DISSERTATION

# BOOTSTRAPPING A TRUSTWORTHY AND SEAMLESS DIGITAL ENGINEERING APPLIANCE

Submitted by

James S. Wheaton

Department of Systems Engineering

In partial fulfillment of the requirements

For the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Summer 2025

Doctoral Committee:

Advisor: Daniel R. Herber

Steven J. Simske
Erika E. Gallegos
Vinayak S. Prabhu

# ABSTRACT


## BOOTSTRAPPING A TRUSTWORTHY AND SEAMLESS DIGITAL ENGINEERING APPLIANCE

Digital engineering is an organizational effort that currently relies on the complex networked integration of heterogeneous computer hardware and software components to maintain a cohesive digital model of the system-of-interest: its Authoritative Source of Truth. The unfortunate truth is that these computer components contain myriad known and unknown defects and their many interfaces cause severe fragility, adding significant cost, risk, and schedule to projects and to the information technology infrastructure that supports them. In this situation, additional resources must be allocated to defect remediation and the gluing of software and data components together with ad-hoc solutions, or to rely on expensive third-party solutions that further accrete the infrastructure. While interoperability by means of Application Programming Interfaces exists in islands of support and is offered by, e.g. the Systems Modeling Language version 2, as the way forward, the foundations upon which these software-intensive systems are built are nevertheless untrustworthy and critically vulnerable.

This dissertation argues that clean-slate approach is necessary to address this mess: a system of interrelated problems. The needs of digital engineering stakeholders motivate a computing system architecture that guarantees consistency and coherence, is independently auditable, and is trustworthy based on strong evidence gathered from a full-source bootstrap and end-to-end formal verification, to achieve correctness-by-construction of itself and of the systems-of-interest it is employed to specify. Such a cyber-system, specifically designed for digital engineering activities, is termed a seamless digital engineering information appliance, and represents a grand challenge in digital engineering research. By applying a transdisciplinary systems engineering approach to this problem space, the assured preservation of bidirectional

traceability of the stakeholder needs and requirements, detailed design specifications, and verification proof certificates becomes achievable. A systems approach to human factors and security will then result in a high-assurance, malleable human-computer interface with fine-grained security controls suited to the needs of digital engineering.

Seamless Digital Engineering is defined as a digital engineering tooling paradigm, contrasted with existing digital engineering integration patterns, and characterized with the seamless integration pattern and a set of architecture tenets to guide surveys of the solutions space. Rationale of the grand challenge is presented. The natural language definition is further clarified using the expressive power of formal ontologies, resulting in ontological definitions-by-relations based on relevant systems and software engineering standards. The concept of seamless is disambiguated using the SQuaRE product quality model, separating it into seamless integration and seamless interaction capability quality characteristics, and seamless quality-in-use characteristics. The Seamless Digital Engineering Ontology includes over 500 concepts and is published open-source in a standard machine-readable format. An open-source SysML profile for digital requirements engineering is presented and validated in real-world projects, representing the preferred model-based technique for developing requirements in the seamless digital engineering context. Finally, the Seamless Digital Engineering Reference Architecture defined in SysML v2 is presented, which captures essential digital engineering stakeholder goals, objectives, and needs. This reference architecture specifies multiple trustworthy bootstrap paths for the proposed seamless digital engineering appliance, with the explicit goal of bootstrapping a powerful, high-assurance digital engineering meta-language. Together, these open-source models form the basis of understanding the grand challenge so that a detailed definition of the Seamless Digital Engineering Reference Architecture can proceed.

# ACKNOWLEDGEMENTS

I am blessed by the support of my peers, mentors, advisors, colleagues, friends, and family. The development of this dissertation was a long and challenging process, and during that time, these people have cheered me on, shared their ideas and patiently listened to my own, and offered their valuable time to review my work.

I would like to thank my research advisor, Daniel Herber, for patiently guiding me to success as a researcher and doctoral student. His support has been invaluable throughout this process, and I will continue practicing what he taught me in contributing to the advancement of systems engineering research. I want to thank my first research advisor, Wade Troxell, who took me on as a fresh student and got me started on my research journey. My doctoral committee, too, have lent me their time and expert judgment so that I may improve the quality of my research and present it accordingly.

Special thanks to my internship mentor, Paulo Younse, for guiding me through real-world systems engineering at NASA Jet Propulsion Laboratory. That experience enriched my life and supported my continued success as a researcher and systems engineer. Thanks to the Mars Returned Sample Handling project team members for supporting my professional development, and eagerly listening to my lectures on requirements engineering and MBSE.

The opportunity to develop a formal digital engineering ontology accelerated my research and took it into new territory, while satisfying an old dream of finding a practical application with which to learn this fascinating skill. For that, I want to thank Joe Gregory who invited me to join the INCOSE DEIX Ontology Working Group and present my early research. The group's feedback has been invaluable for me to iterate on the ontology, and learn to present this technical topic to a variety of audiences.

Thanks to my hacker friends online in the fediverse whose friendship kept me going through the toughest times, whose stimulating conversations led to many ideas presented in this dissertation, and whose diversity of ideas enriched my understanding of the world and

motivated this research. Special thanks to Frédéric Jouvin for his kindness and dedication to sharing unorthodox digital architecture concepts and analysis.

Lastly, I want to thank my family for supporting me throughout this process, no matter how small it may have seemed at times — every bit counts. To my lovely wife, Angel, who has always been there to listen to me talk endlessly about my research and share my struggles, and who has sacrificed our adventure time so I could stay inside to write and grade assignments. To our dog, Gracie, who we rescued and who rescued us. And to my late dad, who mentored my curiosity of the world and who literally taught me physics — I couldn't have done it without you.

# DEDICATION

*To my lovely, supportive wife, Angel.*

*And to my late dad, John.*

TABLE OF CONTENTS

ABSTRACT . . . . . ii
ACKNOWLEDGEMENTS . . . . . iv
DEDICATION . . . . . vi
LIST OF TABLES . . . . . ix
LIST OF FIGURES . . . . . x
LIST OF LISTINGS . . . . . xii
LIST OF ACRONYMS . . . . . xiii

Chapter 1 Introduction . . . . . 1
1.1 Overview . . . . . 1
1.2 Research Questions . . . . . 4
1.3 Structure of this Dissertation . . . . . 4

Chapter 2 A Grand Challenge Driven by Needs . . . . . 6
2.1 Introduction . . . . . 6
2.2 Related Work . . . . . 9
2.3 Characterizing the Problem Space . . . . . 11
2.4 Defining Seamless Digital Engineering . . . . . 24
2.5 A Grand Challenge Driven by Needs . . . . . 29
2.6 Conclusions . . . . . 31
2.7 Summary . . . . . 33

Chapter 3 Ontological Definition of Seamless Digital Engineering . . . . . 35
3.1 Introduction . . . . . 35
3.2 Background . . . . . 37
3.3 Methodology . . . . . 40
3.4 Results . . . . . 42
3.5 Conclusions . . . . . 52
3.6 Summary . . . . . 53

Chapter 4 Digital Requirements Engineering . . . . . 56
4.1 Introduction . . . . . 56
4.2 Model-Based Structured Requirement (MBSR) . . . . . 62
4.3 INCOSE-Derived MBSR for Digital Requirements Engineering . . . 69
4.4 Discussion . . . . . 80
4.5 Related Work . . . . . 84
4.6 Limitations . . . . . 88
4.7 Conclusions . . . . . 88
4.8 Summary . . . . . 90

Chapter 5 Reference Architecture for Seamless Digital Engineering . . . . . 91

5.1 Introduction . . . 91
5.2 Stakeholder Goals . . . 92
5.3 Quality Attributes . . . 95
5.4 Operational Concept . . . 99
5.5 Needs Analysis . . . 108
5.6 Seamless Digital Engineering Meta-Language . . . 122
5.7 Full-Source Bootstrap Strategy . . . 128
5.8 Conclusions . . . 138
5.9 Summary . . . 139

Chapter 6 Conclusions and Future Work . . . 140
6.1 Summary . . . 140
6.2 Research Contributions . . . 143
6.3 Future Work . . . 145

Bibliography . . . 150

LIST OF TABLES

2.1 Definitions of related engineering terms (**emphasis** added). . . . . 8
2.2 Stakeholders of a DES. . . . . 13
2.3 Transforming tool integration questions [30] into system requirements. . . . . 27
2.4 Evaluation against criteria for a grand challenge [62]. . . . . 32

3.1 OWL DL summary of symbols of logic and their descriptions . . . . . 39
3.2 Sources used to define ontology classes/concepts (see Bibliography). . . . . 55

4.1 Benefits and perceived impacts of using INCOSE-derived MBSR for DRE. Scale: ● High, ◐ Medium, ◔ Low. . . . . 82
4.2 Challenges and perceived impacts of using INCOSE-derived MBSR for DRE. Scale: ● High, ◐ Medium, ◔ Low. . . . . 83
4.3 INCOSE Characteristics of well-formed sets and individual needs and requirements [117] with mapping to NASA and ISO guidance [119, 140]. . . . . 84

LIST OF FIGURES


1.1 Digital engineering lifecycle reference model (adapted from [2]). . . . . 2
1.2 Cumulative total number of Common Vulnerabilities and Exposures (CVE) by year (data from [5]). . . . . 3
1.3 Dependency iceberg — software dependency graph of 8 selected DE packages, with 3,641 package dependencies, 80,716 dependency links, and 137 layers (data visualization code from [6]). . . . . 3

2.1 Generic Problem-Solution Space (SysML v2 depiction). . . . . 12
2.2 Softgoal Interdependency Graph (SIG) [35] of DE Goals (decomposed in Figures 2.4, 2.5, 2.6, 2.7, 2.8), Supergoals, and Claims (derived from [22]). . . . . 15
2.3 SIG of decomposition of DE Supergoals from Figure 2.2 (derived from [22]). . . 15
2.4 SIG of DE Goals and Objectives of G-DE-1 from Figure 2.2 (derived from [22]). 16
2.5 SIG of DE and Objectives of G-DE-2 from Figure 2.2 (derived from [22]). . . . . 16
2.6 SIG of DE Goals and Objectives of G-DE-3 from Figure 2.2 (derived from [22]). 17
2.7 SIG of DE Goals and Objectives of G-DE-4 from Figure 2.2 (derived from [22]). 17
2.8 SIG of DE Goals and Objectives of G-DE-5 from Figure 2.2 (derived from [22]). 18
2.9 Import/Export method of DE tool integration (SysML v2 depiction). . . . . 20
2.10 Custom Shim method of DE tool integration (SysML v2 depiction). . . . . 21
2.11 SysML v2 API method of DE tool integration (SysML v2 depiction). . . . . 21
2.12 The Error Avalanche (adapted from [49]). . . . . 24
2.13 Seamless design pattern of DE tool integration (SysML v2 depiction). . . . . 29
2.14 Seamless DE Architecture Tenets as packaged concern definitions (SysML v2 depiction). . . . . 30

3.1 Ontology concept definition process using standard sources. . . . . 42
3.2 SQuaRE quality characteristics from Ref. [89] defined as Product Capabilities . 44
3.3 SQuaRE quality-in-use characteristics as defined in Ref. [90] . . . . . 45
3.4 Proposed import hierarchy of DE-related domain ontologies, based on BFO and CCO. . . . . 46
3.5 Summary of ontological relations of Seamless Digital Engineering concepts. . . . 47

4.1 Classical SysML requirements modeling using standard relationships to system model elements. . . . . 64
4.2 SysML Stereotype definitions for Requirement Expression and Requirement Set. Refer to Figure 4.5 for further detail on the INCOSE GtWR Attributes. . . . . 70
4.3 Relation Map of example MBSR and related system architecture SysML elements. 71
4.4 SysML meta-model of INCOSE-derived MBSR classifiers derived from Figure 4 of [117]. . . . . 72
4.5 Model-based INCOSE GtWR Attributes of Well-Formed Needs and Requirements. Refer to Figure 4.2 for further relationship definitions. . . . . 74

4.6 A Requirements Satisfaction Matrix of INCOSE GtWR Rules (abbreviated), with Violate relationships shown in red. . . . . 75
4.7 Single example MBSR with INCOSE GtWR Attributes, Characteristics, and Rules. 77
4.8 UML 2.5 XMI definition of the example MBSR shown in Fig. 4.7 (newlines added & attributes reordered for readability). . . . . 78
4.9 INCOSE-derived MBSRs in a requirement table with model-based legend highlighting. . . . . 79

5.1 Booting up the Seamless DE Appliance. . . . . 100
5.2 Choosing your first activity. . . . . 101
5.3 Starting a new project — Step 1. . . . . 102
5.4 Starting a new project — Step 2. . . . . 103
5.5 Starting a new project — Step 3 (graphic adapted from [187]). . . . . 104
5.6 Writing requirements in a tabular editor. . . . . 105
5.7 Researching how to model system architectures. . . . . 106
5.8 Using transclusion in a document. . . . . 107
5.9 Textual/visual bidirectional editing. . . . . 108
5.10 Unified Engineering Environment with collaboration. . . . . 109
5.11 The correspondence between SysML v2 graphical syntax and LISP-like definitions of function specification and implementation. . . . . 124
5.12 Major parts decomposition of the Seamless DE Appliance (cf. Figure 5.13). . . . 127
5.13 Reference full-source minimized bootstrap paths to Seamless DE Meta-language (Section 5.6) and HCI. . . . . 129
5.14 Overall view of Seamless DE Appliance bootstrap with 3 alternate bootstrap paths (Figs. 5.15, 5.21, 5.22) . . . . . 131
5.15 Overall view of bootstrap path #1 starting with the availability of a POWER9-based computer . . . . . 132
5.16 Stage0 of bootstrap path #1 (Figure 5.15). . . . . 133
5.17 Stage1 of bootstrap path #1 (Figure 5.15). . . . . 134
5.18 Stage2 of bootstrap path #1 (Figure 5.15). . . . . 135
5.19 Stage3 of bootstrap path #1 (Figure 5.15). . . . . 135
5.20 Stage4 of bootstrap path #1 (Figure 5.15). . . . . 136
5.21 Stage0 of alternate bootstrap path #2 which depends on a WebAssembly VM or ASIC. . . . . 137
5.22 Stage5 of alternate bootstrap path #2 (dependencies to other stages not shown). 138

LIST OF LISTINGS


3.1 **'Seamless Digital Engineering Paradigm'** is a subclass of **Paradigm** equivalent to: 48
3.2 **Correct-by-Construction** is a subclass of **'Assurance Goal'** equivalent to: . . . . 48
3.3 **'Seamless Digital Engineering System'** is a subclass of **'Engineered System'** equivalent to: . . . . . 49
3.4 **'Seamless Quality Claim'** is a subclass of **'Quality Claim'** equivalent to: . . . . . 49
3.5 **'Seamless Integration'** is a subclass of **'Product Capability'** equivalent to: . . . 50
3.6 **'Seamless Interface'** is a subclass of **'Interface'**, a subclass of 'Information Bearing Artifact', and is equivalent to: . . . . . 50
3.7 **'Seamless Interaction Capability'** is a subclass of **'Product Capability'** equivalent to: . . . . . 51
3.8 **'Seamless Quality-in-Use'** is a subclass of **'Quality-in-Use'** equivalent to: . . . . 51
3.9 **Trustworthiness** is a subclass of **'Acceptability'** equivalent to: . . . . . 52

5.1 Definition of MGO goal in the SDE Profile as a SysML v2 specialized requirement with semantic metadata . . . . . 93
5.2 Expected DE benefit reformulated as a stakeholder concern (SysML v2) . . . . . 94
5.3 Excerpt of DE goal framing stakeholder concern of Listing 5.2 (SysML v2) . . . . . 94
5.4 The 'Seamless DE Mission' as a SysML v2 requirement with semantic metadata . . . 95
5.5 Excerpt of the 'Seamless DE Appliance' quality goal as a SysML v2 requirement with semantic metadata . . . . . 96
5.6 Datskovskiy's "Seven Laws of Sane Personal Computing" [185] modeled as stakeholder expectations in SysML v2 . . . . . 98

# LIST OF ACRONYMS

**ABC** Activity-Based Computing *on page(s):* 99, 101, 106

**ACL** access-control list *on page(s):* 117

**AIAA** American Institute of Aeronautics and Astronautics *on page(s):* 146

**API** application programming interface *on page(s):* x, 20, 21, 148

**APT** advanced persistent threat *on page(s):* 24

**ARPA** Advanced Research Projects Agency *on page(s):* 9

**ASIC** application-specific integrated circuit *on page(s):* xi, 116, 137, 148

**ASoS** Acknowledged System-of-Systems *on page(s):* 19, 20, 22

**ASoT** Authoritative Source of Truth *on page(s):* 1, 6, 8, 10, 11, 13, 14, 22, 25, 28, 29, 33, 36, 45, 56, 58, 64, 69, 89, 91, 92, 107, 112, 142, 144, 148

**ATP** automated theorem prover *on page(s):* 125

**BFO** Basic Formal Ontology *on page(s):* x, 35, 39–41, 43, 46, 53, 141, 146

**CAE** computer-aided engineering *on page(s):* 1, 3, 7, 14, 22, 23, 92

**CASCaDE** Collaborative Artifact, Specification, Context and Data Exchange *on page(s):* 146

**CASE** computer-aided software engineering *on page(s):* 10

**CCO** Common Core Ontologies *on page(s):* x, 35, 40–43, 46, 49, 53, 141, 146

**CPU** central processing unit *on page(s):* 2, 23, 31, 114, 116, 120, 130

**CRAFT** Circuit Realization At Faster Timescales *on page(s):* 3, 7

**CRASH** Clean-Slate Design of Resilient, Adaptive, Secure Hosts *on page(s):* 3, 7

**CSV** comma-separated values *on page(s):* 20, 75

**CVE** Common Vulnerabilities and Exposures *on page(s):* x, 3

**DARPA** Defense Advanced Research Projects Agency *on page(s):* 3, 7, 36

**DAU** Defense Acquisition University *on page(s):* 36, 55

**DE** digital engineering *on page(s):* x–xii, 1–4, 6–38, 40–43, 45–48, 53, 54, 56–58, 89, 91–97, 99, 100, 103, 106–123, 125–131, 133, 137–146, 148, 149

**DEE** digital engineering environment *on page(s):* 7, 8, 19, 26, 45, 92, 93, 143, 145

**DEIX** Digital Engineering Information eXchange *on page(s):* 35, 37, 48, 53, 141, 146

**DES** digital engineering system *on page(s):* ix, 7–9, 12, 13, 25, 27, 28, 31–33, 108, 110

**DL** Description Logic *on page(s):* ix, 38, 39

**DMBoK** Data Management Body of Knowledge *on page(s):* 43

**DoD** Department of Defense *on page(s):* 32

**DRE** digital requirements engineering *on page(s):* 57, 58, 62, 63, 65, 71, 89, 144, 146

**DSL** domain-specific language *on page(s):* 106

**DT** digital transformation *on page(s):* 36

**EAL** Evaluation Assurance Level *on page(s):* 143

**ESA** European Space Agency *on page(s):* 57, 61, 90, 144

**ETL** extract-transform-load *on page(s):* 20

**FOL** first-order logic *on page(s):* 38

**FPGA** field-programmable gate array *on page(s):* 116, 117

**GPU** graphics processing unit *on page(s):* 116

**GtWR** Guide to Writing Requirements *on page(s):* x, xi, 59–62, 64–66, 68–75, 77–81, 84, 86, 88, 89, 147, 148

**GUI** graphical user interface *on page(s):* 23, 38

**HCI** human-computer interface *on page(s):* xi, 26, 27, 33, 36, 50, 99, 107, 113, 114, 118, 119, 123, 126, 129

**HSM** hardware security module *on page(s):* 117, 118

**HTML** HyperText Markup Language *on page(s):* 112

**ICE** Information Content Entity *on page(s):* 40, 47–49, 53

**IEC** International Electrotechnical Commission *on page(s):* 35, 37, 43, 49, 52–55, 58, 65, 67, 104, 110, 121, 141, 144

**IEEE** Institute of Electrical and Electronics Engineers *on page(s):* 37, 49, 52–55, 58, 65, 67, 104, 110, 141, 144

**INCOSE** International Council on Systems Engineering *on page(s):* x, xi, 4, 5, 35, 37, 41, 48, 52, 53, 55–57, 59, 61, 62, 65, 69, 70, 72–81, 84, 86–90, 102, 141, 142, 144, 146–148

**ISA** instruction set architecture *on page(s):* 120, 130

**ISO** International Organization for Standardization *on page(s):* 35, 37, 42, 43, 49, 51–55, 58, 65, 67, 84, 104, 110, 120, 121, 141, 144

**IT** information technology *on page(s):* 14, 49, 55, 92, 141, 144

**ITP** interactive theorem prover *on page(s):* 125

**JPL** Jet Propulsion Laboratory *on page(s):* 19, 78, 79, 89, 142

**JSON** JavaScript Object Notation *on page(s):* 20

**LISP** LISt Processor *on page(s):* xi, 9, 11, 99, 106, 124, 137

**MARTE** Modeling and Analysis of Real Time and Embedded systems *on page(s):* 115

**MBE** model-based engineering *on page(s):* 24, 35, 36

**MBSE** model-based systems engineering *on page(s):* 3, 4, 7, 8, 23, 25, 31, 32, 36, 37, 52, 56–58, 60, 63, 65, 68, 85, 86, 89, 112, 115, 140–142, 144, 146

