolution

Systems engineering is an evolutionary practice, evolving at the macro level as a discipline by necessity to support and enable the ever-growing complexity of systems, and evolving at the micro level throughout any particular system life cycle to continually unify and drive system success (Dove and LaBarge 2014). As noted in Section 1.2, recent systems engineering visionary documents, including the International Council On Systems

Engineering's (INCOSE's) *Systems Engineering Vision 2025* (INCOSE 2014), and the U.S. Department of Defense's *Digital Engineering Strategy* (U.S. Department of Defense 2018), place an increasing emphasis on agility (the ability to handle change) with a clear “call-to-action” to push the boundaries of the systems engineering discipline across the entire life cycle of systems. This, coupled with continual and accelerating advancements in Digital Engineering toolchains, has prompted systems engineers to explore evolving practices and processes to enable a digital life cycle thread (an interrelated toolchain for managing systems engineering process and products throughout the life of a system) and to develop the requisite systems engineering specific tools, processes, and methodologies to promote this digital transformation. To date, and as articulated through the detailed literature review throughout Chapter 2, this transformation has been concentrated most notably on the development stages of the system life cycle (as defined in the taxonomy presented in Section 1.1). This has enabled significant improvements in the areas of system design and test, poised the discipline for further improvements to model-based approaches during system operations and sustainment.

1.2.2 Full Life Cycle Need for Model-Based Systems Engineering

Despite the acknowledgement of application within systems engineering doctrine on the *full life cycle*, details and guidance on MBSE-specific products, processes, and practices, beyond the development stages of a system are limited (Madni and Sievers 2018). As noted by Blanchard and Blyer in *Systems Engineering Management*, “Although a great deal of emphasis has been placed on minimizing the costs associated with the procurement and acquisition of systems, little attention has been paid to the costs of system operation and support” (Blanchard and Blyer 2016). Along those lines, visual

representations of current widely accepted life cycle models such as the Systems Engineering Vee, Waterfall, Spiral, and even the more recent Agile model generally conclude with an identified hand-off to system operations and sustainment without detailed representation of the stages beyond (Douglass 2016). Section 2.4 investigates this in more detail, including a review of the general strengths and shortcomings with respect to systems engineering applications across a variety of widely accepted and broadly practiced life cycle models.

Given that “the utilization and support stages of a system usually account for the largest portion of the total LCC” (life cycle cost) (INCOSE 2015), why has the focus on a model-based transformation within the discipline not been extended to support, enhance, and streamline the “largest portion” of the total life cycle cost? One potential answer, per NASA’s “Systems Engineering Engine,” is that once in the operational stage, any upgrade or capability development invokes a restart of earlier life cycle stages to leverage systems engineering development processes and products (NASA 2017). Whereas this may be a logical approach, it assumes the structure, personnel, and most importantly contract vehicles are in place for development teams leveraging the necessary processes and products to support operational systems. In reality, this restart often reveals a disconnect within the organization exemplified by a lack of available resources including lack of personnel, out-of-date systems engineering products and lack of funding and prioritization for systems engineering team involvement during operational stages.

In addition to the noted organizational challenges, how should procedural and documentation updates in response to faults and failures during system operations be handled, which are neither system upgrades nor new capabilities? According to the *NASA*

Systems Engineering Handbook, systems engineering efforts, resources, and schedules during operations are typically constrained, resulting in system operations and maintenance teams assuming the responsibility for more traditional systems engineering tasks and management of associated system level products (NASA 2017). This results in increased risk during system operations due to complexities associated with necessary adaptation and ad hoc development, many times performed by non-development and non-systems-engineering-versed personnel, to sustain a system in dynamic environments and in response to faults and failures. Therefore, systems operations and maintenance teams are many times left without the complete tools and processes to adequately, robustly, and quickly address the continual adaptation and improvement required during the operational life of a system resulting in lost opportunities for agility and increased costs to sustain a system.

1.2.3 A Related Problem-Space – Lessons from Software Engineering

Similar disconnects and discrete hand-offs from development teams to sustainment teams have been encountered in the software domain. In recent years, the delineation between development and operations teams has been steadily dissolving and the environments and tools with which they work have been increasingly merging in order to handle the growing need for agility, responsiveness, and faster-to-market capabilities. The widely growing practice of DevOps is enabling rapid evolution in the software domain and emphasizes continuous development, integration, and delivery. Additionally, through very closely linking the software development environment with the operational software environment, a realistic and continuously representative platform is established for maintaining, updating, testing, and incrementally rolling-out code. The result is companies

such as Amazon who are fully entrenched in the DevOps process, are deploying snippets of code to their operational systems every 11.7 seconds, on average (Null 2020). Sections 3.2 and 3.3 explore adapting the DevOps approach of integrating and streamlining development capabilities, products, processes, and tools with a model-based systems engineering construct for operational systems resulting in a proposed framework to support maintenance and modification of procedures and scripts seamlessly during system operations.

1.3 Thesis Statement

Based on the identified need for MBSE implementation to evolve and take on additional forms at later stages in a system life cycle, a modified life cycle model is both necessary and readily possible. This modified life cycle must focus on promoting agility by enabling a continuous and multi-form, or polymorphic, existence of systems engineering through MBSE. Polymorphism, in this sense, emphasizes the need to support continual system development and modification during system operations. Leveraging methodological advancements and applications in the software engineering domain, this approach can be applied to the systems engineering domain with confidence of success through the use of tools designed for collaboration and digital connectivity.

A model-based approach to systems engineering process and product management throughout a system life cycle cultivates the strengths of MBSE achieved to date in the system design and development stages. Expanding this approach into a dynamic and extensible framework for all life cycle stages, including the operations and sustainment stages of complex systems, improves agility by providing a common digital ecosystem and, in fact, a multi-faceted interface fabric for polymorphic systems engineering.

1.4 Research Questions & Objectives

1.4.1 Research Questions

In order to demonstrate the assertion that a continuous life cycle model built on a model-based product core will provide the methodology and framework for evolving systems engineering into a polymorphic and agile discipline throughout an entire system life cycle, the following research questions were derived and drove the investigation that follows.

1. Can the generic systems engineering life cycle model be expanded to promote the same rigorous, centralized and interrelated model-based approach leveraged in system development into and throughout system operations and sustainment?
2. Can the MBSE methodology and associated toolchains be adapted to the operational stages of a system's life cycle?
3. Can this adaptation of MBSE provide an adequate framework for enabling continuous model-based system development during the operational stages of a system thereby improving efficiency, agility, and operational availability over traditional system engineering practices and methods?

1.4.2 Objectives

Out of these questions, the following two research objectives were formulated and served as guiding concepts through literature reviews, methodology and framework development, and data collection and analysis.

1. Develop a modified life cycle model inclusive of 1) the traditional development stages and supportive of the existing approaches to systems

engineering implementation of these stages as well as 2) the operations stages in such a way as to promote agility, adaptability, and polymorphism through multiple directed paths and interactions between those stages.

2. Develop a model-based framework for systems engineers to leverage for developing the products and implementing the process of this modified life cycle model within a digital ecosystem.

1.5 Hypothesis

The hypothesis for the research to follow is that defining a modified life cycle model and associated methodology along with a new framework for hosting and dynamically linking descriptive, model-based system elements from all stages, including most notably the operational support elements (i.e. operational scripts/procedures/functions) will enable greater agility and reduced life cycle costs compared to current methods. The prior statement is focused on the systems engineering functions executed during system operations, including adaptation to environmental evolution and system aging.

A byproduct of normalizing the modeling methodology and tools across all life cycle stages, inclusive of system development and system operations, is the bidirectionality of information flow via a common modeling environment to improve execution at any life cycle stage. In other words, system development models can be directly linked to operational support products in descriptive model-space and modifications to system elements in operations can be tied back to model-based development products to inform future system adaptations and evolutions, as required. This is explained in more detail in Section 3.2.6.

1.6 Proposed Solution to Enable Polymorphic Systems Engineering: The SEDevOps Lemniscate & MBSO

The concepts introduced in Chapter 3 and demonstrated in Chapter 4 build on the basis of traditional system life cycle models by incorporating specific elements from more recent models and draw from current advancements in the closely related field of software engineering to propose a variation on the view of the system life cycle from a primarily MBSE perspective. This is based on principles employed in the DevOps mindset: a collaborative merger of development and operational tools, processes, products, and practices.

The resulting contributions to the body of knowledge by this work are:

1. A modified systems engineering life cycle model and associated methodology focused on improving agility throughout an entire life cycle by formally interconnecting model-based elements from system development with newly introduced model-based elements from operations and sustainment, coined Systems Engineering DevOps (SEDevOps). (Addressing Objective #1)
2. A domain-specific modeling language extending the application of MBSE to system operations, enabling the realization of a full-duplex digital thread from development through operations and back, labeled Model-Based System Operations (MBSO). (Addressing Objective #2)

These products are the result of a wide survey of recent advancements in systems engineering and applications in software engineering, described in detail in Sections 2.2 and 2.3.

1.7 Significance, Findings, and Conclusions

As demonstrated through the literature review, the need for MBSE evolution continues to expand. Systems continue to grow in complexity, almost entropically, timelines continue to be compressed, and performance continues to be driven to the maximum as technological advancements provide both evolutionary and revolutionary catalysts. The exploration of methodological applications beyond traditional systems engineering provides a plethora of proving grounds for process and toolchain advancements. Combining proven advancements from other fields into the systems engineering discipline and, more specifically the application of MBSE as described through the concepts herein, drives the adaptability of the methods and processes onward enabling systems engineers to advance their focus on solving system problems of escalating complexity going forward.

Along those lines, the benefits of the proposed methodology are demonstrated through a use case extending MBSE tools and products for use in the operational stages of a system to facilitate an improved full life cycle approach to systems engineering through MBSE. Resultant findings corroborate the hypothesis that a fully interrelated continuum, promoting continual development and test during operations through an associated model-based toolchain improves system awareness, responsiveness and adaptability to events encountered during system operations.

Through the life cycle model proposed and demonstrated by implementing the associated MBSO framework, systems engineering is empowered to leverage a common digital ecosystem and model-based products and processes in multiple states during both sequential and simultaneous life cycle stages, therefore moving the systems engineering discipline one step closer towards polymorphic functionality.

1.8 Summary of Dissertation

The remainder of this manuscript is organized as follows:

Chapter 2 starts with the evolution of this particular topic as it relates to existing literature, describing the desired outcome at the onset followed by preliminary findings which resulted in the advancements presented in Chapter 3. Chapter 2 goes on to provide a detailed review of systems engineering and system life cycles, the call to action for enhancements to digital and model-based systems engineering, particularly during the latter stages of a system's life cycle, and an introduction to relevant methodological advancements in the software engineering domain which serve as the foundation for the concepts to follow.

Chapter 3 presents an evolution on the generic systems engineering life cycle model through applying principles employed in the software domain's DevOps practice, termed SEDevOps. This is accompanied by an introduction to an ontology and a model-based framework enabling the implementation of this methodology as an extension to MBSE, coined MBSO.

Chapter 4 describes an instantiation of SEDevOps leveraging the descriptive modeling elements of MBSO. By merging the system operations element environment, composed of operational support products (i.e. command procedures, configurations, etc.), with a system model for the notional FireSat-II spacecraft, the applicability and benefits from a systems engineering perspective are demonstrated.

Finally, Chapter 5 concludes with an explanation of the advancements presented in this work, a description of areas of opportunity for continued and future work, including an

extrapolation to possible advancements, and a discussion of the benefits as well as potential challenges and drawbacks of SEDevOps and MBSO.

Chapter 2: Literature Review

The initial question sparking the exploration into MBSE applications throughout life cycles and model-based instantiation of general systems engineering doctrine was:

Can systems engineering support specific to operations and maintenance/sustainment be codified in a manner to enable autonomic system operations?

In other words, can nominal and, more importantly, off-nominal system operations be performed with not only the systems engineers fully out-of-the-loop (i.e. not involved), but also the system operations and maintenance team out-of-the-loop as well; can a system truly self-adapt? As a note, the use of roles here for systems engineers versus operations and maintenance personnel is derived from the Logistics & Operations (LO) role introduced in Sarah Sheard's Twelve Systems Engineering Roles (Sheard 1996). A tangible example of the above application is deep space exploration (Truszkowski, et al. 2009). Can a space system hundreds of millions of miles away and dozens of minutes delayed in communications perform self-sustained agile system operations in any scenario, including off-nominal, failure circumstances? Today, systems of this nature are designed to respond with pre-programmed if/then responses to isolate off-nominal equipment, without true self-awareness or prognostic behavior in order to configure to a "safe" state while awaiting further investigation and direction from earth-bound systems engineers. In an attempt to leverage the strength in response to change that MBSE brings during system design and development stages (Delligatti 2014), the driving motivation to the above concept was to codify the process to make necessary changes to operational support products during system operations through a model-based toolchain implemented during the system operations and sustainment stages.

The concept proposed is rather straightforward but relies on a foundational assumption that model-based processes and products leveraged by systems engineers and operations and maintenance personnel during the system operations stage already exist, are equally as robust as the model-based processes leveraged during the design, development, and deployment stages, and are in a common language to prior life cycle stages.

What was quickly found in practice through a deep dive into literature is that a clear implementation of MBSE methods, tools, and processes focused on active use during the operations and sustainment stages of a system life cycle does not yet exist. As will be presented in detail in this chapter, widely accepted life cycle models in systems engineering doctrine, on which current MBSE languages and tools are based, focus heavily on the early stages of a system's life cycle and present an opportunity to expand focus to model-based implementations in later life cycle stages. In this deep dive, it was also found that advancements in systems engineering applications leveraging Digital Engineering innovations are increasing rapidly for the *design of, and preparation for* system operations, however continued use and support *during operations* was not readily discovered. This addresses the case within the Logistics and Operations role noted by S. Sheard where systems engineers can serve as operations and maintenance personnel (Sheard 1996) and can ideally provide more rapid response to necessary adaptation during system operations with the proper tools and products available.

Therefore, in order to address the initial question regarding codifying and automating the adaptability required during system operations, a wider net was cast to broaden the review on systems engineering methodologies, processes, and tools. The resulting theme

articulated in the ensuing sections shows that practitioners and academics alike consistently identify two key areas of need and opportunity:

- 1) MBSE advancement to later life cycle stages
- 2) Increased agility through a digital transformation for systems engineering as a discipline

This chapter explores foundational elements of both of these topics in detail and concludes with a summary and reiteration of the call-to-action to enhance systems engineering on these two fronts (INCOSE 2014), (U.S. Department of Defense 2018). Based on these findings, before autonomic applications can be developed and deployed for complex system operations, appropriate MBSE methodologies and tools must be introduced to enable this evolution. The overlap in a Venn diagram of the two areas of opportunity identified above, systems engineering and aspects of Digital Engineering, is where Chapter 3 focuses.

2.1 Literature Review Map

As identified, the two primary areas of focus in the ensuing review of relevant work are Systems Engineering and Digital Transformation supported through advancements in applications within the broader Digital Engineering domain. What was quickly found is the reliance on and in fact coexistence of these two topics with Software Engineering (and software itself) and vice versa. Figure 1 depicts a notional representation of the major discipline relationships in blue and introduces nuanced elements of the deeper literature review performed across additional fields, in white. As can be noted in the figure, there is considerable overlap in topics and disciplines, therefore elements of each sub-topic may appear across discussions of each discipline.

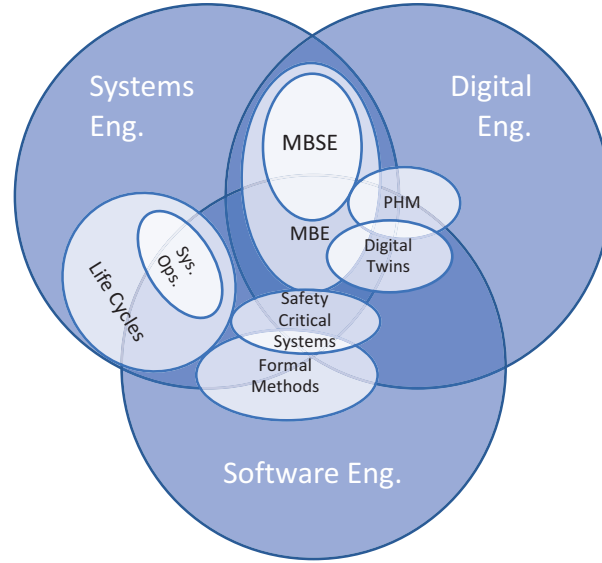


Figure 1 - Literature Map

2.1.1 Primary Disciplines

As noted in Section 1.2, recent calls to action for evolving the systems engineering domain are focused on broadening model-based systems engineering applications across system life cycles and enhancing the digital ecosystem leveraged by systems engineers. Coupling this with the topic on autonomy introduced at the start of this chapter, the literature review focuses on topics encompassed by the primary disciplines of Systems Engineering and Digital Engineering. Closely related and overlapping these two areas, the quickly advancing discipline of Software Engineering is also a primary topic area, including methods, tools, and processes.

2.1.2 Secondary & Intersecting Disciplines

The three primary disciplines have many shared and overlapping sub-topics and sub-disciplines. These include prognostics and health management (PHM), digital twins,

applications of formal methods for streamlined verification purposes, life cycle management, and many more. The noteworthy areas considered in this review are identified as smaller elements on the literature map in Figure 1 and are discussed throughout this chapter.

2.1.3 Method of Literature Presentation

Findings from the detailed literature review are presented by primary discipline. Secondary topics are grouped within the most closely correlated primary discipline or, where fully shared by all three, documented in the Cross-Cutting subsection (Section 2.6). For example, PHM is a topic employed in both systems engineering and software engineering applications however it is primarily enabled by Digital Engineering tools, therefore it is presented as part of the Digital Engineering review.

System life cycles are introduced briefly in both systems and software engineering for reasons that will be apparent as the content is presented and is therefore covered in more depth in a dedicated subsection. System life cycle models have deep roots in systems engineering, while in more recent years, have been informed and evolved through advancements in software engineering applications.

2.2 Systems Engineering

2.2.1 Systems Engineering Definition

Systems Engineering is a widely practiced engineering discipline bridging all facets of realizing successful systems from conception to decommission and is defined as such throughout technical literature, (ISO 2015), (INCOSE 2015), (NASA 2017), (Blanchard and Blyer 2016), (Blanchard and Fabrycky 2006), (Douglass 2016). Perhaps most

concisely articulated in ISO/IEC/IEEE Standard 24765-2017, Systems Engineering is an “interdisciplinary approach governing the total technical and managerial effort required to transform a set of stakeholder needs, expectations, and constraints into a solution, and to support that solution throughout its life” (ISO 2017). The INCOSE Fellows recently modified the definition of Systems Engineering to “a transdisciplinary and integrative approach to enable the successful realization, use, and retirement of engineered systems, using systems principles and concepts, and scientific, technological, and management methods” (SEBoK Editorial Board 2020). Common to these definitions is the open acknowledgement of “integrative” and “total” support *throughout the life of a system*. With the advent of MBSE and the growing application, it is reasonable to expect the integrative and full life cycle application of MBSE.

Despite this, current widely accepted implementations of MBSE tools, processes, and products focus most notably on the use in specification, design, development, and testing of systems and have less focus on standardized products, processes, and general use beyond the deployment and transition to operations of a system. Furthermore, specifics on the continuum of model-based products from the development stages to the operational stages is, in most cases, not specified and is an area of opportunity for evolving the application of MBSE throughout the full life cycle.

Dove and LaBarge provide a slight alternative on the definition of systems engineering with a focus on the cross-discipline problem solving and adaptability required for complex systems, including the often-experienced emergent behavior of such complex systems. Dove and LaBarge go on to identify that systems engineering is continually evolving and work is in progress to enable adaptive and agile systems engineering, stating “we should,

as practitioners and as researchers, identify and define design and operational guidance for adaptive system engineering processes” (Dove and LaBarge 2014). Given that MBSE is considered the application of modeling to support systems engineering implementation throughout the life of a system (SEBoK Editorial Board 2020), it is logical to conclude that MBSE should support and promote adaptability and agility in systems engineering implementation.

To begin to frame the relevancy of system life cycle methodologies to systems engineering implementation and, consequently, MBSE implementation, INCOSE provides a useful description of systems engineering tasks and responsibilities:

“Systems engineering (SE) tasks are usually concentrated at the beginning of the life cycle, but both industry and government organizations recognize the need for SE throughout the systems’ life span, often to modify or change a system product or service after it enters production or is placed in operation. Consequently, SE is an important part of all life cycle stages. During the utilization and support stages, for example, SE executes performance analysis, interface monitoring, failure analysis, logistics analysis, tracking, management, etc. that is essential to ongoing operation and support of the system.” (INCOSE 2015)

With that in mind, the more progressive methodologies of applied Systems Engineering provide powerful, integrated, Digital Engineering and model-centric tools and modeling languages to support systems engineering activities, yet they too are generally limited in scope and applicability to use in defining, developing, and deploying systems (Douglass 2016), (Delligatti 2014), (Friedenthal, Moore and Steiner 2014). This dichotomy is gaining attention and while currently established implementations of MBSE are focused on early life cycle stages, the *Systems Engineering Vision 2025* identifies later life cycle stages as a key focus area in the near term (INCOSE 2014). Similarly, the United States Department

of Defense Digital Engineering Strategy identified a focus of its Goal #1 on “the formalized application of modeling to support all the system life cycle phases from concept through disposal” (U.S. Department of Defense 2018). These “call-to-action” statements, introduced in Section 1.2 and provided in context here, form the basis for much of the additional research and concepts to follow.

Germane to the examination of systems engineering as a full life cycle discipline is a review of the accepted definitions and examples of system life cycle methods, provided in depth in Section 2.4, after exploring relevant advancements in the Software Engineering field.

2.2.2 Recent Developments in Systems Engineering

The systems engineering discipline is advancing on multiple fronts, several of which are relevant to the advancements presented in this work.

2.2.2.1 Model-Based Systems Engineering

MBSE methodologies and supported descriptive and parametric modeling languages and tools facilitate the development of system models and owe much to the software development domain which pioneered the use of unified model-based languages in the 1990’s with the advent of the Unified Modeling Language (UML) (Cloutier, et al. 2015). UML enables not only the representation of modularized pseudocode, it facilitates fully integrated elements through the definition of intricate metamodels, allowing for complex and codified relationships among model elements. Systems engineers recognized the value in representing, and more importantly relating system elements in this manner and adopted UML as the basis for a modeling language, coined the System Modeling Language (SysML) (Cloutier, et al. 2015). SysML empowers complex system architecting, design,

development, and testing through fully integrated elemental modeling concepts previously pioneered in the software domain. According to Friedenthal and Oster:

"SysML can be used to describe the following:

1. The system breakdown as a hierarchy of subsystems and components
2. The interconnection between systems, subsystems, and components
3. The behavior of the system and its components in terms of the actions they perform, and their inputs, outputs, and control flows
4. The behavior of the system in terms of a sequence of message exchanges between its parts
5. The behavior of the system and its components in terms of their states and transitions
6. The properties of the system and its components, and the parametric relationships between them
7. The text-based requirements which specify the mission, system, and components and their traceability relationships to other requirements, design, analysis, and verification"

(Friedenthal and Oster 2017)

Among the many advantages of MBSE using SysML is the ability to capture current, real-time views into the underlying model to provide a state of the system from any number of unique viewpoints, each catered to any particular stakeholder. A limitation of SysML, however, is the language and associated tools are focused almost exclusively on the pre-operational aspects of systems and therefore have limited utility and are seldom employed for continual use throughout the operational stages of a system's life. A deep dive into SysML can be found in a variety of instructional texts, including *SysML Distilled* (Delligatti 2014), and *A Practical Guide to SysML* (Friedenthal, Moore and Steiner 2014).

SysML has become an enabler for digital representations of complex systems and, based

on the stated opportunity to improve the MBSE application for operational aspects of systems, SysML is extended in Section 3.3 to facilitate continued use in later life cycle stages.

Leveraging SysML, there are a multitude of MBSE methodologies employed for performing predefined systems engineering functions throughout system design and development (Ramos, Ferreira and Barcelo 2012). Common to all of the MBSE methodologies is the theme of layered decomposition in problem space followed by coordinated composition in solution space in order to perform the requisite systems engineering tasks throughout system design, development, and test. A key feature noted by Delligatti is that MBSE is most valuable when change happens (Delligatti 2014). The interrelatedness of elements within a descriptive model enables rapid impact identification and change propagation, two concepts leveraged heavily in Section 3.3 with the proposed expansion to a model-based system operations construct.

2.2.2.2 Agile Systems Engineering

Based closely on the advancements in the Software Engineering domain and the development of configurable and extensible tools, Agile Systems Engineering is gaining attention and attraction. The Harmony method, introduced by Douglass, is an instantiation of Agile System Engineering using MBSE tools and features to improve the development of systems (Douglass 2016). Additionally, the term “agile” and associated concepts continually appear in forward-leaning texts encouraging systems engineers to improve the design flexibility and the accuracy with which they perform their roles (Dove and LaBarge 2014).

2.3 Software Engineering

Software systems can be considered “just another system” in the broader sense of the definition. The nuance with software systems is the highly configurable nature of software as compared to hardware. It is based on this feature that software engineering advancements at times appear to outpace general systems engineering advancements. Therefore, it is worth a closer look at recent advancements that may positively influence the way in which systems engineers perform their roles and responsibilities.