**MBSR** Model-Based Structured Requirement *on page(s):* x, xi, 4, 5, 61, 62, 68, 69, 71–73, 75–81, 84–90, 142, 144–148

**MBSSE** model-based systems and software engineering *on page(s):* 43, 55

**MGO** mission-goal-objective *on page(s):* xii, 93–95

**MIT** Massachusetts Institute of Technology *on page(s):* 9

**NASA** National Aeronautics and Space Administration *on page(s):* 19, 55, 57–59, 61, 78, 79, 81, 84, 88–90, 142, 144

**NDN** Named-Data Networking *on page(s):* 115

**NIST** National Institute of Standards and Technology *on page(s):* 117, 120

**NLP** natural language processing *on page(s):* 147, 148

**NRM** Needs and Requirements Manual *on page(s):* 65, 72, 84

**NSF** National Science Foundation *on page(s):* 115

**OCAP** object-capability *on page(s):* 117, 118

**OMG** Object Management Group *on page(s):* 38, 39, 59, 110, 146

**OS** operating system *on page(s):* 2, 10

**OWL** Web Ontology Language *on page(s):* 5, 38–40, 43, 87

**PII** personally identifiable information *on page(s):* 117

**POSIX** Portable Operating System Interface *on page(s):* 31, 121

**QAt** quality attribute *on page(s):* 22, 25, 33

**RBAC** role-based access control *on page(s):* 117

**RDF** Resource Description Framework *on page(s):* 38

**RE** requirements engineering *on page(s):* 58, 60–62, 65, 68

**ReqIF** Requirements Interchange Format *on page(s):* 20

**REST** Representational State Transfer *on page(s):* 20

**RMT** requirements management tool *on page(s):* 58–60, 64, 69, 80, 81, 145

**RPC** remote procedure call *on page(s):* 20

**SCION** Scalability, Control, and Isolation On Next-Generation Networks *on page(s):* 115

**SDE** seamless digital engineering *on page(s):* xii, 93, 95

**SE** systems engineering *on page(s):* 8, 9, 32, 33, 62, 67, 81, 84, 89, 141

**SEBoK** Systems Engineering Body of Knowledge *on page(s):* 55

**SIG** Softgoal Interdependency Graph *on page(s):* x, 12, 13, 15–18, 93

**SMT** Satisfiability Modulo Theories *on page(s):* 116, 117, 127

**SoI** system-of-interest *on page(s):* 1–3, 6, 7, 14, 19, 20, 28, 45

**SoS** system-of-systems *on page(s):* 22

**SPARQL** SPARQL Protocol and RDF Query Language *on page(s):* 38

**SQL** Structured Query Language *on page(s):* 20

**SQuaRE** Systems and software Quality Requirements and Evaluation *on page(s):* x, 35, 37, 43–45, 47, 50, 52, 54, 55, 96, 97, 141, 142, 144

**SysML** Systems Modeling Language *on page(s):* x–xii, 4, 5, 12, 20, 21, 23, 28–30, 33, 39, 56, 59–65, 68–73, 75, 76, 78–81, 84–90, 92–96, 98, 102, 110, 120, 122–124, 129, 130, 142–148

**TBD** to be determined *on page(s):* 76

**TLA$^+$** Temporal Logic of Actions *on page(s):* 115

**UML** Unified Modeling Language *on page(s):* xi, 62, 63, 71, 78

**V&V** verification & validation *on page(s):* 58, 62, 65, 69, 73, 75, 76, 80, 85–87, 109, 147

**VM** virtual machine *on page(s):* xi, 137

**W3C** World Wide Web Consortium *on page(s):* 38

**WG** Working Group *on page(s):* 35, 37, 53, 141, 146

**WIMP** Window-Icon-Menu-Pointer *on page(s):* 101

**XMI** XML Metadata Interchange *on page(s):* xi, 20, 78

**XML** Extensible Markup Language *on page(s):* 20, 38, 76

# Chapter 1

# Introduction

> We have also come to realize that no problem ever exists in complete isolation. Every problem interacts with other problems and is therefore part of a set of interrelated problems, a *system of problems*… Furthermore solutions to most problems produce other problems… I choose to call such a system a *mess*.
>
> Russell L. Ackoff [1]

## 1.1 Overview

Engineers and other staff and contractors use tools to develop engineered systems such as integrated circuits, spacecraft, and software that meet stakeholder needs. Primarily, those tools are computer-based and therefore based on the plethora of commercially- and freely-available software running on extant computer hardware. Digital engineering (DE) in particular is an effort to complete this transition and rely solely on a digitized Authoritative Source of Truth (ASoT) of the system-of-interest (SoI) for engineering and associate activities throughout the system development life cycle (Figure 1.1). However, due to the heterogeneity and other complexity factors of those computer-aided engineering (CAE) tools, the achievement of DE is frustrated by integration and usability challenges.

The modern computing industry on which DE depends is experiencing a "normalization of deviance" [3] whereby faulty engineering practices are tolerated and vulnerable components are depended upon, despite their negative impact on project cost, risk, and schedule [4]. Many deficiencies in computing systems are tolerated because workarounds are available and feasible. Defects may lurk inside millions of lines of code for years before they are discovered and corrected. While physical system security may be visualized, digital system security is largely intractable due to the numerous known and unknown defects that cause critical security vulnerabilities (Figure 1.2); the sheer number of components, interfaces, and

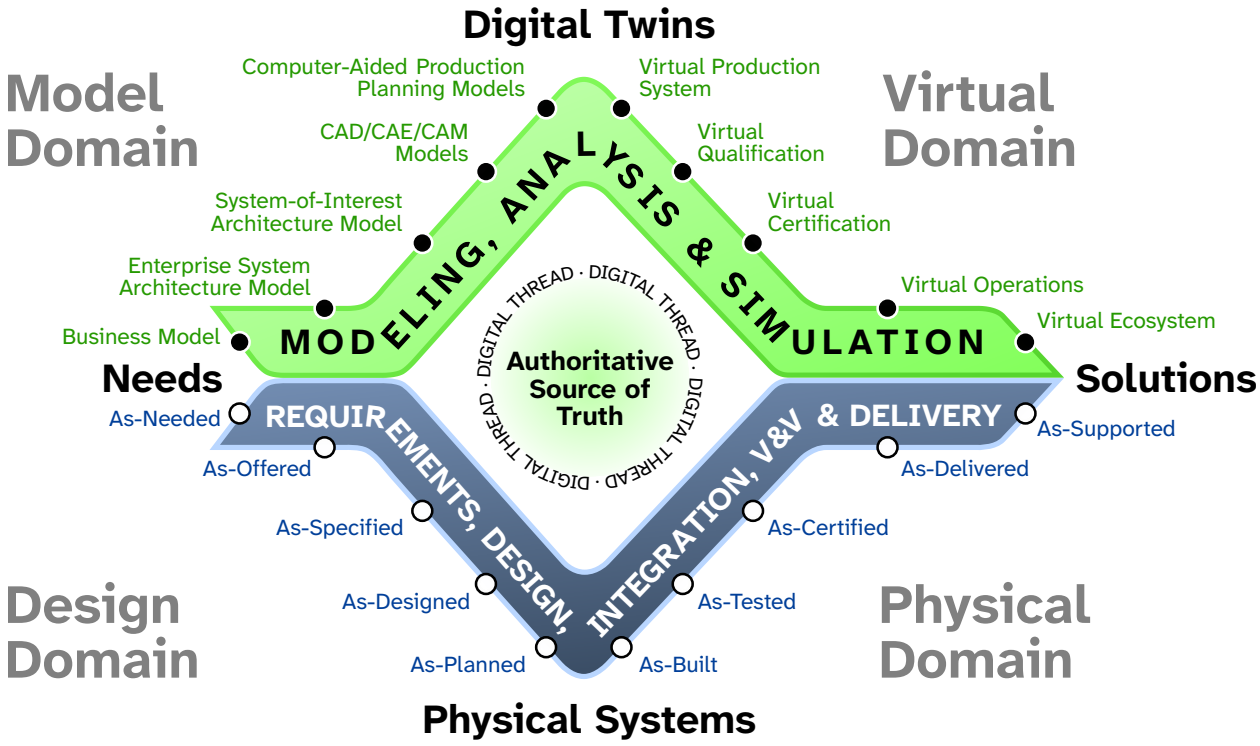



**Figure 1.1:** Digital engineering lifecycle reference model (adapted from [2]).

dependencies (Figure 1.3); the decentralized and inconsistent evolution of those components; and in particular, a profound lack of a cohesive system architecture model that systems engineers and other stakeholders would use to inspect, analyze, validate, verify, and modify the complex digital system.

A useful way of thinking about these digital tooling deficiencies is with the concept of the *reverse salient*, which may be defined as "a set of critical problems" [7]. For example, it may be considered that the computer operating system (OS) or the central processing unit (CPU) is the reverse salient that "fail[s] to deliver the necessary level of technological performance thereby inhibiting the performance delivery of the system as a whole" [8], where the SoI is the DE environment that provides a set of affordances that depend on the underlying computing hardware and software components. The computing system has not been designed to meet the engineers' needs, but rather some software components known as applications have been designed for general use. As systems engineers, we do not tolerate delivering defective systems that frustrate users with unpredictable behavior and random errors, so if we

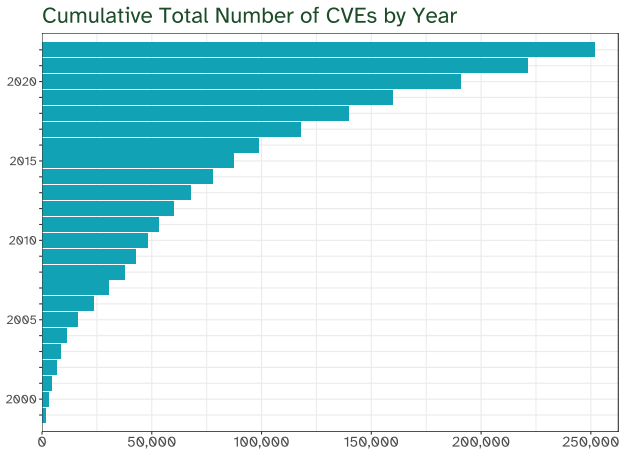



**Figure 1.2:** Cumulative total number of Common Vulnerabilities and Exposures (CVE) by year (data from [5]).

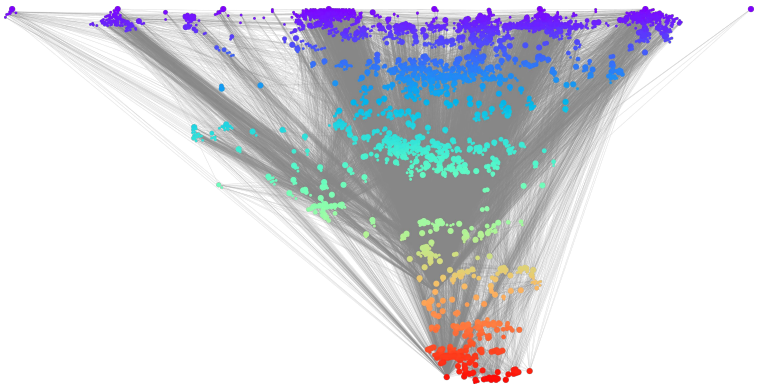


**Figure 1.3:** Dependency iceberg — software dependency graph of 8 selected DE packages, with 3,641 package dependencies, 80,716 dependency links, and 137 layers (data visualization code from [6]).

were contracted to develop our own DE environment, we would naturally apply model-based systems engineering (MBSE), maintain traceability of needs and requirements, and measure Quality Attributes to assure successful delivery of the SoI.

Prior publicly-funded research initiatives have taken a systems approach to this mess. Defense Advanced Research Projects Agency (DARPA) funded such initiatives as Clean-Slate Design of Resilient, Adaptive, Secure Hosts (CRASH) [9] and META-II [10], and later, Circuit Realization At Faster Timescales (CRAFT) [11], recognizing that the systemic vulnerabilities and deficiencies of current computer systems and CAE tools require a clean-slate design to overcome. Reference [12] recognized that "the current paradigm is so thoroughly established

that the only way to change is to start over again". Given the advancements in MBSE, computer science, integrated circuit design, and manufacturing over the past four decades, we have the opportunity to envision a new DE environment that meets our needs for safety, security, integrity, efficiency, and efficacy. Or, as Ref. [12] has put it, "the goal is to move from the current situation of complexity and frustration to one where technology serves human needs invisibly, unobtrusively: the human-centered, customer-centered way", a goal that ought to resonate with systems engineers and computer users alike.

## 1.2 Research Questions

The research questions that motivated this work are the following:

**RQ1** What is Seamless Digital Engineering and what distinguishes it from Digital Engineering?

**RQ2** How do we precisely define Seamless Digital Engineering and its related concepts?

**RQ3** How do we define requirements in Seamless Digital Engineering?

**RQ4** How does the proposed Seamless Digital Engineering Reference Architecture meet the needs of "bootstrapping a trustworthy and seamless digital engineering appliance"?

## 1.3 Structure of this Dissertation

These research questions are answered in sequence in this dissertation. Chapter 2 summarizes the current *mess* in digital engineering, defines Seamless Digital Engineering as a tooling paradigm, and presents a grand challenge to develop a seamless digital engineering environment from a clean slate. Chapter 3 more precisely defines Seamless Digital Engineering using methods in ontology engineering, disambiguating it into constituent concepts, giving meaning to 'seamless' and 'trustworthy'. Chapter 4 presents digital requirements engineering in Seamless Digital Engineering, extending the Model-Based Structured Requirement (MBSR) Systems Modeling Language (SysML) profile with an INCOSE-derived meta-model based on the latest guidance on writing high-quality requirements. Chapter 5 presents a reference

architecture defined in SysML v2 for seamless digital engineering environments. Chapter 6 summarizes the research contributions, outlines future work, and concludes the dissertation.

To summarize, the contributions of this dissertation are the following:

- Definition of seamless digital engineering and identification of it as a grand challenge in digital engineering research, with accompanying design patterns, requirements, and architecture tenets that characterize it;
- Seamless Digital Engineering Ontology defined in Web Ontology Language (OWL) 2, which logically defines bootstrappability, trustworthiness, and seamless quality attributes, and which provides many standards-based ontological definitions in systems engineering and digital engineering;
- Extensions to the Model-Based Structured Requirement (MBSR) SysML (v1) profile, which conform to the latest International Council on Systems Engineering (INCOSE) guidance on writing requirements, defining important features of digital requirements engineering; and
- Reference architecture for trustworthy and seamless digital engineering systems, defined in SysML v2, including a model library for Model-Based Structured Requirement (MBSR).

# Chapter 2

# A Grand Challenge Driven by Needs[1]

> We propose a grand challenge for the formal methods community: build and mechanically verify a practical embedded system, from transistors to software.
>
> J. Strother Moore [14]

This chapter addresses RQ1, which is restated as follows: *What is Seamless Digital Engineering and what distinguishes it from Digital Engineering?*

To address this question, the chapter characterizes Seamless Digital Engineering as a grand challenge in digital engineering research. It defines Seamless DE as a tooling paradigm that leverages end-to-end formal verification to achieve the needed reliability and integration correctness for digital engineering tools. The chapter analyzes digital engineering from the perspective of tooling reliability and compares common integration patterns in digital engineering to the proposed seamless integration pattern. Finally, it offers a base set of requirements derived from prior work and architecture tenets to guide future research and development.

## 2.1 Introduction

Engineers and other staff and contractors primarily use tools (see Table 2.1) to produce engineered systems such as integrated circuits, spacecraft, and software that meet stakeholder needs. Increasingly, those tools are computer-based and therefore dependent on the wide range of commercially- and freely-available computer software and hardware. Digital engineering (DE), in particular, is an effort to complete this transition and rely solely on a digitized Authoritative Source of Truth (ASoT) of the system-of-interest (SoI) for engineering and

[1]The contents of this chapter are based on the paper "Seamless Digital Engineering: A Grand Challenge Driven by Needs" published in AIAA SCITECH 2024 Forum, No. AIAA 2024-1053 [13].

associated activities throughout the system development life cycle. However, due to the heterogeneity and other complexity factors of those computer-aided engineering (CAE) and ancillary tools, the achievement of DE is frustrated by integration and usability challenges.

Many deficiencies in computing systems are tolerated because workarounds are available and then implemented, further entrenching their intractability. The intractability of DE is due to the high number of computing components and interfaces, the decentralized and inconsistent evolution of those components, and in particular, a profound lack of a cohesive system architecture model that systems engineers and other stakeholders would use to inspect, analyze, validate, verify, and modify the complex digital system. A useful way of thinking about these digital tooling deficiencies is with the concept of the *reverse salient*, which may be defined as "a set of critical problems" [7] or components that "fail to deliver the necessary level of technological performance thereby inhibiting the performance delivery of the system as a whole" [8], where the SoI is the digital engineering environment (DEE) that provides a set of affordances that depend on the underlying computing hardware and software components. The computing system has not been designed to meet the engineers' needs, but rather software components known as applications have been designed for general use. As systems engineers, we do not tolerate delivering defective systems that frustrate users with unpredictable behavior and random errors, so if we were contracted to develop our own digital engineering system (DES), we would naturally apply MBSE with full requirements traceability, and track the Quality Attributes to guarantee success.

Prior publicly-funded research initiatives have had similar goals. In the recent past, Defense Advanced Research Projects Agency (DARPA) has funded such initiatives as Clean-Slate Design of Resilient, Adaptive, Secure Hosts (CRASH) [9], META-II [10], and Circuit Realization At Faster Timescales (CRAFT) [11], recognizing that the systemic vulnerabilities and deficiencies of current computer systems and CAE tools require a clean-slate design to overcome. [12] also recognized that "the current paradigm is so thoroughly established that the only way to change is to start over again". Given the advancements in model-based

**Table 2.1:** Definitions of related engineering terms (**emphasis** added).

| Term | Definition |
|---|---|
| Tool | "The external employment of an unattached or manipulable attached environmental object **to alter more efficiently** the form, position, or condition of another object, another organism, or the user itself, when **the user holds and directly manipulates the tool** during or prior to use **and is responsible for the proper and effective orientation** of the tool." [15] |
| Technology | "Artifacts made through a systematic application of knowledge and used to reach practical goals" [16] **"in a specifiable and reproducible way"** [17] |
| Engineering | "The creative application of scientific principles to design or develop structures, machines, apparatus, or manufacturing processes, or works utilizing them singly or in combination; or to construct or operate the same **with full cognizance of their design**; or **to forecast their behavior under specific operating conditions**; all as respects an intended function, economics of operation **and safety to life and property**." [18] |
| Systems engineering (SE) | "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." [19]<br>"Interdisciplinary approach governing **the total technical and managerial effort** required to transform a set of customer needs, expectations, and constraints into a solution and to support that solution **throughout its life**." [20] |
| Model-based systems engineering (MBSE) | "The formalized application of modeling to support system requirements, design, analysis, verification and validation activities beginning in the conceptual design phase and continuing throughout development and later life cycle phases." [21] |
| Digital engineering (DE) | "An enterprise approach for incorporating data-driven infrastructure, methods and tools, innovation, and workforce transformation centered on an **authoritative source of truth (ASoT)** for engineering data." [22]<br>"DE is a transformation of the implementation of all engineering (and associated disciplines) across an enterprise to **take full advantage of the digital integration** of engineering work, data, knowledge, and wisdom across that enterprise." [23] |
| Authoritative Source of Truth (ASoT) | "The authoritative source of truth captures the current state and the history of the technical baseline. It serves as the central reference point for models and data across the lifecycle. The authoritative source of truth will provide traceability as the system of interest evolves, capturing historical knowledge, and connecting authoritative versions of the models and data." [22] |
| Digital engineering environment (DEE) | The "enterprises' interconnected digital environments, stakeholder-networks, and semantic data that allows the exchange of digital artifacts from an authoritative source of truth to serve the stakeholder communities' interests." [24] |
| Digital engineering system (DES) | A theoretical engineered system-of-systems that provides the networked computing resources and human-computer interfaces for an enterprise to accomplish all digital engineering activities for successful realization of engineered systems. |

systems engineering (MBSE), computer science, integrated circuit design, and manufacturing over the past four decades, we have the opportunity to envision a fresh DES that meets our needs for safety, security, comprehensiveness, efficiency, and efficacy. The goal then "is to move from the current situation of complexity and frustration to one where technology serves human needs invisibly, unobtrusively: the human-centered, customer-centered way" [12], a goal that ought to resonate with systems engineers and computer users alike.

The primary contribution of this chapter is to clarify the definition of *Seamless Digital Engineering* in contrast with how DE is currently being implemented and identify it as a grand challenge in DE research. In Section 2.2, historical and related research is summarized in relation to seamless DE. Section 2.3 characterizes the DE tooling problem space and current integration patterns for achieving DE outcomes. Section 2.4 defines seamless DE so it can be subject to further SE and DE research. Finally, Section 2.5 discusses how the achievement of seamless DE would constitute a grand challenge in DE research typically beyond the capability of any one organization. Some research conclusions and a summary of future work follow.

## 2.2 Related Work

The theoretical and practical foundations for DE have been developing for at least a half-century, dating back to ARPA-commissioned research by Massachusetts Institute of Technology (MIT) on LISt Processor (LISP) machines [25–27] which aimed to "help designers get from prototype to product faster" and whose "key [was] an open architecture and highly-integrated development tools". Along with Smalltalk computing systems [28] that represented a similar trajectory in integrated tool-based computing environments, LISP machines are virtually nonexistent today, and we risk forgetting the many lessons and advancements made in those early years of computing. Later research has focused on *ex post facto* integration of computer-based tools.

Reference [29] described a UNIX-based integrated tooling system that assembled basic components provided by the OS in combination with custom software fit-to-task, not unlike what we practice today. [30] provides an essential formalism of tool integration that decomposes the concept into four: 1) process integration, 2) data integration, 3) presentation integration, and 4) control integration. That paper describes system properties, such as consistency, coherence, nonredundancy, and synchronization, that are still relevant today, and in particular, "tools are well integrated when they have a common view of data". Although these efforts focus on computer-aided software engineering (CASE), the principles and lessons are generalizable to DE.