2.3.1 Software Engineering - Why Systems Engineers Care

Software engineering is a constantly evolving and closely correlated discipline to systems engineering, continually advancing methodologies and requisite tools to support and enable the rapid advancement in software development. According to the Systems Engineering Body of Knowledge (SEBoK), “software is a flexible and malleable medium which facilitates iterative analysis, design, construction, verification, and validation to a greater degree than is usually possible for the purely physical components of a system” (SEBoK Editorial Board 2020). From this, systems engineering methodologies continue to be informed of tactics and techniques for advancement based on observed successes.

A recent key example of this is the advent of Agile Systems Engineering noted previously, based on the successes of Agile Software Development (Douglass 2016). As stated by Dove and LaBarge, “the growing acceptance and adaptation of agile software development methods has passed the tipping point in the software world, and is now motivating expectations in broader domain-independent systems engineering” (Dove and LaBarge 2014).

Therefore, any thorough survey of systems engineering capabilities and advancements warrants a further look into software engineering practices and processes.

2.3.2 Recent Advancements in Software Engineering

Modularized software development has become mainstream and associated development methodologies, processes, and tools have adapted. Agile methods were developed specifically on the basis of modularity and focus on continuous development (Beedle, et al. 2001). In order to not only continuously identify and develop code but to quickly deploy it into operational systems, two additional methodological advances were introduced:

- 1) The concept and widespread use of microservices: containerized, seamlessly interchangeable “black box” modules of code to perform isolated, singular tasks of an overall system.
- 2) The introduction of DevOps: the merger of the development and operational environments and engineering teams and, more specifically, the principle of cross-collaboration across both to foster continuous integration, testing, deployment, and monitoring of these smaller microservice elements constituting system enhancements or maintenance patches (Zhu, Bass and Champlin-Scharff 2016).

2.3.2.1 Microservices

The concept of “microservices” was introduced in 2012 as a way to articulate the architectural concept of many small software programs, each performing a single function, working together as a whole to perform a more complex function (Lewis 2012), (Lewis and Fowler 2014). Since that time, the use of microservices has blossomed. Lewis and

Fowler broadened the definition of microservices several years later by stating “In short, the microservice architectural style...is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms...” (Lewis and Fowler 2014). Building on this over recent years, the concept of containerizing has grown in usage to describe the modification or creation of functions into discrete “containers” which can be stacked and leveraged together for more complex functionality.

“Containerizing” and microservices (or microtasks) are not foreign concepts to system engineers and can be equated to clearly defining interfaces and functions of any bounded widget to enable successful late-in-the-flow integration and isolated correction. Driving containerization down to the smallest feasible level is an area of opportunity for systems engineering, particularly in operational support elements and products, such as defining containerized maintenance manual or procedural steps down to singular actions. This enables reuse of a single element (such as a procedure step) across all executable occurrences, alleviating the concern of finding and updating all instances of a particular system command throughout a multitude of maintenance manuals, procedures, and scripts upon a necessary change.

For a simplistic example, establishing a modularized, elemental representation of “power down system” allows reuse of this element in any digital, model-based representation of maintenance manuals requiring power down. If the system is updated for internal battery power vs. supplied ground power, the change in all maintenance manuals is captured in a single location and propagated to all relevant procedures/manuals by pulling content from this single source-of-truth location within an overall system model.

2.3.2.2 DevOps

As previously noted, an additional trend in the software domain is the concept and practice of DevOps. DevOps can be defined as “a set of practices intended to reduce the time between committing a change to a system and the change being placed into normal production, while ensuring high quality” (Bass, Weber and Zhu 2015). Olszewska and Walden apply DevOps to system modeling as a means to couple development, quality assurance, and IT operations for formal model generation and curation, emphasizing the larger infrastructure dictated by the DevOps methodology to enable more effective development, deployment, and operations of software systems (Olszewska and Walden 2015). Combemale and Wimmer introduce a vision of expanding DevOps from pure software development to managing models of cyber-physical systems throughout the continuum of development and operations (Combemale and Wimmer 2019). These works provide a basis for broader systems engineering, and more specifically, MBSE adoption of DevOps principles beyond software only applications, a primary focus of Section 3.2.

Many of the software engineering methodological advancements provide flexibility and adaptability throughout the system development and deployment stages, however DevOps is the first to specifically depict graphically the focus on the operation and sustainment of software in addition to development and deployment, and perhaps more importantly, the continued link between development and operations. This can be seen by a review of broadly accepted life cycle models such as Waterfall, Spiral, Incremental and Iterative Development (IID), and Agile, covered in more detail in Section 2.4 (Dove and LaBarge 2014), (Basili and Larman 2003).

DevOps promotes closely mirroring the operational software environment within a development environment, effectively keeping the two in synch. This practice enables software updates and modifications to be rolled out in small, incremental changes in order to control impact and vulnerability to change, enabling the concepts of Continuous Integration and Continuous Deployment. DevOps also institutes a tight reliance on collaboration between development and operational support personnel, their respective environments, and the relevant support products (Zhu, Bass and Champlin-Scharff 2016), (Bass, Weber and Zhu 2015).

2.3.3 Opportunities for Systems Engineering Advancement through Software Engineering Advancements

The two concepts presented in the prior section, microservices and DevOps, when leveraged together and stacked on the practice of rigorously defined unit-level testing, sparked the movement to continuous integration and continuous automated regression testing, greatly reducing the time-to-operations, and more importantly, the serviceability of any developments (Balalaie, Heydarnoori and Jamshidi 2016). Regression testing can be defined as broadly testing system functionality after a change to ensure there is no “regression” in capability and performance (Kargar and Hanifzade 2018). A key enabler from DevOps is the intentional synching of environments on regular, short intervals resulting in high confidence of successful deployments when implementing frequent small changes versus large overhauls. Merging development and operational environments in DevOps can be equated in the systems domain to accurately representing and actively tracking/managing current configurations of the system of interest and, more importantly in the case presented in this manuscript, the operational support elements required to

operate and sustain the system with the ability to establish automated regression testing. Given the prior adoption of UML into SysML and its use in system modeling and the implementation of agile methods, a logical advancement towards broader use of MBSE throughout a system's life cycle is the application of microservice-like executable building-block modules and DevOps principles to enable continuous testing, maintenance, and deployment of operational support product updates during the operational stages of a system's life cycle.

The SEDevOps adaptation on the DevOps concept of tightly coupling elements of an operational system with surrogates in a development environment is detailed methodologically in Section 3.2 and practically in Section 3.3. SEDevOps enables agility in the operation of the overall system and provides a proper model for enabling systems engineers to be agile in their support to operational systems through the use of common MBSE tools and products throughout the life cycle (Mathieson, Mazzuchi and Sarkani 2020).

Critical to the proper implementation of a model-based digital ecosystem in SEDevOps is the expansion of the system-of-interest boundaries to include operational support elements and products (command procedures, maintenance manuals, scripts, etc.) in the core system model, described in more detail in Section 3.3. Based on the current implementation of MBSE and use of SysML v1.6, these operational support elements are not traditionally incorporated within the boundaries of a system-of-interest descriptive model. The distinction in SEDevOps is the additional focus on the operational stages of the life cycle as part of the overarching model-based life cycle. Therefore, the ability to

permanently incorporate the operational support products into the overall “system” and the representative model is a critical enabler.

2.4 System Life Cycles

With the discussion on systems engineering and MBSE fundamentals as well as software engineering advancements on those fundamentals presented, it is now valuable to review in more detail a variety of widely accepted and practiced system life cycle models.

2.4.1 Life Cycle Definition

A system life cycle can be defined as “a view of a system or proposed system that addresses all phases of its existence to include system conception, design and development, production and/or construction, distribution, operation, maintenance and support, retirement, phase-out and disposal” (Blanchard and Fabrycky 2006). Figure 2 depicts a generic system life cycle from Blanchard & Blyer’s *Systems Engineering Management* text (Blanchard and Blyer 2016).

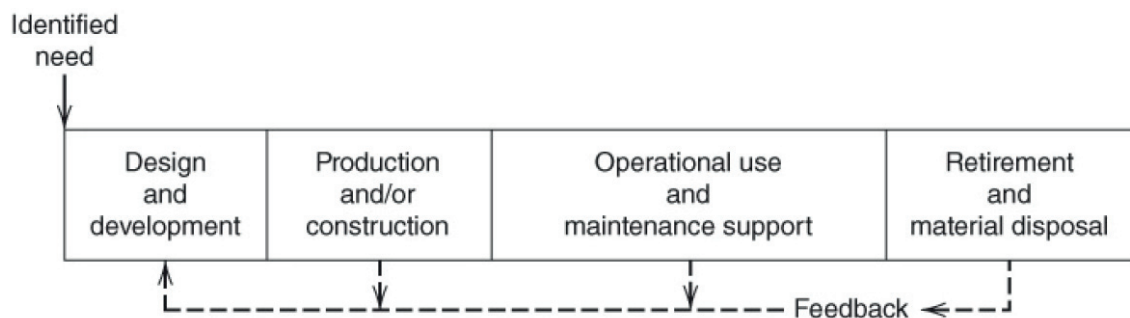


Figure 2 - Generic System Life Cycle (Blanchard and Blyer 2016).

INCOSE defines the key phases, or stages, of a system life cycle as Concept, Development, Production, Utilization, Support, and Retirement, Figure 3, and further defines Systems Engineering as a full life cycle discipline, stating “the role of the systems

engineer encompasses the entire life cycle for the SOI [System of Interest]” (INCOSE 2015). In addition, INCOSE and ISO/IEC/IEEE Standard 24765 (ISO 2017) have defined specific roles and responsibilities of systems engineers in each of these life cycle stages. The following section reviews in more detail various life cycle models and the methods for addressing relevant stages in each.

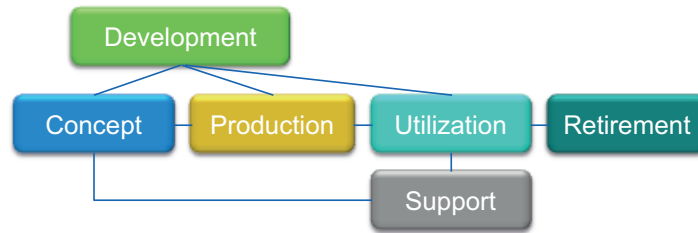








Figure 3 - INCOSE Generic System Life Cycle Stages (adapted from INCOSE 2015).

System life cycle stages can be implemented in a variety of ways incorporating strict sequential plans, iterative and incremental blocks, or frequent function-based releases. Many life cycle models therefore exist to facilitate architecting and executing specific aspects of a system’s life. These include the Vee, Waterfall, Spiral, and Agile, among others. Table 1 describes a variety of widely recognized life cycle methods employed in part or in whole in systems engineering applications (Dove and LaBarge 2014), (Douglass 2016), (ISO 2015), (INCOSE 2015), (SEBoK Editorial Board 2020), (Basili and Larman 2003). Proposed strengths and shortcomings with respect to systems engineering applications are incorporated in the table and described in further detail in the subsections to follow. The life cycle models identified in this table are leveraged in Section 3.2 to develop a representative life cycle model for MBSE implementation throughout the life of a system.

Table 1 - Life Cycle Model Comparison

Life Cycle Category	Method Name	Notional View	Strengths	Shortcomings	Ability to Accommodate		
					System Complexity	Time to Market	Requirement Variability
Sequential & Plan Driven (Pre-specified)	Waterfall		<ul style="list-style-type: none"> Rigorously structured plan High predictability, stability, repeatability High levels of assurance suitable for safety critical systems 	<ul style="list-style-type: none"> Variability poses significant challenges and can balloon costs Infrequent, large gate reviews 	High	Low	Low
	Vee		<ul style="list-style-type: none"> Inherent recursion and layered decomposition of specification Continuous in-process validation enables large scale, complex system development Continuous risk/opportunity assessment 	<ul style="list-style-type: none"> Limited agility and adaptability to change Typically single, large delivery Infrequent, large gate reviews 	High	Low	Low
Evolutionary & Concurrent (Iterative)	IID		<ul style="list-style-type: none"> Early initial capability delivery Iterative development and deployment enables incremental delivery on more frequent centers More frequent gate reviews 	<ul style="list-style-type: none"> Risk of instability due to changing requirements System not fully capable until incremental deliveries are complete Evolution of system may make initial capabilities obsolete Challenges with large, complex systems 	Med	Med	Med
	Spiral		<ul style="list-style-type: none"> Focus on collaborative development and continuous adaptation Near continual onramps for updates and modifications with constant feedback "Time to market" of features 	<ul style="list-style-type: none"> Perception of unstructured approach Potential for inconsistently applied process Implementation challenges for complex systems 	Low	High	High
Interpersonal & Emergent	Agile		<ul style="list-style-type: none"> Focus on collaboration across development and operations teams Key tenets: continuous integration, continuous test, continuous deployment 	<ul style="list-style-type: none"> Unformalized methodology with varying interpretations and applications Generally localized focus rather than global focus 	Low	High	High
	DevOps						

2.4.2 Sequential & Plan-Driven Life Cycle Models

Traditional system development follows a logical, sequential flow in which successive development tasks are performed following completion of prior tasks. The long employed Waterfall model represents the linear flow of completed data and material from one stage to the next. In this model, the development plans are rigorously structured and therefore highly predictable, stable, and repeatable. This becomes a favorable attribute when developing and testing safety critical systems in which variability can pose dynamic and unpredictable impacts. Challenges that arise with a rigid approach of this nature include limited ability to support requirement variability and added scope after initial planning. Systems employing this approach are generally highly complex and potentially system of system implementations requiring well vetted interfaces and bounded constituent capabilities (SEBoK Editorial Board 2020), (INCOSE 2015).

A closely related model developed out of the Waterfall model is the Vee. In this model, decomposition, definition, design, and development occur on the descending left leg of the Vee while integration, test, verification, and validation occur on the ascending right leg. Of note in this model is the correlation of the level of decomposition on the descending leg with the level of integration and test on the ascending leg. In other words, the first step of integration and associated test is at the lowest level, corresponding to the lowest decomposition of design and development. As the program builds/integrates from individual units out to the system level, the requirements verified at each level match the decomposition done on the descending leg. In this way, the Vee is a very logical, sequential implementation with correlation between earlier stages and later stages of development. Figure 4 depicts a traditional Vee model found in numerous sources, including the INCOSE

Systems Engineering Handbook, adapted by Douglass and modified further for incorporation herein to represent the interrelation to and data sharing with products from earlier stages (for integration, verification, and validation) (INCOSE 2015), (Douglass 2016).

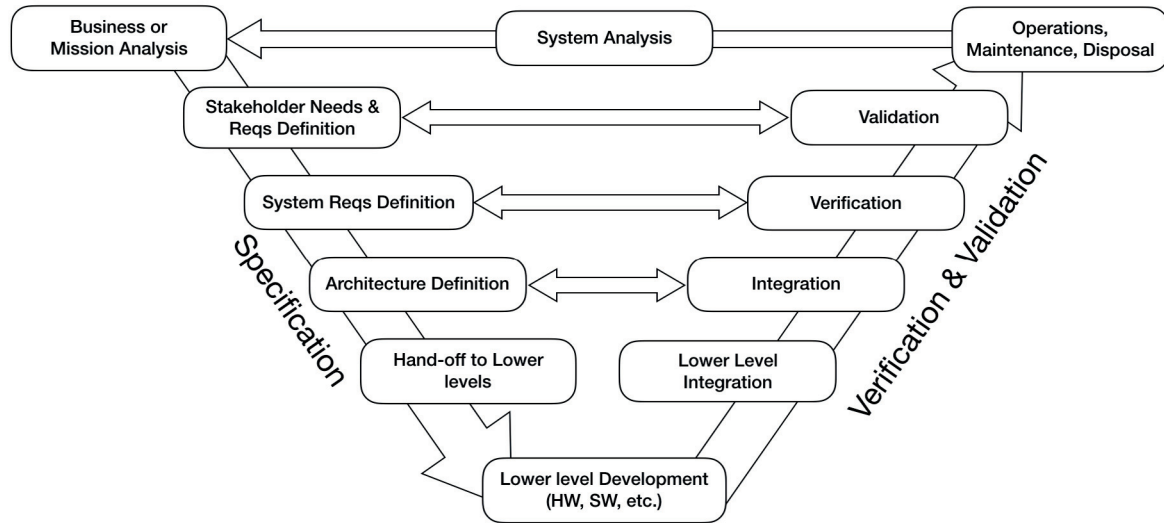


Figure 4 - Vee Life Cycle Model (adapted from Douglass 2016)

Similar to the traditional Waterfall method, the Vee tends to be structured and pre-defined. Advantages include the inherent recursion between levels of decomposition and specification as well as the potential for early validation, at the lowest level of decomposition and subsequent integration. This facilitates continuous risk and opportunity assessment at all levels, supporting complex system development. Due to the pre-planning generally employed with the Vee method, agility and adaptability to change are limited and likely result in broader impacts (SEBoK Editorial Board 2020).

Common to both the Waterfall and Vee are gate reviews and deliveries. For complex systems requiring many parts for initial functionality, this may be an advantage. Otherwise,

for incremental and dynamic systems, methods from the evolutionary & concurrent category may provide better suitability.

2.4.3 Evolutionary & Concurrent Life Cycle Models

Software systems have pushed methodologies towards incremental development and deployment in which capability is developed to an initial, minimum functionality and deployed while more complex functionality is then developed, tested, and ultimately deployed on top of the prior increment(s). In this category of life cycle methodologies, functionality is strategically built in phases, typically with gated milestones associated with each discrete installment of complexity to validate foundational functionality before progressing.

IID builds on the Waterfall and Vee methods and can employ these within a development increment for the purposes of structured planning and verification (Basili and Larman 2003). In the IID method, multiple increments can be developed simultaneously leveraging a common baseline or trunk. Figure 5 provides a visual of the IID process (Forsberg, Mooz and Cotterman 2005).

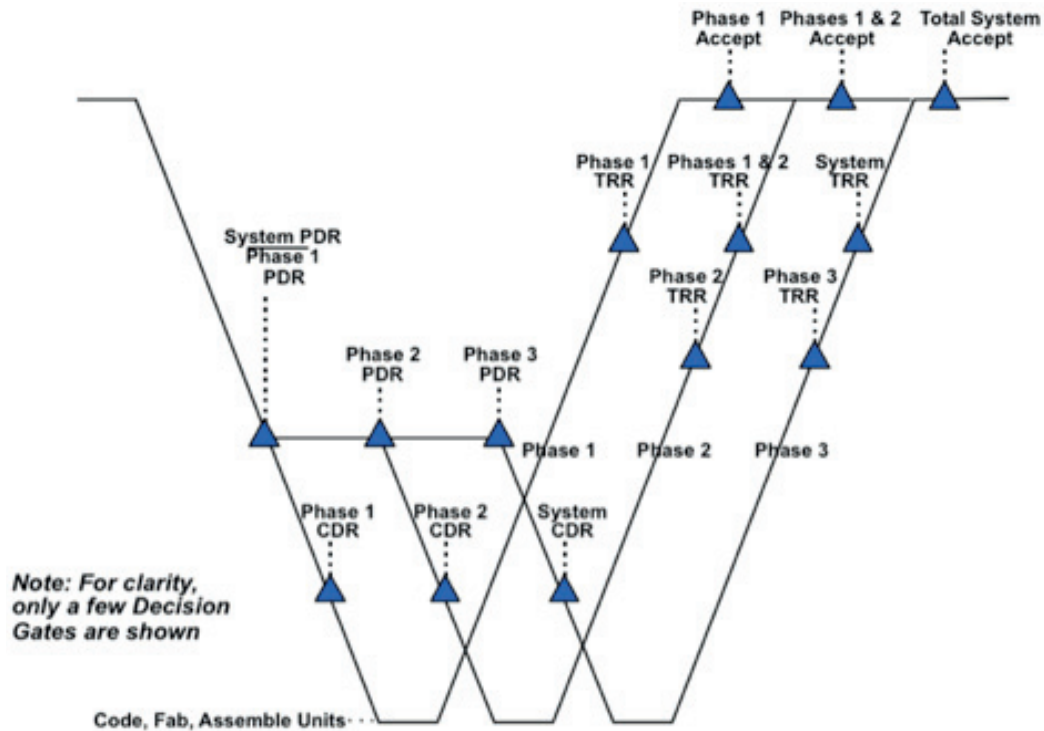


Figure 5 - Iterative and Incremental Development Life Cycle Model (Forsberg, Mooz and Cotterman 2005)

The Spiral model, on the other hand, typically focuses on more discrete transitions from one iteration to the next as building blocks (B. W. Boehm 1988) and can therefore be employed more readily on cyber-physical systems in which functionality can build over time. An example of this is the U.S. Space Development Agency’s acquisition of space-based capabilities in spirals, or “tranches” where each successive tranche builds on the capability of the prior (Space Development Agency 26 June 2020). Figure 6 provides a modified graphic of a single iteration in a spiral life cycle (Douglass 2016).

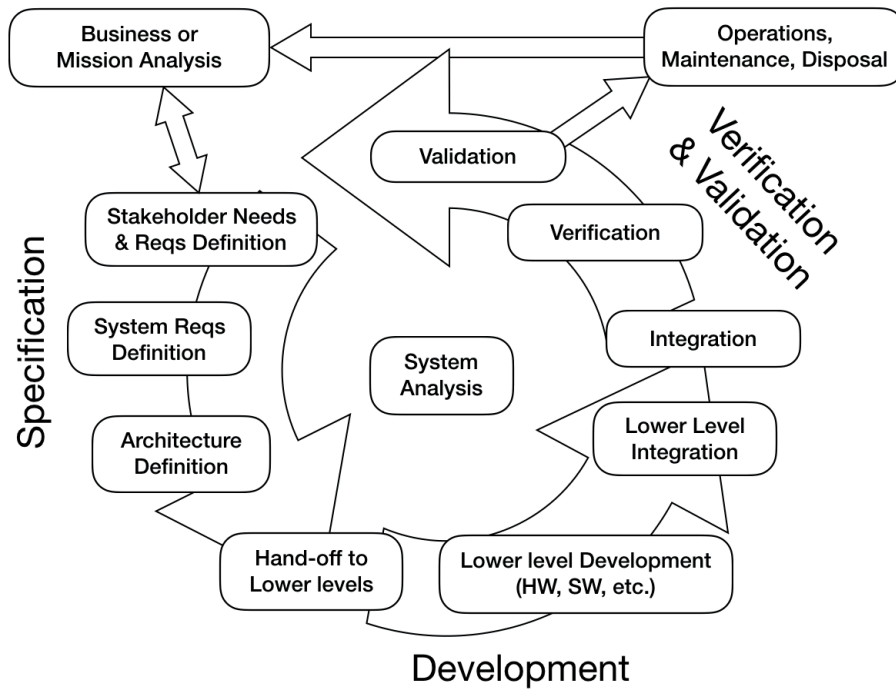


Figure 6 - Spiral Life Cycle Model (adapted from Douglass 2016)

IID and Spiral enable early delivery of capability with incremental updates on more frequent centers than traditional Waterfall and Vee implementations. In addition, more frequent deliveries generally result in more frequent, smaller gate reviews. A byproduct of the shorter cycles and frequent reviews is the potential for added variability and instability due to shifting focus and requirements. In terms of completed capability and functionality, a shortcoming in these methods is the system is generally not fully capable until all deliveries are complete resulting in potential challenges with obsolescence before completion. Overall, evolutionary and concurrent models tend to be best suited for systems of medium complexity, schedule need, and variability (SEBoK Editorial Board 2020).

While Iterative and Incremental Development and Spiral life cycle models provide a means to revisit prior life cycle stages, they typically do so as a predefined logical plan to incrementally introduce new capabilities rather than to focus on adaptation or maintenance

of existing capabilities during the post-deployment operational life of a system (Douglass 2016).

2.4.4 Interpersonal & Emergent Life Cycle Models

The third category of life cycle models introduced by SEBoK is Interpersonal and Emergent (SEBoK Editorial Board 2020). As presented by SEBoK, this is focused primarily on the Agile model however, as introduced in Section 2.3.2.2, DevOps has many similar characteristics and is proposed here as an extension to this category.

Agile, discussed briefly in both the systems engineering and software engineering literature reviews, is structured to “adapt” to evolving stakeholder needs through continual assessment of highest priorities in short, repetitive development cycles, called “sprints”. This, in practice, is less of an iterative and more of a rolling process, resulting in a focus on quick turn priorities typically driven by stakeholders more so than developers. Agile incorporates more than basic process methodology and, as articulated in the Agile Manifesto, is a mindset and conscious prioritization of continual forward progress and working functionality over comprehensive completion before release (Beedle, et al. 2001). Figure 7 depicts a representation of an Agile process flow (Boehm and Turner 2004).