Seamless model-driven systems engineering was first presented in Refs. [31, 32] and later in [33]. [32] identifies three ingredients to seamless integration, i.e., "deep, coherent, and comprehensive integration of models and tools": 1) a modeling theory that serves as the basis of the ASoT's semantic domain, 2) a complete architectural model that describes product and process, and 3) "a manner to build tools that conform to the modeling theory and allow the authoring of the product model", noting that the scientific basis is available but political barriers thus far prevent its realization. [32] contributes critical theories and observations informed by automotive and aeronautical industry practice, such as a conceptual model for an integrated model engineering environment based on horizontal and vertical tooling aspects and the useful notion that currently available tools impose the current situation rather than the ideal case of formally-defined modeling theory dictating tool support. The rationale for this vision may be familiar to DE practitioners, such as: lack of model integration at both the conceptual and tooling level; transitions between artifacts are ad-hoc, unclear, and error-prone; redundancy of models that lose product information, leaving some implicit in engineers' minds; the high costs of internally developing tools and the unsatisfactory tool support for domain-specific modeling in the organization's core business; and that formal verification is essential to producing complex systems that are correct-by-design but that today we only benefit from "islands of success". [32] uses overlapping views of *seamless*: seamless

model-based development, seamless systems engineering, and seamless system development, and offers the following definition:

> "Seamless model-based development promises to lift software development to higher levels of abstraction by providing integrated chains of models covering all phases from requirements to system design and verification. In seamless model-based development, modeling is not just an implementation method but also a paradigm that provides support throughout the entire development and maintenance lifecycle."

This definition is software development-focused and lacks precision with respect to 'higher levels of abstraction'; while it covers the complete system development lifecycle, it refers to "integrated chains of models" that somewhat contrasts with DE's ASoT; lastly, it highlights this approach as a *paradigm* as opposed to solely an implementation method. [33] defines *seamless* as an approach to software and system modeling where "all models and all development steps are in a tight relationship with a clear understanding how they benefit", but this definition appears to lack the precision needed to determine if the modeling approach is in fact seamless. Clarity of what qualifies as a "tight relationship" among models and steps would improve our understanding of seamless model-based development. Finally, the first published use of the term "seamless digital engineering" was by [34], which describes knowledge integration as a key component to DE, but that nevertheless leaves the definition implicit and subject to the reader's interpretation of seamless.

## 2.3 Characterizing the Problem Space

As Ref. [12] succinctly notes, "The personal computer is perhaps the most frustrating technology ever. The computer should be thought of as infrastructure". Perhaps we do not think of computers as infrastructure but rather as suites of products. This mental model may be clouding our vision of future computing technology that suits our needs as engineers and communicators. Where much research after LISP machines has assumed the von Neumann

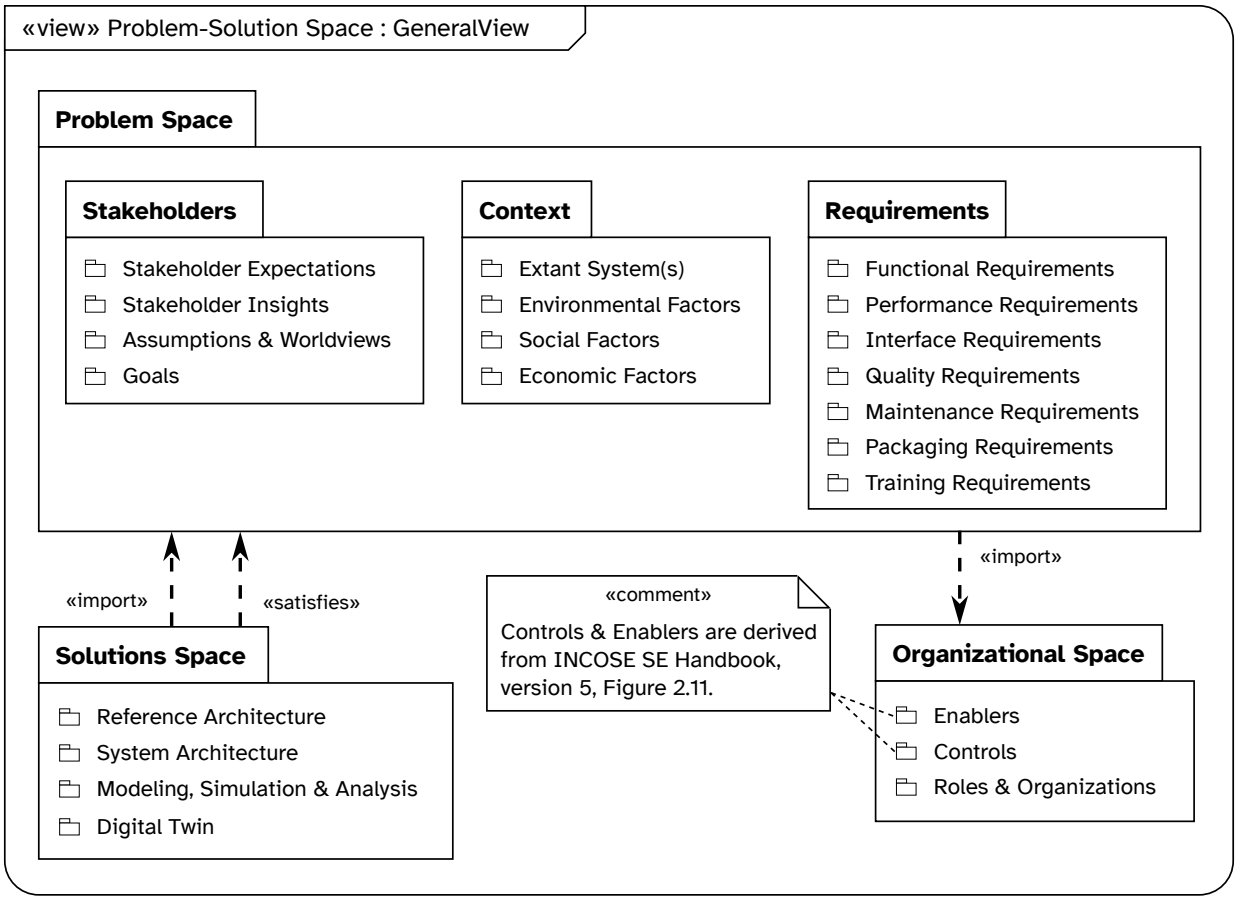



**Figure 2.1:** Generic Problem-Solution Space (SysML v2 depiction).

computer architecture and widely-deployed operating systems as inviolable, a clean-slate approach favors the systems engineering practice of not making point-design decisions too early in the system development lifecycle. Let us first identify and characterize the problem space and then move on to proposing a solution space (see Figure 2.1).

### 2.3.1 Stakeholders and Goals[2]

Since DE encompasses the entire system development lifecycle, the stakeholders of a DES are varied and many. We provide here a list (see Table 2.2) of potential stakeholders for a DES as a starting point, noting that some stakeholders listed, such as Engineer, may be further specialized depending on the role.

[2]This section has been expanded from the original to include goals analysis using Softgoal Interdependency Graph (SIG).

**Table 2.2:** Stakeholders of a DES.

| | | |
|---|---|---|
| Engineer | Project/Program Manager | Accountant |
| Acquirer | Contractor | Auditor |
| Manufacturer | Technician | Designer |
| Supplier | Regulator | Technical Writer |

The original DE goals [22] are the visionary statements used to guide any DE implementation effort. The goals provide enough detail for us to link Stakeholder Expectations to them in a hierarchical, traceable relationship structure. Enumerated below, the DE goals are among the first elements to add to our reference architecture model.

`G-DE-1` Formalize the development, integration, and use of models to inform enterprise and program decision-making.

`G-DE-2` Provide an enduring, Authoritative Source of Truth (ASoT).

`G-DE-3` Incorporate technological innovation to improve the engineering practice.

`G-DE-4` Establish a supporting infrastructure and environments to perform activities, collaborate, and communicate across stakeholders.

`G-DE-5` Transform the culture and workforce to adopt and support DE across the lifecycle.

While some aspects of the above DE goals depend on the organization's execution of the DE processes, e.g., "transform the culture and workforce", those goals may nevertheless be supported by a comprehensive DES that incorporates documentation and training resources.

Using Ref. [22] as the primary source of DE goals, Softgoal Interdependency Graph (SIG) notation [35] was used to analyze the goals, decompose them, and link their explicit and implicit relationships, giving short names to goals for greater readability (Figure 2.2). Four super-goals were identified by analysis of the primary source text, each with their own decompositions (Figure 2.3). The expected benefits of DE described in Ref. [22] were modeled as Claim Softgoals [35] in the SIG language, which contribute toward the 5 DE goals and their implicit or explicit interdependencies. This analysis was performed for all 5 goals, leading to up to four levels of decomposition down to Operationalizing Softgoals [35] (Figures 2.4,

2.5, 2.6, 2.7, 2.8). While analysis of the DE goals in this way revealed hidden depth and breadth, the DE goal of incorporating technological innovation (G-DE-3) (Figure 2.6) in particular lacked Operationalizing Softgoals. This result may indicate a weak point in the DE strategy as described in Ref. [22] is that it underrepresents the complexity of the challenge of developing supporting IT infrastructure and infusing technological innovations to support digital engineering (DE).

### 2.3.2 DE is an Adaptation of Existing Tools and Available Computing Systems

Decades of competitive innovation in CAE tools have led to thousands of different software products in various functional areas throughout the product development life cycle. Startups entering high-tech fields such as integrated circuit design must first acquire a set of expensive software tools to begin their design work, with no guarantee that the various tools will be able to maintain a coherent system model. DE is an emerging paradigm where an ASoT of the SoI is maintained for the entire product development lifecycle, supporting Digital Thread and Digital Twin [36, 37], and promising to streamline the engineering of complex systems. However, it prescribes no particular way of accomplishing this streamlined DE environment.

The burden of maintaining this organization-wide system of systems drains organization resources away from engineering systems: requiring data engineers and information technology administrators, more computing equipment, and the operational cost of vendor contracts that provide (and may eventually terminate) licenses and technical support. Moreover, staff is frequently switching between tools to accomplish their tasks, requiring productivity-draining context switches and training in each software product while being subject to vendors' upgrades that often modify or deprecate functions that have become integral to the organization's workflow.

[38, 39] outline the OpenCAE tooling system, which amounts to an instantiation of the DE concept. Over 50 software products and languages are identifiable from the presentation

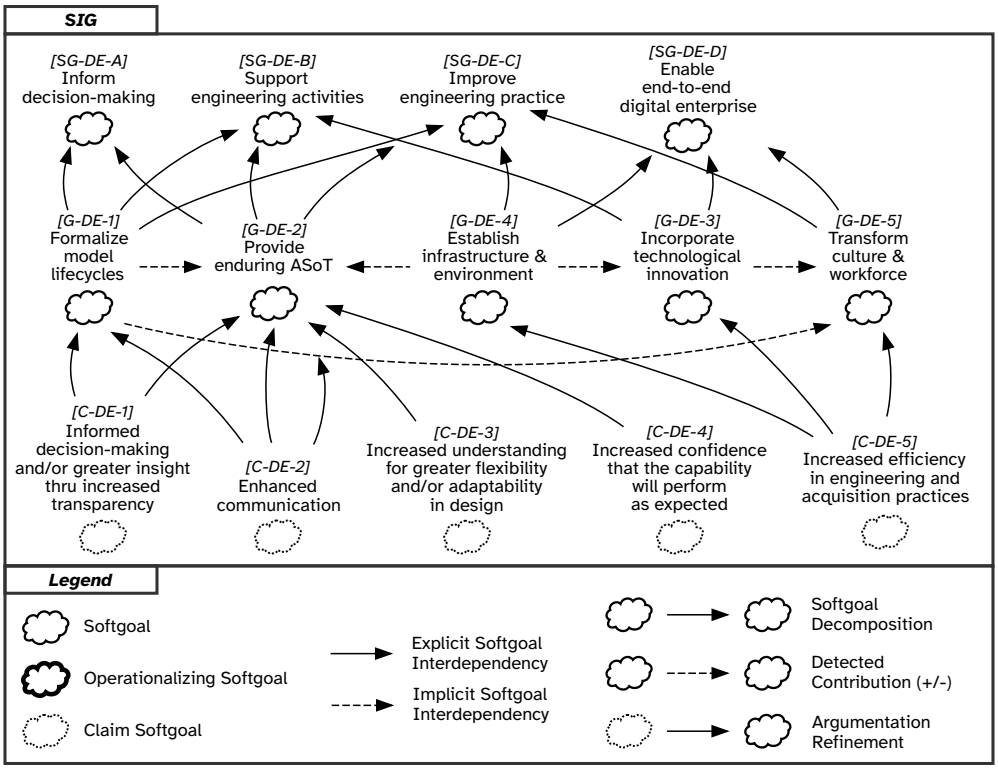



**Figure 2.2:** Softgoal Interdependency Graph (SIG) [35] of DE Goals (decomposed in Figures 2.4, 2.5, 2.6, 2.7, 2.8), Supergoals, and Claims (derived from [22]).