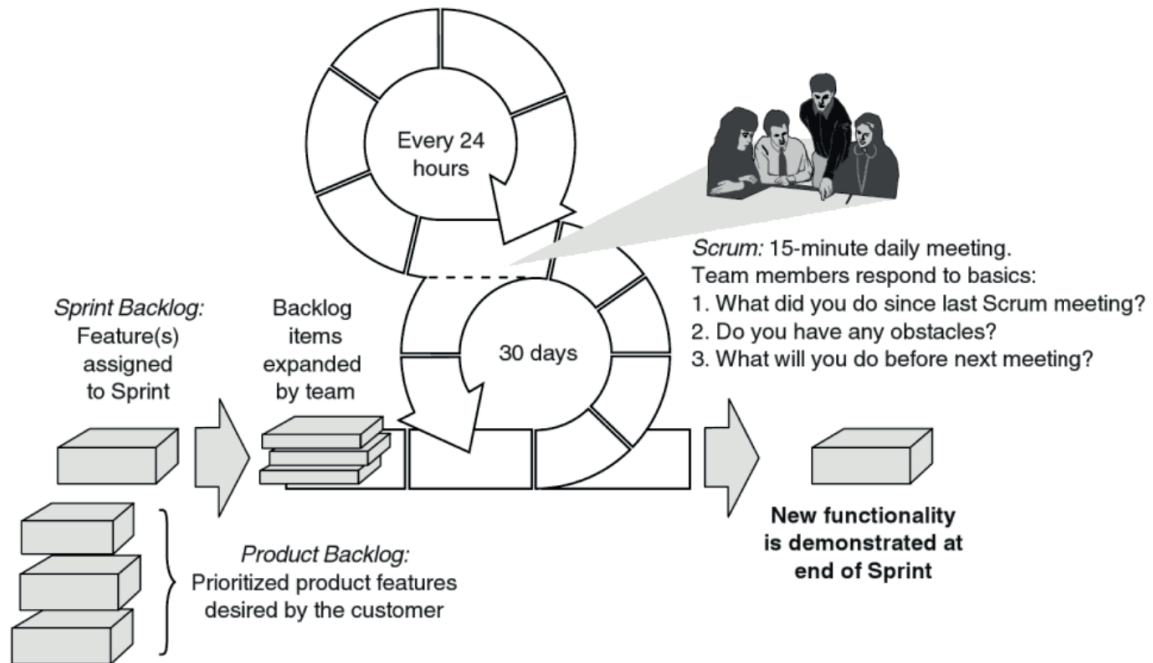


Figure 7 - Agile Methodology (Boehm and Turner 2004)

Agile focuses on continuous adaptation to changing customer/stakeholder priorities and therefore is receptive to variability in requirements and needs. Additionally, the methodology and process emphasize collaboration which further improves time to market for capabilities. A result of the limited planning and structure in Agile is the potential for unstructured approaches to system architecture and inconsistency throughout the development process. Similar to the previously presented life cycle methods, Agile is also focused on product delivery and does not explicitly incorporate elements of post deployment system management into the core tenets of the life cycle model.

DevOps is a model recently gaining traction and broader application in the software domain and was discussed in more detail previously, in Section 2.3.2.2. From a life cycle model standpoint, it combines the flexibility and adaptability of Agile development with a deliberate and continuous path from operations back into development, promoting

continual maintenance and evolution. Figure 8 provides a view of DevOps as a cyclic feature generation process (Compuware 2019).

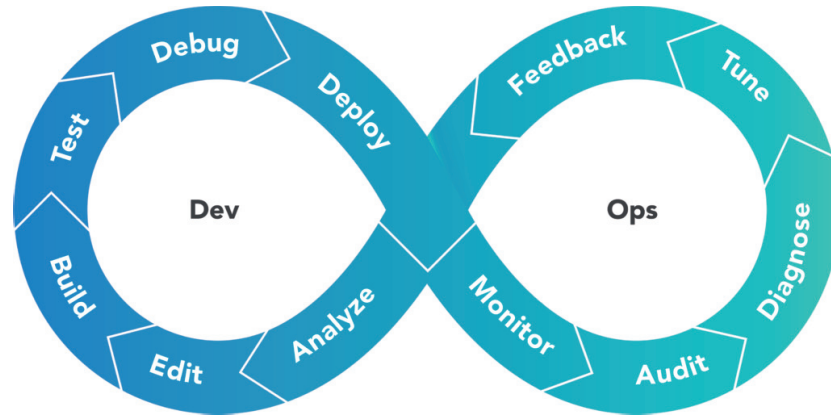


Figure 8 - Representative DevOps Life Cycle Process (Compuware 2019)

DevOps emphasizes collaboration not just among development teams but more importantly across development and operations/sustainment teams to enable continuous integration, test, deployment, and monitoring (Riungu-Kalliosaari, et al. 2016). An area for potential improvement with respect to DevOps is the consistent application across broader system instantiations. Due to the localized focus on maturing, deploying, and monitoring system features on the feature scale rather than system scale (typically in containerized microservices), there is the potential of local optimum solutions rather than global optimum solutions. Based on this, broad applicability to highly complex cyber-physical systems is still in question.

Of the life cycle models discussed herein and presented in Table 1, DevOps is the first to visually integrate aspects of operations, monitoring, and maintenance into the life cycle model depiction and is the one currently not yet formalized in model-based systems

engineering practices. Section 3.2 explores an adaptation on DevOps for systems engineering applications and more specifically for MBSE application throughout a system's life cycle.

2.5 Digital Engineering

2.5.1 Definition of Digital Engineering

Digital Engineering, as defined in the taxonomy in Section 1.1, is enabled by “the creation of computer readable models to represent all aspects of the system and to support all the activities for the design, development, manufacture, and operation of the system throughout its life cycle” (SEBoK Editorial Board 2020). Therefore, Digital Engineering relies on the conversion of systems, subsystems, functions, behaviors, interfaces, documentation, etc. into fully digital representations capable of interrelation at the metadata level. Another way to look at it is that Digital Engineering is organized and formatted data hosted in a common and accessible repository or computing system.

A sub-discipline of Digital Engineering is model-based engineering (MBE) which extends formalism of system or discipline specific modeling techniques to the data. As defined by the NDIA MBE Subcommittee and further articulated by the US DoD MBE Infusion Task Team, “Model-Based Engineering is an approach to engineering that uses models as an integral part of the technical baseline, including the requirements, analysis, design, implementations, and verification of a capability, system, and/or product throughout the acquisition life cycle” (NDIA Systems Engineering Division M&S Committee 2011) (Puchek, et al. 2017).

Within MBE, MBSE is a further sub-discipline focused on modeling and more importantly digitally linking the tasks, processes, and artifacts of systems engineers.

Friedenthal, Moore, & Steiner define MBSE as “the formalized application of modeling to support systems requirements, design, analysis, verification, and validation activities beginning in the conceptual design phase and continuing throughout the development and later life cycle phases” (Friedenthal, Moore and Steiner 2014).

Digital Engineering, MBE, and MBSE are in the early stages and evolving rapidly, as are the tools to instantiate and maintain system models. This can be seen in the focus and attention given to digital and model-based engineering in current visionary and strategic documents (INCOSE 2014), (U.S. Department of Defense 2018). Looking forward, it is then logical to trend towards fully digital representations of systems in the form of models as an enabler for establishing and managing inter-relations of system elements from womb to tomb (including procedures, scripts, maintenance manuals, etc.), as introduced in Section 3.3.

As a point of note, the first purpose of modeling has been identified as characterizing an existing system to facilitate maintenance, including support for training, knowledge capture, and system design evolution, all of which are critical aspects of sustaining and evolving operational systems (INCOSE 2015). Despite the attention across the community, the existing MBSE tools tend to focus on products and processes in line with early life cycle stage definitions, roles, and responsibilities (Friedenthal, Moore and Steiner 2014), (Delligatti 2014), presenting an opportunity for evolution into the operational stages of system life cycles.

2.5.2 Recent Developments in Digital Engineering

Work towards a digital transformation of systems engineering has been ongoing for some time. There is significant literature available describing novel applications of Digital

Engineering and model-based practices to all stages of system development and operations, including, among other topics:

- 1) The design of system operations (Gans 2017), (Uhlemann, et al. 2017)
- 2) Prognostics & health management applications (Sutharssan, et al. 2015), (Codetta-Raiteri and Portinale 2015)
- 3) Fault management design and test to create resilient systems (Castet, Bareh, et al. 2016), (Castet, Bareh, et al. 2018), (Wagner, et al. 2012), (Rabelo and Clark 2015)
- 4) Documenting and tracking Maintenance (Crane, et al. 2017)
- 5) The development and maintenance of digital twins to accurately track system status over the life cycle (General Electric Corporation 2018)

Common to many of these topics is the use of MBSE methodologies and descriptive models to represent and support the design and development of the system in preparation for system operations, however applications of active use of descriptive models leveraging SysML during operations presents an opportunity for advancement.

2.6 Cross-Cutting Topics

Digitally representing systems and constituent elements in descriptive, interrelated, and executable models creates a plethora of opportunities to fully leverage the models for advancing systems engineering applications. The following two subsections are two specific examples where research is being performed with a potential for positive impacts to system development and, as proposed herein, to system operations and sustainment.

2.6.1 Formal Methods

Formal methods can be defined as “mathematic/logic methods to specify, develop, and verify systems” and are used to “...evaluate the compliance of a system specification to a set of constraints defining correctness properties” (Madni and Sievers 2018). Toure, et al propose representing software systems through domain specific modeling languages to enable the use of axiomatic semantics for pre- and post- condition verification (Toure, et al. 2017). The concept of axiomatic semantics is a subtype of formal methods focusing on discrete pre- and post- conditions, or states, surrounding a discrete task, function, or action. As will be described in Section 3.3.5, applying this approach more broadly beyond software systems to cyber-physical systems establishes a means to validate system configuration pre- and post- actions (i.e. a constraint check prior to execution of a procedural step and successful outcome following a step).

The concept and application of Petri Nets is a form of axiomatic semantics. As articulated by Huang et al (Huang, McGinnis and Mitchell 2019) and Graves & Bijan, (Graves and Bijan 2011), DevOps and microservice-like behavior models in SysML each individually enable a format, such as Petri Nets, for formal verification of system operations support products within an executable modeling environment. Graves & Bijan state that “formal methods have the potential for determining information consistency and change impact” (Graves and Bijan 2011), which are both key elements to agility and adaptability during operations and sustainment stages.

The use of formal verification of descriptive modeling artifacts is fairly recent and leaves room for further expansion. This is addressed briefly in a discussion on future research opportunities in Section 5.3.1.

2.6.2 Safety Critical Systems

Building on the application of DevOps principles with formal methods of verification establishes the necessary framework to support safety critical system design and operation, as articulated by Olszewska and Walden (Olszewska and Walden 2015). Research is underway to expand on the usage of MBSE and SysML for developing and verifying safety critical systems and is discussed briefly in Section 5.3.1 as a next step to build on the concepts introduced in the following chapter.

2.7 Bringing It All Together – Summarizing the State of the Art with a Call to Action

As noted at the start of the literature exploration, the more recent systems engineering methodologies are driving towards the use of integrated digital and model-centric tools to structure and manage systems engineering tasks and products. However, they are generally focused on and optimized for the first “half” of systems’ life cycles and rarely leveraged beyond system deployment (Douglass 2016), (Delligatti 2014), (Friedenthal, Moore and Steiner 2014). The INCOSE *Systems Engineering Vision 2025* identifies formalizing systems engineering and, more specifically MBSE, for later life cycle stages and the U.S. DoD *Digital Engineering Strategy* calls out a need to evolve digital and model-based tools to improve agility, adaptability, and overall scalability to more complex systems (INCOSE 2014), (U.S. Department of Defense 2018). Madni and Sievers identify the many benefits of “living” system models, as well as the current shortcomings in modeling tools, and go on to articulate the need for advancements and evolution in the MBSE methodology, processes, and tools, stating: “MBSE methods must cover the full system life cycle. Extending MBSE methods will require both methodological advances and development of supporting processes and tools” (Madni and Sievers 2018). While the systems engineering

discipline is still considered immature in its widespread use of models, it is expected in the next decade that MBSE will play an increasing role in the practice of systems engineering, particularly based on the growing interdependency of software within systems engineering (Ramos, Ferreira and Barcelo 2012). Along those lines, Dove and LaBarge recognize the need for an agile systems engineering life cycle model, introduce the start to one, and identify an opportunity to better build this model into a workable methodology (Dove and LaBarge 2014).

Based on the broadly acknowledged applicability of systems engineering to all life cycle stages, the identified opportunities for improvement to systems engineering focus and agility, and the needed improvement to digital and model-based systems engineering tools for later life cycle stages, a modified systems life cycle model is introduced in Section 3.2 and an extension to toolchains to support this model is addressed in Section 3.3.

Chapter 3: Research Methods, Resulting Life Cycle Methodology & Framework

3.1 Research Methodology Overview

Before embarking on a detailed discussion on the concepts introduced by this research, the methods employed to develop the new material, and the specific contributions this research brings, a brief dialogue on the Inventor's Paradox is useful. This will help frame the path chosen in this research based on the initial question proposed at the onset of Chapter 2 for codifying system operations products and systems engineering support during system operations into a descriptive modeling environment to improve agility and ultimately enable autonomicity.

3.1.1 The Inventor's Paradox

As introduced by George Polya in *How to Solve It*, the inventor's paradox implies that in order to solve what one sets out to do, one may have to solve additional problems along the way (Polya 1945), (Ruan, et al. 2010). The initial intent of this research focused on improving model-based applications of system engineering during system operations in order to harvest the advantages seen to date in MBSE applications. What was quickly found and articulated through the literature review throughout Chapter 2 is that MBSE is rarely applied during the operational stages of a system's life cycle. Therefore, to leverage a model-based support structure for operational systems, a larger problem must first be solved: a life cycle model and associated tools must be created to foster the use of MBSE into and throughout the operational stages.

The concepts introduced throughout this chapter provide a normalization for MBSE support throughout the life cycle to enable, as this manuscript's title implies, a step towards polymorphic systems engineering. This is introduced as a modified life cycle model and is

followed by a descriptive modeling framework to normalize the interface of MBSE information and products throughout the life cycle.

3.1.2 Architecture, Methodology, Framework Development

The methodology applied to expanding MBSE applications to system operations is in the general architecture development category. Therefore, the problem-solving approach converts heuristics into a more qualitative and foundational product rather than a quantitative and exact scientific application. The result is a modified life cycle model to aid and guide systems engineering support, through the application of MBSE, throughout the life of any system. The modified life cycle model is a graphical representation of critical stages of any system (consistent with the generic life cycle model introduced in Section 2.4.1) with a focus on the relationship of one stage to another and an emphasis on key enablers for each transition. As will be noted in detail in this chapter, the focus most notably adds direction for MBSE application and use into and during system operations and sustainment.

The second application of research methodology and problem solving is in the development of a more detailed descriptive modeling framework to support instantiating this modified life cycle model. This too falls into an architecture development category with a resultant visual representation of the meta-model, i.e. the inherent interrelation of newly created descriptive model elements, imbedded in the modeling framework.

3.1.3 Methodology Map

Figure 9 provides an overview of the research methodology employed. There are three major elements in this process, identified by the three numbered circles. Each element builds on the prior, resulting in a simulation environment (3) in which to perform a use

case to determine the utility of the newly introduced life cycle model (1) using the descriptive modeling framework (2). The chapters in this manuscript in which the material in the methodology map in Figure 9 is covered is noted by the color coding of each block, identified in the legend.

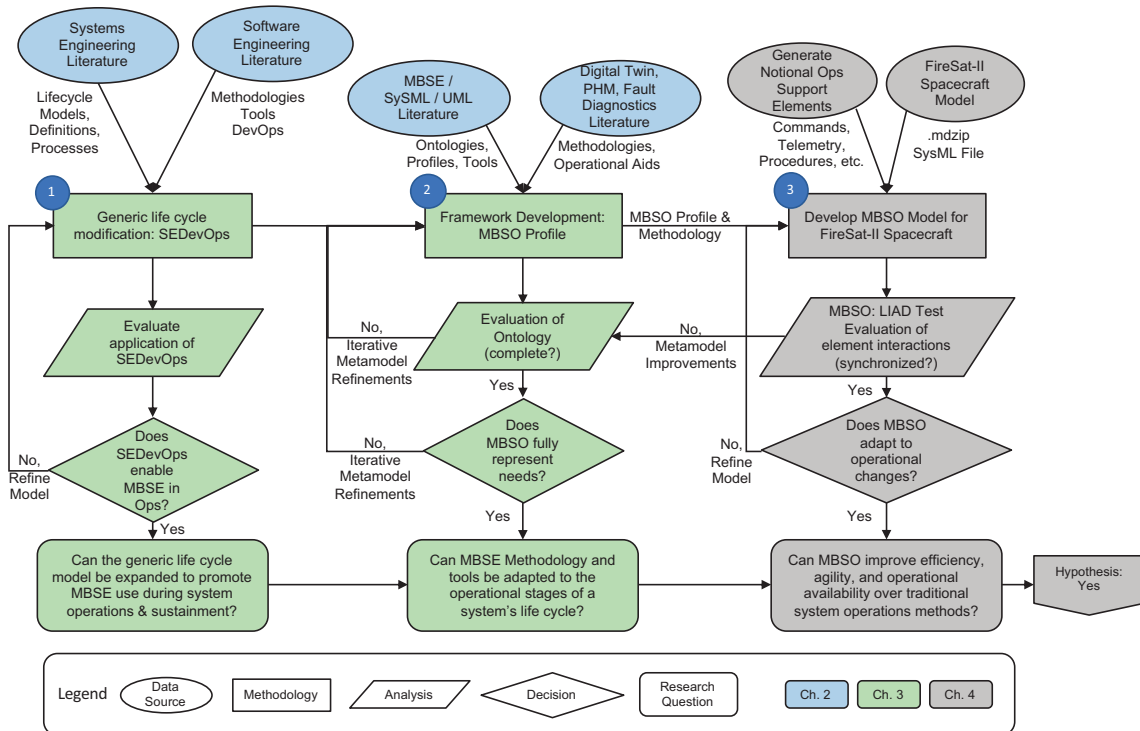


Figure 9 - Research Methodology Map

Based on this, it is now possible to build on the literature review performed throughout Chapter 2 towards a solution for the problem statement introduced in Section 1.2.

3.2 Introducing the Systems Engineering DevOps Lemniscate

3.2.1 Putting It Together into SEDevOps

As noted throughout the literature review, evolving the use of MBSE beyond system deployment is an area of opportunity to improve agility and reduce costs during the

operational stages of a system’s life cycle. This opportunity comes about in part due to the limited definition of descriptive modeling elements and specifics on modeling methodology beyond deployment and, furthermore, the limited continuation of MBSE models from the development stages to the operational stages. To support and promote the opportunity for greater MBSE utilization throughout a system life cycle, the SEDevOps Lemniscate, introduced in Figure 10, was created.

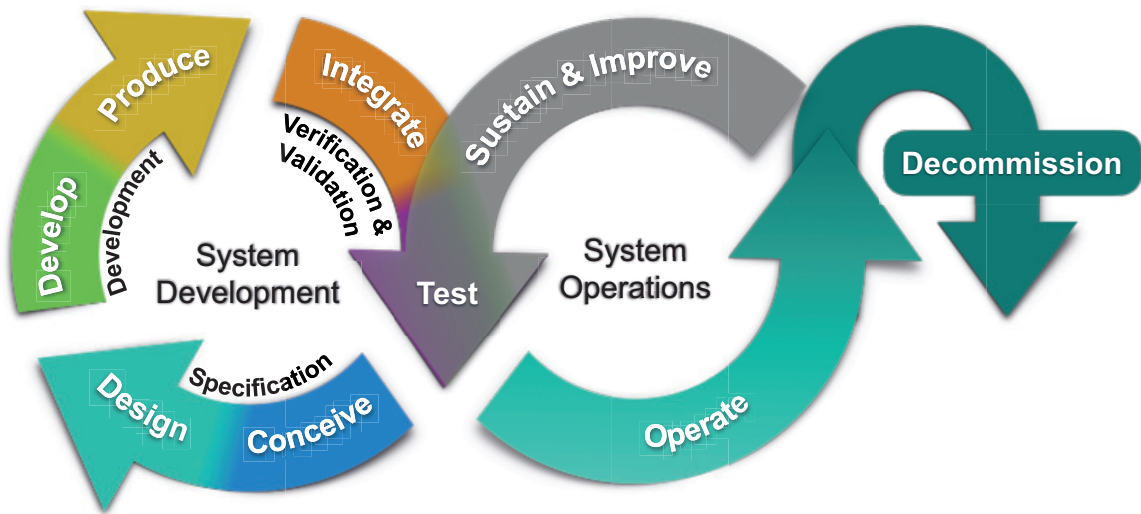


Figure 10 - The SEDevOps Life Cycle Model

Based on the successes seen in DevOps implementations in the software domain, the SEDevOps Lemniscate starts with the lemniscate shape from DevOps models and overlays the more traditional system life cycle stages across the lemniscate as compared to the software-centric stages seen in DevOps models, and example of which is depicted previously in Figure 8. A key element of DevOps is the underlying common toolchain that facilitates continuity of data and products throughout all life cycle stages. SEDevOps is built to leverage MBSE tools and modeling languages as this common toolchain throughout the life cycle, articulated in more detail in Section 3.3. With the underlying

application of MBSE in mind, the SEDevOps life cycle model is a merger of key life cycle stages from traditional systems engineering life cycle models, described in Sections 2.4.2 and 2.4.3, along with features derived from more recent emergent life cycle models overlaid on a DevOps continuum or lemniscate extracted from several recent variants of DevOps models. Figure 11 depicts the source of features from the life cycle categories described in Table 1 on the SEDevOps lemniscate.

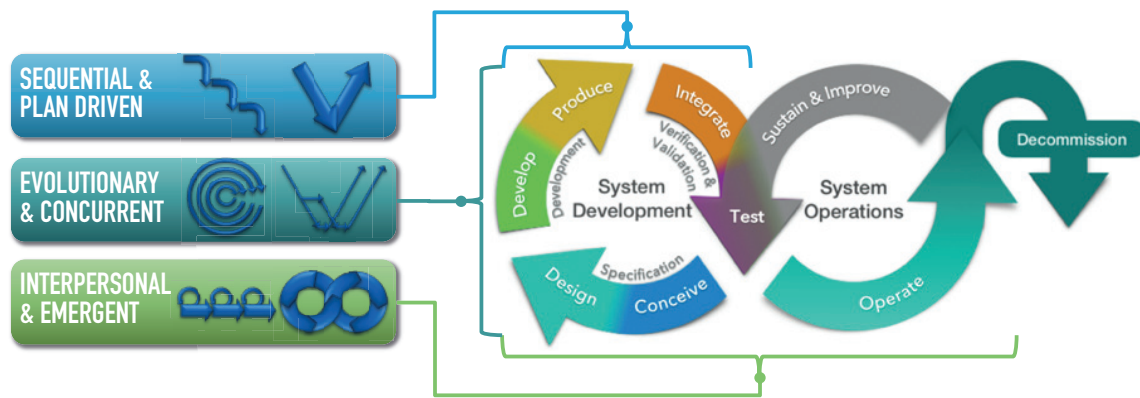


Figure 11 - Feature Sources in the SEDevOps Life Cycle Model

Existing MBSE methodologies and tools address the traditional systems engineering life cycle stages of Concept, Design, Develop, Integrate, and Test and therefore these stages were sourced from the generic life cycle models described in Section 2.4.1. The agile feature of promoting multiple design and deployment cycles was derived from agile systems engineering models presented by Dove & LaBarge (Dove and LaBarge 2014), and Douglass (Douglass 2016) which differs from traditional DevOps by introducing a branch from Test back to Conceive to support subsequent cycles in parallel to Operations. Finally, the DevOps base lemniscate, which represent the continuum of capability creation, integration, verification, deployment, monitoring, and improvement, was adapted from a

variety of DevOps life cycle models currently in use in the software domain (Combemale and Wimmer 2019), (Compuware 2019), (Atlassian 2019).

3.2.2 Description of Parts from Existing Life Cycle Models

The SEDevOps life cycle model embodies the symbiotic relationship between system development activities and system operations and sustainment activities. SEDevOps is designed to specifically encompass the development and management of descriptive models not only of the core system of interest, but also the necessary operational support elements within the boundary of the overall system and, more importantly, incorporated as part of the requisite MBSE model and continuously managed and tested throughout the life cycle, a concept depicted notionally in Figure 12. This ensures the complete system *and the necessary support elements to operate and sustain the system* are properly accounted for in modeling space, continuously addressed, and accurately maintained. MBSO, introduced later in this chapter, provides an approach to incorporate these operational support elements into the broader descriptive system model.

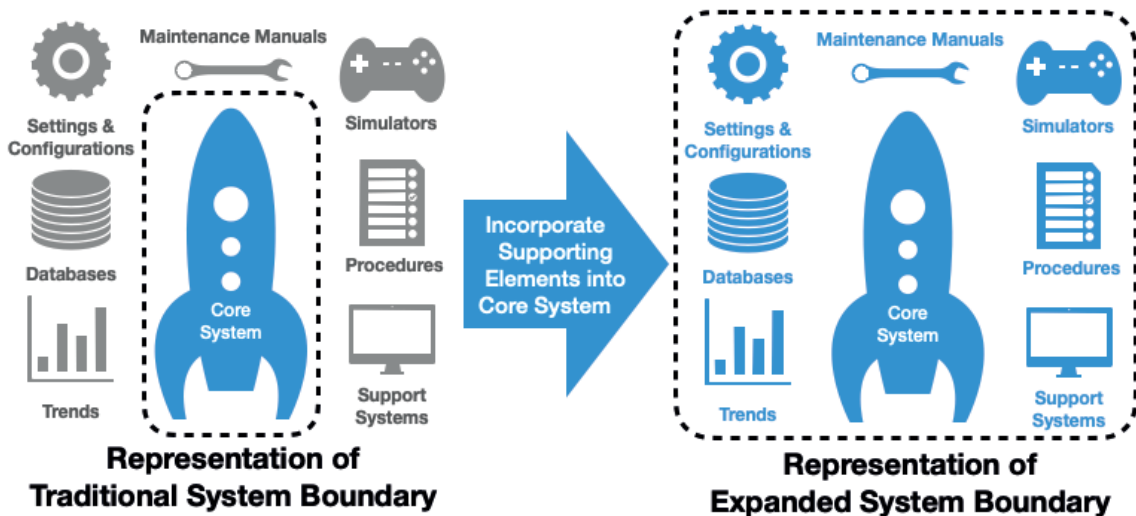


Figure 12 - Notional Expansion of System Boundaries in SEDevOps

In the SEDevOps life cycle model, system development progresses through the conventional sequential, plan-driven stages on the left in Figure 10, starting with Conceive then Design (grouped together as system Specification), Develop then Produce (grouped together as system Development), and Integrate then Test (grouped together as system Verification & Validation). These development stages are characteristic of a methodical systems engineering approach to system development, as derived from the sequential, plan-driven methods presented in Table 1 and discussed in Section 2.4.2. MBSE, as broadly implemented to date, contains the relevant modeling elements and prespecified interrelationships to support the product and content development throughout these stages (Friedenthal, Moore and Steiner 2014).