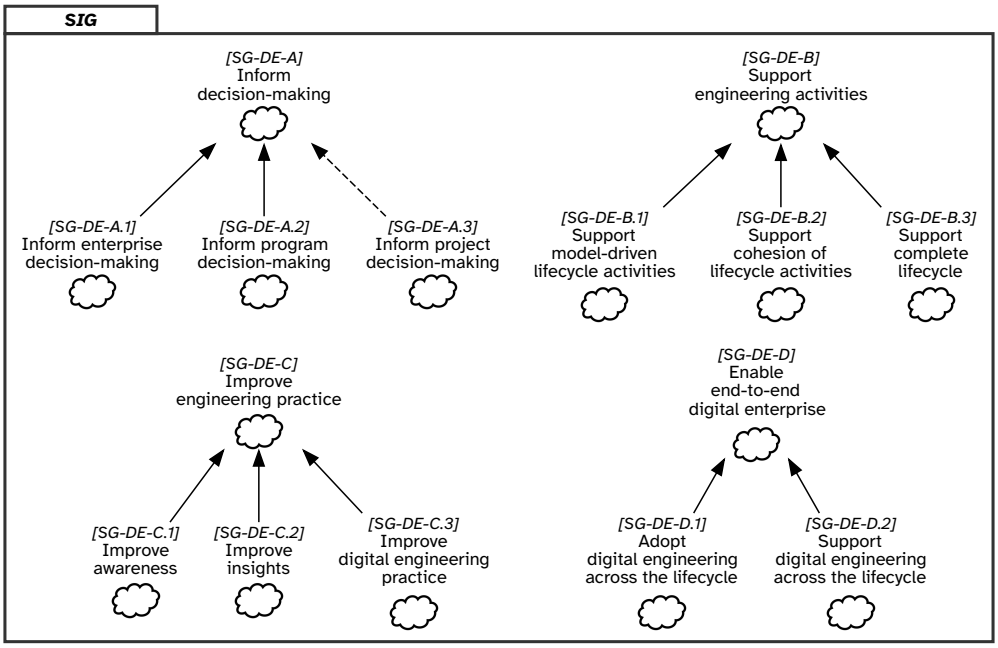



**Figure 2.3:** SIG of decomposition of DE Supergoals from Figure 2.2 (derived from [22]).

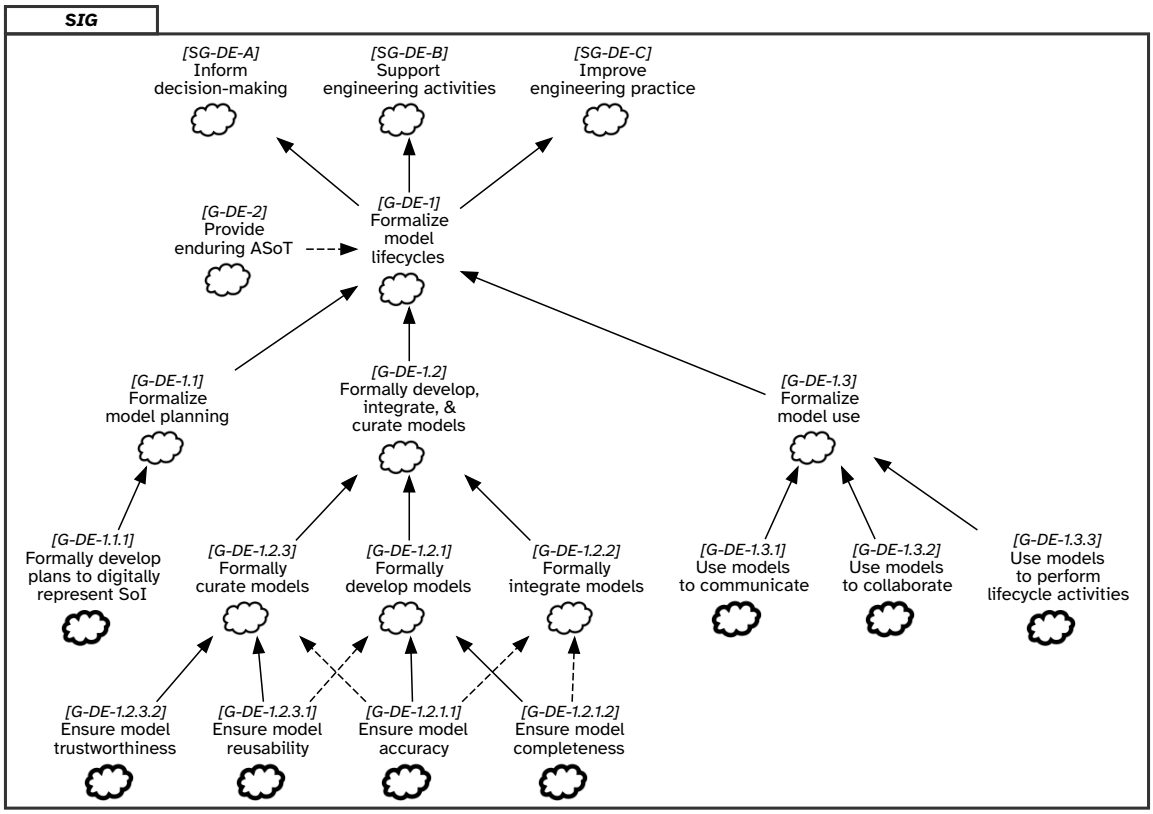



**Figure 2.4:** SIG of DE Goals and Objectives of G-DE-1 from Figure 2.2 (derived from [22]).

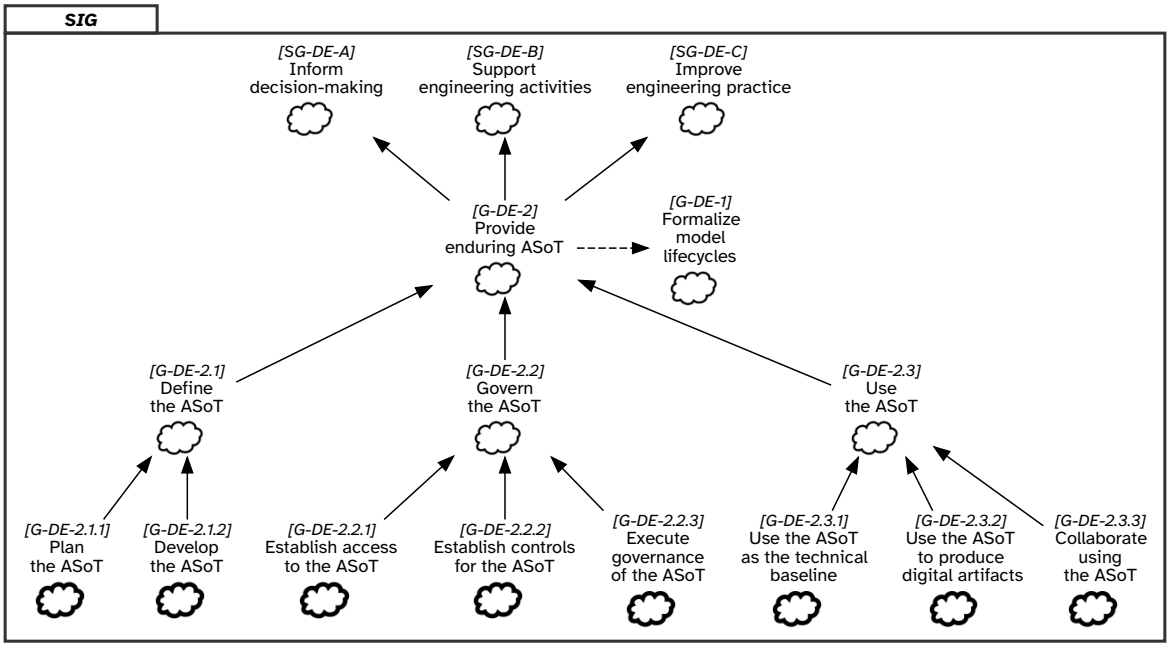



**Figure 2.5:** SIG of DE and Objectives of G-DE-2 from Figure 2.2 (derived from [22]).

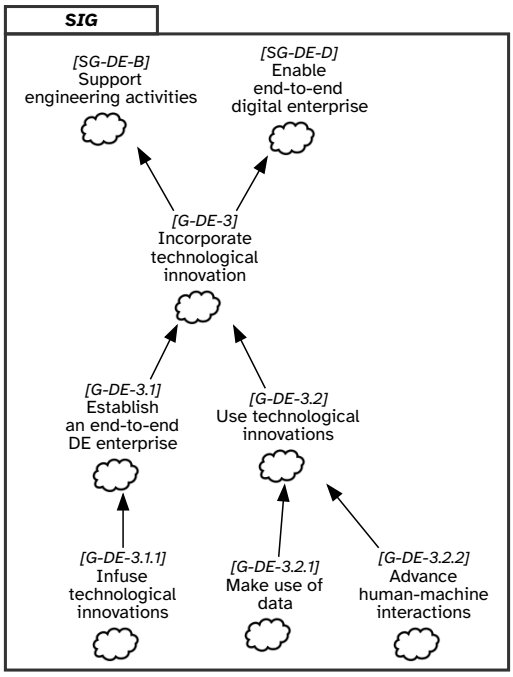



**Figure 2.6:** SIG of DE Goals and Objectives of G-DE-3 from Figure 2.2 (derived from [22]).

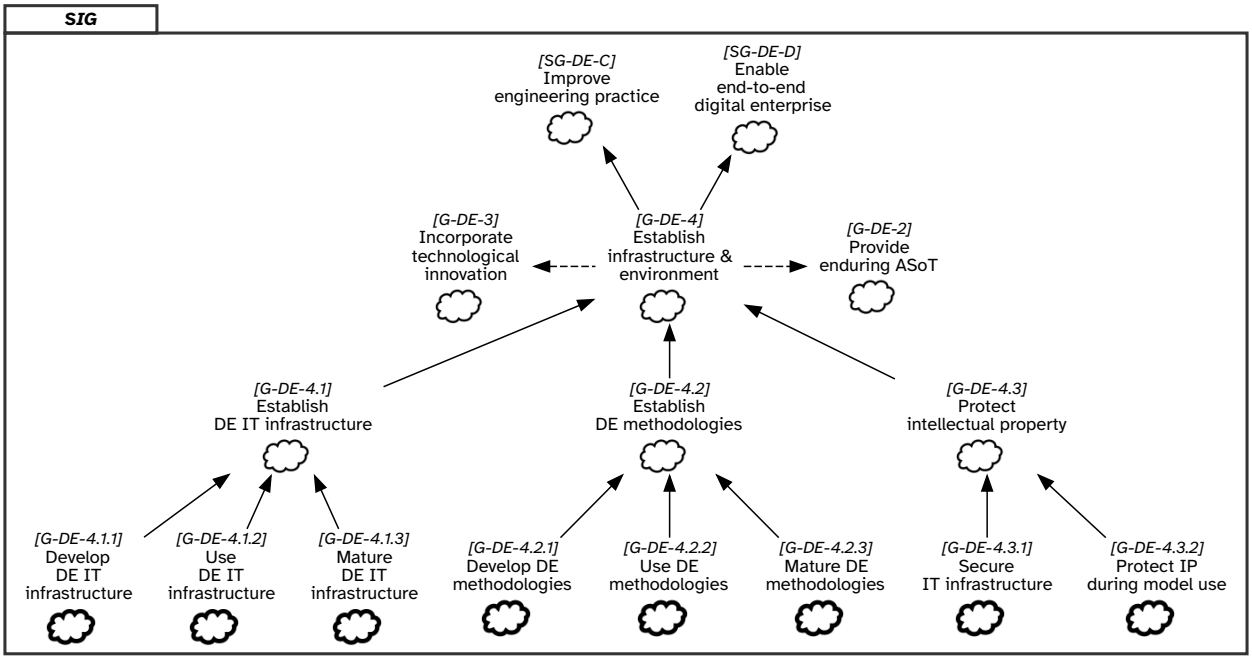



**Figure 2.7:** SIG of DE Goals and Objectives of G-DE-4 from Figure 2.2 (derived from [22]).

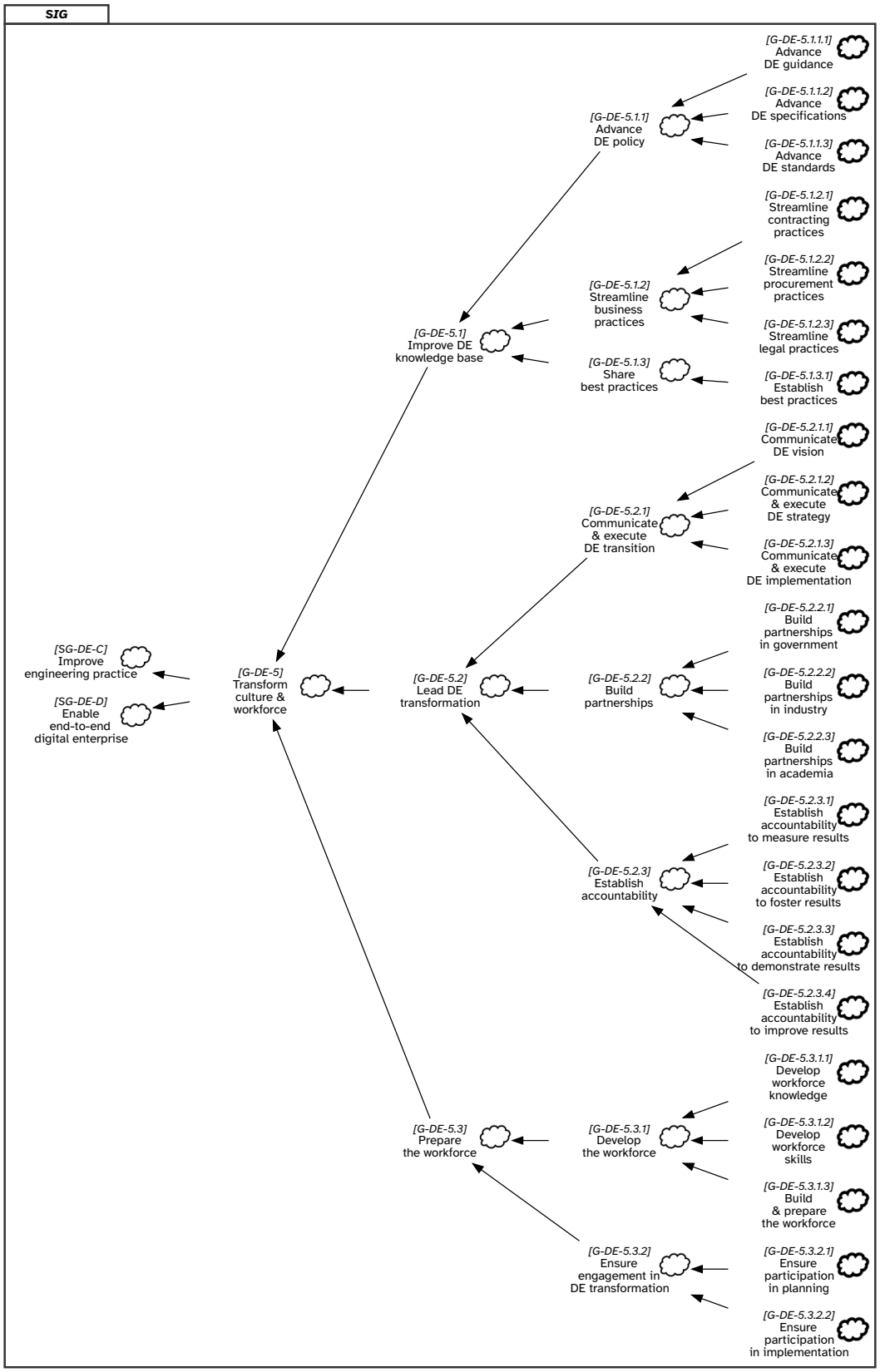



**Figure 2.8:** SIG of DE Goals and Objectives of G-DE-5 from Figure 2.2 (derived from [22]).

in 2019. The presenter notes that the NASA Jet Propulsion Laboratory (JPL) uses "14 server set-ups – over 200 servers" in cloud compute infrastructure and software-as-a-service to achieve their DEE. An Acknowledged System-of-Systems (ASoS) of this size and heterogeneity is likely to exhibit emergent fragility as the various software products are updated out-of-sync with one another. The academic and industry communities would benefit from more published research that studies the incongruities of computing system-of-systems of this kind. While the details of these deployed DEE are often proprietary, here we can acknowledge their complexity and the effort required to integrate the various components to achieve DE's goals.

### 2.3.3 Common DE Integration Patterns

Software components can be made to work together typically using what is referred to as a 'glue' or shim component. Even software that exposes the common UNIX-style plain text interface must nevertheless be integrated using a shim or glue component that parses the text, interprets any data, assembles a model, and then performs model transformations while striving to preserve model invariants, though typically without any formal guarantees. The more humans are involved in these integration processes, the more human error is introduced, resulting in SoI defects that may compromise cost, schedule, and risk budgets.

#### Import/Export Pattern

The most basic pattern of DE integration is import/export, where the engineering model is exported to a standardized or proprietary file format and then imported into another tool (see Figure 2.9). This pattern relies on extra functionality in the tools, a clear specification of the file format, and an accurate implementation thereof. Nevertheless, the pattern suffers when model information is lost during export and/or import due to incomplete implementations and incompatible meta-models.

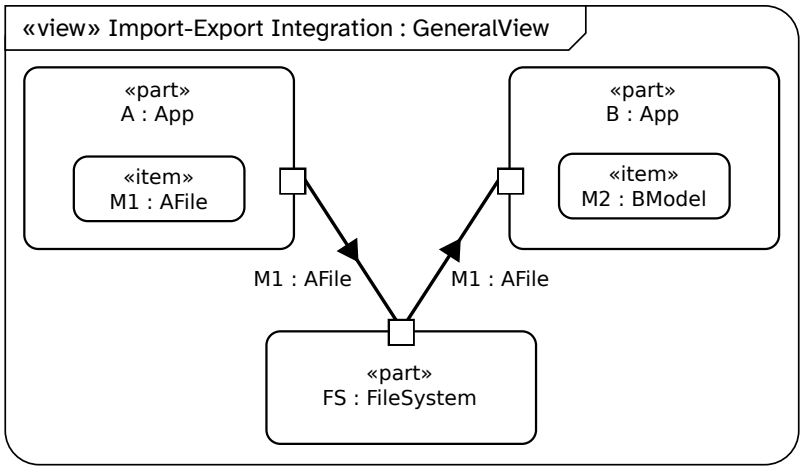



**Figure 2.9:** Import/Export method of DE tool integration (SysML v2 depiction).

### Custom Shim Pattern

When DE tools produce engineering artifacts with known schemas that are stored in known formats, custom shim software may be written to automate extract, transformation, load (ETL) operations (see Figure 2.10). Examples of such artifacts include XML-based formats such as XML Metadata Interchange (XMI) and Requirements Interchange Format (ReqIF), comma-separated values (CSV) or Excel spreadsheets, standard Structured Query Language (SQL) databases, and JavaScript Object Notation (JSON) data that may be loaded into a NoSQL database. The common drawback of this method, besides the additional software engineering effort, is that file formats and database schemas frequently change and often without due notice, resulting in brittle shims that must be urgently corrected to maintain the DE ASoS. Custom shims are also written when integrating software tools that expose RESTful or RPC-style APIs, with similar drawbacks when API specifications change.

### API Pattern

SysML v2 champions an API-based method of integration by exposing a RESTful interface (see Figure 2.11) to the SysML model of the SoI [40]. Other DE tools are meant to use this RESTful API to query attributes and properties of the SoI model such that they are no

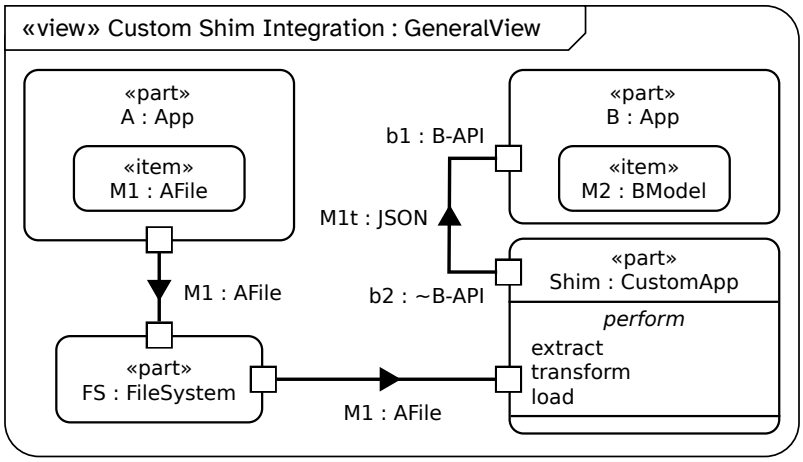



**Figure 2.10:** Custom Shim method of DE tool integration (SysML v2 depiction).

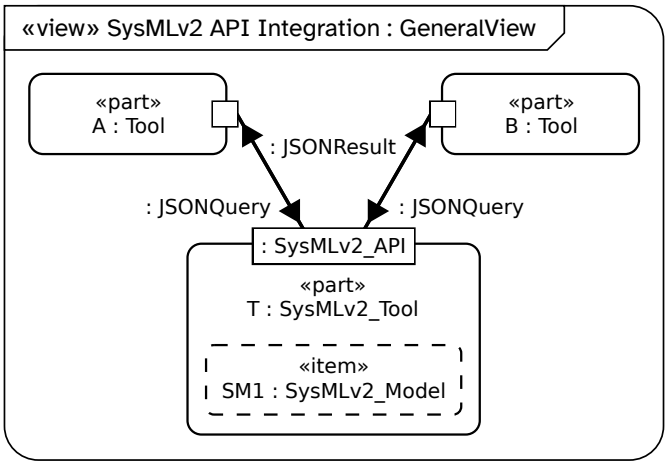



**Figure 2.11:** SysML v2 API method of DE tool integration (SysML v2 depiction).

longer duplicated in the separate tools, thereby reducing opportunities for emergent model definition divergence.

### 2.3.4 DE As-Implemented Cannot be Elegant or Seamless

Before the commodification of computer hardware and software, computer-aided tools were designed-to-purpose (SAGE [41] is offered as an illustrative example). Although the cost of such systems was extremely high due to custom development, they did not have as many computing standards and conventions to support and thus avoided the complexity of supporting and interfacing with non-essential or generalized technologies. Nowadays, it is often assumed that computing systems must support the plethora of computer technologies

provisioned by Microsoft Windows or GNU/Linux to avoid 'reinventing the wheel'. This decision, while pragmatic, can suffer from a sunk-cost fallacy that limits the effectiveness of the deployed system and results in *kludge*[3].

Existing DE and CAE tooling systems lack elegance. *System elegance* is a quality attribute (QAt) defined as "a system that is robust in application, fully meeting specified and adumbrated intent, is well-structured, and is graceful in operation" [43–46]. The four characteristics of elegant systems are [47]:

- System efficacy,
- System efficiency,
- System robustness, and
- Minimizing unintended consequences.

DE as an ASoS lacks efficacy because the tools are nevertheless unaware of the ASoT and liable to introduce defects. It lacks efficiency because the ASoS must itself be developed, deployed, and supported as a fixed and operational cost whose only opportunity for recoupment is the efficiency and quality gains not otherwise available. Finally, its unintended consequences derive from the independent governance and development lifecycles of the adopted software components, whose emergent interactions as the SoS ages are impossible to predict with precision.

CAE tools, in general, do not consider the complete development lifecycle activities to be in-scope and only have plug-in and import/export support for other languages and tools. This scope limitation and typical reliance on proprietary data formats limit its efficacy and efficiency since the engineer must context-switch and adapt to the recalcitrant tool rather than the tool being adaptable to the engineer's needs. Additionally, import/export capabilities often have unintended consequences as even international standards may have ambiguity

[3]Kludge is defined as "a badly assembled collection of parts hastily assembled to serve some particular purpose (often used to refer to computing systems or software that has been badly put together)" [42].

and often explicitly leave areas of the standard implementation-defined, so tools will have differing interpretations causing unintended consequences.

Generally speaking, seamless is defined as "perfectly consistent and coherent" and "not having or joined by a seam or seams" [42], i.e., "the fissure or gap formed by the imperfect union of two bodies laid or fastened together" [48]. In systems engineering terms, the seam is the system architecture element that joins two or more interfaces and may employ parts and behaviors of its own to satisfy its traced requirements. So to be seamless, this joining element must not be necessary. This criterion has only limited satisfaction today, where multiple different model-based activities are possible from within the same tool (e.g., executable SysML and instrumented GUIs for system behavior simulations and early prototypes).

### 2.3.5 Existing Computer Platforms are Nonviable

A driving model for this research is The Error Avalanche as described in [49] and shown in Figure 2.12: that not only will errors be introduced into the delivered system during the system development life cycle as a consequence of human error — even with the support of CAE tools — but the computer hardware and software used to design that system may also compromise the integrity of the system model and introduce subtle defects into the system. The existence of *mercurial cores* that silently give the wrong calculation [50] illustrates this second point, a rather alarming discovery following the early Pentium FDIV bug [51] and remotely-exploitable Meltdown [52] and Spectre-class [53] CPU microarchitecture flaws that were predicted in [54].

By now, it should be well-known that endpoint security is the weakest link in networked computer architecture and that the current approach to cybersecurity is nonviable [55–59]. And so uncritical adoption of existing computing technologies will eventually compromise enterprise security and thus compromise the robustness and 'minimizing unintended consequences' properties of the DE tooling system. During a time when international competition is driving the adoption of MBSE and DE, and corporate espionage and data sabotage are

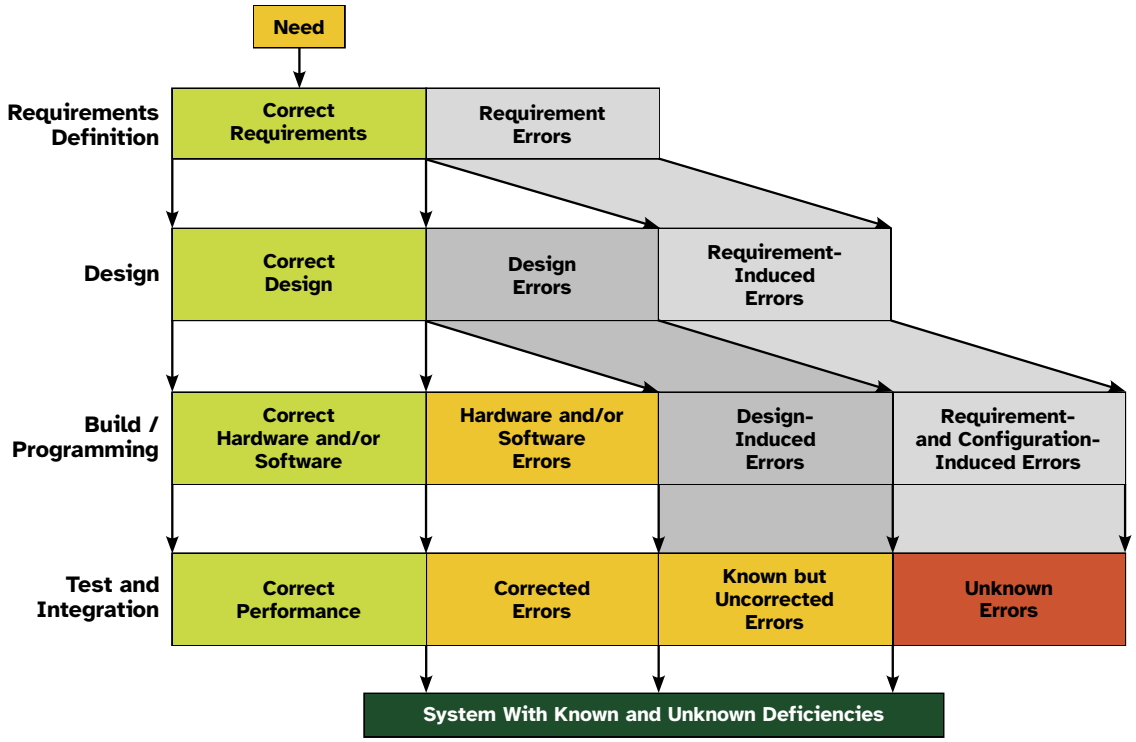



**Figure 2.12:** The Error Avalanche (adapted from [49]).

serious risks in the presence of advanced persistent threats (APTs), a clean-slate DE design must specify the full stack of computing hardware, firmware, and software to avoid making the same mistakes that we normally retain for backward-compatibility.

## 2.4 Defining Seamless Digital Engineering

Prior descriptions of seamless DE or seamless model-based engineering (MBE) environments have left the contextual definition of seamless implicit [32, 34]. In this chapter, the following definition of Seamless DE is offered with the additional criterion of end-to-end formal verification to guarantee seamless, error-free operation:

> **Seamless Digital Engineering** is a digital engineering tooling paradigm that guarantees model coherence and integrity by affording an elegant human-computer interface for systems modeling that is end-to-end formally verified down through the computer hardware.

### 2.4.1 Seamless DE is a Refinement of DE

In order to avoid the unmitigated complexity, hidden costs, and risks of existing computer systems, we must work to design a clean-slate DES that is capable of meeting stakeholder needs. There is a great opportunity here to avoid previous compromises and backward compatibility that have plagued existing computer systems, which are now the reverse salient that holds back the envisioned capability of DE and MBSE. Seamless digital engineering refines the definitions of DE quoted in Table 2.1 by adding the *seamless* requirement for achieving the *elegance* QAt as discussed above. We propose that this refinement may guide future research and collaboration toward a standardized and generational DES.

In light of the research summarized in Section 2.2, a critical property of a successful DES must be that the ASoT or model integrity is guaranteed. A failure of integrity is an unintended change of the model data or "the condition in which data are identically maintained during any operation, such as transfer, storage, and retrieval" [60]. Whether such a failure is corrected or not, it will inevitably add cost and risk to the system development project. The DES should be capable of detecting and automatically correcting errors wherever possible, leaving humans to their design activities rather than fixing and cleaning up after broken tools. The promise of computers in this respect has not yet been fully realized.

All the DE goals [22] are addressable by an adequate system architecture. A seamless DES would be designed with those goals defined in the system architecture model to support their bidirectional traceability for verification and validation. Although DE goal `G-DE-5` regarding "transform the culture" is beyond the scope of the system, an adequate architecture model of the Seamless DES would include an operational viewpoint showcasing how common operational processes (workflows) would be accomplished using the system. And while DE metrics are now collated and processed with great effort, a capable Seamless DES would provide toggleable instrumentation for process and product quality measurements with automated, configurable dashboards for monitoring DE effectiveness across the organization or project.