Upon successful completion of Test within Verification & Validation, the system is deployed to an operational environment at which time “Operate” is the primary focus. Concurrently, SEDevOps promotes iterative and evolutionary development, particularly with respect to MBSE model curation and use, with a directed branch back into system development at this point, as derived from both IID and Spiral methods discussed in Section 2.4.3. In order to continue operations through system aging, changing environmental influences, and failure scenarios, system maintenance, sustainment, and improvement is required. These stages are not new to systems engineering as a discipline. However, the continued use of MBSE descriptive models throughout the operational stages as the common and collaborative toolchain, akin to DevOps collaboration tools for software development and maintenance, is where SEDevOps provides a new emphasis.

Throughout system sustainment, near-continual regression and diagnostics testing is critical for ensuring the continued accuracy and completeness of all necessary operational

support elements (i.e. ensuring procedures and scripts are accurate for the current system configuration and environmental conditions, they represent the proper transition from primary to redundant equipment if a failure occurs, etc.). The right side of the SEDevOps lemniscate in Figure 10 introduces the divergence from the traditional systems engineering life cycle models of Table 1 and incorporates the DevOps concept of a directed path back to system development from the system operations stages, as scenarios warrant during system sustainment.

As described in Section 3.2.1, the merger of features from each of the life cycle models noted in Table 1 into the SEDevOps model brings a number of strengths from each life cycle category together, resulting in a combination of the strengths, enabling not only rigorous development structure but also adaptability in both development and operations. Table 2 provides an addendum to Table 1 describing the relevant characteristics of SEDevOps, including strengths and potential shortcomings.

Table 2 - SEDevOps Characteristics

Life Cycle Category	Method Name	Notional View	Strengths	Shortcomings	Ability to Accommodate		
					System Complexity	Time to Market	Rqmt Variability
Model-Driven, Continuous	SEDevOps		<ul style="list-style-type: none"> Enables layered, sequential development approach Supports inherent recursion and cyclic development construct, including incremental capability delivery Drives operational focus for MBSE Enables and emphasizes continuous validation and regression testing Near continual onramps for updates and modifications with constant feedback Focus on collaboration across development and operations teams Streamlines impact identification and mitigation in operations Provides intrinsic configuration management in model-space Provides a common toolchain and data interface throughout life cycle 	<ul style="list-style-type: none"> Requires rigorous model management Potentially complex infrastructure to support model development and management Potential adoption challenges, including adequate training and infrastructure Designed to primarily support software-based operational support element representations in modeling space; Potential challenges adapting to hardware management 	High	Med	High

Additionally, the fork in the life cycle flow between the Test stage and the Operate stage as well as between Sustain and Operate promotes the concept of MBSE support occurring and continuing simultaneously in different stages. Development of enhancements and system fixes in MBSE model-space is encouraged to occur in parallel to continued system operations and maintenance of operational support products within a common model-space. Definition of multi-faceted and simultaneous systems engineering support through this form of MBSE application is a first step towards a polymorphic discipline.

3.2.3 Focus on Test to Enable Continuous Integration, Test, & Deployment

An advancement that SEDevOps brings is the visual emphasis on “Test” residing in the middle of the graphic with all paths, apart from Decommission, passing through this stage. This stresses the importance of not only testing prior to deployment of any system element or operational support element upgrade, but more importantly near-continual validation and regression testing throughout operations and sustainment. As a noted focus of SEDevOps is on the model-based support structure of MBSE throughout the life cycle, this focus on Testing is an inherent feature in many MBSE toolchains and therefore can be implemented seamlessly within model-space in order to provide continuous regression testing on existing modeled elements as well as validation testing as configurations change over the life of a system. Testing has long been identified as a critical activity prior to deployment in the systems engineering process and is represented as such in existing systems engineering process definition (INCOSE 2015) as well as in MBSE modeling language and tool composition (Friedenthal, Moore and Steiner 2014). It is equally as critical as systems age and operational environments evolve over time to ensure the system

continues to be operated successfully and the operational support elements (procedures, scripts, databases, etc.) are continually tested for accuracy and completeness in response to system and environment evolution.

An example of a simplified operational support element (i.e. procedure) modification, test, and redeployment case during the Operate stage is the response to a hardware unit failure in an operational system. In this scenario, the physical system is configured, as designed, to a back-up unit with reduced capability requiring operational procedure updates based on the reduction in performance, followed by testing of these procedural updates and finally release of the updates to properly poise the system and the support team for the change in functionality and capability. The focus in this scenario is on the operational support products which, as proposed with SEDevOps, exist in the MBSE model space. The modeling framework to host and manage operational procedures in MBSE models is proposed and described in detail in Section 3.3. This scenario exercises the right-side of the SEDevOps lemniscate in Figure 10.

An alternate and more complex scenario could entail transitioning an operational software system from a bare-metal, rack-mounted server architecture to a cloud-enabled, containerized, virtual architecture. In a case such as this, it requires a full, formal redevelopment cycle in which significant aspects of the system are redesigned, redeveloped, reintegrated, reverified, and redeployed. This example is a more involved case in which the system adapts to advancements in infrastructure capability and evolves its architecture to improve performance and maintain relevancy. Similar to the scenario above, the focus in describing this scenario here is to articulate the ability for the MBSE tools supporting SEDevOps to handle revisiting core elements of the system's descriptive

model to rearchitect and follow the existing and previously employed systems engineering (and MBSE) processes to redeploy the updated and evolved system. This represents traversing the entirety of the SEDevOps lemniscate in Figure 10.

In each of these cases, the resulting actions pass through the Test stage as a gate before returning to the Operate stage, emphasizing the continuous testing needs during system operations to prepare for further modifications, adaptations, and evolutions, as needed. As noted earlier with the focus of SEDevOps on MBSE applications across the life cycle, implementing adequately detailed system models allows for automating the continued validation and regression testing of operational support products to ensure compatibility and proper management of system configurations. This is discussed in Section 3.3 with an implementation leveraging SysML.

3.2.4 Addressing Decommissioning

Ultimately, the operational stage results in a need to decommission the system, represented by the off-ramp on the far right of the SEDevOps lemniscate in Figure 10. Decommissioning is a topic considered briefly during system development when driven by requirements. For example, space systems in low-earth orbits have requirements levied upon them to deorbit within some period of time following mission completion or certain failure criteria dictating a necessary deorbit. In other words, the systems must be removed from the operational environment and, in doing so, have to abide by strict safety requirements to ensure any potential loss of life and property damage due to reentering the Earth's atmosphere is minimized. In this scenario, design considerations are made to account for material selection, redundant safe modes to control reentry, and other critical factors to ensure the requirement can be met at end of life. As the system ages over its

operational life, the initial decommissioning procedure may be unfeasible due to extraordinary failures resulting in the need to redevelop and test a modified decommissioning procedure. The prescribed path from Operate through Sustain & Improve back to development enables this modification in parallel to continued Operations prior to decommissioning. This again is intended to address the use of MBSE models to develop and manage system behaviors and relevant operational support products in response to operational circumstances.

3.2.5 Continuity & Collaboration Throughout the Life Cycle

While SEDevOps is directly derived from the DevOps principles of continuity and collaboration between development and operations leveraging tools and technologies to facilitate this, the focus of SEDevOps is on MBSE model implementation at the full system level (i.e. macro-level). DevOps, applied in the software domain, generally provides the framework to manage the life cycle of individual features or groupings of features (i.e. micro-level) through the requisite phases seen in Figure 8: Plan, Create, Verify, Package, Configure, and Monitor before moving on to the next feature development. Depicting a broader system life cycle graphically with a traditional DevOps lemniscate would therefore show a multitude of successive (and possibly parallel) lemniscates (cycles) throughout a system's life, similar to a view of the Agile model implemented throughout the life of a system, seen in the notional graphic for Agile in Table 1.

SEDevOps incorporates a fork in the cycle where events during system operations can result in a return to the development stages as warranted by the complexity of the updates needed to the model and operational support products, or can remain in a continuous cycle of Operate-Sustain-Improve-Test-Operate without a necessary return to System

Development. Noted here for clarity, this implementation of SEDevOps is focused on the instantiation and continuous management of MBSE artifacts throughout the life cycle of a system as a means to create, model, test, and manage the products necessary to operate a system. This distinction is important in establishing a framework for interrelating operational support elements with development elements in system artifacts, namely in a descriptive modeling environment, as introduced in Section 3.3 with a modeling extension. An additional distinction to articulate is the resulting ability for operations and sustainment personnel to manage and implement changes to operational support products in model space without a full path back through the traditional systems engineering development and deployment stages based on the management of operational support products within an overarching system model.

To articulate the broadened focus of SEDevOps as compared to DevOps in the software domain, the following example is presented. DevOps can support the roll-out of a software application hosting framework without necessitating the incorporation of all apps in the framework on day one. A vignette here is a smart phone operating system (framework) and app store (application catalog). DevOps can then monitor the base functionality and cycle back to develop applications individually, feeding back lessons learned into updates to the framework as needed and as application complexity increases over time.

A spacecraft is a relevant example of a cyber-physical system. Due to the complexities and extremes of space, it is unfeasible to launch a spacecraft without all necessary subsystems (i.e. apps) populated, integrated, tested, and working. If a spacecraft is launched without an attitude control system, for example, the base functionality is irrelevant and unusable. Therefore, adapting DevOps principles to cyber-physical system

development and deployment requires modification to the methodology to apply on a broader time scale and a broader feature scale. SEDevOps is designed to broaden this view and the associated support to enable the benefits seen at the microservice level in DevOps to the full system level throughout the life of a system.

3.2.6 Evolution & Adaptation

As systems are deployed in an operational environment, scenarios arise which necessitate responses to faults and failures, updates for changing environmental conditions, and critical system enhancements to ensure continued success of the system. This requires adapting system configurations and the associated support elements to address the need. Such adaptation can range in complexity from simple procedural updates and associated testing, centered on maintaining the baseline system capabilities in dynamic and evolving environmental conditions, to more complex scenarios and system evolutions involving a formal development cycle (conceive, develop, integrate, verify, etc.), based on the severity of the adaptation and evolution required. The SEDevOps life cycle model promotes this range in adaptation and evolution throughout the system operations & sustainment stages, including the direct path back to a full specification, development, verification, and redeployment cycle if the complexity of the adaptation and evolution warrant, as depicted by the transition back to the left-side of the SEDevOps lemniscate. Figure 13 provides a spin on the SEDevOps life cycle graphic representing the corresponding adaptation & evolution cycles a system may go through over the life of a single product or more so, over the life of a product line as the need for enhancements and evolution are learned over the operational life of prior versions/deployments and factored into the development of future products. Therefore, as systems are required to adapt to changing conditions, updates and

enhancements are developed and deployed resulting in an evolution on the overall system over time.

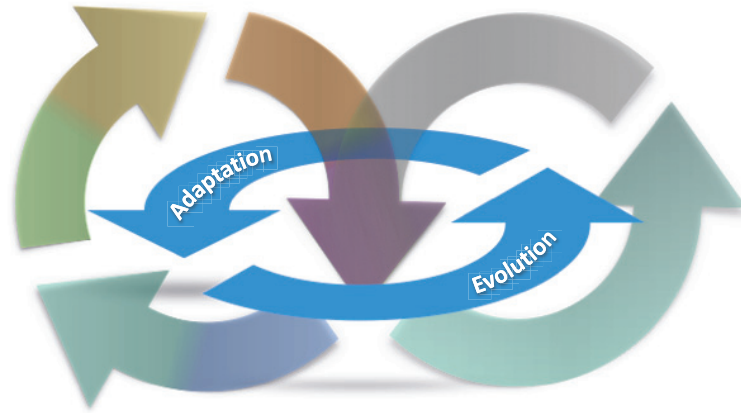


Figure 13 - System Adaptation & Evolution via the SEDevOps Life Cycle Model

3.2.7 SEDevOps Summary

Two fundamental principles of DevOps, and key enablers of the continuum between development and operations, are:

1. Continued curation of the development environment to enable accurate testing
2. Tight collaboration between operations and development

Therefore, to properly implement SEDevOps, there must be a means to accurately represent the operational system in an environment in which the development of capabilities and features can take place at any point. Tying back to the push for digital transformation of systems engineering, this is where a descriptive modeling environment provides the platform for this continuum as it can be created during system development, maintained during system operations, and leveraged for continued development and testing of a system's and its associated support element capabilities throughout. MBSE provides

this platform and the necessary extension to enable use in operations and sustainment stages is presented next.

3.3 A Framework for Applied Methodology: Model-Based System Operations

3.3.1 Summary

Implementing the concept of SEDevOps (i.e. continuous collaboration and simultaneous model-based systems engineering support throughout the life of a system) requires both a clear model, described in the prior section, and a framework for developing and curating the surrogate operational environment in which to create and maintain the operational support products. In the case of cyber-physical systems, the simplest instantiation of a surrogate operational environment consists of a detailed representation of the operational configuration of the system at any given point in time coupled with the actions necessary to transition between configurations (i.e. procedures, commands, maintenance manuals, etc.). In a sense, this becomes an operational emulation environment.

MBSE tools and methodologies have driven large portions of the system life cycle into this digital ecosystem however, as noted previously, the portions not adequately addressed by MBSE methods and tools to date and critical to successful implementation of SEDevOps are those supporting the operations and sustainment of systems.

MBSO builds on the base of UML/SysML and the MBSE approach through an ontology defining and relating operational support elements in a descriptive model, depicted in Figure 14. Unless otherwise noted in figure captions, SysML figures presented here and throughout Chapter 4 are originals created in the development of the MBSO framework and for the supporting use case detailed in Chapter 4.

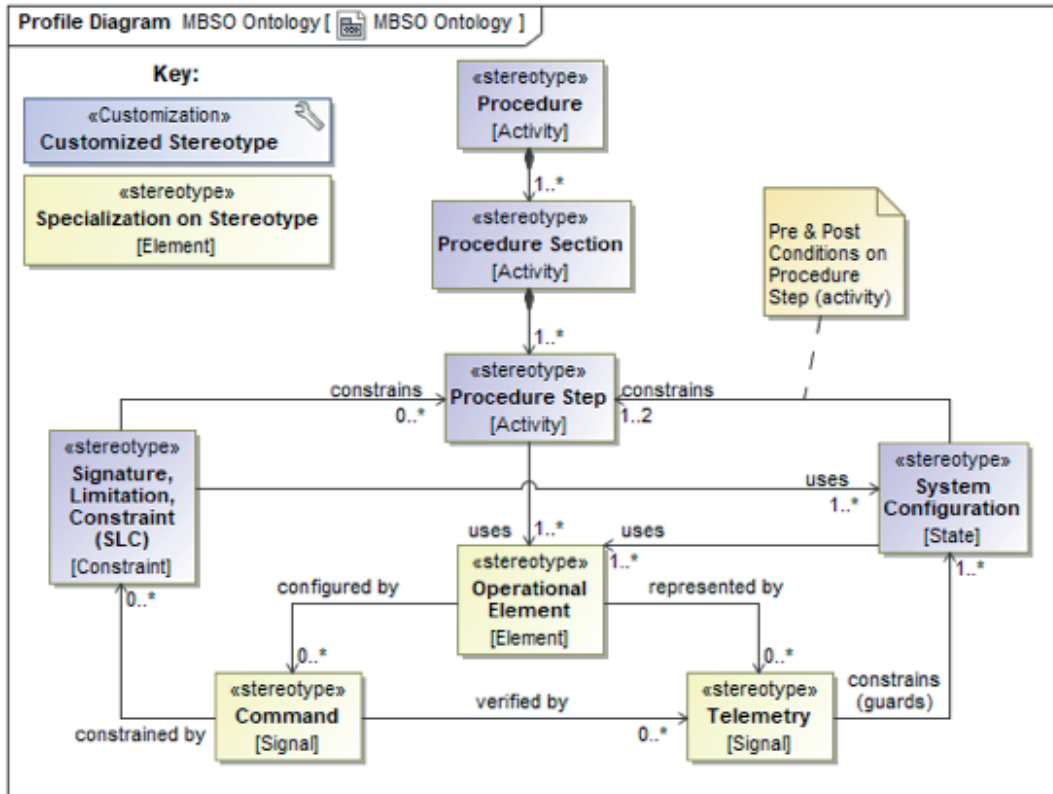


Figure 14 - MBSO Ontology Represented in SysML Profile Diagram

The MBSO elements introduced in this ontology represent support products necessary to manage and configure the state of a system within its environment. The result is a descriptive modeling framework designed specifically to support the operations and sustainment stages of a system life cycle, whereby support elements critical to the continued operation of a system can be continually and accurately maintained, updated, adapted, and verified within a digital, modeling environment throughout the system's life.

When viewing the SEDevOps life cycle graphic in Figure 10, traditional MBSE frameworks are designed to primarily support and enable the left side of the lemniscate (the system development stages) while MBSO supplements those existing frameworks to enable the right side of the lemniscate (the system operations and sustainment stages) and,

more importantly, improve the interaction/iteration between the two sides through continual adaptation and evolution of a system over its operational life. Figure 15 provides a visual of this.

According to Friedenthal et al, “Formal representation may be referred to as an ontology, a conceptual model, or a metamodel. This representation can then be used to define domain specific extensions to the language” (Friedenthal, Moore and Steiner 2014). Based on the ontology introduced in Figure 14, it is possible to build a Domain-Specific Language (DSL) to be leveraged in the implementation of SEDevOps.

3.3.2 MBSO Domain Specific Language Components

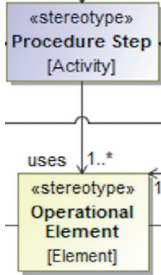
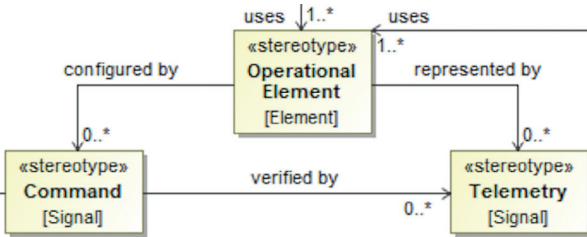

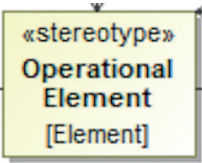
MBSO extends the base SysML profile by creating a DSL designed for extending the active use of detailed descriptive system models from the development stages into the operational stages of a system’s life cycle. This includes defining and supporting the process associated with operational support product development, testing, deployment, and maintenance, including adaptation required in response to changing environments, aging system elements, and evolving system needs. A key principle of SEDevOps shown earlier in Figure 12 is the fundamental expansion of a system’s boundary beyond the traditional physical system elements and intrinsic behaviors to more broadly include the enabling system elements and extrinsic behaviors as part of the overall integrated system model. In other words, this focuses on incorporating the products, artifacts, documents, and processes needed to operate the system within an environment into the model, thus including procedures, scripts, commands, configuration snapshots, etc. into the single-source-of-truth system model. A different way to articulate this is procedures are simplified representations of state transitions of a system requiring very methodical control, which is

implemented and exercised through a formal modeling language. The ontology presented in Figure 14 provides a representation of these key elements in the MBSO framework and the relationship between the elements in a SysML Profile Diagram.

The DSL comes from transcribing the proposed ontology into a formal SysML profile for MBSO (Castet, Rozek, et al. 2015) in order to syntactically enforce the stated associations. This is accomplished through the customizations and stereotypes of existing SysML elements identified in the “<< >>” brackets in Figure 14. These modifications to the base SysML elements enable the creation of the MBSO specific elements defined in more detail in Table 3.

Table 3 - Customized Elements in the MBSO Ontology

MBSO Element	Details	Graphical Depiction
Procedures	Accomplish a pre-defined and complete function or system transition. Composed of any number of Procedural Sections each designed to be self-contained reconfigurations usable in multiple procedures	<pre> graph TD A["«stereotype» Procedure [Activity]"] -- "1..*" --> B["«stereotype» Procedure Section [Activity]"] </pre>
Procedure Section	Perform a grouping of steps to accomplish a logical element of an overall procedure. Composed of one to many <i>Procedure Steps</i> (analogous to software microservices and ideally containerized or cleanly bounded functions) to perform singular actions	<pre> graph TD A["«stereotype» Procedure Section [Activity]"] -- "1..*" --> B["«stereotype» Procedure Step [Activity]"] B -- "col" --> C["1..1"] </pre>

MBSO Element	Details	Graphical Depiction
Procedure Step	Execute a singular action to configure one to many operational elements by executable <i>commands</i> to enact a change on the system	 <pre> classDiagram class ProcedureStep["«stereotype» Procedure Step [Activity]"] class OperationalElement["«stereotype» Operational Element [Element]"] ProcedureStep "1..*" --> "1" OperationalElement : uses </pre>
Commands & Telemetry	<i>Commands</i> configure operational elements and are verified through related <i>telemetry</i> to confirm correct system <i>configuration state</i> at any point in time	 <pre> classDiagram class OperationalElement["«stereotype» Operational Element [Element]"] class Command["«stereotype» Command [Signal]"] class Telemetry["«stereotype» Telemetry [Signal]"] OperationalElement "1..*" --> "0..*" Command : configured by OperationalElement "1..*" --> "0..*" Telemetry : represented by Telemetry "0..*" --> "0..*" Command : verified by </pre>
System Configuration States Signatures, Limitations, & Constraints	<i>System Configuration States</i> are used as pre- and post- configuration checks for any procedure step, poising the framework for eventual formal methods of procedural test and verification. Combined with “ <i>Signatures, Limitations, & Constraints</i> ” (SLCs) to allow any <i>procedural step</i> to be validated at the time of procedure construction and any time thereafter as the system model and system states change.	 <pre> classDiagram class SystemConfiguration["«stereotype» System Configuration [State]"] class SLC["«stereotype» Signature, Limitation, Constraint (SLC) [Constraint]"] </pre>
Operational Element	Stereotype applied to elements of a system model. Allows for emulating various states of system elements in the system model in order to exhibit defined behaviors when interacted with, such as in a procedure	 <pre> classDiagram class OperationalElement["«stereotype» Operational Element [Element]"] </pre>

Signatures, Limitations, & Constraints (SLCs) are a concept employed in space system development and operations documenting unique system features where, for example, commands must be sent in a particular order to avoid catastrophic results. A fully interrelated metamodel enables this automated, continual, and formal consistency and validity regression testing of procedures against current and potential system configurations and product updates. Continual validation checks of element relationships and constraints is a feature inherent in SysML modeling tools.

3.3.3 Life-In-A-Day (LIAD) Testing

Coupling the concept of executable composition, inherent in SysML, with the ability to simultaneously represent past, present, and future states of a system in its environment results in the creation of a DevOps-like environment to perform development and testing. This provides systems engineers with a tool to manage operational support through system configuration curation.

This facilitates a concept introduced here as “Life-In-A-Day” (LIAD) testing in which simulated off-nominal life events encountered throughout system operations and sustainment (such as component aging & degradation, environment evolution, etc.) can be represented as configuration properties within the system model, and impacts to the broader system (most notably, the operational support products) can be assessed, including regression and validation testing against modified states. As a result, appropriately detailed operational support products can be developed and thoroughly tested in advance to streamline the response in real-time to conceived off-nominal events. This enables traversal of the full SEDevOps Lemniscate through simulation with the ability to feedback findings for incorporation to system development. This is not unlike simulating potential future

states using a digital twin of a system however the focus here is on the management and validity of the operational support products in the simulated future state.

This concept of LIAD testing to simulate and capture system agility and adaptability is analogous to Day-In-The-Life (DITL) testing performed during traditional system verification and validation to capture and validate system performance against stated mission requirements. LIAD is designed to validate the system can continue to operate through changing environments and in response to changing system behaviors. As noted previously, with MBSE best representing the stages and artifacts on the left side of the SEDevOps life cycle and MBSO representing and enabling the stages and products on the right side, DITL testing supports the culmination of design and development of a system on the left side of the SEDevOps life cycle to prove the right system has been fielded for the intended use while LIAD testing facilitates the extension to the right side of the life cycle model to validate the operational support elements provide agility and adaptability to potential system and environment evolution, a relationship visually depicted in Figure 15. LIAD testing therefore provides validation that the system has the proper support products to ensure continued operations throughout the life of the system.

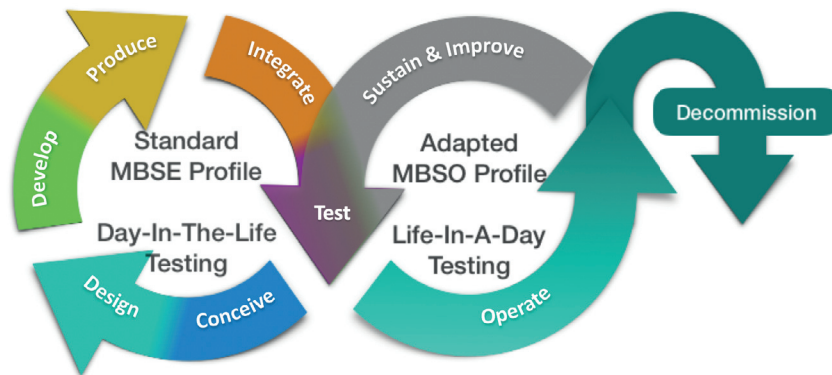


Figure 15 - MBSE and MBSO Focus in the SEDevOps Life Cycle Model

3.3.4 MBSO Modeling Methodology

This subsection outlines the modeling methodology by which to apply the MBSO framework, broken into *Modifications to System Context* followed by *Add-Ins for Operational Context*, representing the two main categories in the MBSO profile. The MBSO domain-specific customization elements, as well as a number of unique stereotypes used to identify status of operational elements, are captured in these two categories within the MBSO profile, shown in Figure 16.

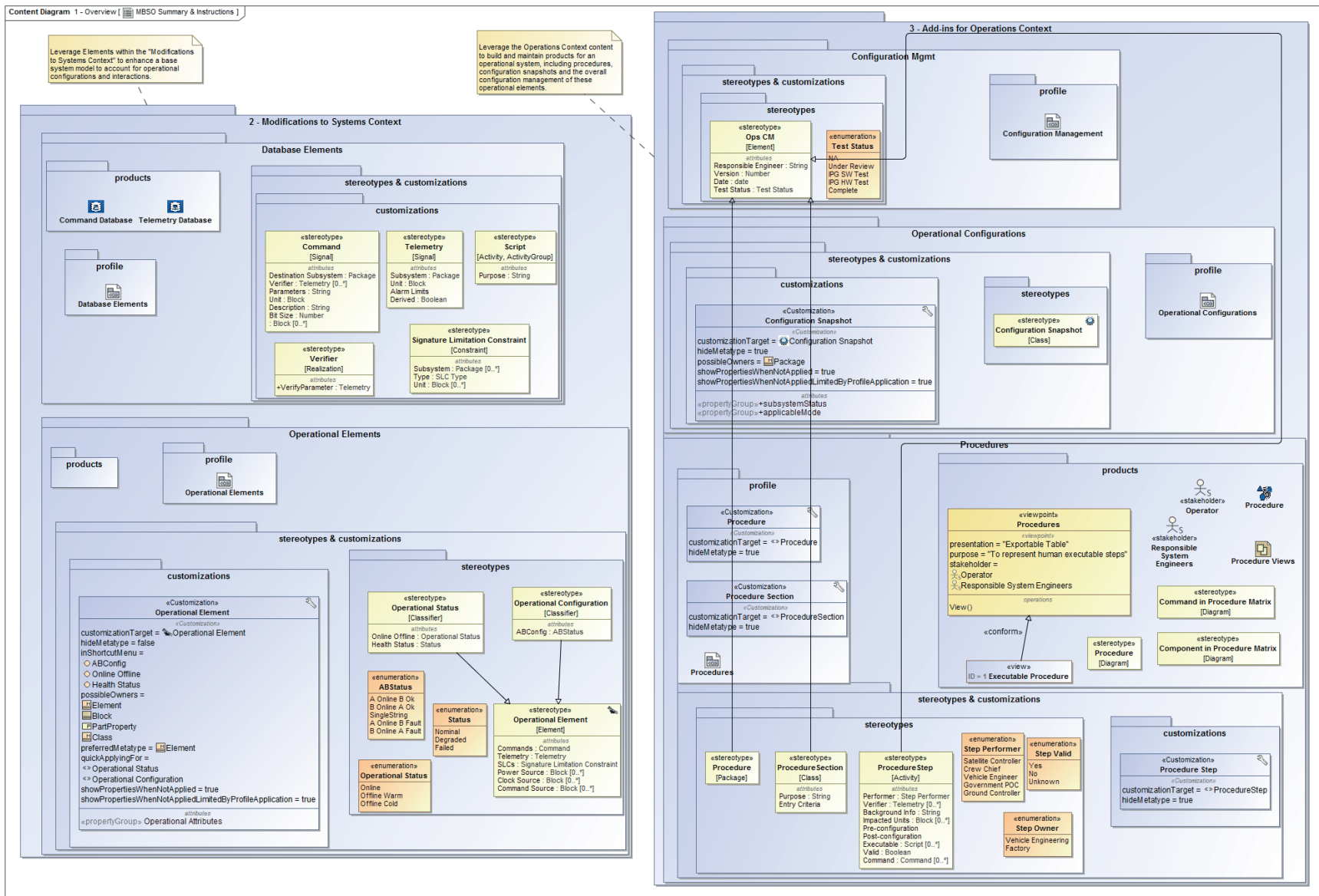


Figure 16 - View of MBSO Profile Organization

The purpose of this division is to establish a mechanism to apply MBSO to both existing and new system models in a layered approach. If a detailed SysML descriptive model is pre-existing, the MBSO “Operational Element” stereotype, depicted in Figure 17, can be applied to the existing model elements to enable building interrelated configurations within the model. This enables applying status of individual units such as “online”, “offline”, etc. and subsystems (i.e. “nominal,” “degraded,” etc.). Figure 17 is a SysML Profile Diagram representing a stereotype of the metaclass “Element” with enumerated attributes for Operational Status and Operational Configuration. The enumerated attributes are represented in the orange blocks on the bottom and referenced in the classifiers within the figure.

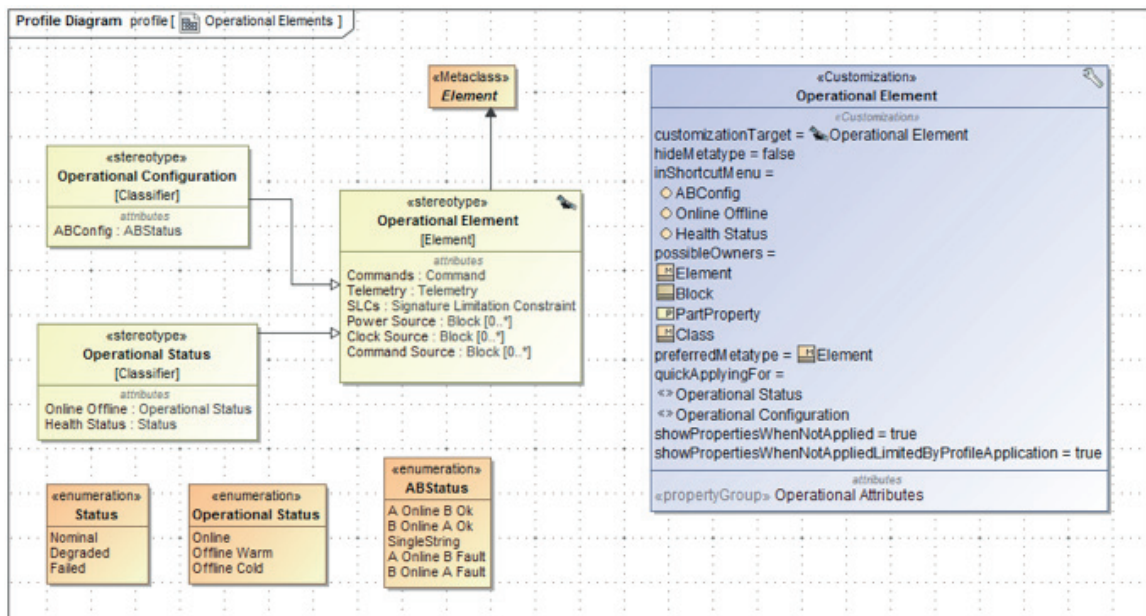


Figure 17 - MBSO Operational Element Stereotype Details in SysML Profile Diagram

Figure 18 shows the application of these attributes to physical elements (i.e. blocks) within a system model in a SysML Block Definition Diagram (bdd) and Figure 19 provides

a view of the modified quick-select menu for operational elements, used to apply status attributes to an operational element in an Internal Block Diagram (ibd). The model depicted in both Figure 18 and Figure 19 is leveraged in the use case described in Chapter 4 and represents the FireSat-II notional spacecraft architecture with its subsystems.

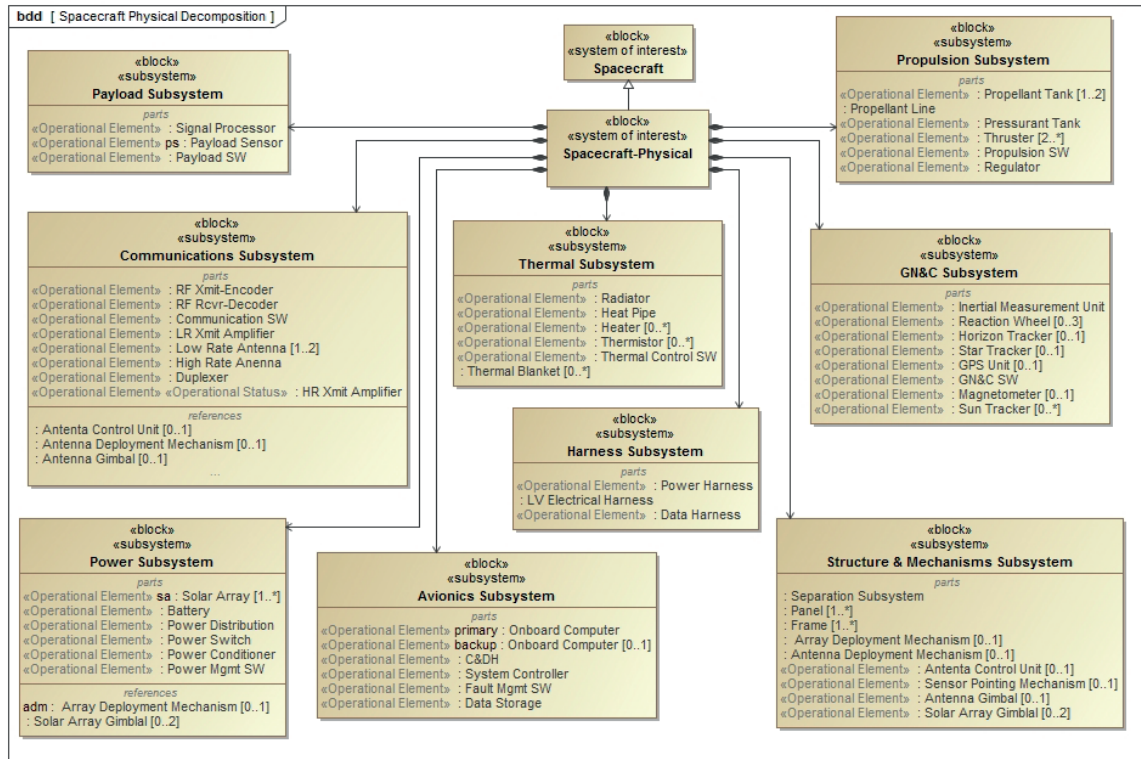


Figure 18 - MBSO Operational Element Stereotype Applied to Physical Architecture in a SysML bdd (adapted from Friedenthal 2017)

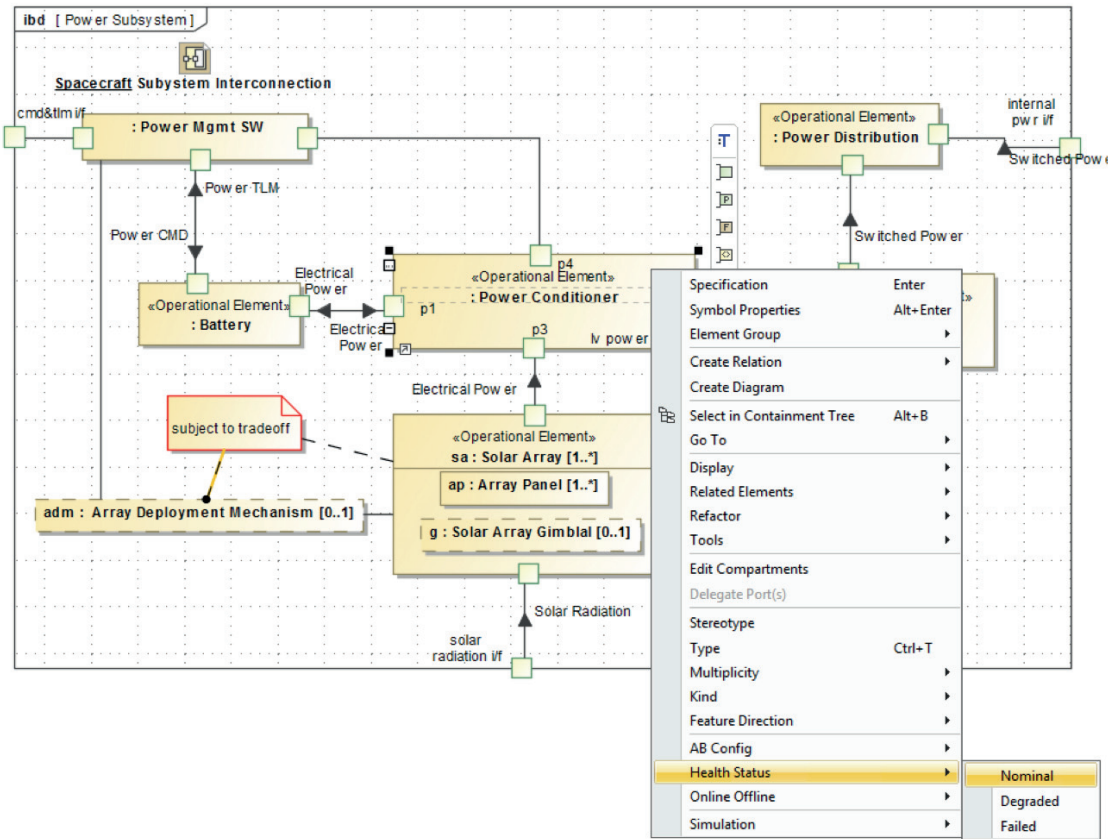


Figure 19 - Applying Operational Status Attributes in a SysML ibd (adapted from Friedenthal 2017)

If a system model does not yet exist, the first step is the development of a descriptive physical architecture model in a *System Context* to which the Operational Element stereotype can then be applied on an element by element basis. Once the system model has sufficient fidelity to represent the operational state of constituent parts, pre-existing operational procedures, commands, telemetry items, etc. can be transcribed from native formats (such as stand-alone documents or tables) into the modeling language and linked directly to the system elements in the model. The result of building operational elements within a system model is a single, interrelated source of truth to support both the development domain and the operational domain, as represented in the SEDevOps life

cycle model and as methodologically driven by DevOps principles on which SEDevOps is based.

As a note on the MBSO profile shown in Figure 16, several additional profile elements have been incorporated into the profile to support more detailed management of operational content. Examples of these additional elements include a Configuration Management stereotype and the addition of several potential actors which can be documented as performers of various procedural steps. The specific utility of these two elements was not assessed in the ensuing use case as they are examples of widely used inherent features of SysML and associated tools. These can be considered elements for refinement in future iterations of the MBSO profile.

3.3.5 MBSO Extension with Formal Methods

Extending the concept of model formalism in MBSE (Madni and Sievers 2018), MBSO is designed as both “a descriptive and executable (dynamic)” formal model. MBSO therefore lends itself to a formal methods approach to system operations based on concepts proposed by (Wang and Dagli 2014), (Huang, McGinnis and Mitchell 2019), and (Graves and Bijan 2011), discussed briefly in Section 2.6.1. Additionally, establishing a formal method of verification within a DevOps-like construct enables the applicability of this approach to high-criticality and safety-critical system management, as proposed by Olszewska and Walden (Olszewska and Walden 2015) and discussed briefly in Section 2.6.2. Implementation of Petri Net behavioral flow representations (Wang and Dagli 2014), (Huang, McGinnis and Mitchell 2019) and application to critical systems is identified as an area of future research and development.

This concept of formally verifiable operational support products is the basis for autonomic system operations as initially sought in the question posed at the onset of Chapter 2. The question: *Can systems engineering support specific to operations and maintenance/sustainment be codified in a manner to enable autonomic system operations?* drove the initial research and the identification of an opportunity and a need to extend model-based systems engineering support into system operations. The advancements introduced in this chapter, SEDevOps and MBSO, provide a foundation to codify operational support in a modeling environment and promote progress towards autonomicity in system operations. Additionally, leveraging a descriptive modeling framework as a common interface for systems engineering products throughout a system life cycle and encouraging continued collaboration between development and operations drives the discipline a step closer towards a polymorphic existence.

Chapter 4: Data Collection & Analysis

In order to demonstrate the SEDevOps model and prove or disprove the initial hypothesis regarding the utility of extending model-based systems engineering processes and products to system operations, a use case was established. As the aerospace industry continues its digital transformation progression, a spacecraft system operations scenario provides a relevant opportunity for leveraging detailed, digital, descriptive system models from system development into the operations stage. Additionally, spacecraft system operations and sustainment tasks are traditionally performed by systems engineers adding further applicability and utility to the example.

A well-known manuscript in the spacecraft engineering domain is *Space Mission Engineering: The New SMAD* by J. Wertz, D. Everett, and J. Puschell (Wertz, Everett and Puschell 2011). In this manuscript, the authors walk the readers through a detailed, multi-disciplinary spacecraft system design as a means to teach the relevant and critical steps of spacecraft design and development. The example spacecraft is called FireSat-II and, due to the broad distribution and wide acceptance of the manuscript by Wertz et al, it was an excellent and widely known starting point for this use case. Additionally, a detailed SysML model of the notional FireSat-II spacecraft has been built by S. Friedenthal as an example model for his and C. Oster's book *Architecting Spacecraft with SysML* (Friedenthal and Oster 2017). This SysML model is publicly available (Friedenthal 2017) and provided a natural and detailed basis on which operational products were built and the utility of SEDevOps and MBSO were tested.

As SEDevOps incorporates the traditional system development stages common to many existing life cycle models, the system development progression in the life cycle is

not covered specifically in the description of this use case. Instead, the FireSat-II model provided a starting point, post-system-development, in which an MBSE methodology was applied to generate the detailed model. This provided a robust point of departure for the right side of the SEDevOps lemniscate with the application of MBSO to the model in order to facilitate further simulation and assessment.

To perform this simulation and assessment, three separate scenarios are considered in this chapter to demonstrate the utility of the SEDevOps model and the application of the MBSO framework. The three scenarios are:

1. A simulated failure of a critical spacecraft hardware unit resulting in a need to reconfigure operational support products (i.e. executable command procedures) to address the transition to and continued operation on a back-up, or redundant, hardware unit.
2. A scenario included for discussion in Section 4.2.4.1 is a case in which intermittent and autonomous hardware unit toggles are present (A-side to B-Side and back, or Primary to Redundant and back). This scenario references spacecraft onboard flight computers which are susceptible to repeated space radiation-induced bit-flips and subsequent power-on-resets. In this case, operational support products (scripts, procedures, etc.) must dynamically update to address the current online and in-use unit.
3. An additional scenario provided for discussion in Section 4.2.4.2 is one in which a catastrophic failure in a spacecraft battery pack results in permanent degradation to mission capability due to limited system power. The result is a need to re-develop the system Concept of Operations (CONOPS) and

associated operational support products to address the new system constraints not initially designed into the system architecture or CONOPS.

The primary use case described here and leveraged for data collection, metrics, and analysis (number 1 in the scenario list), represents a streamlined application of SEDevOps and MBSO for the notional FireSat-II spacecraft system demonstrating the concepts, with a focus on impact assessment of operational support products to the simulated unit failure. By applying the MBSO profile to expand the FireSat-II spacecraft descriptive model in SysML, it is shown that impact assessment and maintenance of operational support products can be streamlined. A hardware reconfiguration due to a power distribution unit fault was simulated to initiate this use case and the results of procedural impact assessment are presented. This simulated hardware failure illustrates the right-side of the SEDevOps lemniscate in which an external influence during operations resulted in the need for system reconfiguration and the consequential maintenance of enabling support elements (i.e. command procedure update and test). The descriptive model used in this scenario was created using the SysML tool suite in NoMagic's Cameo Systems Modeler software and figures presented herein are exports from the model. Where figures are directly presented herein or adapted from the original FireSat-II model from Friedenthal, notation is included in the figure caption. Otherwise, figures are originals created by leveraging MBSO.

The additional use cases are included in Section 4.2.4 for discussion to articulate the utility of SEDevOps, MBSO, and the overall evolution to a more polymorphic approach to systems engineering leveraging model-based systems engineering.

4.1 Data Source: FireSat-II Modified through MBSO

4.1.1 FireSat-II Base Model

FireSat-II is a notional Earth-observing spacecraft designed to identify and track forest fires. As noted previously, the detailed and widely available FireSat-II model, engineered in *Space Mission Engineering: The New SMAD* (Wertz, Everett and Puschell 2011), was initially built in SysML, as described in *Architecting Spacecraft with SysML* (Friedenthal and Oster 2017). The base model includes traditional spacecraft and ground subsystems needed to perform an Earth-observing mission in a low-altitude orbit. The model also includes elements of the necessary ground support equipment to command and control the system as well as to interface with key stakeholders to task the system to perform a sensor collection and subsequently deliver the collected data to the appropriate destination.

The boundaries of the initial model are focused on key architectural elements in the logical data flow concept of operations (CONOPS), from a task request on the ground to a command sent to the spacecraft to data collection on-board the spacecraft to data downlinking to the ground and finally data delivery to the initial requester. This approach is considered standard in system development and omits key operational support elements in the model required to execute the stated CONOPS, such as procedures to generate and transmit commands to the spacecraft, maintenance manuals on system elements such as the ground station antennas, and material to train the operators on the system configurations and capabilities, all of which are critical to the success of the stated mission, but are not traditionally resident within a system model. Figure 20 through Figure 23 provide views of the FireSat-II model for context and to articulate the level of detail at which this use case started.