The vision of computers as invisible infrastructure [12, 61] still seems worlds away when compared to our daily experience. And so, with respect to the presentation and process integration described in [30], seamless integration must also encompass human-computer interface (HCI) ergonomics. We may even strive for computing systems that are *delightful* to use and help guide and focus the user on their activities and tasks by incorporating HCI and human factors engineering research into the design.

### 2.4.2 Seamless Integration Requirements

Tool integration problems are central to DE transformation efforts at organizations, and they influence workforce attitudes toward change. Stakeholders become familiar with their tools — including the quirks — and how they fit together to satisfy their needs. So naturally, proposed changes to those workflows are often met with skepticism and resistance. An important goal of Seamless DE is to smooth out those seams (defined in Section 2.3.4) so stakeholders can focus on productive work rather than developing workarounds that are coping mechanisms for deficient tooling.

The tool integration evaluation questions from [30] may be transformed into system requirements as in Table 2.3. These requirements serve to define the problem space (Figure 2.1) with respect to Seamless DE tool integration. One aspect commonly discussed in the context of DEE is interoperability, and integration patterns such as a custom shim (Figure 2.10) are often developed to transform data without having access to data schemas and definitions. The transformed requirement with respect to interoperability states that Seamless DE Tools shall define the data schema used according to a standard such that other tools may parse it directly and avoid rework and mistakes. Likewise, the Control Integration dimension leads to the requirement that Seamless DE Tools provide what is essentially a model-generated Interface Control Document for other tools to consume and use directly. This level of architecture openness and model-based interface documentation would help resolve many tool integration challenges.

**Table 2.3:** Transforming tool integration questions [30] into system requirements.

| Dimension | Aspect | Question | Transformed Requirement(s) |
|---|---|---|---|
| Presentation Integration | Appearance and behavior | To what extent do two tools use similar screen appearance and interaction behavior? | The Seamless DE Tools shall conform to the [Seamless DE HCI Standard]. |
| | Interaction paradigm | To what extent do two tools use similar metaphors and mental models? | Following HCI Customization by the Operator, the Seamless DES shall apply the Operator HCI Configuration uniformly. |
| Control Integration | Provision | To what extent are a tool's services used by other tools in the environment? | The Seamless DES shall provide common functionality as formally-verified System Libraries. |
| | Use | To what extent does a tool use the services provided by other tools in the environment? | The Seamless DE Tools shall reuse System Libraries for Tool-specific functions.<br>The Seamless DE Tools shall provide Tool functions conforming to the [Seamless DE Tool Interface Standard]. |
| Process Integration | Process step | How well do relevant tools combine to support the performance of a process step? | The Seamless DES shall provide the Workflow Definition Tool. |
| | Event | How well do relevant tools agree on the events required to support a process? | The Workflow Definition Tool shall define the Process Event using [Seamless DE Data Definition Standard]. |
| | Constraint | How well do relevant tools cooperate to enforce a constraint? | During Workflow Definition & Execution, the Workflow Definition Tool shall enforce the Process Constraint on the Workflow-Defined Tool. |
| Data Integration | Interoperability | How much work must be done for a tool to manipulate data produced by another? | The Seamless DE Tools shall define the applicable Data Schema conforming to the [Seamless DE Data Definition Standard]. |
| | Nonredundancy | How much data managed by a tool is duplicated in or can be derived from the data managed by the other? | The Seamless DES shall provide the ACID-compliant System Database. |
| | Data consistency | How well do two tools cooperate to maintain the semantic constraints on the data they manipulate? | The Seamless DES Database shall maintain semantic constraints of data where available. |
| | Data exchange | How much work must be done to make the nonpersistent data generated by one tool usable by the other? | The Seamless DE Tools shall write transactions to the Seamless DES Database for each change performed by the Operator. |
| | Synchronization | How well does a tool communicate changes it makes to the values of nonpersistent, common data? | The Seamless DES Database shall provide the Transaction Log applicable to the Seamless DE Tool. |

As SysML v2 requirement definitions, the transformed requirements in Table 2.3 may then be reused as requirement usages in various levels of the system architecture providing bidirectional traceability. SysML v2 requirements are modeled as constraints so that the system either satisfies the requirement or not. While the questions in Table 2.3 often ask "how well?", in a clean-slate architecture, the components can be designed to satisfy these measures as requirements. The tool integration questions may also be modeled directly as SysML v2 *concerns* that are similar to requirements but include one or more stakeholders and allow the relevant requirement(s) to *frame* the concern. Further SysML v2 meta-modeling could capture the dimensions and aspects in Table 2.3, giving them ontological definitions and relations consistent with systems engineering practice.

### 2.4.3 Seamless Integration Design Pattern

A seamless integration design pattern (see Figure 2.13) uses system-wide standard interfaces, data schemas, and object representations in a live DES to accomplish the integration of SoI models to form the ASoT. By drastically reducing the number of interfaces, tools are designed to speak the same language. With self-documenting interfaces, tools may query other tools to discover and perform the available functions. When the ASoT is queried by a tool, a view is returned including the unique identifiers — or rather — handles of live objects. In this integration design pattern, it is important to note that data is not copied and divorced from its source which requires later merging and reconciliation.

### 2.4.4 Seamless DE Architecture Tenets

Seamless DE is based on a set of insights or architecture tenets that will guide its detailed architecture definition. These tenets are derived from observations of extant computing systems and existing research in computer science, human-computer interaction, and cybersecurity. Figure 2.14 shows how these architecture tenets may be modeled as SysML v2 concerns with their own subjects, stakeholders, attributes, and constraints — to be framed by system requirements in a reference architecture model. In summary, the architecture tenets

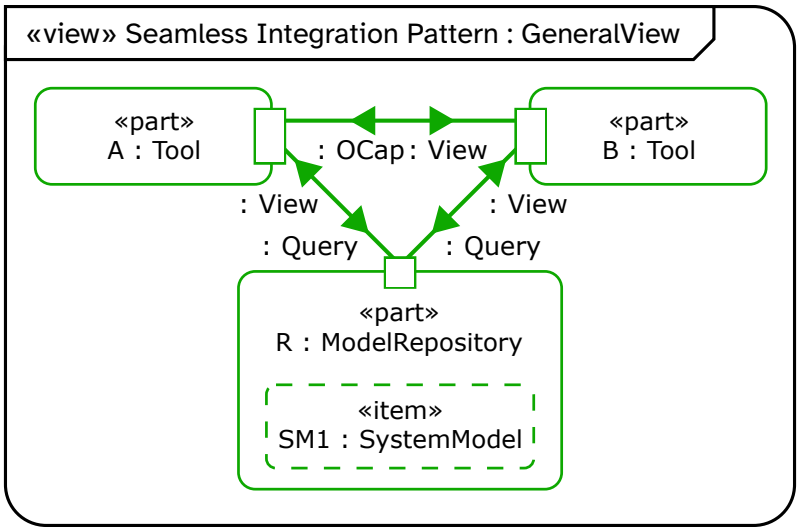



**Figure 2.13:** Seamless design pattern of DE tool integration (SysML v2 depiction).

that enable Seamless DE are: 1) Seamless Models, 2) Composable Data, 3) Live Objects, 4) Seamless Workflows, and 5) Clean-slate Cybersecurity.

By packaging and presenting these insights together, they may be better understood than if analyzed individually. In accordance with systems engineering principles, capturing system architecture information as model data that is then presented in a variety of stakeholder views is superior to doing one or the other alone. Again, further SysML v2 meta-modeling may better capture, connect, and present the Seamless DE architecture tenets — our intent is to demonstrate how modeling these subjects can aid stakeholder understanding that improves system development outcomes. Rather than existing as implicit knowledge harbored by system designers, the Seamless DE architecture tenets are explicit knowledge captured in the ASoT.

## 2.5 A Grand Challenge Driven by Needs

Grand challenges have historically been proposed to identify important research directions in a field and to help focus research and development efforts. [62] has clarified the criteria of what makes a research challenge a 'grand challenge' in particular. Table 2.4 summarizes how Seamless DE, as proposed in this chapter, fits those criteria. By proposing Seamless DE as a

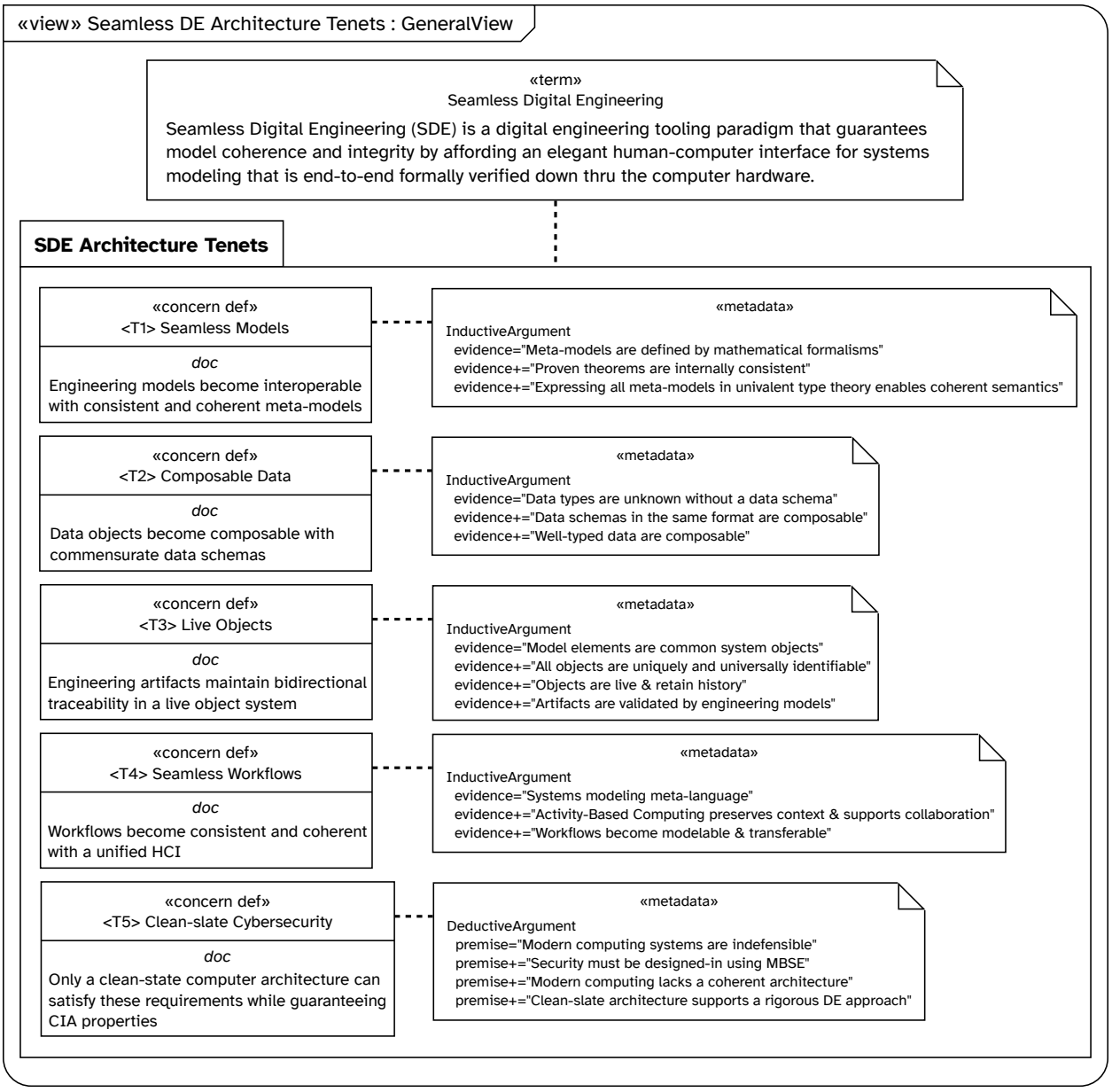



**Figure 2.14:** Seamless DE Architecture Tenets as packaged concern definitions (SysML v2 depiction).

DE research direction using this framework, it may be better understood in the context of past grand challenges.

A grand challenge inspires collaboration across multiple fields and challenges researchers to adopt a transdisciplinary approach. Transdisciplinary systems engineering [63] is especially well-suited for tackling the mess [64] of DE. Seamless DE draws on research in computer science, computer engineering, human-computer interaction, knowledge engineering, formal methods, and MBSE to envision a DES designed to meet stakeholder needs. Systems engineering researchers are encouraged to learn from these fields and apply that learning with systems praxis [65] to the Seamless DE grand challenge.

As a grand challenge, the work involved is beyond the capability of any one enterprise. The economics are challenging and would require investment from multiple organizations working in concert. The formal verification effort, in particular, has not been achieved at the proposed scale — although we may rest assured that a sufficiently specified component that is formally verified is 'finished,' unlike the software we use from day-to-day, allowing engineers to progress steadfastly. Another salve is that with clean-slate design, workarounds of legacy systems are unnecessary, and we may dream beyond such reverse salients as the central processing unit (CPU) and Portable Operating System Interface (POSIX) standard interfaces [66]. Emulating open-source software projects may not be enough to manage the complexity, and new approaches to open-source systems engineering may need to be tested. Lastly, thinking on a generational time scale may alleviate the apprehension of accomplishing such a megaproject.

## 2.6 Conclusions

Rather than coping with fragile, byzantine computing systems for implementing DE, we could have a DES by virtue of a computing system that is designed to satisfy user needs. Systems engineers are especially well-suited for tackling this design problem, but it will require a transdisciplinary systems engineering [63] and an international collaborative

**Table 2.4:** Evaluation against criteria for a grand challenge [62].

| Criterion | Rationale |
|---|---|
| *Fundamental* | Elegant and correct engineering tools are fundamental to development of complex systems. |
| *Astonishing* | An all-encompassing DE tooling system does not exist, but it is at least conceivable and astonishing if realized. |
| *Testable* | Systems engineering test & evaluation methods apply, and the prototype system shall be continuously verified and validated. |
| *Inspiring* | People want better tools because it makes their work easier. The scope is grand and inspiring. |
| *Understandable* | Varieties of computer-aided tools are widely in use, and they may be understandably synthesized. |
| *Useful* | The application of SE in this way shall inform computer science and related fields, as well as being eminently useful as a tool. |
| *Historical* | This challenge is related to early research in computing contracted by the US DoD, and ties in with former grand challenges in computing. |
| *International* | Everyone is welcome to contribute as everyone may potentially benefit from an open-source tool. |
| *Revolutionary* | Systems, software and computer engineering could experience a paradigm shift as the approaches augment each other within a cohesive tooling system. |
| *Research-directed* | Advanced research is drawn upon and brought into an elegant system architecture in a scope commercially infeasible. |
| *Challenging* | While isolated tools and methods are identifiable, their synthesis will present new challenges and open new inquiries for research. |
| *Feasible* | Recent advancements in MBSE, programming languages and formal verification make a Seamless DES development feasible. |
| *Incremental* | By using MBSE to specify system components and behaviors, and the program itself, progress may be made incrementally and in parallel efforts. |
| *Co-operative* | Researchers are already working in the relevant fields, and may wish to contribute in this new context. |
| *Competitive* | Point-design decisions are unclear in some cases, such as the best type system or virtual machine, and competitive entries may be evaluated against the architecture metrics. |
| *Effective* | Potentially it will change how we advance DE research. |
| *Risk-managed* | Even failures are valuable research outcomes. By using SE, we can reorient at every spiral. |

approach. While many would contest a clean-slate architecture approach as infeasible, there are interacting emergent effects of modern networked computing systems that cannot be resolved with patches or by adding more components: namely, cybersecurity, reliability, and HCI considerations. By incorporating decades of research in computing and adjacent fields, we can design in the required functionality at the system level and eliminate unnecessary complexity that triggers The Error Avalanche (Figure 2.12). A clean-slate design effort of this nature has been proposed in the past [9, 55], but it warrants renewed investment and research through the DE lens.

We contend that under the umbrella of "Seamless Digital Engineering Grand Challenge", the necessary resources can be assembled to design a DES that meets DE practitioners' needs. In this chapter, we defined Seamless DE as a paradigm (Section 2.4) to help guide future research, and we distinguished it from prior uses in the literature with the inclusion of the elegance QAt and end-to-end formal verification for guaranteeing ASoT coherence and integrity. Seamless integration requirements (Table 2.3) were derived from tool integration concerns in the literature to demonstrate how SE methods may address this problem. A seamless design pattern of DE tool integration (Figure 2.13) was depicted in SysML v2 to contrast this paradigm with existing tool integration patterns (Figs. 2.9, 2.10, and 2.11) that cope with tooling seams. Finally, Seamless DE architecture tenets (Figure 2.14) were derived to provide a concise set of concerns that guide system design.

## 2.7 Summary

This chapter addressed RQ1, restated as: *What is Seamless Digital Engineering and what distinguishes it from Digital Engineering?*

This work pulled together threads of prior work in model-based systems engineering, computer science, and software engineering. It highlighted how prior work approached related problems, and redefined the problem in the context of digital engineering while proposing a grand challenge in digital engineering research. Finally, it presented essential requirements,

the seamless integration design pattern, and architecture tenets of Seamless DE that will be reused later in Chapter 5.

Seamless Digital Engineering is defined above in Section 2.4 as:

> **Seamless Digital Engineering** is a digital engineering tooling paradigm that guarantees model coherence and integrity by affording an elegant human-computer interface for systems modeling that is end-to-end formally verified down through the computer hardware.

This chapter explains how digital engineering, as conceived today, cannot be elegant or seamless, due to its critical dependency on heterogeneous software and hardware components that must be integrated *ex post facto*. Critical issues of security, reliability, and correctness are highlighted as reasons for this operational limitation, which cannot be adequately addressed through fixes and/or accrual of functionality. Due to the high risk and high cost of such digital engineering systems, Seamless DE is presented necessarily as a clean-slate approach to this *mess*.

By treating DE as a problem and opportunity using a systems engineering approach, we avoid common pitfalls of system development projects that lack systems engineering rigor. These pitfalls include the failure to enumerate stakeholder concerns and needs; failure to define, develop, and verify requirements; and insufficient verification and validation of the realized system against stakeholder needs, to name a few. The problem reframing presented in this chapter provides an alternative formulation of an essential DE topic that goes unaddressed in the literature.

# Chapter 3

# Ontological Definition of Seamless Digital Engineering[4]

> All problems in computer science can be solved by another level of indirection, except for the problem of too many layers of indirection.
>
> David J. Wheeler [68]

This chapter addresses RQ2, which is restated as follows: *How do we precisely define Seamless Digital Engineering and its related concepts?*

To address this question, the chapter uses ontology engineering methods to disambiguate the Seamless Digital Engineering concept as defined in Chapter 2. The chapter presents an essential fragment of the developed Seamless Digital Engineering Ontology based on multiple international standards and other canonical sources as shown in Table 3.2. By basing the ontology on the standard Basic Formal Ontology (BFO) [69] and the Common Core Ontologies (CCO) [70], and by collaborating with the INCOSE Digital Engineering Information eXchange (DEIX) Ontology Working Group (WG), significant progress was made in defining concepts in DE. Further details on the ontological definitions of these general DE concepts can be found in [71].

## 3.1 Introduction

Since the introduction of digital engineering (DE) as a well-defined concept in 2018, enterprises and working groups have been working out how to transform existing model-based engineering (MBE) and Digital Twin / Digital Thread strategies and architectures to

[4]The contents of this chapter are based on the paper "Ontological Definition of Seamless Digital Engineering Based on ISO/IEC 25000-Series SQuaRE Product Quality Model" published in INCOSE International Symposium 2025 [67].

encompass the whole system lifecycle as envisioned by the DE strategy [22]. Many challenges exist, including tooling capability, integration capability, process digitalization, data and metadata management, and adaptation of existing digital transformation (DT) programs to accommodate the DE vision. Tool vendors and standards committees are actively working to fill the void of unmet needs when it comes to an organization implementing their own DE strategy. We are now in an exciting era of development for MBSE and DE that is supported by professional societies, standards bodies, tool vendors, and engineering organizations, so we may soon be practicing DE and realizing its many promised benefits.

As engineers, we rely on precise, prescriptive language to carry out our work, and DE is no exception. Position papers and reference models on Digital Twin / Digital Thread have been published that further describe these concepts in the context of DE [36, 37, 72], along with explication of the concept of Authoritative Source of Truth (ASoT) [73]. While these efforts are important steps in understanding and implementing DE, we now have multiple definitions of DE concepts that may overlap or conflict. Examples include 'Digital Engineering Ecosystem' and 'Digital Engineering Environment', which may have been used interchangeably, but only DE Ecosystem is defined by the Defense Acquisition University (DAU) Glossary [24], while the INCOSE Systems Engineering Handbook discusses the DE Ecosystem concept as dependent on scope and context, without mentioning DE Environment [74]. To avoid confusion during DE transformation efforts, and to avoid imprecision of data in production environments — to realize ASoTs — ontologies have been identified as a critical component of MBE and DE strategies [75].

To address the integration and systems security challenges of DE owing to its reliance on insolvent [76, 77] digital infrastructure, Seamless DE was proposed by the authors as a grand challenge in DE research [13]. Following the clean-slate approach first proposed by DARPA [9], Seamless DE aims to provide an alternative path to the brittle integration of DE Environments, instead focusing on end-to-end formal verification, metamodel and metalanguage coherence, and unified human-computer interface (HCI) to meet the needs

of DE practitioners. This clean-slate approach to DE has the potential to stimulate DE research based on first principles, while taking advantage of recent advancements in MBSE and computer science to deliver real synergy of capabilities unattainable by traditional digital integration methods. To clarify these ideas in the context of DE, an ontological development approach is taken and the mature ISO/IEC/IEEE 25000-series Systems and software Quality Requirements and Evaluation (SQuaRE) quality and related standards are used to elucidate the meaning and potential of Seamless DE.