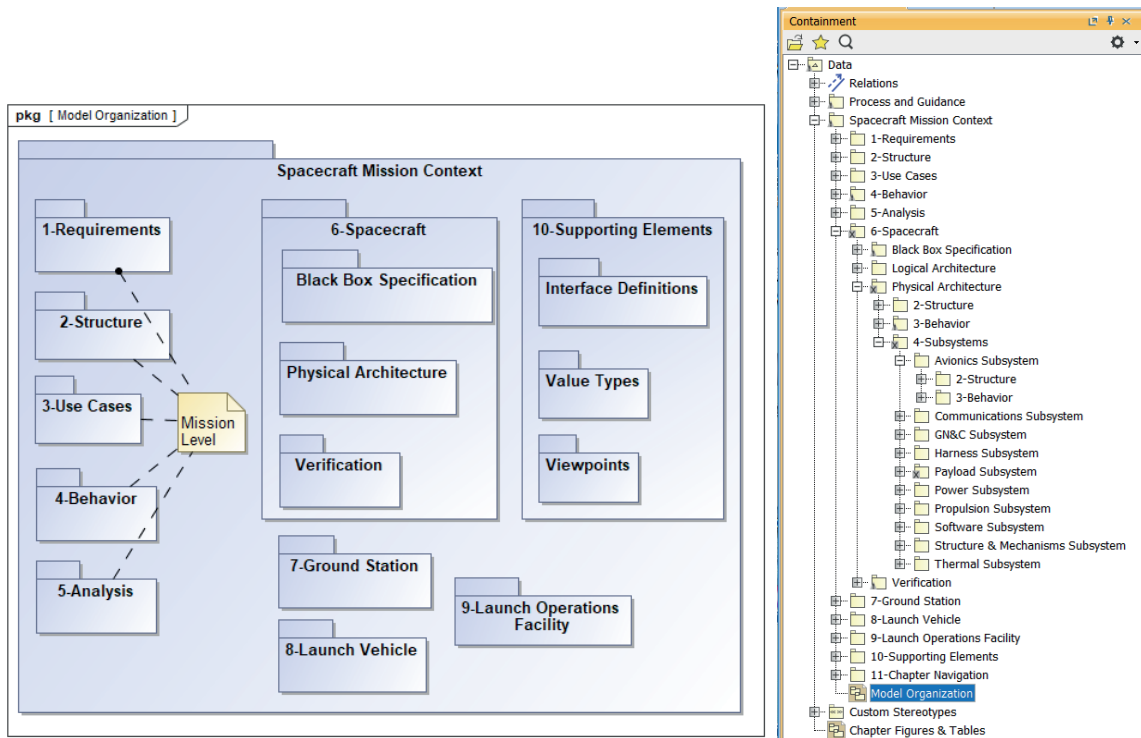


Figure 20 - FireSat-II Model Organization prior to MBSO Integration (Friedenthal 2017)

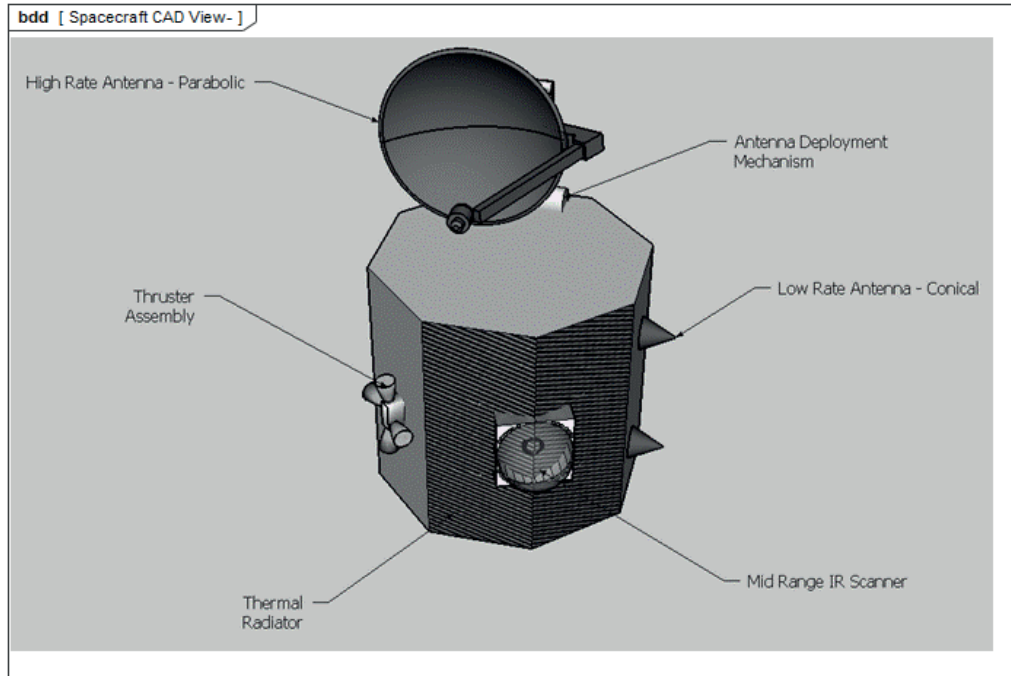


Figure 21 - FireSat-II Spacecraft (Friedenthal 2017)

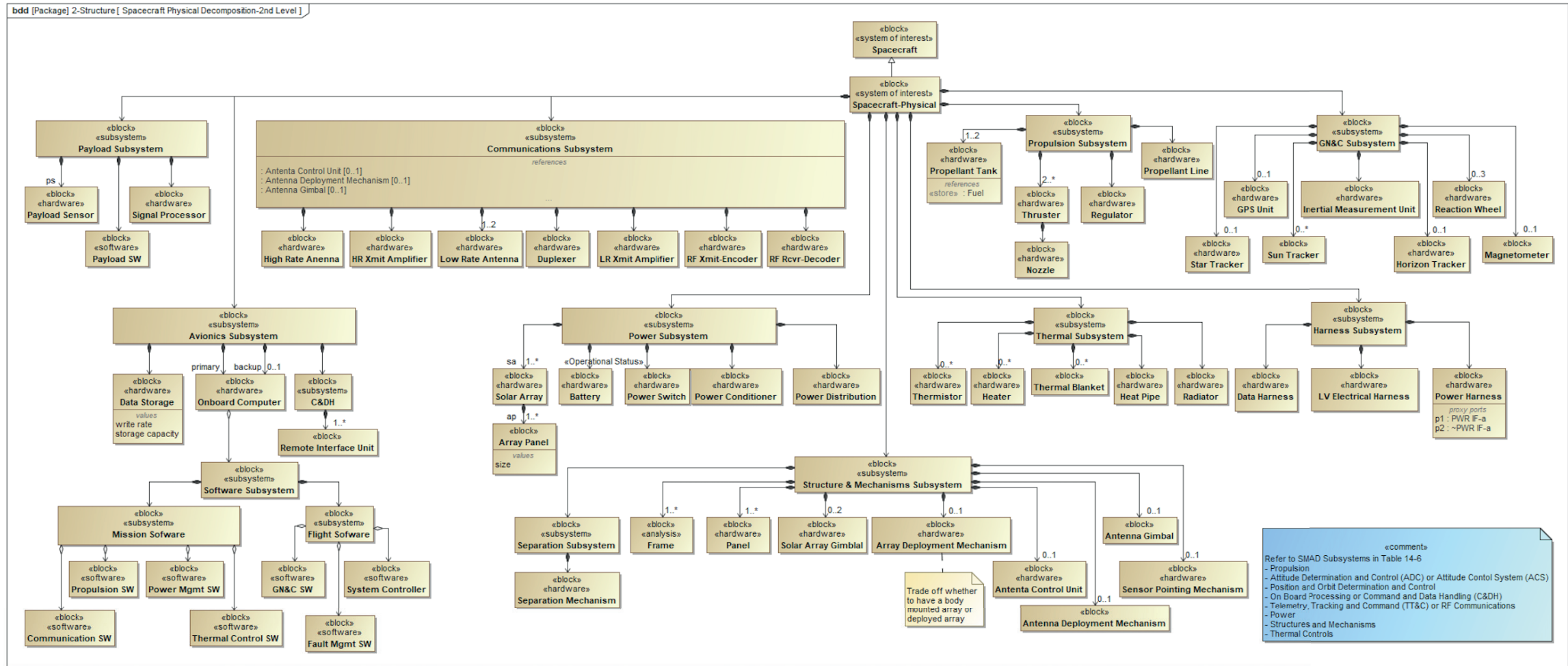


Figure 22 - FireSat-II Spacecraft Physical Decomposition, 2nd Level (Friedenthal 2017)

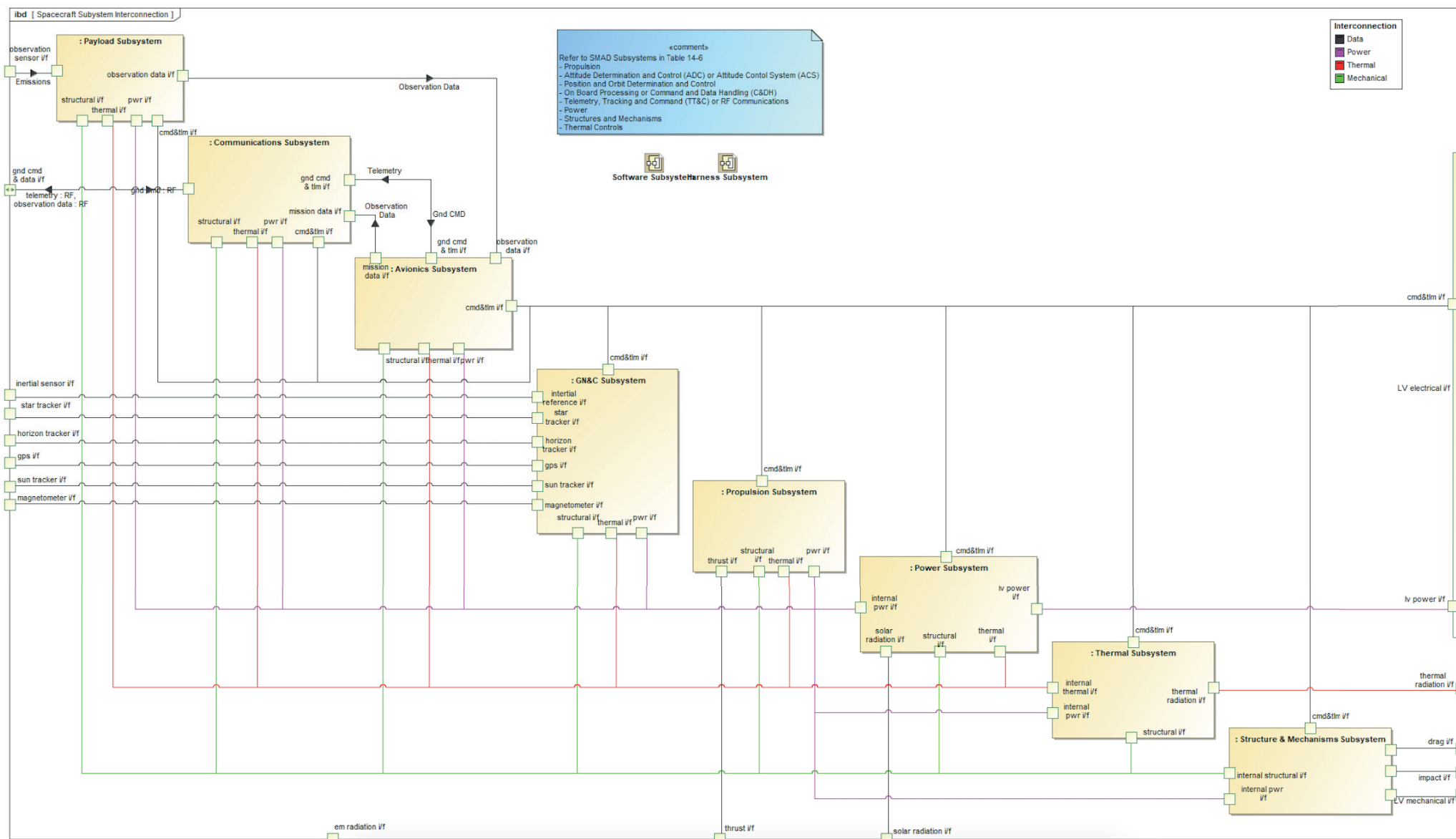


Figure 23 - FireSat-II Spacecraft Subsystem Interconnection (Friedenthal 2017)

4.1.2 MBSO Extension to FireSat-II Model

Using the MBSO profile described in Section 3.3.1 and the FireSat-II Cameo Systems Modeler “.mdzip” file (Friedenthal, Architecting spacecraft with SysML FireSat-II model 2017), MBSO stereotypes were applied to existing model elements and new customized operational elements were created using the methodology outlined in Section 3.3.4 and described step-by-step below.

1. The Operational Element stereotype was first applied to existing physical elements of the FireSat-II spacecraft architecture, seen in the SysML bdd in Figure 24. This enabled identification of relevant elements to then apply command and telemetry items to as well as to track for closer configuration management purposes, as is done on operational spacecraft system elements.
2. Next, appropriate status attributes (online, offline, nominal, degraded, etc.) were allocated to the operational elements in the internal block diagram inset to Figure 24 using the MBSO Modifications to System Context features and leveraging the options added through the MBSO profile to the quick-select menu on the SysML block definition diagrams and internal block diagrams.

As a note, several additional stereotypes are present in the block definition diagram in Figure 24 and were pre-applied in the FireSat-II model during the prior stages of system design/development. These stereotypes are used to properly define and structure the provided spacecraft model and include “system of interest” and “subsystem” stereotypes.

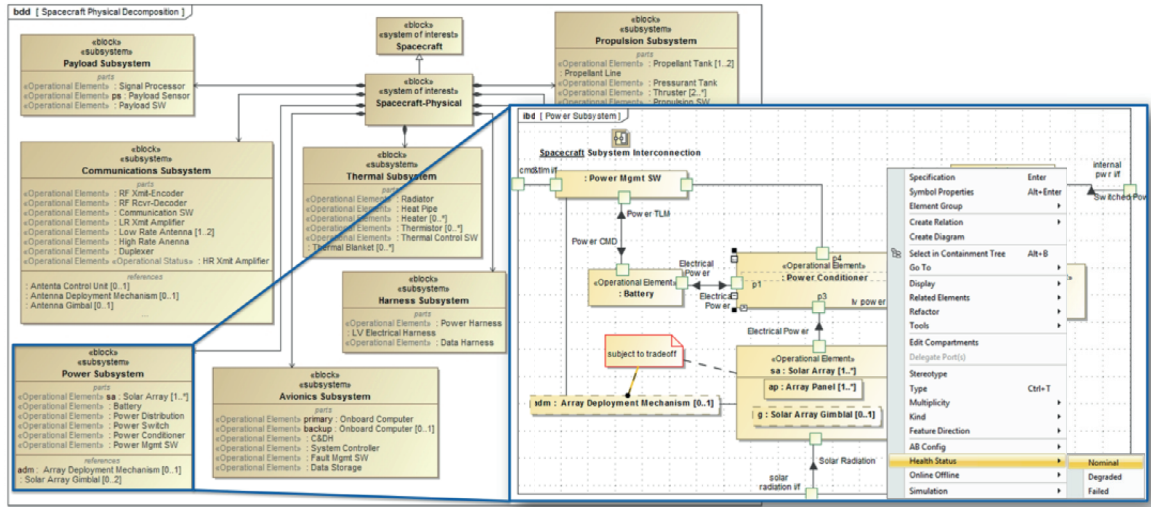


Figure 24 - MBSO Operational Element Stereotype Applied to Existing Physical Architecture in SysML bdd and ibd (adapted from Friedenthal 2017)

3. Leveraging the MBSO profile and the additional customizations and stereotypes introduced earlier in the MBSO Ontology, Figure 14, a new SysML diagram, “Command Database” with notional hardware-specific configuration commands was created and linked (i.e. associated) directly to physical elements of the FireSat-II model. These notional spacecraft commands were generated specifically for this use case and were assigned to one or more relevant hardware units, destination subsystem(s), configuration parameters, and telemetry verifiers. Figure 25 provides a view of the command table generated for this scenario. For context, command databases of this nature are not traditionally included in descriptive system models. With the advancements in MBSE and the call-to-action for digital transformation within the systems engineering domain, there will be a widespread shift to capture this level of data in the system model. MBSO provides a framework to initiate this transition.

4. Next, an additional new SysML diagram, “Telemetry Database” with telemetry parameters, used to verify the state of hardware units and ultimately the success of commands and the reconfiguration actions initiated, was created to represent notional data points to monitor for status using the MBSO Add-Ins for Operational Context telemetry stereotype. Figure 26 shows an excerpt from the telemetry database created for this scenario. Similar to the note on command databases, telemetry databases are not traditionally incorporated in descriptive system models.
5. Following creation of both command and telemetry databases, elements from each were linked (“allocated”) together through a “verify” relationship which dictates specific telemetry elements as the source of verification of command implementation and the actions they invoke within the associated hardware units. Figure 27 depicts a SysML dependency matrix used to perform the linkage as well as to provide a quick visualization artifact of interrelations between command and telemetry items.

#	△ Name	◇ Description	◇ Verifier	◇ Parameters	◇ Destination Subsystem	◇ Unit
1	downlink.highrate.transmit.OFF	High Rate Xmit Amplifier OFF	downlink.highrate.transmit.amplifier.ONOFF		Power Subsystem	HR Xmit Amplifier
2	downlink.highrate.transmit.ON	High Rate Xmit Amplifier ON	downlink.highrate.transmit.amplifier.ONOFF		Power Subsystem	HR Xmit Amplifier
3	eps.pwrdist.AoffBon	Toggle Power Converter and Distribution from A to B	eps.pwrdist.Online		Power Subsystem	Power Distribution
4	eps.pwrdist.BoffAon	Toggle Power Converter and Distribution from B to A	eps.pwrdist.Online		Power Subsystem	Power Distribution
5	fsw.gnc.attitudecontrol.DISABLE	Disable GNC attitude control	fsw.gnc.attitudecontrol.ENDIS		Software Subsystem	GN&C SW
6	fsw.gnc.attitudecontrol.ENABLE	Enable GNC attitude control	fsw.gnc.attitudecontrol.ENDIS		Software Subsystem	GN&C SW
7	fsw.gnc.attitudeprocessing.DISABLE	Disable GNC attitude processing	fsw.gnc.attitudeprocessing.ENDIS		Software Subsystem	GN&C SW
8	fsw.gnc.attitudeprocessing.ENABLE	Enable GNC attitude processing	fsw.gnc.attitudeprocessing.ENDIS		Software Subsystem	GN&C SW
9	gnc.gps.conv	Control GPS Unit power source selection	gnc.gps.conv.sel	ConvA ConvB	GN&C Subsystem	GPS Unit
10	gnc.gps.power	Control GPS Unit On/Off status	gnc.gps.power	ON OFF	GN&C Subsystem	GPS Unit
11	gnc.hor.power	Control Horizon Tracker On/Off Status	gnc.horizon.power	ON OFF	GN&C Subsystem	Horizon Tracker
12	gnc.horizon.conv	Control Horizon Tracker power source selection	gnc.horizon.conv.sel	ConvA ConvB	GN&C Subsystem	Horizon Tracker
13	gnc.imu.conv	Control IMU power source selection	gnc.imu.conv.sel	ConvA ConvB	GN&C Subsystem	Inertial Measurement Unit
14	gnc.imu.power	Control IMU On/Off status	gnc.imu.power	ON OFF	GN&C Subsystem	Inertial Measurement Unit
15	gnc.magnetometer.conv	Control Magnetometer power source selection	gnc.magnetometer.conv.sel	ConvA ConvB	GN&C Subsystem	Magnetometer
16	gnc.magnetometer.power	Control Magnetometer On/Off status	gnc.magnetometer.power	ON OFF	GN&C Subsystem	Magnetometer
17	gnc.rwa.num.conv	Control Reaction Wheels power source selection (1, 2, & 3)	gnc.rwa.1.conv.sel gnc.rwa.2.conv.sel gnc.rwa.3.conv.sel	Wheel # ConvA ConvB	GN&C Subsystem	Reaction Wheel
18	gnc.rwa.num.power	Control Reaction Wheels On/Off status (1, 2, & 3)	gnc.rwa.1.power gnc.rwa.2.power gnc.rwa.3.power	Wheel # ON OFF	GN&C Subsystem	Reaction Wheel
19	gnc.sta.conv	Control Star Tracker power source selection	gnc.sta.conv.sel	ConvA ConvB	GN&C Subsystem	Star Tracker
20	gnc.sta.power	Control Star Tracker On/Off status	gnc.sta.power	ON OFF	GN&C Subsystem	Star Tracker
21	gnc.sun.conv	Control Sun Tracker power source selection	gnc.sun.conv.sel	ConvA ConvB	GN&C Subsystem	Sun Tracker
22	gnc.sun.power	Control Sun Tracker On/Off status	gnc.sun.power	ON OFF	GN&C Subsystem	Sun Tracker

Figure 25 - Notional FireSat-II Command Database created with MBSO Profile Elements, in a Customized SysML Table Diagram

#	Name	Subsystem	Unit	Alarm Limits	Derived
1	downlink.highrate.transmit.amplifier.ONOFF	Communications Subsystem	HR Xmit Amplifier	OFF = Alarm ON = Normal	<input type="checkbox"/> false
2	fsw.gnc.attitudecontrol.ENDIS	Software Subsystem	GN&C SW	ENABLE = Normal DISABLE = Alarm	<input type="checkbox"/> false
3	fsw.gnc.attitudeprocessing.ENDIS	Software Subsystem	GN&C SW	ENABLE = Normal DISABLE = Alarm	<input type="checkbox"/> false
4	eps.pwrdist.Online	Power Subsystem	Power Distribution	A Online = Normal B Online = Alarm	<input type="checkbox"/> false
5	eps.pwrdist.Voltage	Power Subsystem	Power Distribution	4.8 - 5.2V = Normal < 4.8V = Alarm > 5.2V = Alarm	<input type="checkbox"/> false
6	gnc.gps.conv.sel	GN&C Subsystem	GPS Unit	A = Normal B = Alarm	<input type="checkbox"/> false
7	gnc.gps.power	GN&C Subsystem	GPS Unit	ON = Normal OFF = Alarm	<input type="checkbox"/> false
8	gnc.sta.conv.sel	GN&C Subsystem	Star Tracker	A = Normal B = Alarm	<input type="checkbox"/> false
9	gnc.sta.power	GN&C Subsystem	Star Tracker	ON = Normal OFF = Alarm	<input type="checkbox"/> false
10	gnc.sun.conv.sel	GN&C Subsystem	Sun Tracker	A = Normal B = Alarm	<input type="checkbox"/> false
11	gnc.sun.power	GN&C Subsystem	Sun Tracker	ON = Normal OFF = Alarm	<input type="checkbox"/> false
12	gnc.imu.conv.sel	GN&C Subsystem	Inertial Measurement Unit	A = Normal B = Alarm	<input type="checkbox"/> false
13	gnc.imu.power	GN&C Subsystem	Inertial Measurement Unit	ON = Normal OFF = Alarm	<input type="checkbox"/> false
14	gnc.horizon.conv.sel	GN&C Subsystem	Horizon Tracker	A = Normal B = Alarm	<input type="checkbox"/> false

Figure 26 - Notional FireSat-II Telemetry Database created with MBSO Profile Elements, in a Customized SysML Table Diagram

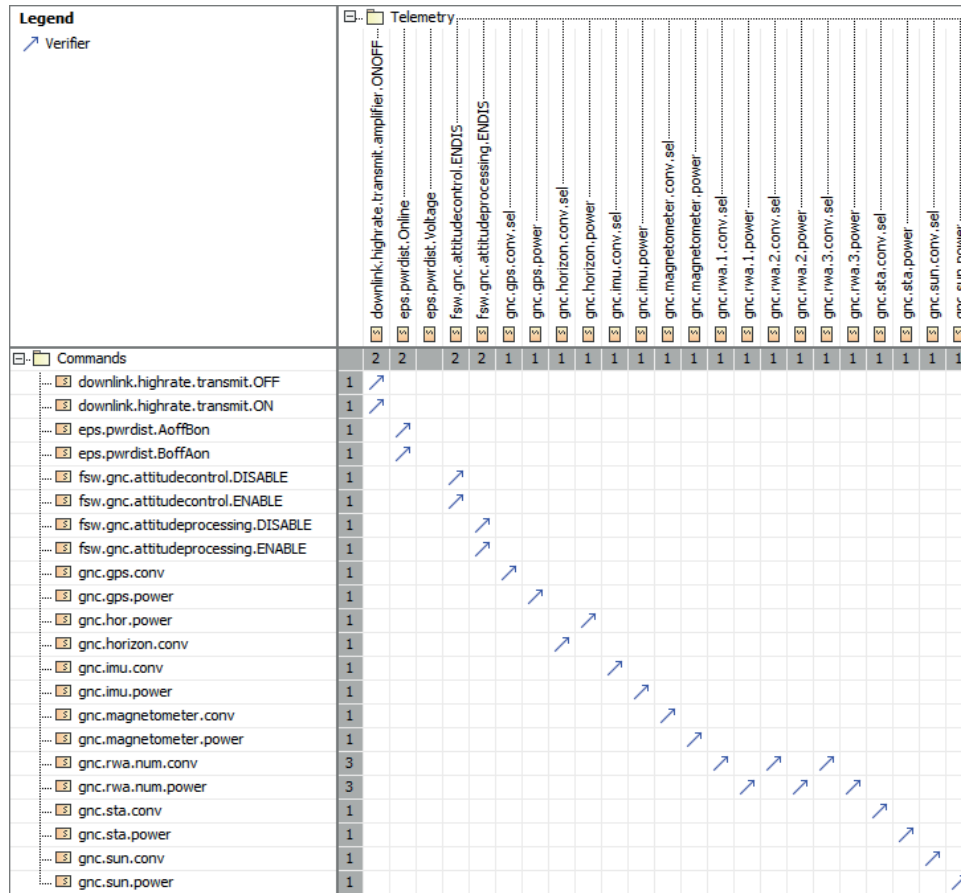


Figure 27 - FireSat-II Notional Command & Telemetry Element Interrelation in a SysML Dependency Matrix

- Finally, the SysML Block Definition Diagrams in Figure 28 and Figure 29 show the elemental relationship created between physical hardware elements in the FireSat-II system and the command and telemetry elements created using the MSBO profile elements. The orange <<command>> and <<telemetry>> elements represent the uniquely created content for this simulation and align to the previously mentioned command and telemetry database visualizations (Figure 25 and Figure 26, respectively).