The contributions of this chapter include: 1) ontological definition of 'seamless' and Seamless DE in the context of systems engineering standards and based on established top- and mid-level ontologies and the ongoing work of the INCOSE Digital Engineering Information eXchange (DEIX) Ontology Working Group (WG), 2) assimilation and presentation of ISO/IEC/IEEE standards information including the ISO/IEC 25000-series on quality into an ontology used to define these seamless DE concepts, and 3) an open-source, machine-readable Semantic Web standards-based ontology for DE and MBSE applications.

Section 3.2 provides additional background on ontology engineering in the context of DE. In Section 3.3, we outline the methodology used to develop the present ontology. Section 3.4 presents the primary results, including ontological definitions for 'Seamless Quality Claim', 'Seamless Integration', 'Seamless Interaction Capability', 'Seamless Quality-in-Use', 'Seamless Digital Engineering Paradigm', and 'Seamless Digital Engineering System'[5]. Section 3.5 concludes the paper and outlines future work.

## 3.2 Background

### 3.2.1 Ontology Engineering

Ontologies are formal representations of knowledge domains, wherein concepts are defined through their relations and thereby disambiguated. Ontologies have the potential to link

[5]The concept label 'Seamless Digital Engineering System' has been changed from the original 'Seamless Digital Environment' to match the terminology introduced in Chapter 2.4 without modifying its ontological definition.

together adjacent domains of knowledge to facilitate the interchange of structured information among domain experts. While many ontology languages exist, the mathematical decidability of an ontology is an important contributor to its usefulness. Based on the axioms provided in a given ontology, inferences of knowledge may be made with a suitable reasoner, and structured queries may be made that give deterministic results. Overall, ontologies are suitable for providing the accuracy and reliability that engineers need when navigating complex, emerging knowledge domains [78].

The Protégé open-source software tool developed by Stanford University [79] has proven itself an indispensable tool in ontology engineering work. Protégé is based on the Web Ontology Language (OWL) version 2 standardized by the World Wide Web Consortium (W3C) [80] as a Semantic Web technology. By using OWL 2, Protégé interoperates with the digital ecosystem of Semantic Web technologies, including Resource Description Framework (RDF), Extensible Markup Language (XML), and SPARQL Protocol and RDF Query Language (SPARQL). These open standards enable organizations to use existing software libraries and to write their own tools to handle Semantic Web data for their DE Environments. Similar to how tool vendors implement the XML-based standards published by the Object Management Group (OMG), Protégé provides a familiar graphical user interface (GUI) for authoring an ontology and importing other ontologies from the Semantic Web. One need only to open an ontology file and begin navigating its entities and relations to familiarize oneself with the tool's capabilities — ontology imports are handled automatically by the tool to provide a cohesive view of the 'imports closure'.

### 3.2.2 Description Logic

The mathematical formalism offered by OWL 2 and Protégé is known as a Description Logic (DL), which is a subset of first-order logic (FOL) using propositional logic that ensures its decidability [81, 82]. The particular DL used by OWL DL subset is characterized by $\mathcal{SHOIN}(\mathbf{D})$, summarized in Table 3.1.

**Table 3.1:** OWL DL summary of symbols of logic and their descriptions

| Symbol | Description |
|---|---|
| $\mathcal{AL}$ | (Attributive Language) Inclusion, equivalence, intersection, and complex definition of classes |
| $\mathcal{ALC}$ | (with Complement) Adds to $\mathcal{AL}$ the empty, complement, union classes [83] |
| $\mathcal{S}$ | Adds the transitivity of relations to $\mathcal{ALC}$ |
| $\mathcal{H}$ | Inclusion and equivalence between relations |
| $\mathcal{O}$ | (One of) Classes created with list of all and only the individuals contained |
| $\mathcal{I}$ | (Reverse) Inverse property |
| $\mathcal{N}$ | (Number) Cardinality restriction |
| $\mathcal{D}_n$ | (Countable domain) Definition of domains (data types) |

In short, OWL 2 ontologies consist of classes, object properties, annotation properties, individuals, and data properties, and are checked by OWL reasoners which implement OWL DL or other profiles described in [80]. Individuals are instances of classes and inherit their subclass and equivalence axioms, enabling object property assertions with other individuals. Individuals provide the capability to ground the ontology in familiar real-world objects, while additionally checking the consistency of the ontology. One example of an individual in our ontology would be OMG, which has type class 'Standards Organization' and 'is carrier of' Systems Modeling Language (SysML), an individual of type class 'Architecture Description Language'. OWL DL can be represented in the Manchester syntax [84], which provides a human- and machine-readable syntax for authoring axioms.

### 3.2.3 Top-Level and Mid-Level Ontologies

While the development of an ontology in isolation provides the benefits afforded by the DL and the Semantic Web digital tool ecosystem, those benefits are enhanced by importing existing ontologies that model higher-level concepts and relations. The Basic Formal Ontology (BFO) [69,85] provides the top-level ontology of our domain ontology, including 36 classes based on the principles of ontological realism, fallibilism, and adequatism. The BFO class hierarchy starts with 'entity' and is subdivided into 'continuant' and 'occurrent', i.e., entities that have a continuous existence and entities which are processes and other temporally-bounded entities.

Continuants are divided into 'generically dependent continuants', 'independent continuants', and 'specifically dependent continuants'. These types of continuants and occurrents form the basis of the realistic classes of over 350 ontologies. It is important to note that although BFO is based on the principle of realism (entities exist, have existed, or will exist), it does not preclude certain types of modeling in digital engineering of entities that do not exist, such as in simulations, because the digital objects nevertheless exist with a material basis of the constituent computer components, and the target objects are also supported by generically dependent continuants such as Information Content Entities (ICEs).

The Common Core Ontologies (CCO) [70] is a mid-level ontology suite based on BFO. It contributes eleven ontologies that extend the usability of BFO while remaining domain-agnostic. These include: Extended Relations, Units of Measure, Time, Quality, Information Entity, Geospatial, Facility, Event, Currency Unit, Artifact, and Agent. Among these, the Extended Relation, Quality, Information Entity, Artifact, and Agent ontologies have been immensely useful in defining the Seamless Digital Engineering domain ontologies.

## 3.3 Methodology

The present ontology was developed following the methodology presented in Ref. [86]. Starting with our DE knowledge domain, BFO and CCO top- and mid-level ontologies, we begin by identifying concepts and defining classes within the existing class hierarchy. We then move on to defining relations or object properties, and instances of the classes to further validate the ontology. The steps of the methodology described in [86], adapted to OWL terminology, are as follows:

1. Determine the domain and scope of the ontology
2. Consider reusing existing ontologies
3. Enumerate important terms in the ontology
4. Define the classes and the class hierarchy
5. Define the object properties

6. Define the domains/ranges of object properties
7. Create instances of classes

The process becomes iterative as the reasoner validates each step, and missing concepts are identified. We are using the HermiT 1.4 reasoner [87] to check the consistency of the ontology. When a change is made to an 'equivalent to' or 'subclass of' axiom, the reasoner is synchronized with the ontology and checks its consistency. This way, erroneous definitions are caught immediately and flagged as `owl:Nothing` inferences, which may then be explained using the reasoner's capabilities.

To be a valid axiom accepted by the Protégé software and its reasoners, every class and object property used in an axiom must already have a valid definition. Ontology development proceeds iteratively in this way until all the associated concepts reach satisfactory and stable definitions. Refactoring may then proceed with automated validation, checking each change to maintain the consistency of the ontology. This process of adding new classes in the BFO and CCO class hierarchy requires careful consideration of how a new class follows the conventions and axioms. Without these top- and mid-level hierarchies, the domain ontology can easily lack rigor, which adversely affects its reusability and reproducibility.

Domains and ranges of object properties are especially important in this process, as they restrict the possible classes and provide the consistency checking we need to develop the ontology. At this stage of development, the addition of new object properties (Steps 5-6) was avoided to ensure the BFO and CCO were fully used. After careful consideration of the class hierarchy and use of existing object properties, more precise relations may be identified and developed to clarify the meaning of some concepts. However, complex hierarchies of object properties should be avoided to simplify reasoning.

To develop the concepts, we referenced international standards, INCOSE technical products, and other canonical sources, but did not reuse any existing DE domain ontology. By referencing international standards in the systems engineering and related domains, we were able to confidently build from recognized meanings of common concepts. However, following

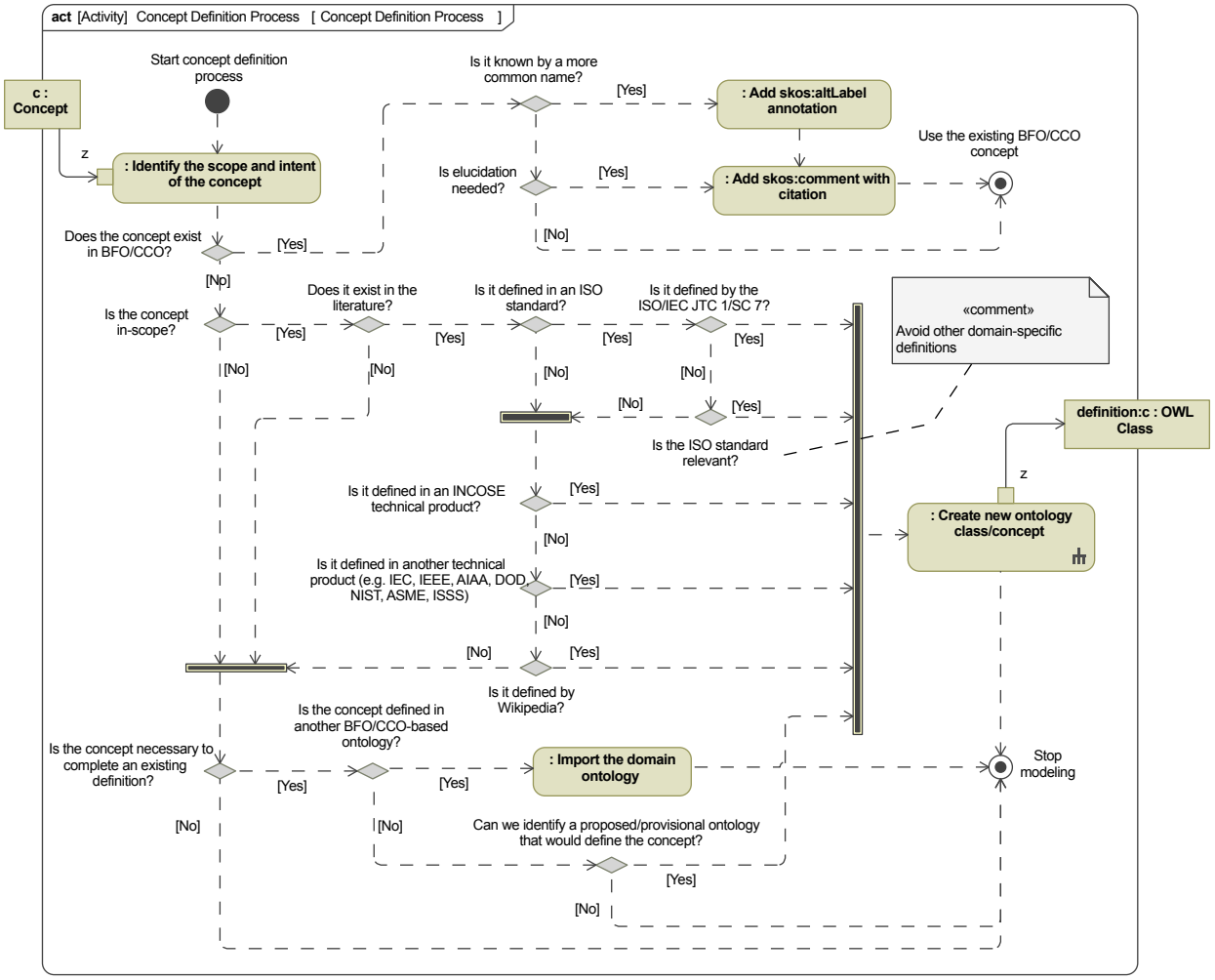



**Figure 3.1:** Ontology concept definition process using standard sources.

the process in Fig. 3.1 and the practice demonstrated by CCO, when a concept belongs to a more general domain and is not defined by ISO, it often suffices to use Wikipedia or an English dictionary as the source. This concept definition process prioritizes internationally-recognized terminology while leaving room for emerging concepts undergoing consensus definition processes, such as those in DE.

## 3.4 Results

The Seamless Digital Engineering Ontology [88] consists of over 500 classes and over 150 'equivalent to' axioms, while adhering to a strict zero new object properties defined to ensure

logical consistency with BFO and CCO at this stage of development. Definitions related to 'seamless' are based on the SQuaRE product quality (Figure 3.2) and quality-in-use models (Figure 3.3), which are modeled as realizable entities (subclass of specifically dependent continuant) in the ontology. While the BFO provides the 'quality' class, it differs from 'realizable entity' in that qualities need no further processes to be realized. The quality characteristics according to the standard are defined as capabilities and therefore modeled as capabilities with relations to Information Quality Entities according to CCO.

Concepts from the following standards and other canonical sources (Table 3.2) were used in the development of the Seamless Digital Engineering ontology, reusing BFO and CCO concepts wherever possible to avoid conflicts and duplication. The "Count" column indicates the number of concepts whose definition derives from that source. Notably, the highest count is for concepts defined in [91]. Although many attempts at developing a systems engineering domain ontology are documented in the literature [92, 93], a recognized standard does not yet exist. Lacking a publicly available systems engineering domain ontology developed in OWL using BFO and CCO, these ontology development efforts were reproduced for this research. The next largest share of definition sources are the ISO/IEC 25000-series SQuaRE product quality standards [89, 90, 94, 95], owing to the number of product quality and quality-in-use characteristics defined therein.

The diverse domains and subject areas in Table 3.2 were drawn from to support the definition of the concepts essential to DE in general, and to Seamless DE in particular. Ideally, these domains would be represented in importable CCO-based ontologies of their own, supporting separation of concerns and reusability. Systems engineering, for example, would be its own domain ontology imported by the DE ontology and would have sub-domain ontologies such as for requirements engineering, systems architecture, and quality engineering. Figure 3.4 depicts a non-exhaustive proposed import hierarchy, and shows that DE relies on concepts in trustworthy secure systems, model-based systems and software engineering (MBSSE), and digital enterprise, including the extensive topics described by the Data Management

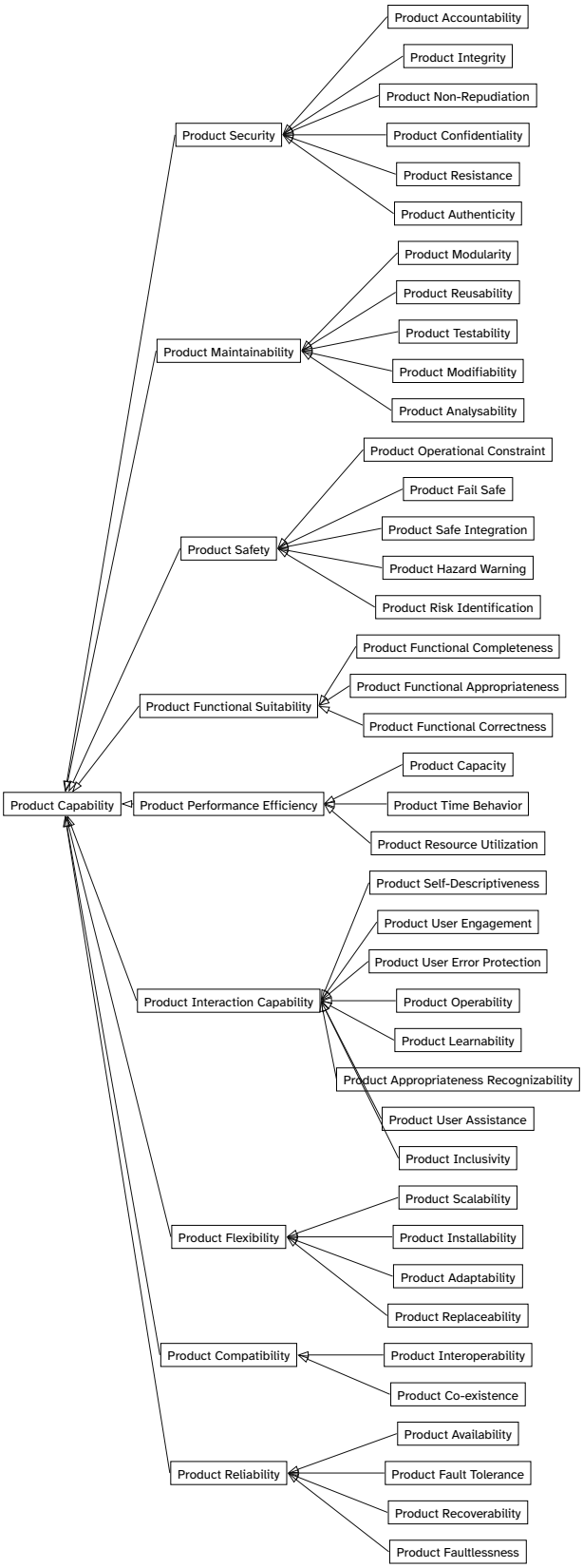



**Figure 3.2:** SQuaRE quality characteristics from Ref. [89] defined as Product Capabilities

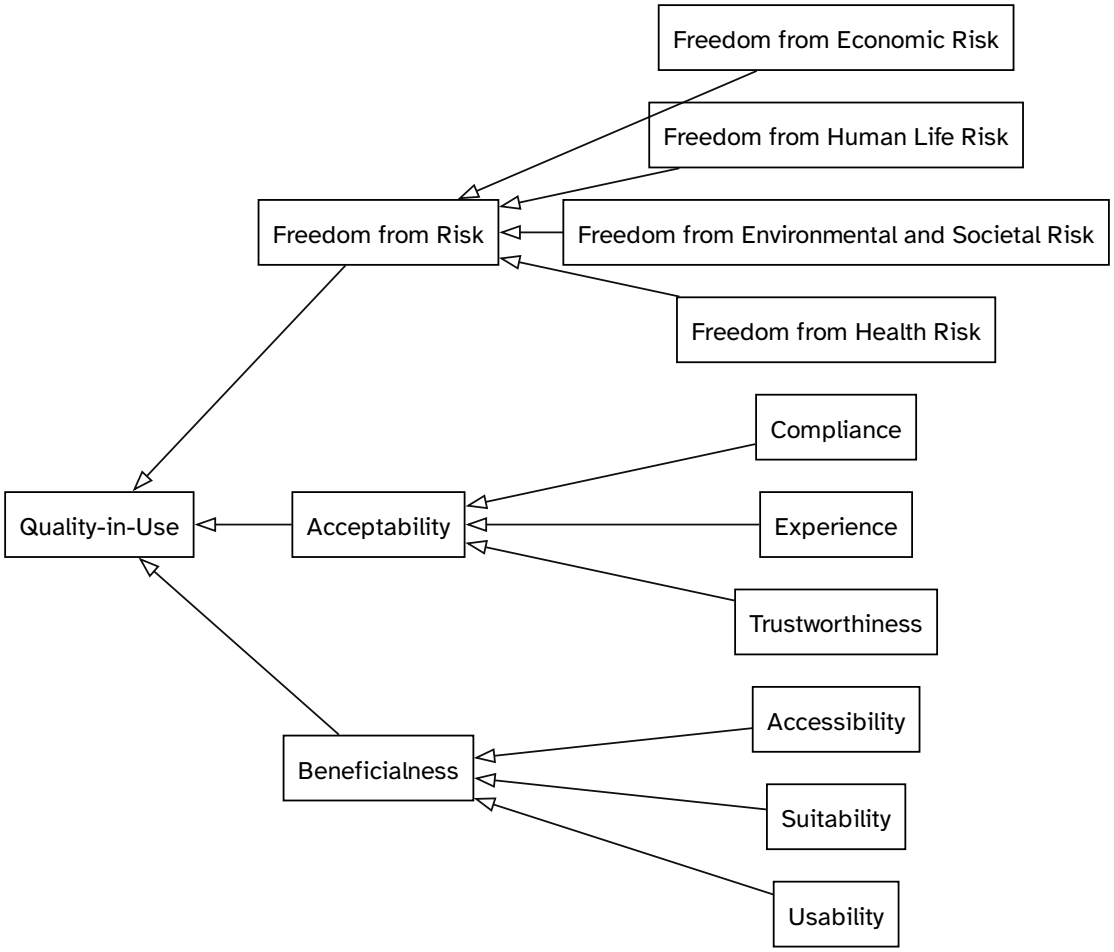



**Figure 3.3:** SQuaRE quality-in-use characteristics as defined in Ref. [90]

Body of Knowledge (DMBoK) [120]. Such breadth and depth enable precise knowledgebase queries, Authoritative Source of Truth (ASoT) consistency evaluators, and support system architecture modeling activities, including of the Seamless digital engineering environment (DEE) system-of-interest (SoI).