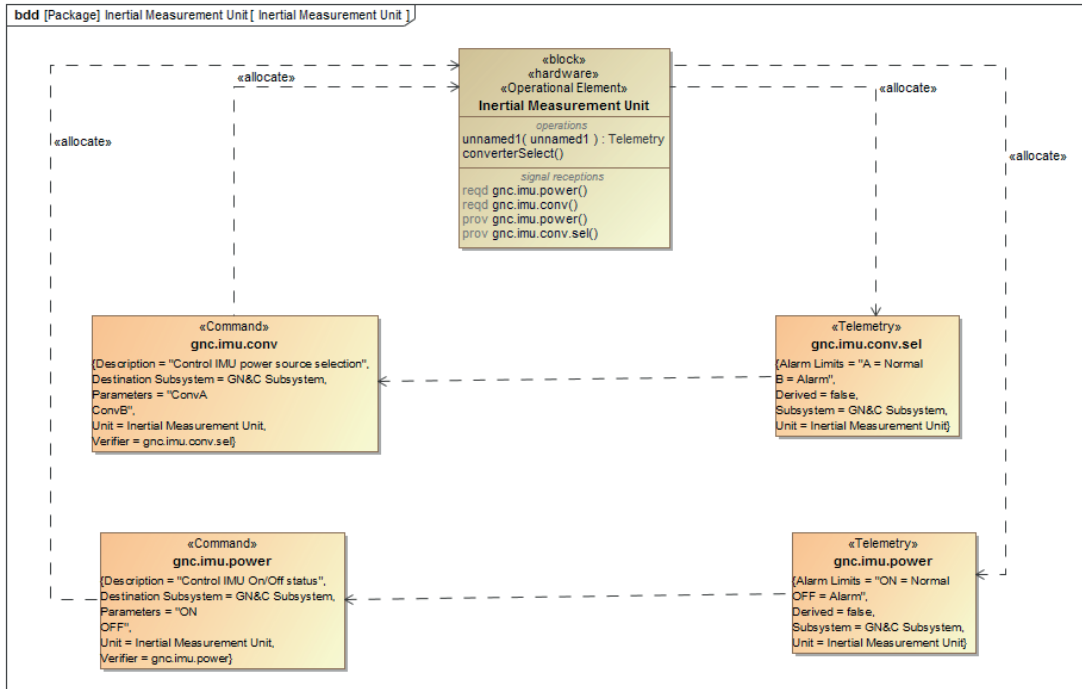


Figure 28 - Command & Telemetry Interrelationship to Inertial Measurement Unit in SysML Block Definition Diagram

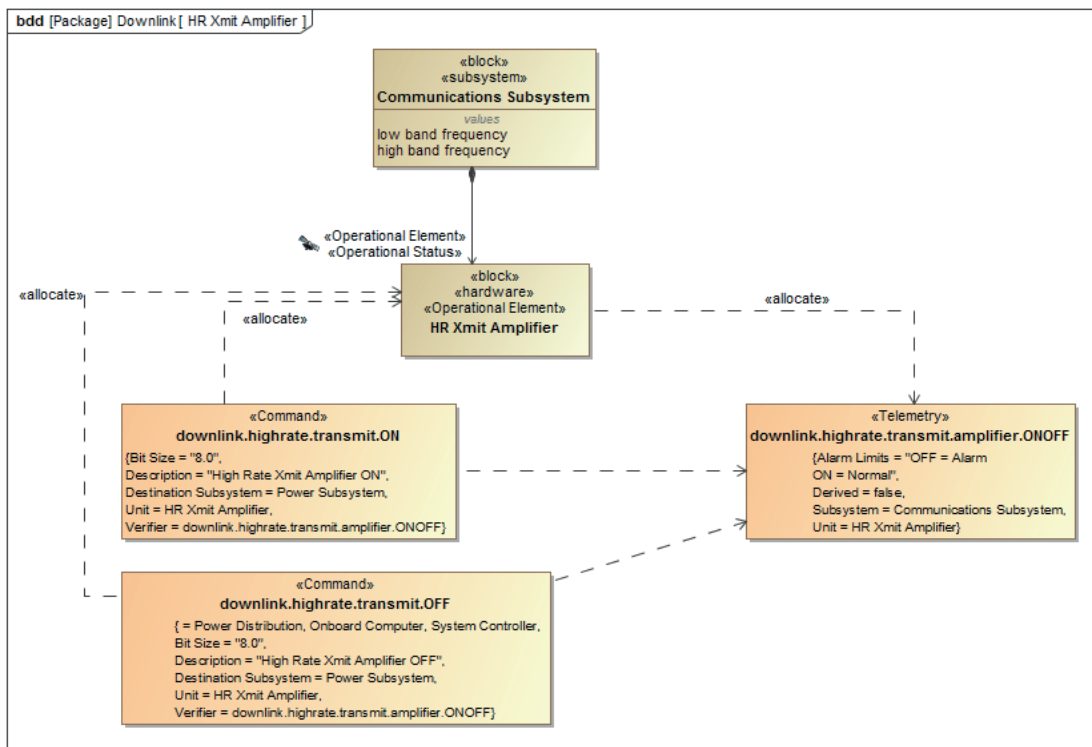


Figure 29 - Command & Telemetry Interrelationship to High Rate (HR) Transmit Amplifier Unit in SysML Block Definition Diagram

Based on the steps outlined in this section, the MBSO-modified FireSat-II model was in a state ready for the generation of sample operational procedures to be leveraged in the use case to follow. For comparison purposes with Figure 20, Figure 30 below is a representation of the FireSat-II model after incorporation of the MBSO profile and generation of associated model elements.

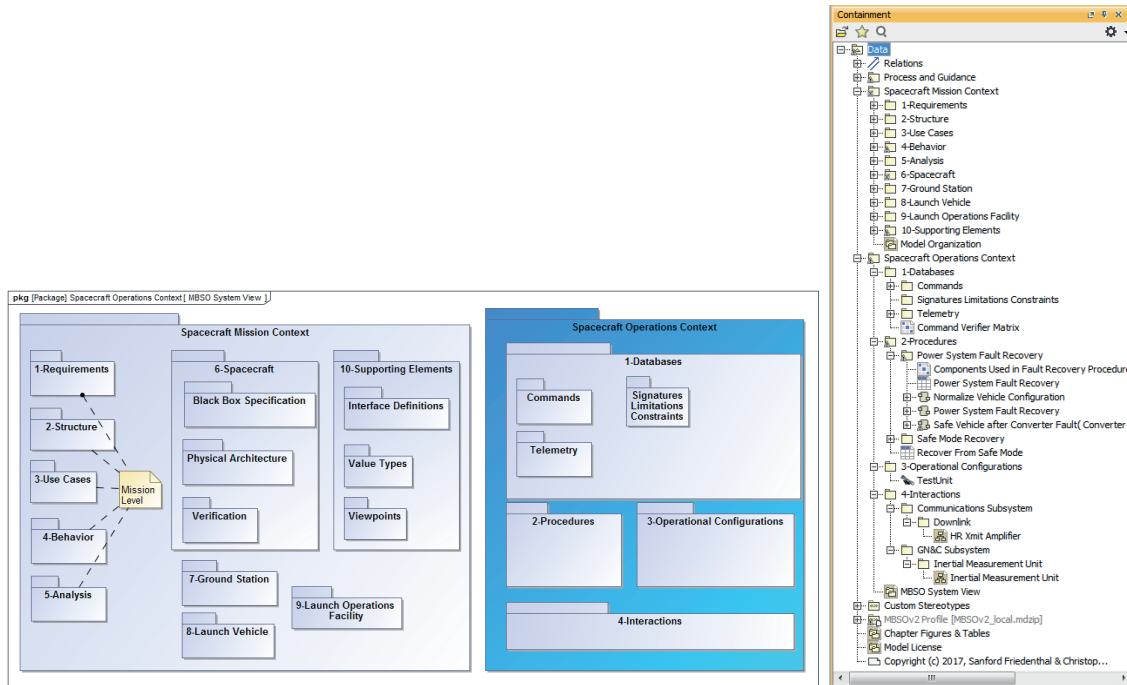


Figure 30 - FireSat-II Model Organization Following MBSO Integration (adapted from Friedenthal 2017)

4.2 Use Case: Life-In-A-Day Simulation & Response

4.2.1 Impact to Critical Unit Failure & Simplified Ops Product Roll-Out

To fully configure the FireSat-II model for use in this simulation, representative operational procedures were created, using the procedure, procedure section, and procedure step customizations introduced in the MBSO ontology in Figure 14, to characterize the basic response to a failed power distribution unit and the ensuing transition to a redundant

unit as the new “online” and in-use unit. Figure 31 provides a SysML activity diagram view of the command procedure generated for this scenario and Figure 32 provides a customized MBSO table view of the same procedural steps.

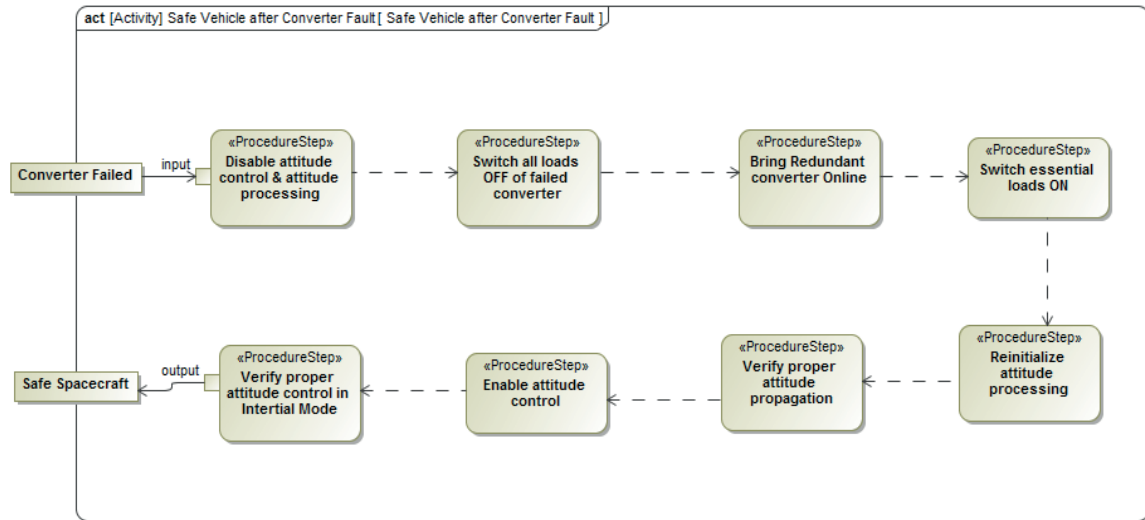


Figure 31 - SysML Activity Diagram representing the Flow of Actions for Command Procedure to Reconfigure Power Loads for Failed Converter Unit

#	Name	Purpose	Performer	Command	Verifier	Impacted Units
1	Safe Vehicle after Converter Fault	Confirm isolated power converter fault and toggle to redundant co	Satellite Controller			
2	Disable attitude control & attitude processing	Limit propagation of errors in GNC SW by disabling further attitude	Satellite Controller	fsw.gnc.attitudecontrol.DISABLE	fsw.gnc.attitudecontrol.ENDIS fsw.gnc.attitudeprocessing.ENDIS	Onboard Computer Reaction Wheel GN&C SW Fault Mgmt SW Software Subsystem
3	Switch all loads OFF of failed converter	Toggle units off of secondary power converter in preparation for c	Satellite Controller	gnc.gps.power gnc.gps.conv gnc.horizon.conv gnc.imu.conv gnc.magnetometer.conv gnc.rwa.num.conv gnc.sta.conv gnc.sun.conv gnc.hor.power gnc.imu.power ...	gnc.gps.conv.sel gnc.gps.power gnc.horizon.conv.sel gnc.imu.conv.sel gnc.magnetometer.conv.sel gnc.rwa.1.conv.sel gnc.rwa.2.conv.sel gnc.rwa.3.conv.sel gnc.sta.conv.sel gnc.sun.conv.sel ...	Power Conditioner Power Distribution Power Switch GPS Unit Horizon Tracker Inertial Measurement Unit Magnetometer Reaction Wheel Star Tracker Sun Tracker ...
4	Bring Redundant converter Online	Swap power converters to redundant unit	Satellite Controller	eps.pwrdist.BoffAon	eps.pwrdist.Online	Power Distribution Power Switch
5	Switch essential loads ON	Swith essential attitude sensing & control units onto new converter	Satellite Controller	gnc.rwa.num.conv gnc.imu.conv gnc.sun.conv gnc.rwa.num.power gnc.imu.power gnc.sun.power	gnc.imu.conv.sel gnc.rwa.1.conv.sel gnc.rwa.2.conv.sel gnc.rwa.3.conv.sel gnc.sun.conv.sel gnc.imu.power gnc.rwa.1.power gnc.rwa.2.power gnc.rwa.3.power gnc.sun.power	Power Conditioner Power Distribution Power Switch Inertial Measurement Unit Sun Tracker Reaction Wheel
6	Reinitialize attitude processing	Reenable attitude processing once sensors are powered. Reinitiali	Satellite Controller	fsw.gnc.attitudeprocessing.ENABLE	fsw.gnc.attitudeprocessing.ENDIS	

Figure 32 - Customized SysML Table Diagram representing the Command Procedure to Reconfigure Power Loads for Failed Converter Unit

In generating the procedural representation, several redundant/back-up hardware units were created and incorporated into the existing FireSat-II physical architecture model, enabling linkage and simulation of a broader system reconfiguration in response to this simulated fault.

In order to quickly and visually capture the impact assessment to operational procedure changes necessary to account for the simulated hardware failure, a SysML Dependency Matrix was generated. This view shows the specific elemental interrelation between procedural step, command, telemetry, subsystem, and hardware unit, seen in Figure 33, therefore establishing a dynamic, real-time mechanism for tracking all procedural steps interacting with the power distribution hardware unit and which procedural steps require updating for use of the back-up, or redundant, unit.

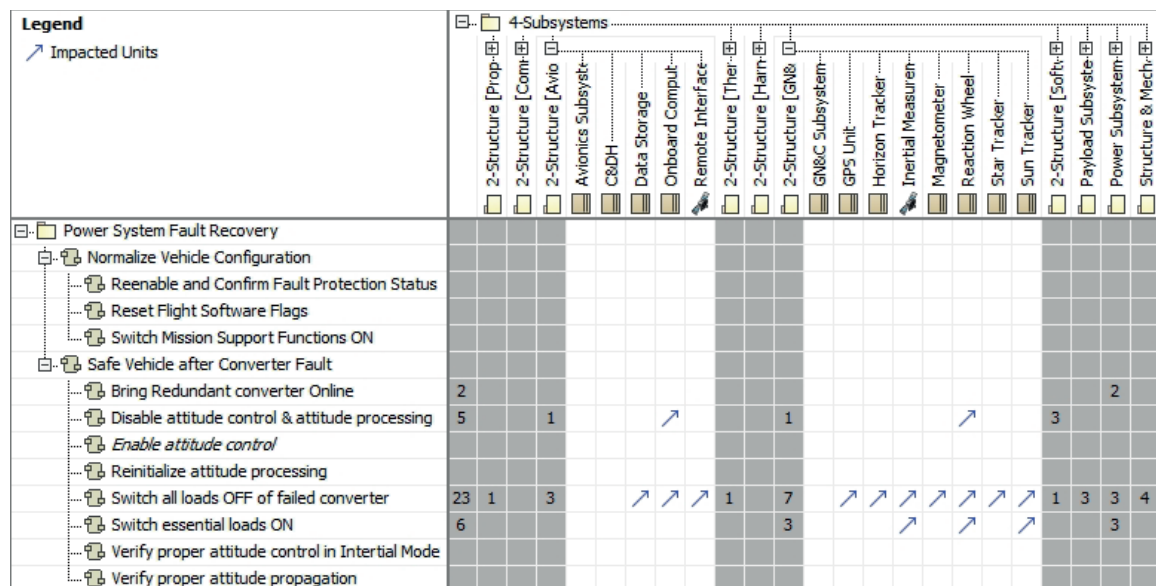


Figure 33 - FireSat-II Procedural Step Dependency Matrix in SysML

Configuration of the model in this way facilitated exercising a sample LIAD test case by assessing the impact of the primary power distribution unit failure throughout the necessary support elements. This method of inherent traceability in a model simplifies the review of procedures, scripts, and commands for necessary changes. The result was the identification of numerous “mis-configurations” in the support products with the ability to quickly correct the necessary procedure steps.

In this manner, the update to any number of procedures, scripts, and commands can be identified and corrected with a small number of touch points. In contrast to practices without interrelated, model-based representations, this process is a task performed by the system operations team to search all products for impacted procedural steps, assess the impact, propose and review the required updates, test the updates for accuracy and completeness for the new configuration, regression test related procedural sections to ensure no inadvertent misconfigurations were introduced, and deploy the change to the operational system. MBSO provides the ability to perform this process with minimal touch points through established inter-relationships and re-use of common steps, elements, and pre/post-configuration constraints isolating procedural steps in a manner similar to containerized microservices in a software suite. The result is a modeling tool designed to support and enable the sustainment and adaptation described in Chapter 3 and depicted visually in the SEDevOps lemniscate seen in Figure 10.

The example provided was a simplified adaptation of existing operational support products (i.e. procedures) where the resolution was easily identifiable (swap a primary A-Side unit for a redundant B-Side unit and update procedures to address the new unit) and did not require a more thorough concept, design, develop, integrate, test cycle. More

complex scenarios are introduced in the following subsections for discussion and comparison purposes.

4.2.2 Discussion on Cost & Implications

Life cycle costs continue to be a focus of attention. NASA, for example, identifies spacecraft operations costs as an area of increasing concern resulting in significant looks into ways to reduce manual support and increase automation to reduce overall sustainment costs (Truskowski, et al. 2009). To generate a relative metric to assess cost savings through the use of MBSO for an operational system, the example fault response provides a starting point. Leveraging the FireSat-II spacecraft and simulating a power distribution unit fault in the manner presented, it is possible to identify 23 units in the spacecraft which directly source power from the failed unit. As a note, in order to establish this level of data for this use case, the FireSat-II model was modified to incorporate additional lineage for critical interfaces for hardware units, including power (relevant to this example), command paths, and telemetry paths.

Based on this additional detail, for each procedure where the downstream units are commanded or powered on/off (i.e. interface with the power distribution unit), there exists a procedural step, script, and command to address and configure the unit appropriately. For the purposes of this comparative cost exercise, it is assumed this is covered by 3 procedures at a minimum, one for system start-up (i.e. power-on upon initialization or following a low bus voltage event), one for a contingency response to a power system fault, and one for system shut-down. In each of these procedures, there exist at a minimum 23 occurrences of steps addressing the failed distribution unit requiring an update, one for each unit sourcing power from the failed unit. This results in roughly 70 procedural updates

traditionally performed manually by a systems engineer providing operations support. In addition to the direct procedural updates, the changes require testing, review, and deployment/release for use. If a notional unit of effort is assigned for each single touch labor point, it is possible to generate a relative labor ratio between a traditional, manual approach and the MBSO approach. For this exercise, the manual modification of each step is designated as a touch point. The testing of all changes in a single procedure is grouped into one unit of effort as a single procedural test and similarly the review of one procedure is grouped into a single unit, and finally the deployment of one procedure is designated as a single unit. This results in 69 procedure step updates, 3 procedure tests, 3 procedure reviews, and 3 procedure deployments for a total of 78 notional “units” of touch-labor effort. In comparison, leveraging MBSO results in the following interactions with the system model:

1. Acknowledge the hardware failure by modifying the status of the online power distribution unit as a configuration change in the model (i.e. toggle online/offline status of the unit).
2. Review a predefined dependency matrix in the model for identification of impacted procedural steps in which the power distribution unit is referenced, as seen in Figure 33.
3. Select the now online unit as the destination unit which, through element relationships in the model, is propagated to each affected step in each procedure.

4. Deploy the updated procedures to the operational system (traditionally a configuration controlled repository) through an export or “publish” action from the system model.

Implicit in the stated MBSO steps is the immediate propagation of element relationships and updates (creating the ability for continual comparison of procedures against the system configuration and resultant regression and validation testing) which resulted in the noted impacts. The result is a comparison of units of manual touch-labor effort of almost 80 for the traditional manual approach vs. 4 for the MBSO approach, or a nearly 20 to 1 reduction in required interactions and therefore inferred level of effort.

While this is a notable reduction in the level of interaction required to identify, test, and deploy a change, it does not consider the level of effort required to build this capability into a system. There is, inherently, non-recurring engineering required to transition existing, non-model-based operational products into a MBSO model. There is also a non-zero level of effort required to create operational support products in MBSO during system development, however this may be deemed equivalent to the level of effort required to initially generate the products in traditional formats such as documents, tables, and spreadsheets. Lastly, there is a learning curve associated with SysML and related profiles which comes with both a labor impact to gain familiarity as well as an organizational hurdle for general acceptance of a new practice. Understanding there is sometimes drastic change required to implement a model-based approach, many MBSE texts devote sections and even chapters to documenting ways to best navigate the stigma associated with this type of change (Friedenthal, Moore and Steiner 2014). This is identified as a potential challenge to implementation in a discussion on challenges in Section 5.4.3.

4.2.3 Use Case Summary

The example provided was a straightforward modification of existing system support products where the resolution was easily identifiable and did not require a more in-depth systems engineering effort for a conceive, design, develop, integrate, and test cycle. Viewing the SEDevOps lemniscate, this involved the “Operate” stage where the simulated failure occurred, followed by the “Sustain/Improve” stage where the impacted procedures were improved to account for operations on the redundant unit. Finally, prior to releasing the updated procedures into operations, they would require Testing, at the center of SEDevOps.

A properly and thoroughly constructed descriptive model supports continuous regression and validation testing enabling the updated procedures to be validated against thorough constraint models and available to execute upon completion of the updates in the model. As a note, the use case provided here was an early validation of MBSO to a probable scenario. As described in Section 5.3, further research and development is warranted to expand the model to include detailed system configuration state constraint enforcement, automated procedure updates, continuous validation and regression testing, and ultimately a formal methods application for change verification.

4.2.4 Discussion on Additional Use Cases

In addition to the adaptation represented by a one-time failure and update to corresponding operational support products in the previous scenario, this automated impact assessment and streamlined product update can be applied to other types of adaptations required during the operational life of a system, such as for recurring system configuration

toggles and the exploration and development of mission utility expansion, both described here.

4.2.4.1 Managing Recurring Unit Toggles Use Case – A Candidate for Formal Methods Application

Recurring configuration toggles can occur in a system with built-in fault protection designed to respond to an off-nominal scenario by toggling, or “failing-over” to a spare unit to isolate the identified problem and continue system operations with limited impact. For example, on a spacecraft, single-event upsets (i.e. bit-flips) in on-board computer memory due to the harsh radiation environment in space can result in an automated fail-over to a back-up computer. Upon reboot, the originally affected computer memory is refreshed to clear the undesired bit-flips, deemed functional, and established as the new backup for the next toggle. In each occurrence, procedures and commands must be updated to properly address the current online computer (i.e. a different physical destination on the spacecraft, many times with different command addresses and paths for added fault protection purposes) and therefore require continual procedure maintenance and management for accurate system command and control. Automating the procedure composition in response to current system configurations allows for minimal engineering rework and test upon each occurrence. In response to the current state of the system at any point in time, the procedures can be generated directly out of the system model with the correct destination unit in the proper step.

Based on concepts proposed by (Wang and Dagli 2014) and (Huang, McGinnis and Mitchell 2019) discussed in Section 2.6.1, this approach of leveraging descriptive system models for state and behavior management supports the representation of procedures and

ensuing system behavior as Petri Nets – one example of a formally verifiable elemental construction. This is a viable next step in MBSO directly applicable to the scenario discussed in this subsection and is a concept with potentially significant implications to evolving the manner in which system operations is performed, including the potential for run-time procedure generation. Tying back to the concepts of microservices and containerization which have enabled almost continual integration and continual deployment in the software realm, this type of implementation in support of cyber-physical system operations is a logical application.

4.2.4.2 A Mission Expansion Use Case – A Return to the Development Cycle

For a more complex operational scenario, the following was considered and is presented here for context but was not simulated. If, for example, the FireSat-II spacecraft failed several cells in its battery pack, resulting in limited and insufficient power available for mission data collection during an orbit, a more complex system-wide adaptation would require a conceive-design-develop-integrate-test cycle (the left-side of the SEDevOps Lemniscate, Figure 10) to trade and redesign how energy is managed for the remainder of the system life. This scenario drives a new system operating mode where non-essential hardware units are powered off when not in use to save power, a mode not initially designed, developed, or tested prior to system deployment and one that necessitates propagating procedural changes throughout many support products, including a significant test campaign to ensure viable performance and to verify no additional regression or adverse conditions have been introduced.

Considering the SEDevOps life cycle model, in this scenario the battery failure occurs in the Operate stage, causing a transition to the Sustain/Improve stage to assess and

determine path forward. Following an assessment, the systems engineers transition back to the Conceive stage to initiate a new development cycle for modified capability based on the impact and permanence of the failure. Requirements must be reevaluated and viable operating modes must be designed, developed, integrated, tested, and deployed. This type of adaptation and evolution cycle is therefore represented by the return to the system development stages on the left side of the SEDevOps Lemniscate to leverage well-established systems engineering processes and tools for development, including MBSE models tied to development products. This would be followed by appropriate verification testing to confirm successful development and the redeployment of necessary enabling system elements to system operations on the right side of the lemniscate. Note that for a spacecraft in orbit, the spacecraft hardware is not undergoing a re-development cycle however the states and modes of the spacecraft are revisited, reevaluated, and addressed in new operational support elements to use those hardware elements in a different and initially unintended manner based on new system constraints. This is enabled by the incorporation of support elements into the core system model.

4.3 Summarizing the Insight Gained Through Use Cases and Addressing Hypothesis

In addition to implementing the capabilities described in this chapter in response to real-time events during the operational life of a system, all of the described scenarios can be simulated and tested in advance of experiencing an event during system operations, thereby providing the systems engineering team with additional capabilities in assessing the readiness for operations of a system and its support products prior to system deployment. As previously noted, this is introduced as LIAD testing and provides a method for full operational life validation analogous to DITL testing typically performed during

system validation before transitioning to a fully operational state of a system. In order to properly assess viability of a system for later life cycle stages, a LIAD test approach provides early validation of adaptability and agility of a system and its support elements in a condensed and simulated manner. This offers an advancement beyond the traditional benefits of the DevOps methodology in the software domain and represents the driver for a SEDevOps approach to life cycle management and a mechanism for systems engineering to exist in multiple stages simultaneously in a polymorphic sense.

Revisiting the initial research questions posed in Section 1.4.1, answers can be identified based on the data collected and analyzed through the use case in this chapter. Table 4 below provides answers to the proposed research questions.

Table 4 - Research Questions Answered

Research Question	Answer
1. Can the generic systems engineering life cycle model be expanded to promote the same rigorous, centralized and interrelated model-based approach leveraged in system development into and throughout system operations and sustainment?	Yes. Drawing from a number of existing and widely implemented life cycle models, a logical adaptation interleaving strengths of each was created with an equal focus on System Operations in combination with System Development. This SEDevOps model promotes a continuum from development to operations and back to foster a continuous model-based approach and common data products and processes.