The following listings provide the description logic axioms that relate the Seamless DE concepts, including those from the SQuaRE product quality and quality-in-use models [89,90]. A summary relational diagram is shown in Figure 3.5 that highlights how the salient 'seamless'

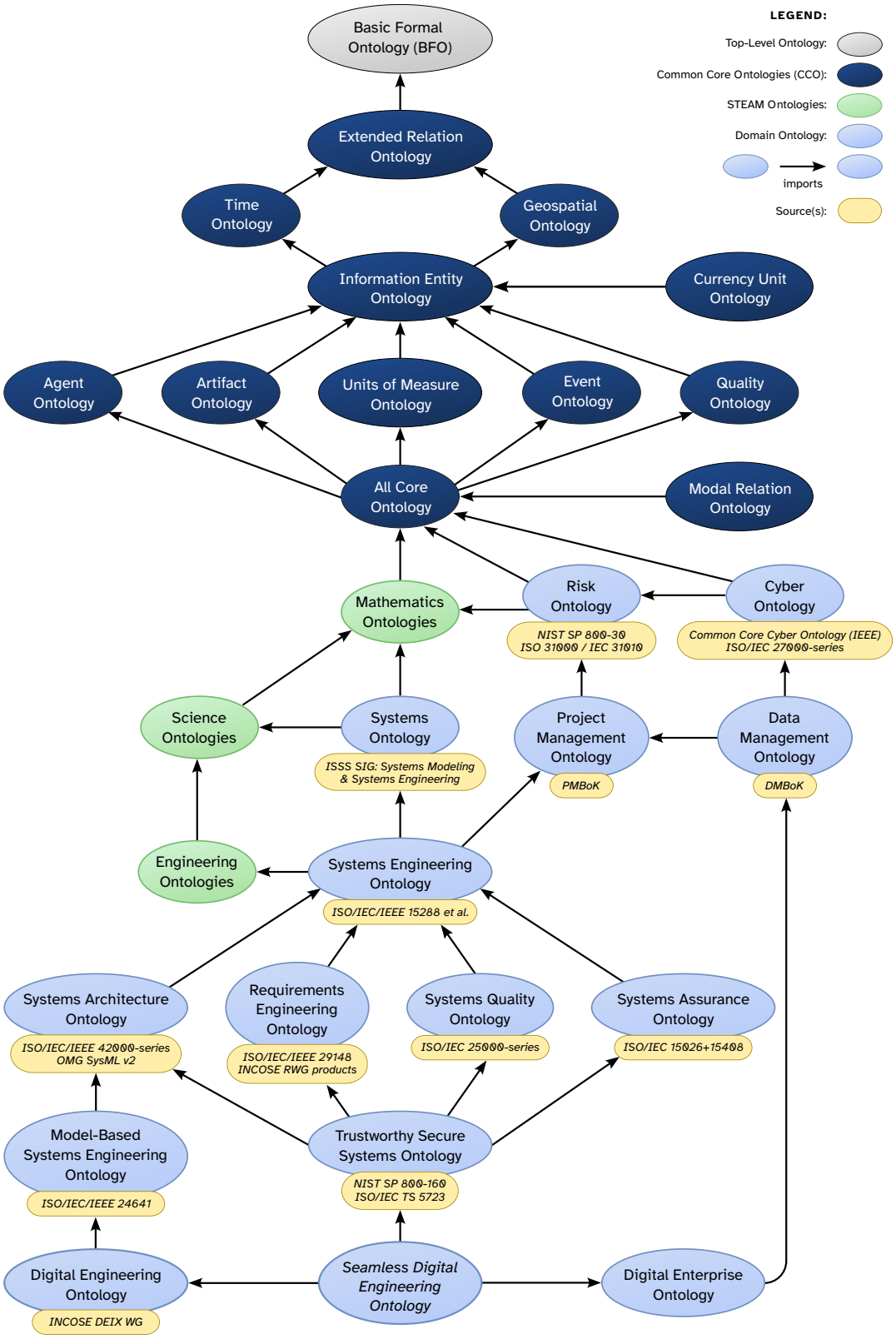



**Figure 3.4:** Proposed import hierarchy of DE-related domain ontologies, based on BFO and CCO.

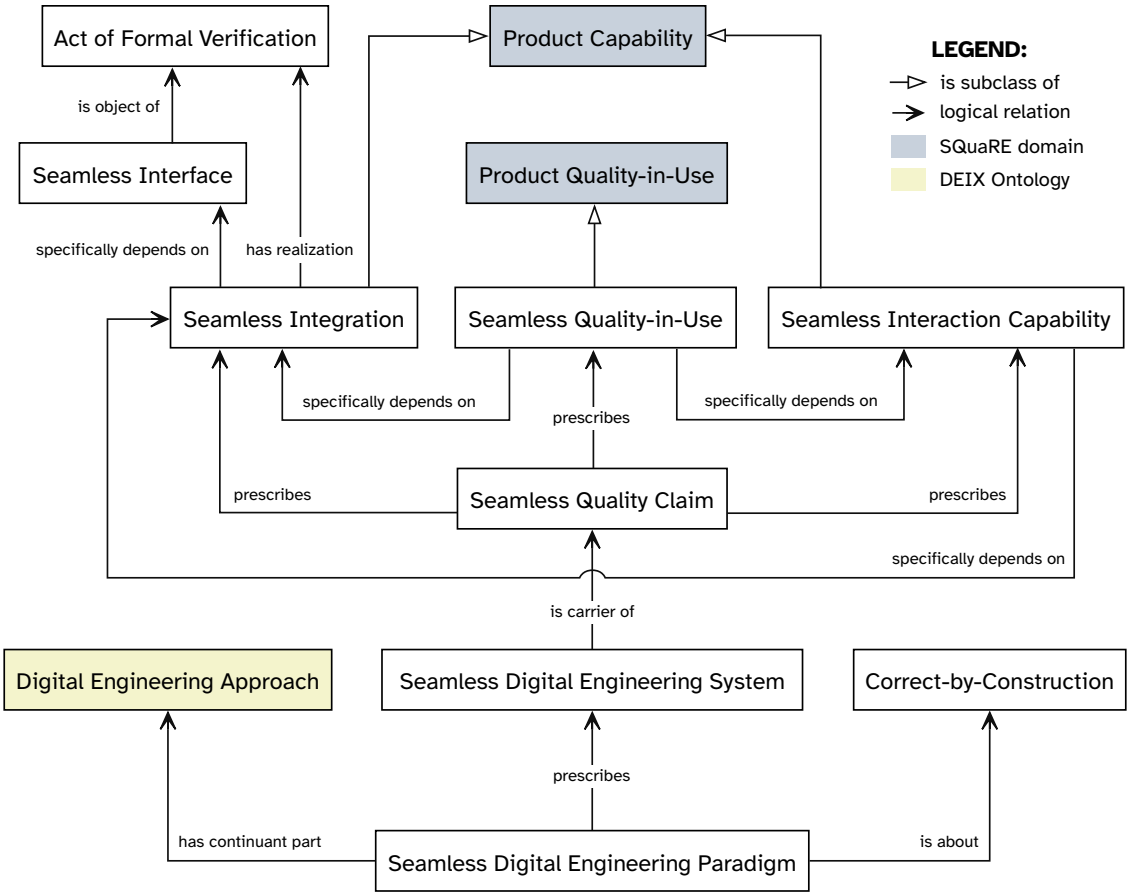



**Figure 3.5:** Summary of ontological relations of Seamless Digital Engineering concepts.

concepts relate to one another. The SQuaRE concepts are at the top-center, and not all concepts and relations are shown. Refer to the listings below for more details.

Seamless Digital Engineering was identified as a DE tooling paradigm by [13], where the theory developed from related work in 'seamless' model-driven systems engineering [31–33]. Further development of these concepts using an ontological framework and reasoner helped clarify their meaning and relations. Concepts such as 'seamless' and 'paradigm' may be interpreted in different ways depending on the person, but assigning their meaning in an ontology disambiguates them through definition-by-relations. Paradigm is defined as a Directive ICE with continuant parts of Approach, Method, and Principle, which are all subclasses of Directive ICE, i.e., information content that prescribes some entity.

Instead of defining 'Seamless Digital Engineering' as a distinct concept, it needs to be decomposed into related concepts that fit appropriately as subclasses in the class hierarchy. Therefore, 'Seamless Digital Engineering Paradigm' (Listing 3.1) is defined as a subclass of Paradigm, although it may later be revised to be a subclass of Engineering Paradigm or similar when sibling classes are identified and defined.

**Listing 3.1: 'Seamless Digital Engineering Paradigm'** is a subclass of **Paradigm** equivalent to:

```
    Paradigm
and ('has continuant part' some 'Digital Engineering Approach')
and ('is about' some Correct-by-Construction)
and (prescribes some 'Seamless Digital Engineering System')
```

Digital Engineering Approach is simply an Engineering Approach that prescribes some Act of Digital Engineering. Act of Digital Engineering is an Act of Engineering that has a Digital Artifact participant and an Authoritative Source of Truth as its object. Detailed definitions of these DE concepts are omitted here, and may be found in publications by the INCOSE DEIX Ontology Working Group. The essential concepts of the Seamless Digital Engineering Paradigm concept here are Correct-by-Construction (Listing 3.2) described by [121] and the Seamless Digital Engineering System (Listing 3.3).

**Listing 3.2: Correct-by-Construction** is a subclass of **'Assurance Goal'** equivalent to:

```
    'Assurance Goal'
and ('is concretized by' some 'Integration Process')
and ('is concretized by' some 'Loss of Error')
and (prescribes some 'High-Integrity Level')
and (prescribes some 'Process Outcome')
```

Correct-by-Construction is modeled as an Assurance Goal, a descendant subclass of Directive ICE, suitable for referential use in system architecture models. The natural language definition (`skos:definition`) is derived from [122]: "An Assurance Goal to eliminate Integration Process-time errors emerging from undesirable cross-layer interactions". As a natural language elucidation, its component parts correspond to the ontology axiom above

(Listing 3.2). High-Integrity Level is defined according to ISO/IEC/IEEE 15026 [96] and is modeled as an Ordinal Measurement ICE from the CCO.

Seamless Digital Engineering System (Listing 3.3) is an Engineered System, indicating it is designed, built, and validated to satisfy enumerated Stakeholder Needs. This acquisition approach contrasts with those commonly used for enterprise IT infrastructure supporting Digital Engineering Environments today, characterized by a series of "build-or-buy" decisions that typically lack the basis of an enterprise system architecture model. Rather than relying on third-party assessments that constitute a Trusted Computing Base, a Seamless Digital Engineering Environment relies on the independently-verifiable Seamless Quality Claim (Listing 3.4) of a Trustworthy Computing Base whose Trustworthiness is supported by a Complete Assurance Case Report defined by ISO/IEC/IEEE 15026 [96] and provided as output of an Act of Assurance Evaluation defined by ISO/IEC 15408 [106].

**Listing 3.3: 'Seamless Digital Engineering System'** is a subclass of **'Engineered System'** equivalent to:

```
    'Digital Engineering System'
and 'Engineered System'
and ('is carrier of' some ('High-Integrity Level Claim' and 'Seamless Quality
   Claim'))
and ('has member part' some 'Trustworthy Computing Base')
```

Seamless Quality Claim (Listing 3.4) is based on the concept of Claim defined by ISO/IEC/IEEE 15026 [96], which prescribes some Attribute, Condition, and Uncertainty, and is a descendant subclass of Constraint for use in a System Architecture Model. Seamless Quality Claim is the top-level Quality Claim for a Seamless Digital Engineering Environment, and links together the seamless composite Quality and Quality-in-Use concepts.

**Listing 3.4: 'Seamless Quality Claim'** is a subclass of **'Quality Claim'** equivalent to:

```
    'Quality Claim'
and (prescribes some 'Seamless Integration')
and (prescribes some 'Seamless Interaction Capability')
and (prescribes some 'Seamless Quality-in-Use')
```

Seamless Quality Claim is split among related composite qualities or Key Quality Attributes, and these results show how we disambiguate the term 'seamless' using the SQuaRE quality [89] and quality-in-use models [90]. Seamless Integration (Listing 3.5) is 'described by' a Quality Characteristic and modeled as a Product Capability, a realizable entity, which relates Quality Sub-characteristics from SQuaRE to a correct Integration Process. Seamless Integration is realized by an Act of Formal Verification and specifically depends on Seamless Interface(s) (Listing 3.6), defined as "An Interface prescribed by a System Architecture Model which is the object of an Act of Formal Verification that produces a Proof Certificate of its Product Functional Correctness" — a SQuaRE Quality Sub-characteristic of Production Functional Suitability.

**Listing 3.5: 'Seamless Integration'** is a subclass of **'Product Capability'** equivalent to:

```
    'Product Capability'
and ('has realization' some 'Act of Formal Verification')
and ('has continuant part' some
        ('Product Analysability'
     and 'Product Faultlessness'
     and 'Product Functional Correctness'
     and 'Product Integrity'
     and 'Product Safe Integration'))
and ('specifically depends on' some 'Seamless Interface')
```

**Listing 3.6: 'Seamless Interface'** is a subclass of **'Interface'**, a subclass of 'Information Bearing Artifact', and is equivalent to:

```
    Interface
and ('has continuant part' some 'Proof Certificate')
and ('prescribed by' some 'System Architecture Model')
and ('is object of' some 'Act of Formal Verification')
```

Seamless Interaction Capability (Listing 3.7) relates to Product Interaction Capability with additional Product Functional Suitability sub-characteristics, and specifically depends on 'Seamless Integration'. This concept captures the intuitive notion when a human-computer

interface (HCI) is seamless from the point-of-view of the Operator, and complements 'Seamless Integration' which applies to Interfaces throughout the Seamless Digital Engineering Environment that the Operator may not interact with directly. Seamless Interaction Capability specifically depends on successful Seamless Integration.

**Listing 3.7: 'Seamless Interaction Capability'** is a subclass of **'Product Capability'** equivalent to:

```
    'Product Interaction Capability'
and ('has continuant part' some
        ('Product Compatibility'
     and 'Product Functional Appropriateness'
     and 'Product Functional Completeness'))
and ('specifically depends on' some 'Seamless Integration')
```

Seamless Quality-in-Use (Listing 3.8) cannot be formally proven as it relies on the Operator's experience, but as with any Product Quality-in-Use, it should be traced to Stakeholder Needs in an appropriate Authoritative Source of Truth. When products claim to provide a 'seamless experience', it is this Seamless Quality-in-Use comprising the ISO 25019 [90] Qualities-in-Use, Experience and Trustworthiness (subclasses of Acceptability), and Suitability and Usability (subclasses of Beneficialness). Trustworthiness (Listing 3.9) is further decomposed to relate to the evaluation assurance concepts introduced above, and is therefore evidence-based.

**Listing 3.8: 'Seamless Quality-in-Use'** is a subclass of **'Quality-in-Use'** equivalent to:

```
    Quality-in-Use
and ('has continuant part' some
        (Experience
     and Suitability
     and Trustworthiness
     and Usability))
and ('specifically depends on' some
        ('Seamless Integration'
     and 'Seamless Interaction Capability'))
```

**Listing 3.9: Trustworthiness** is a subclass of **'Acceptability'** equivalent to:

```
    Acceptability
and ('specifically depends on' some Bootstrappability)
and ('specifically depends on' some
        ('Complete Assurance Case Report'
    and ('is carrier of' some 'Trustworthiness Quality Claim')))
```

Concepts from ISO/IEC/IEEE 15026 [96–98] and ISO/IEC 15408 ("Common Criteria") [106, 107] were carefully defined within the ontological framework. These concepts are essentially divided between Information Content Entities and their Information Bearing Artifact/Entity counterparts, supported by risk-related concepts defined as subclasses of Change, Stasis, Effect, and disposition. The ontology may be used to develop Complete Assurance Case Reports using Information Parts such as Evidence Items that conform to the standard. More complete systems engineering domain ontology development, including the many inputs/outputs/processes from ISO/IEC/IEEE 15288 [91] will further enhance the usefulness of the ontology.

## 3.5 Conclusions

We have presented concepts from the open-source Seamless Digital Engineering Ontology [88] which includes over 500 classes and over 150 axioms based on 30 existing international standards and INCOSE technical products, in an attempt to align with the latest knowledge of the field. This chapter presented concepts that helped elucidate Seamless Digital Engineering, focusing on aspects of trustworthy systems, high assurance and high integrity, and product quality of the Digital Engineering System as an Engineered System. The Trustworthiness Quality-in-Use Characteristic was selected for its importance in Seamless Quality-in-Use and its relation to existing standards in the systems engineering domain that define concepts in evaluation assurance. 'Seamless' has been an appealing word to use when describing products, but it had not been given a precise meaning that would assist in MBSE-based development of clean-slate engineered systems. By following the SQuaRE model, we prepare

the Seamless DE grand challenge for further development using Quality Need Expressions and Quality Requirement Expressions that shall be incorporated into the Seamless DE Reference Architecture (Chapter 5).

Harmonization of the concepts remains a difficult area, as terms in the literature may not be used consistently, or more importantly, do not match the base ontological definitions provided by BFO and CCO. Salient examples include different types of models which are often defined as "representations", but in CCO, a Representational ICE is distinct from an Artifact Model, which is a kind of Directive ICE. In these cases, the intent of the canonical sources was interpreted within the CCO framework, and the natural language definition was adjusted to fit the ontological axioms. Another challenge was disambiguation, including overloaded terms such as Authoritative Source of Truth, which required months of deliberations by the first author and INCOSE DEIX Ontology WG. We found that splitting concepts up according to the BFO and CCO framework was useful in clarifying their meaning and making further use of the concepts in relations. One such example is Traceability defined by ISO/IEC/IEEE 15288 [91] which has been provisionally disambiguated into four related concepts: Traceability Measurement ICE, Traceability Relation, Traceability Observation Artifact Function, and Act of Establishing Traceability. The concept of System is defined as an object aggregate, but its subtler meanings and uses may require many revisions to incorporate it usefully in the CCO framework and to be useful to a variety of fields. Such challenging ontology development requires repeated discussions among subject matter experts, but the effort is worth it to produce standardized machine- and human-readable artifacts that have been machine-verified for logical consistency. We expect this harmonization work to continue as stakeholders in our field recognize the importance and power of ontologies in solving DE challenges.

## 3.6 Summary

This chapter addressed RQ2, restated as: *How do we precisely define Seamless Digital Engineering and its related concepts?*

Seamless Digital Engineering was disambiguated into the following concepts using ontological engineering and freely-available glossaries of international standards and other canonical sources (Table 3.2): 'Seamless Digital Engineering Paradigm' (Listing 3.1), Correct-by-Construction (Listing 3.2), 'Seamless Digital Engineering System' (Listing 3.3), 'Seamless Quality Claim' (Listing 3.4), 'Seamless Integration' (Listing 3.5), 'Seamless Interface' (Listing 3.6), 'Seamless Interaction Capability' (Listing 3.7), 'Seamless Quality-in-Use' (Listing 3.8), and Trustworthiness (Listing 3.9). Many other ontological definitions were omitted from the publication due to space and scope limitations, but are present in the freely-available Seamless Digital Engineering Ontology [88] including Bootstrappability, Information Appliance, Act of Formal Verification and Proof Certificate, and Convivial and Elegant Key Quality Attributes.

The ISO/IEC 25000-series standards on the SQuaRE product quality model [89,90,94,95] provided the critical basis from which to complete this work, as 'seamless' was identified as a Key Quality Attribute which incorporated multiple aspects of product quality. The SQuaRE distinction of quality and quality-in-use was instrumental in clarifying 'seamless' in the DE context, and the ontology class hierarchy provided the necessary logical conventions to disambiguate the 'seamless' concepts. The standardized product quality characteristics in SQuaRE provided a taxonomy of closely related but distinct concepts that together defined the higher-level Key Quality Attributes.

The 'seamless' qualifier was then interpreted as an assurance claim according to ISO/IEC/IEEE 15026 [96–98] which provided the ontological basis for defining how the Seamless Quality Claim may be evaluated. The SQuaRE 'Trustworthiness' quality-in-use characteristic is defined by relations using assurance concepts from ISO/IEC/IEEE 15026, further clarifying the relationship of trustworthiness and seamless. Ontologically-defined concepts from ISO/IEC 15408 ("Common Criteria") [106,107] supported the definition of Correct-by-Construction, a concise term from the literature that accurately captures the high-integrity quality described in the Seamless DE definition (Section 2.4).

**Table 3.2:** Sources used to define ontology classes/concepts (see Bibliography).

| Standard Identifier | Domain | Subject Area | Count |
|---|---|---|---|
| ISO/IEC/IEEE 15288 | Systems and software engineering | System life cycle processes | 82 |
| ISO/IEC 25000 | Systems and software engineering | Guide to SQuaRE | 3 |
| ISO/IEC 25002 | Systems and software engineering | SQuaRE — Quality model overview and usage | 14 |
| ISO/IEC 25010 | Systems and software engineering | SQuaRE — Product quality model | 50 |
| ISO/IEC 25019 | Systems and software engineering | SQuaRE — Quality-in-use model | 13 |
| ISO/IEC/IEEE 15026 | Systems and software engineering | Systems and software assurance | 29 |
| ISO/IEC/IEEE 24765 | Systems and software engineering | Vocabulary | 18 |
| ISO/IEC/IEEE 24748 | Systems and software engineering | Life cycle management | 8 |
| ISO/IEC/IEEE 21841 | Systems and software engineering | Taxonomy of systems of systems | 4 |
| ISO/IEC/IEEE 24641 | Systems and software engineering | Methods and tools for MBSSE | 1 |
| ISO/IEC/IEEE 26514 | Systems and software engineering | Design and development of information for users | 2 |
| ISO/IEC/IEEE 42010 | Software, systems and enterprise | Architecture description | 8 |
| ISO/IEC/IEEE 42020 | Software, systems and enterprise | Architecture processes | 8 |
| ISO/IEC 15408 | Information security and cybersecurity | Evaluation criteria for IT security | 27 |
| ISO/IEC/IEEE 41062 | Software engineering | Software acquisition | 3 |
| ISO 56000 | Innovation management | Fundamentals and vocabulary | 2 |
| ISO 9000 | Quality management systems | Fundamentals and vocabulary | 2 |
| ISO 5127 | Information and documentation | Foundation and vocabulary | 6 |
| ISO/IEC 38500 | Information technology | Governance of IT for the organization | 2 |
| ISO/IEC/IEEE 12207 | Systems and software engineering | Software life cycle processes | 1 |
| ISO 10795 | Space systems | Programme management and quality | 3 |
| ISO 10303-2 | Industrial automation systems | Product data representation and exchange | 1 |

| Other Canonical Source | Count |
|---|---|
| INCOSE Systems Engineering Handbook v5 [74] | 4 |
| INCOSE Needs and Requirements Manual [116, 117] | 13 |
| Systems Engineering Body of Knowledge (SEBoK) [118] | 5 |
| NASA Systems Engineering Handbook [119] | 14 |
| DAU Glossary [24] | 7 |
| Wikipedia | 11 |
| Academic literature | 4 |
| Dictionaries [42] | 6 |

# Chapter 4

# Digital Requirements Engineering[6]

> We build our computer systems the way we build our cities— over time, without a plan, on top of ruins.
>
> Ellen Ullman [124]

This chapter addresses RQ3, which is restated as follows: *How do we define requirements in Seamless Digital Engineering?*

To address this question, the chapter presents the theory, formulation, and practical application of digital requirements engineering using the Model-Based Structured Requirement SysML (v1) profile [125] based on INCOSE guidance [117]. This research is based on prior work [126, 127], extending it with an INCOSE-derived meta-model and aerospace domain example. It distinguishes itself from related work in model-based requirements by its practical use of SysML model elements to define attributes found in the latest INCOSE guidance on requirements.