Research Question	Answer
<p>2. Can the MBSE methodology and associated toolchains be adapted to the operational stages of a system's life cycle?</p>	<p><i>Yes.</i> Leveraging MBSE tools, processes, and common output products, a natural extension into system operations was developed, designated MBSO. This extension focuses on the incorporation of operational support products that can be modeled and, more importantly, exercised within a descriptive modeling framework. This creates a model-based development environment for cyber-physical system operations akin to the software development environment leveraged for a DevOps approach to software systems.</p>
<p>3. Can this adaptation of MBSE provide an adequate framework for enabling continuous model-based system development during the operational stages of a system thereby improving efficiency, agility, and operational availability over traditional system engineering practices and methods?</p>	<p><i>Yes.</i> Establishing a common platform for modeling aspects of systems operations within a MBSE toolchain (one that is growing ever more familiar to systems engineers) facilitates a transition to a fully digital ecosystem for systems engineers through the life cycle of any system. It was shown this interrelation within a common tool greatly streamlines efforts and work product maintenance for systems engineers.</p>

Factoring in the answers to the three research questions above, the hypothesis in Section 1.5 was confirmed: a model-based approach to full life cycle management improves agility (i.e. responsiveness to change) in system operations and provides an opportunity to reduce life cycle costs as compared to current methods without active use of MBSE during system operations.

Chapter 5: Conclusions

5.1 What was Accomplished: Contributions to the Field

The two major contributions presented in this research to address the stated research objectives in Section 1.4.2 are:

1. The SEDevOps life cycle model to emphasize the continuity of a model-based approach to system development and operations throughout the full life cycle of a system
2. The MBSO framework to implement this life cycle approach within a model-based, digital ecosystem

SEDevOps is based on the principles of DevOps implemented in the software domain and closely links system development products, processes, and tools to system operations with a focus on collaboration and continuous integration, testing, deployment, and monitoring. SEDevOps prescribes interrelated model-based systems engineering processes, and more importantly products and artifacts, throughout an entire system life cycle. This focus goes beyond the traditional system-development-heavy MBSE concentration, where support to system operations and sustainment builds on a descriptive model from system development and maintains this model throughout system operations with a direct path back to development processes and tools as sustainment tasks require. Additionally, system development teams have a feed-forward mechanism to system operations with the ability to validate system and support product feasibility in a forward looking, LIAD simulation and testing approach. The LIAD testing concept introduced herein enables validation of operational support products through simulation of variable and evolving environments in addition to system aging and elemental morbidity. The

outcome of testing in this manner is the ability to validate that operational products can adequately handle variability as well as establishing a mechanism to further explore and test operational CONOPS early in the system development processes.

The MBSO descriptive modeling framework builds on the continually increasing momentum of MBSE to carry descriptive system models into active use during system operations and sustainment. This is accomplished through the expansion of SysML to include operational support product elements (i.e. procedures, maintenance manuals, command and telemetry databases, etc.). This expansion provides the ability to adapt and evolve elaborately interrelated system configurations and modular executable products over time, both proactively and reactively. Furthermore, this approach inherently lends itself to continual validation and regression testing at the time of model construction and at the time of any ensuing updates providing further utility to the systems engineering team in developing and sustaining a system and its support elements throughout the full system life cycle.

The utility and benefits of active use of descriptive system models to represent, curate, and maintain executable procedures was demonstrated through a spacecraft system operations use case. In this, it was demonstrated that linking a detailed spacecraft system model to the executable procedural steps to reconfigure the system during operations and sustainment streamlines the response to events encountered during the life of a system and proves the feature of improved agility as the ability to handle change. In this use case, a critical hardware unit failure was simulated and the identification of all relevant operational products impacted by this failure and requiring update were rapidly identified. An extrapolation to the positive effects this implementation has on life cycle costs during

system operations was presented. This use case provides early endorsement of a single viable interface for systems engineering tasks throughout the life of a system given the fact that the model leveraged for this use case was developed as a product during the notional design stage of a system life cycle and proposed herein for active use during system operations. Additionally, based on the application explored in this use case, several other use cases were posited with viability and benefits identified.

5.2 What was Not Accomplished and Limitations of Findings

A known limitation of MBSO as the framework for applying SEDevOps is the overall scope, which is restricted in this initial implementation to software-based operational support elements, where updates, impacts, testing, and deployment can be performed autonomously with little touch labor. Hardware updates are, by their nature, more physical in the level of interaction required, and therefore the benefits of this approach and the findings articulated in this research are limited to the products surrounding, enabling, and operating hardware systems rather than updates to the physical hardware elements themselves. Additionally, the simulation presented in the data collection use case was limited to development and maintenance of operational support products and did not extend to formalized system configuration state management.

The end result of meeting the research objectives and proving improved agility to system operations events and an opportunity to reduce operational costs was not impacted by the two limitations noted above. Much of the modeling infrastructure to perform the additional configuration state curation was developed in the MBSO profile and can be coupled with the inherent functionality in most MBSE tools, and specifically in Cameo Systems Modeler used for this development, and is therefore identified here as the next

step in advancing the contributions presented and is therefore an area of future research noted below.

5.3 Areas of Future Research

Further research into improvements and future applications of SEDevOps and MBSO is merited. The following expansions on these concepts were considered in the body of research however not fully formulated or vetted at the time this manuscript was compiled. Reaching back to the concept introduced at the start of Chapter 2, in order to develop and deploy fully autonomic systems, a normalizing foundation of products and processes is required. SEDevOps and MBSO provide this foundation and the concepts presented here for further research expand upon this foundation, making fully autonomic systems one step closer.

5.3.1 Formal Methods & Safety Critical System Operations

Further research and development is warranted on expanding the incorporation of formal methods directly into the MBSO profile and applying the modified profile to safety critical systems. Formal methods allow for constraint-driven configuration checking to mathematically prove a model is both correct and complete (Blanchard and Fabrycky 2006) which, in the case of MBSO, enables automating system support product update, testing, and deployment with high confidence and assurance, making safety critical system management in this manner viable.

5.3.2 Autonomic System Operations

Based on this approach for formal method implementation, a logical next step is to incorporate further degrees of autonomy and, ultimately, autonomicity (Truszkowski, et al.

2009) into system operations to the extent of enabling adaptability with formal pre-verification as well as run-time verification of executable products. This is based on the clear definition of and strict adherence to pre-configuration and post-configuration states managed within the model and directly interrelated to the operational products. This enables correct-by-construction building blocks for operational support element development and execution, effectively an application of formal methods.

Extrapolating one step further, once adequate and accurate signatures, limitations, and constraints are built into an expanded system model and coupled with the relevant command and telemetry databases, it is feasible to apply machine learning algorithms to find the optimal path between two known (or partially known) configuration states, therefore autonomously and thoroughly developing operational procedures able to be formally verified a priori and at run-time. This is a fertile area of research and one not yet addressed with respect to SEDevOps and MBSO.

5.3.3 Prognostics, Diagnostics, & Data Trending

Another area for continued research is in adding prognostic capabilities into MBSO where system health data is continuously processed, trended, and compared with known configurations and behaviors for signs or trends of abnormality. System operations support teams can prepare and test proactive responses within the modeling environment prior to deploying configuration changes to the physical system resulting in the ability to influence trends as needed for successful operations before fault conditions are triggered. Likewise, adding diagnostic capabilities into the system model for cases where proactive measures are either not possible or not implemented prior to a fault event would enable generating procedures in real-time in response to events, computing new valid configurations and

transitions between them, and implementing fault responses as needed to ensure system reliability against encountered events.

5.4 Research Benefits & Potential Implementation Challenges

The SEDevOps life cycle model and the MBSO framework provide numerous benefits throughout system life cycles, supporting the overall digital transformation initiative for systems engineering, and providing a basis for the single model-based interface point to enable a more polymorphic systems engineering discipline.

5.4.1 Benefits of SEDevOps

The SEDevOps model is a logical merger of various life cycle approaches spanning sequential, evolutionary, and emergent methods (SEBoK Editorial Board 2020). As noted in Table 1, each life cycle method brings certain strengths for systems engineers and SEDevOps is designed to leverage strengths from each. The individual development stages of SEDevOps support a layered, sequential approach to capture the strengths and rigor of plan-driven methods. The cyclic development construct promotes iterative and evolutionary development to field and continually adapt capabilities to evolving needs. The overall lemniscate integrates an operational focus with established development tools, processes, and artifacts to improve adaptability throughout a full system life cycle. A notable enabler of this paradigm for operational systems is the focus on continual testing and validation of executable products against system configurations with the Test stage emphasized at the center of the SEDevOps model.

5.4.2 Benefits of MBSO

Extending the validation of executable products noted previously to managing configuration changes, MBSO enables identification of impacts, potential mitigations to these impacts, the ability to test changes to system configurations for semantic and syntactic correctness, and the capability to regression test updates for unintended impacts based on those changes. These tasks, inherent in MBSO, are traditionally manual actions requiring touch-labor on operational products and, without robust configuration management and testing capabilities, present a risk of incomplete impact assessment, overlooked procedures or procedural steps, and incomplete testing and validation.

An additional benefit of MBSO is the configuration management capabilities intrinsically built into descriptive modeling tool suites based on the check-out/commit and trunk/branch method of software development which enables detailed accounts of model updates and the ability to quickly and easily roll-back changes (i.e. “commits”) to prior versions. This also enables systems engineers to work in separate branches of a model to develop and test feature improvements during system operations without impacting the operational baseline. As DevOps principles, and SEDevOps by adoption, focus on toolchains and collaboration, this feature of MBSO is a critical enabler for more efficient operations and sustainment.

Using the MBSO framework, this configuration management benefit can be extended to knowledge items, formalizing what has been traditionally considered "tribal knowledge" associated with systems (Douglass 2016). Expanding upon this, many configuration changes during the operational life of a system are driven by reactions to events, both internal and external to the system of interest. Background and supporting knowledge

which resulted in the chosen path forward is many times held in memory by a small number of personnel involved and captured in documentation on a non-interfering basis. The result is lost fidelity over time on why configuration changes were made, what impacts were assessed (and not assessed), and erosion of general system historical knowledge as personnel transition off of projects over the course of the system life cycle. MBSE descriptive modeling tools include built-in attributes to capture notations and detailed information, providing a means to document historical data in a single location for quick reference at any point in time, directly associated with a specific system element (such as leveraging the “Documentation” elemental attribute in Cameo Systems Modeler) or configuration change (by documenting rationale and information with any model update during the “commit”).

5.4.3 Potential Implementation Challenges

While SEDevOps and MBSO have the potential for improving agility and reducing cost in adaptable system operations, they are not without potential implementation challenges, including:

Adoption – While DevOps continues to gain traction and supporters in the software domain, it is not without adoption challenges, including availability of requisite tool suites and properly trained personnel. SEDevOps likely faces similar adoption challenges in the systems engineering domain. An area for future development to improve adoption and implementation is the creation of a detailed SEDevOps modeling plug-in, broadened beyond just MBSO, akin to the Unified Architecture Framework in which users are visually directed through a number of steps to build-out the required level of detail within a modeling framework. Implementing a broader SEDevOps modeling language profile

would foster early adoption and creation of necessary products throughout an entire system life cycle.

Investment – As with any process improvement, investment is required to enable progress. Building a detailed system model leveraging MBSO will require investment in requisite tools, training, and personnel to support the transition and ramp-up. As noted in systems engineering literature, the transition to MBSE, and in this case its extension to MBSO, requires investment in infrastructure, process, and training (Friedenthal, Moore and Steiner 2014), (Madni and Sievers 2018), (Ramos, Ferreira and Barcelo 2012).

Maintenance – Once operational support products become part of the overall system model, maintenance of the model and continually synching with the operational state becomes critical to success of the expanded system. Without rigorous configuration management and infrastructure in place, SEDevOps and MBSO success may be challenged.

5.5 Final Remarks

The benefits of the SEDevOps life cycle model implemented through the MBSO framework were presented in a use case in which a simulated unit failure was assessed via intrinsic model features for impacts to operational support products during the notional operations and sustainment stages of a spacecraft system. As the scale and complexity of systems continue to increase, the risk to operational support product maintenance and update in response to dynamic events also increases. Establishing a model-centric, single source of truth approach to linking and managing system support products built on detailed descriptive models improves consistency, reduces risk, and drives down manual touch

points in operations thereby offering an opportunity to reduce life cycle costs while improving overall agility of both the system and the systems engineers.

With that, the research detailed herein provides a step towards polymorphic systems engineering by establishing a model-based continuity across the entirety of a system's life cycle.

Chapter 6: Bibliography

- Alberts, David S. 2011. *The Agility Advantage*. Washington, DC, USA: DoD Command and Control Research Program.
- Atlassian. 2019. *How to choose the right DevOps tools*. Atlassian. Accessed June 2019. <https://www.atlassian.com/blog/devops/how-to-choose-devops-tools>.
- Balalaie, Armin, Abbas Heydarnoori, and Pooyan Jamshidi. 2016. "Microservices architecture enables devops: Migration to a cloud-native architecture." *IEEE Software* 33 (3): 42–52.
- Basili, Victor R., and Craig Larman. 2003. "Iterative and incremental developments: A brief history." *Computer* 36 (6): 47–56.
- Bass, Len, Ingo Weber, and Liming Zhu. 2015. *DevOps: A Software Architect's Perspective, 1st Ed.* Old Tappan, NJ, USA: Addison-Wesley.
- Beedle, Mike, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, Jim Highsmith, Andrew Hunt, et al. 2001. *The agile manifesto: Principles behind the agile manifesto*. Accessed February 2018. <https://agilemanifesto.org/principles.html>.
- Blanchard, Benjamin S., and John E. Blyer. 2016. *Systems Engineering Management, 5th ed.* Hoboken, NJ, USA: Wiley.
- Blanchard, Benjamin S., and Wolter J. Fabrycky. 2006. *Systems Engineering and Analysis, 4th ed.* Saddle River, NJ, USA: Prentice Hall.
- Boehm, Barry W. 1988. "A Spiral Model of Software Development and Enhancement." *Computer* 61-72.
- Boehm, Barry, and Richard Turner. 2004. *Balancing agility and discipline: a guide for the perplexed*. Crawfordsville, Indiana: Addison-Wesley.
- Bryson, Jeff. 2010. "Polymorphic System Architecture." *INCOSE International Symposium*. 370-385.
- Castet, Jean-Francois, Magdy Bareh, Jeffery Nunes, J. Steven Jenkins, and Gene Lee. 2016. "Fault management ontology and modeling patterns." *Proc. AIAA Space Astronaut. Forum Expo*. Long Beach, CA, USA. 1–17.

- Castet, Jean-Francois, Magdy Bareh, Jeffery Nunes, Shira Okon, Larry Garner, Emmy Chacko, and Michel Izygon. 2018. "Failure analysis and products in a model-based environment." *Proc. IEEE Aerosp. Conf.* Big Sky, MT, USA. 1-3.
- Castet, Jean-Francois, Matthew Rozek, Michel Ingham, Nicolas Rouquette, Seung Chung, Aleksandr Kerzhner, Kenneth Donahue, et al. 2015. "Ontology and modeling patterns for state-based behavior representation." *Proc. AIAA Infotech Aerosp.* Kissimmee, FL, USA. 1-22.
- Cloutier, Robert, Brian Sauser, Mary Bone, and Andrew Taylor. 2015. "Transitioning systems thinking to model-based systems engineering: Systemigrams to SysML models." *IEEE Transactions on Systems, Man, & Cybernetics: Systems* 45 (4): 662–674.
- Codetta-Raiteri, Daniele, and Luigi Portinale. 2015. "Dynamic Bayesian networks for fault detection, identification, and recovery in autonomous spacecraft." *IEEE Trans. Syst., Man, Cybern., Syst.* 45 (1): 13–24.
- Combemale, Benoit, and Manuel Wimmer. 2019. "Towards a model-based DevOps for cyber-physical systems." *Proc. 2nd Int. Workshop Softw. Eng. Aspects Continuous Develop. New Paradigms Softw. Prod. Deployment.* Villebrumier, France. 1-11.
- Compuware. 2019. *Mainframe DevOps*. Compuware. Accessed June 2019. <https://www.compuware.com/lifecycle-overview/>.
- Crane, Jeremiah, Vinodini Sundaram, Justin Malek, and Leonard Brownlow. 2017. "MBSE for sustainment: A case study of the air force launch and test range system (LTRS)." *Proc. AIAA Space Astronaut. Forum Expo.* Orlando, FL, USA. 1-9.
- Delligatti, Lenny. 2014. *SysML distilled – A Brief Guide to the Systems Modeling Language*. Upper Saddle River, NJ, USA: Addison-Wesley.
- Dictionary.com. 2020. *Polymorph*. Accessed September 4, 2020. <https://www.dictionary.com/browse/polymorph>.
- Douglass, Bruce Powel. 2016. *Agile Systems Engineering*. Waltham, MA, USA: Morgan Kaufmann.
- Dove, Rick, and Ralph LaBarge. 2014. "8.4.1 & 8.4.2 Fundamentals of agile systems engineering - Parts 1 & 2." *Proc. INCOSE Int. Symp.* 24 (1): 859–896.

- Erickson, Martin J. 2011. *Beautiful Mathematics*. MAA Spectrim, Mathematical Association of America. Accessed September 4, 2020. <https://en.wikipedia.org/wiki/Lemniscate>.
- Forsberg, Kevin, Hal Mooz, and Howard Cotterman. 2005. *Visualizing Project Management: Models and Frameworks for Mastering Complex Systems, 3rd Edition*. John Wiley & Sons.
- Friedenthal, Sanford. 2017. "Architecting spacecraft with SysML FireSat-II model." Accessed November 2018. <http://sysml-models.com/spacecraft/models.html>.
- Friedenthal, Sanford, Alan Moore, and Rick Steiner. 2014. *A Practical Guide to SysML. Third Edition: The Systems Modeling Language*. Boston, MA, USA: Morgan Kaufmann.
- Friedenthal, Sanford, and Christopher Oster. 2017. *Architecting Spacecraft with SysML: A Model- Based Systems Engineering Approach*. San Bernardino, CA, USA: CreateSpace Independent Publishing Platform.
- Gans, Howard D. 2017. "Development of space vehicle CONOPS using SysML & UPDM." *Proc. AIAA Space Astronaut. Forum Expo*. Orlando, FL, USA. 1–13.
- General Electric Corporation. 2018. *The Digital Twin: Compressing time-to-value for digital industrial companies*. Accessed October 2018. https://www.ge.com/digital/sites/default/files/download_assets/The-Digital-Twin_Compressing-Time-to-Value-for-Digital-Industrial-Companies.pdf.
- Graves, Henson, and Yvonne Bijan. 2011. "Using formal methods with SysML in aerospace design and engineering." *Annals of Mathematics and Artificial Intelligence* 63 (1): 53–102.
- Huang, Edward, Leon F. McGinnis, and Steven W. Mitchell. 2019. "Verifying SysML activity diagrams using formal transformation to Petri nets." *Systems Engineering* 23 (1): 118–135.
- INCOSE. 2015. *Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities, 4th ed*. Hoboken, NJ, USA: Wiley.
- INCOSE. 2014. *Systems Engineering Vision 2025*. San Diego, CA, USA: International Council on Systems Engineering.
- ISO. 2015. *ISO/IEC/IEEE International Standard - Systems and Software Engineering – System Life Cycle Processes*. Standard, Geneva: ISO/IEC/IEEE Standard 15288.

- ISO. 2017. *ISO/IEC/IEEE International Standard - Systems and Software Engineering—Vocabulary*. Standard, Geneva: ISO/IEC/IEEE Standard 24765: 2017(E).
- Kargar, Mohammad Javad, and Alireza Hanifzade. 2018. "Automation of regression test in microservice architecture." *2018 4th International Conference on Web Research (ICWR)*. Tehran, Iran. 133-137.
- Lewis, James. 2012. *Micro services - Java, the Unix Way*. March 19. Accessed September 12, 2020. <http://2012.33degree.org/talk/show/67>.
- Lewis, James, and Martin Fowler. 2014. *Microservices*. March 25. Accessed September 12, 2020. <https://martinfowler.com/articles/microservices.html#footnote-etymology>.
- Madni, Azad M., and Michael Sievers. 2018. "Model-based systems engineering: Motivation, current status, and research opportunities." *Systems Engineering* 21 (3): 172–190.
- Mathieson, John T.J., Thomas Mazzuchi, and Shahram Sarkani. 2020. "The Systems Engineering DevOps Lemniscate and Model-Based System Operations." *IEEE Systems Journal* 1-12.
- NASA. 2017. *NASA Systems Engineering Handbook, Revision 2*. Washington, D.C., USA: National Aeronautics and Space Administration.
- NDIA Systems Engineering Division M&S Committee. 2011. *Final Report of the Model Based Engineering Subcommittee*. Final Report, Arlington, VA, USA: National Defense Industrial Association (NDIA).
- Null, Christopher. 2020. *TechBeacon*. Accessed September 12, 2020. <https://techbeacon.com/devops/10-companies-killing-it-devops>.
- Olszewska, Marta, and Marina Walden. 2015. "DevOps meets formal modelling in high-criticality complex systems." *Proc. 1st Int. Workshop Qual.-Aware DevOps*. New York, NY, USA.
- Polya, George. 1945. *How to Solve It*. United States of America: Princeton University Press.
- Puchek, Beth, Michael Bisconti, Philomena Zimmerman, Jaime Guerrero, Pamela Kobryn, Geethesh Kukkala, Clif Baldwin, Joe Hale, and Midh Mulpuri. 2017. *Digital Model-based Engineering: Expectations, Prerequisites, and Challenges of Infusion*. Washington DC, USA: Model-Based Engineering (MBE) Infusion Task

Team, Office of the Undersecretary of Defense for Acquisition & Sustainment, US Department of Defense.

Rabelo, Luis, and Tom Clark. 2015. "Modeling space operations systems using SysML as to enable anomaly detection." *SAE International Journal of Aerospace* 8 (2): 189-194.

Ramos, Ana Luisa, Jose Vasconcelos Ferreira, and Jaume Barcelo. 2012. "Model-based systems engineering: An emerging approach for modern systems." *IEEE Transactions on Systems, Man, & Cybernetics, Part C* 42 (1): 101–111.

Riungu-Kalliosaari, Leah, Simo Makinen, Lucy Ellen Lwakatare, Juha Tiihonen, and Tomi Mannisto. 2016. "DevOps Adoption Benefits and Challenges in Practice: A Case Study." *17th International Conference on Product-Focused Software Process Improvement, PROFES*. Trondheim, Norway: Springer. 590-597.

Ruan, Haowei, Craig Anslow, Stuart Marshall, and James Noble. 2010. "Exploring the Inventor's Paradox: Applying Jigsaw to Software Visualization." *SOFTVIS'10 - Proceedings of the 2010 International Symposium on Software Visualization, Co-located with VisWeek 2010*. Salt Lake City, UT, USA: Association for Computing Machinery. 83-92.

SEBoK Editorial Board. 2020. *The Guide to the Systems Engineering Body of Knowledge (SEBoK)*, v. 2.2. Edited by R.J. Cloutier (Editor in Chief). The Trustees of the Stevens Institute of Technology. Accessed May 20, 2020. www.sebokwiki.org.

Sheard, Sarah. 1996. "Twelve Systems Engineering Roles." *Proceedings, Sixth Annual International Symposium of the International Council on Systems Engineering*. Boston, MA: INCOSE.

Space Development Agency. 26 June 2020. *Broad Agency Announcement - Mission Specific Application Prototypes, HQ085020S0002*. Washington, DC, USA: Space Development Agency.

Sutharssan, Thamo, Stoyan Stoyanov, Chris Bailey, and Chunyan Yin. 2015. "Prognostic and health management for engineering systems: A review of the data-driven approach and algorithms." *Journal of Engineering* 7: 215–222.

Toure, El Hadji Bassirou, Ibrahima Fall, Alassane Bah, Mamadou Samba Camara, and Mandicou Ba. 2017. "Consistency preserving for evolving megamodels through axiomatic semantics." *2017 Intelligent Systems and Computer Vision (ISCV)*. Fez, Morocco.

- Truszkowski, Walt, Lou Hallock, Christopher Rouff, Jay Karlin, James Rash, Michael G. Hinchey, and Roy Sterritt. 2009. *Autonomous and Autonomic Systems with Applications to NASA Intelligent Spacecraft Operations and Exploration Systems*. New York, USA: Springer.
- U.S. Department of Defense. 2018. *2018 Digital Engineering Strategy*. Arlington, VA: U.S. Department of Defense Office of the Deputy Assistant Secretary of Defense for Systems.
- Uhlemann, Thomas H.J., Christoph Schock, Christian Lehmann, Stefan Freiberger, and Rolf Steinhilper. 2017. "The digital twin: Demonstrating the potential of real time data acquisition in production systems." *Procedia Manufacturing* 9: 113–120.
- Wagner, David A., Matthew B. Bennett, Robert Karban, Nicolas Rouquette, Steven Jenkins, and Michel Ingham. 2012. "An ontology for state analysis: Formalizing the mapping to SysML." *Proc. IEEE Aerospace Conference*. Big Sky, MT, USA. 1-6.
- Wang, Renzhong, and Cihan H. Dagli. 2014. "Executable system architecting using systems modeling language in conjunction with colored petri nets in a model-driven systems development process." *Syst. Eng.* 14 (4): 383–409.
- Wertz, James R., David F. Everett, and Jeffery J. Puschell. 2011. *Space Mission Engineering: The New SMAD*. Hawthorne, CA, USA: Microcosm Press.
- Zhu, Liming, Len Bass, and George Champlin-Scharff. 2016. "DevOps and its practices." *IEEE Software* 33 (3): 32–34.