## 4.1 Introduction

As engineered systems grow in complexity, the demand for more cost-effective system development programs grows in turn. A critical point of leverage in reducing system development costs is by improving requirements engineering processes and the quality of its outputs [119]. Digital engineering (DE) promises to improve quality and reduce costs through increased access and connectivity of digital artifacts in a central data store called the Authoritative Source of Truth (ASoT) [23]. model-based systems engineering (MBSE)

[6]The contents of this chapter are based on the paper "Digital Requirements Engineering with an INCOSE-Derived SysML Meta-model" first published in The Proceedings of the 2024 Conference on Systems Engineering Research [123], and submitted as an invited extended paper for the CSER 2024 special issue of *Systems Engineering* journal.

improves upon document-centric systems engineering by incorporating formal digital models throughout systems engineering processes and products, by leveraging those digital system models to precisely represent design values and relationships, and by computing model validity based on modeling language and design rules. MBSE practice develops digital requirements engineering (DRE) as a response to increasing the system architecture model connectivity toward complete traceability of stakeholder needs and system requirements in support of DE goals.

With MBSE being central to the INCOSE systems engineering vision dating back to 2007 [21] and DE later being added in 2014 [128], and carried forward to the 2035 Vision [129], systems engineers should apply effort to incorporate model-based, digital methods in their practice. The effectiveness of MBSE compared to document-centric systems engineering continues to be studied, but results already corroborate its expected benefits [130–133], making it the most important systems engineering practice toward achieving DE goals in the organization. As a motivating example, the NASA-ESA Mars Sample Return campaign is "an ambitious and complex space system engineering endeavor" [134] with multiple interfacing space systems necessary to coordinate the safe return of Martian gas and solid core samples for further study on Earth. The Mars Sample Return program completed its second Independent Review Board assessment that included a probable program life cycle cost estimate of US$8-11 billion, "strong irrefutable evidence" that strong systems engineering (SE) is a crucial factor for mission success, and recommendations to refactor the program architecture to control costs [135]. Strong systems engineering is increasingly model-based, and while DRE is not a substitute for good requirements engineering practice, it highlights a paradigm shift that includes other digital engineering trends.

### 4.1.1 Digital Engineering

Recognition of the current and potential impact of digital models, including those used in MBSE, has led to the development by the US Department of Defense of a strategy for

taking greater advantage of digital models to transform the system development process. DE is "an integrated digital approach that uses authoritative sources of system data and models as a continuum across disciplines to support lifecycle activities from concept through disposal" [22]. As an engineering leader, NASA has invited the future of digital workflows by publishing a Digital Transformation strategy [136], MBSE strategy [137], and DE Acquisition Framework Handbook [138]. DE is not a new discipline of engineering but rather an intentional transformation of how an organization integrates and performs its engineering activities to achieve higher quality and efficiency [23].

One of the DE goals is to provide an enduring ASoT of the system to improve communication and decision-making. The system architecture model is one component of the ASoT, typically integrated in a centralized repository, and the system requirements may be created in the architecture model or synchronized with the model from a requirements management tool (RMT). DRE further integrates requirements with the ASoT, enabling formal verification & validation (V&V) activities that may be automated to improve model confidence and ease stakeholder reviews [139].

### 4.1.2 Requirements Engineering

requirements engineering (RE) is a subset of systems engineering that encompasses requirements development and requirements management. The ISO/IEC/IEEE 29148:2018 standard defines requirements engineering as "an interdisciplinary function that mediates between the domains of the acquirer and supplier or developer to establish and maintain the requirements to be met by the system, software or service of interest. Requirements engineering is concerned with discovering, eliciting, developing, analyzing, verifying (including verification methods and strategy), validating, communicating, documenting and managing requirements" [140]. The range of RE activities necessitates the use of metadata to organize information about each requirement, emphasizing that the familiar "shall" statement is only one attribute of a well-managed and model-connected requirement.

The INCOSE Guide to Writing Requirements (GtWR) provides a current perspective of well-formed requirements, and it defines a requirement statement as "the result of a formal transformation of one or more sources, needs, or higher-level requirements into an agreed-to obligation for an entity to perform some function or possess some quality within specified constraints with acceptable risk" [117]. The GtWR emphasizes that the requirement statement forms the basis of contractual language, and then presents a rules-based structured format for facilitating that communication. It defines a requirement expression as the requirement statement and its attributes. The GtWR recommends a data-centric practice using a RMT, as opposed to spreadsheets or documents, to model and present requirement expressions using diagrams and tables for stakeholder-tailored views.

Systems engineering handbooks provide another important reference for requirements engineering activities, complementing the detailed guides, manuals, and standards cited above. The INCOSE *Systems Engineering Handbook - Fifth Edition* provides updated Sections 2.3.5.2 and 2.3.5.3 that incorporate the latest INCOSE guides and manuals on needs and requirements engineering [74]. The NASA Systems Engineering Handbook [119] describes the traditional NASA requirements definition and management processes in Sections 4.2 and 6.2, respectively, emphasizing bidirectional traceability, and including a checklist in Appendix C and an informal set of characteristics similar to those defined in the latest INCOSE GtWR. The INCOSE GtWR and NASA sets of characteristics are compared in Section 4.4 and found to be complementary.

### 4.1.3 Systems Modeling Language

The Systems Modeling Language (SysML) by the Object Management Group (OMG) [141] is a standard language for modeling system architectures, which provides the capability to model the solution space as structure, behavior, and rules and requirements as digital model elements in a directional graph constrained by SysML semantics. SysML version 1.7 is expected to be the final version in the 1.x series as the standards development effort shifts to

the new version 2. SysML provides rudimentary facilities for modeling system requirements, including the primary attributes: ID, name, and text; and the relationships: derive, refine, satisfy, verify, and trace (which is discouraged in favor of the more precise relationships). Requirement type and rationale are not attributes provided by the SysML standard but are customizations of the SysML profile often provided by systems architecture modeling tools. According to the standard, SysML Requirements may be shown in a Requirements Diagram, or placed on other SysML diagrams to highlight relationships for certain stakeholder views; requirements tables and matrices are non-normative.

Today RE is often practiced with the help of an RMT which stores the requirements in a database and provides structured access to them for management activities. Since this RE practice uses digital models in a computer database, it may be considered MBSE or DE. However, the threat remains of duplicative system model elements interfering with traceability due to the RMT lacking the SysML meta-model used to define the system architecture. Controlled import/export or RMT data connector synchronization cycles alleviate this problem but may cause issues due to the non-standard interfaces between tools [13]. Although SysML v2 has addressed this issue by standardizing the API providing access to the system architecture model [142], due to disparate tool development lifecycles and enterprise adoption timelines, it will be years before it sees widespread adoption inside organizations [143]. SysML v2 improves upon v1 by modeling requirements as constraints that must be satisfied by parts of the system-of-interest, and it uses the consistent distinction between a requirement definition and requirement usage to aid in reuse [144].

Due to the perceived inadequate facilities for modeling and managing requirements using SysML, the GtWR cautions against its use [117]. Its apparent advantages with respect to its graphical syntax (diagrams) and traceability support have not been enough to satisfy the critical needs of managing and distributing possibly thousands of requirements. In addition, due to accessibility and training challenges associated with SysML modeling tools, the GtWR notes that stakeholders "will still prefer and demand" electronic/printed documents "for the

foreseeable future". However, SysML makes it possible to extend the language through Stereotypes, thus availing the systems engineer with meta-modeling capabilities to define custom elements and to use them to model RE concepts directly. SysML tools also provide means to generate artifacts such as requirements documents from the architecture model, ameliorating the need for specialized software for stakeholder review. This SysML meta-modeling capability specifically enables the GtWR-derived Model-Based Structured Requirement (MBSR) approach described in this chapter, while model-based artifact generation enabled benefits and encountered challenges discussed in Section 4.4.

### 4.1.4 Overview

This chapter presents an extension to the MBSR approach developed by [126] and [127] that incorporates the INCOSE GtWR ontology and the [140] standard pattern for requirement statements (Figure 4.4) using a notional space flight system to demonstrate the use of the SysML profile.

Contributions of the chapter are the following:

- We present a complete open-source SysML profile for digital requirements engineering based on structured need/requirement statements.
- We demonstrate application of the profile using an example model in the aerospace domain.
- We validate this approach to digital requirements engineering in a real-world NASA-ESA Mars Sample Return project in Pre-Phase A [119] and discuss the benefits and challenges.

The rest of the chapter is organized as follows: Section 4.2 describes the prior MBSR SysML language extension that connects model elements to their respective statement pattern slots; Section 4.3 extends the prior MBSR with our INCOSE-derived SysML meta-model and presents a representative example and summarized case study; Section 4.4 discusses the benefits and challenges of this approach compared to unstructured document-based and classical SysML requirements in the context of a real project in Pre-Phase A; Section 4.5

presents a literature review and compares SysML- and other model-based requirement techniques to INCOSE-derived MBSR; and Section 4.7 concludes with limitations and future work, and reflects on the value in the context of DRE.

## 4.2 Model-Based Structured Requirement (MBSR)

A strong SE practice is a countervailing force against cost overruns and system development project cancellations, and it is bolstered by a strong embedded RE practice from the beginning of the project. Where there were undefined, unsupported, or undisciplined RE processes in an organization, tailored guidance from INCOSE should fill the knowledge gap. Developing successful complex systems requires participation from diverse sets of stakeholders and organizations, leading to the development of Simplified Technical English [145] to reduce language ambiguity and confusion. Unlike the situation in the late 20th century when software development projects turned into production messes and accumulated technical debt, practitioners today may now benefit from advancements in RE to minimize ambiguity, reduce technical debt, formalize V&V processes, and produce consistently high-quality design output specifications [116, 146].

This section of the chapter describes classical standard SysML requirements modeling and performs a gap analysis (Section 4.2.1), structured requirements and its related work (Section 4.2.2), and INCOSE GtWR-derived meta-model extensions to MBSR from [126] (Section 4.2).

### 4.2.1 Classical SysML Requirements Modeling

Classical SysML represents a requirement as an "indivisible entity" [147] that enriches and is enriched by other system architecture model elements through the use of typed relationships. According to the SysML standard specification, "a requirement is defined as a stereotype of Unified Modeling Language (UML) Class subject to a set of constraints" and includes "properties to specify its unique identifier and text requirement," noting that

"additional properties ... can be specified by the user" [141]. The 'name', 'id', and 'text' properties of SysML requirements are defined as a simple string of characters, lacking any other structure, manipulable only by standard string processing functions of popular programming languages if available. Traceability relationships include containment, and subtypes of the UML Dependency relationship, as follows:

***Containment*** specifies the *Owner* of the requirement in the model containment hierarchy, graphically represented by a circle with two perpendicular lines crossed at the *Owner* end of the connector.

***Derive*** specifies a type of *Trace* that relates a derived requirement to its source requirement at the arrowhead-end of the dashed line.

***Refine*** specifies a directed relationship used "to describe how a model element or set of elements can be used to further refine a requirement" [141].

***Satisfy*** specifies a type of *Trace* that "describes how a design or implementation model satisfies one or more requirements" [141].

***Verify*** specifies a type of *Trace* that relates a test case or other model element to a requirement to identify a verification activity that checks that the system element meets its traced requirements and constraints.

***Copy*** specifies a type of *Trace* that creates a read-only copy of the requirement ID and textual statement on the requirement element at the non-arrowhead end of the dashed line.

***Trace*** is a general-purpose relationship between a requirement and any other model element; its use with more specific traceability relationships listed above is discouraged due to its ambiguity [141].

Classical SysML-based requirements popularized the idea of "cohabitation" where requirements and system model elements exist and are linked together in the same model [148] (Figure 4.1). Practicing DRE with this MBSE approach leads to the notion that requirements engineers and system architects should work in integrated teams rather than organizational

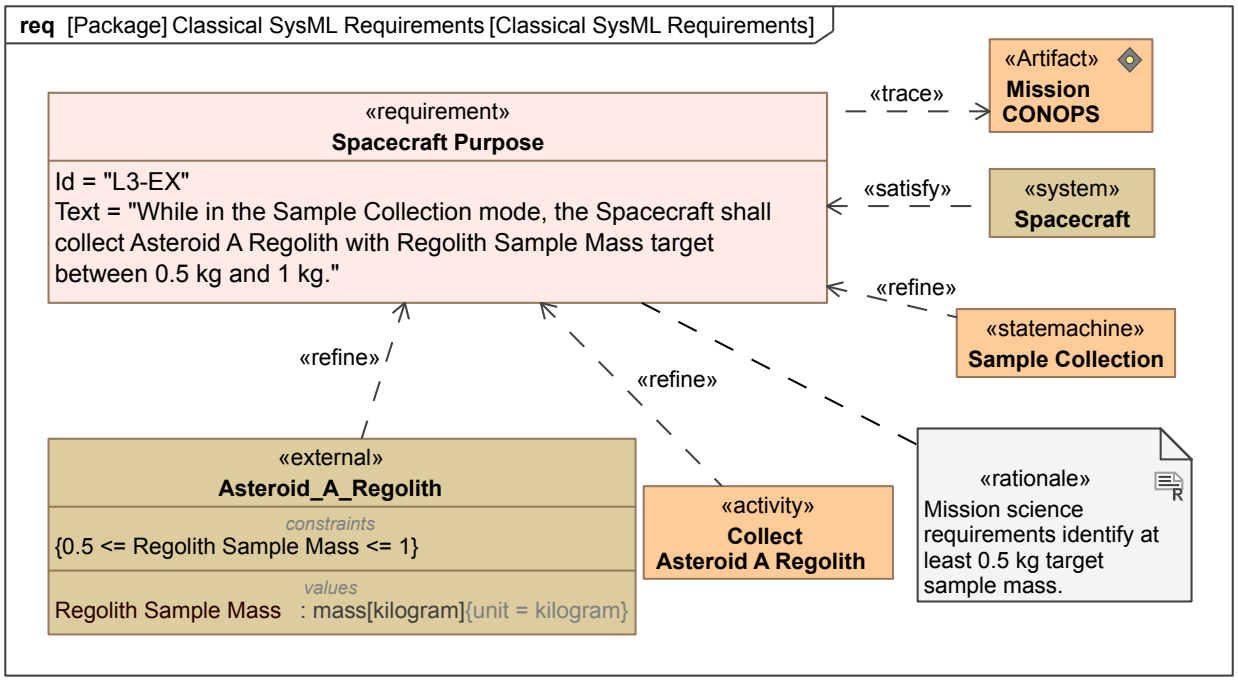



**Figure 4.1:** Classical SysML requirements modeling using standard relationships to system model elements.

silos and separate software tools [116]. However, due to the lack of requirements management facilities (such as software-defined workflows, audit logs, and change management boilerplate), managing requirements in SysML presents additional challenges already addressed by enterprise-grade RMTs. While the GtWR notes that "SysML requirement diagrams … are not well-suited to representing multiple or large sets of requirements" [117], we find that the same may also apply to requirement tables compared to RMTs (see Section 4.4.2).

In order for system architects to access the system requirements and add traceability relationships to the model, data connectors with RMTs or import/export/sync with Excel spreadsheets are used. Synchronization between tools mitigates the change management issues but due to differing data schema and meta-models, important information enriching the requirement expressions may not be available. Synchronized requirement tables will automatically highlight requirements that have changed, but the implication is that organizational silos are loosely cooperating in this manner, with the RMT owning the requirement ASoT and the SysML tool working with copies of that data.

digital requirements engineering (DRE) has outgrown classical SysML requirements due to their lack of precision, leading practitioners and tool vendors to extend standard SysML in various ways to satisfy MBSE objectives [141]. The *Trace* and *Copy* relationships are discouraged, leading to confusion about their use, although this may be mitigated by hiding them in the SysML tool's user perspective. The *Verify* relationship perpetuates the conflation of system V&V with requirements V&V ("verifying a requirement"), a distinction made clear in the INCOSE GtWR. The non-normative extensions, such as the additional requirement stereotypes and risk kinds in [141], may confuse practitioners due to the 'one size fits all' approach, and interface requirements, in particular, are rejected by the GtWR. Although SysML provides the basic utilities for complex system requirements analysis, the implementation of such analyses can take considerable effort, favoring organizations with mature model libraries. In summary, classical SysML requirements lack modeling precision without significant extensions, and introduce ambiguity and confusion in modeling constructs that contradict the latest RE guidance from INCOSE and ISO/IEC/IEEE.

### 4.2.2 Structured Requirement

The INCOSE GtWR and Needs and Requirements Manual (NRM) advocates for "structured, natural language" that treats the familiar 'shall' statement not as an "atomic entity" but as a "grammatical structure appropriate for communicating needs and requirements" [116]. In fact, rule number one (R1) in GtWR is "Structured Statement" which contributes to the quality characteristics: (C3) unambiguous, (C4) complete, (C5) singular, (C7) verifiable, and (C9) conforming (refer to Table 4.3 and the GtWR Summary Sheet). This INCOSE guidance builds upon a history of success using structured language for textual requirements [147, 149–152].

Requirement statements with uniform structure are shown to improve quality [151, 152] and have been adopted and recommended by [140] and [74]. Textual "shall" statements with parts in a standard order are easier to write, parse, and verify due to the regular structure that

guards against ambiguity and complex grammar. A structured requirement defines a pattern of the requirement statement using placeholders to clearly identify the critical features of the requirement. They have been called template requirements or "statement-level templates" [117], structured requirements [151], requirement structures [150], or "boilerplates" [153], but the GtWR uses the term 'requirement pattern' to avoid confusion with other common uses of the word 'template'. A requirement pattern is "represented by a series of building blocks (also called pattern slots) including all the elements envisioned to represent a well-formed, singular, and complete requirement" [117]. We present the following review of structured requirements using recommended patterns from [151] and [140].

*Plain-language Requirement Pattern:*

The [**Who**] *shall* [**What**] [**How Well**] under [**Condition**].

[**Who**] Singular subject of the requirement referring to an entity or agent that provides a capability or performs a function.

[**What**] Singular action-verb performed by the [**Who**], referring to required functionality or quality characteristic.

[**How Well**] Comparison factor(s) specified by constraints on the [**What**] which places feasible limits on the required functionality or quality characteristic, and which are used to verify the [**What**].

[**Condition**] "[M]easurable qualitative or quantitative terms specified by characteristics such as an operational scenario, environmental condition, or a cause that is stipulated for a requirement" [127].

*Plain-language Structured Requirement Example:*

The [**Spacecraft**] shall [**collect Asteroid_A_Regolith**] [**with Regolith_Sample_Mass target between 0.5 kg and 1 kg**] under [**Sample_Collection mode**].

The plain-language requirement pattern is flexible for use with functional requirements and quality requirements, and the pattern slot names may suit stakeholders who are not as comfortable with SE jargon. Standardizing with this requirement pattern across requirement sets in a project will significantly reduce the ambiguity compared to unstructured requirements. The ISO 29148 [140] standard pattern provides different names for the pattern slots and adds one for **[Object]** but is otherwise similar to the plain-language requirement pattern. The standard presents two patterns with an implied "shall" after the **[Subject]**:

*ISO/IEC/IEEE 29148:2018 Requirement Pattern:*

**[Subject]** ***shall*** **[Action]** **[Constraint of Action]**.

*OR*

**[Condition], [Subject]** ***shall*** **[Action] [Object] [Constraint of Action]**.

**[Condition]** "[M]easurable qualitative or quantitative attributes that are stipulated for a requirement, …and provide attributes that permit a requirement to be formulated and stated in a manner that can be validated and verified" [140].

**[Subject]** Singular system element in the same system hierarchy level as the requirement that provides a capability or performs a function.

**[Action]** Singular action-verb performed by the **[Subject]**, referring to required functionality or quality characteristic.

**[Object]** The entity being acted upon by the **[Subject]**; an element of the system or system environment.

**[Constraint of Action]** The "measurable outcome" [117] that "restrict[s] the design solution or implementation of the systems engineering process" [140] by applying feasible limits on the **[Action]** performed by the **[Subject]** on the **[Object]** under the stated **[Condition]**.

*ISO-standard Structured Requirement Example:*

**[While in the Sample_Collection mode]**, the **[Spacecraft]** shall **[collect]** **[Asteroid_A_Regolith]** **[with Regolith_Sample_Mass target between 0.5 kg and 1 kg]**.

Many patterns are possible depending on the domain and should be prescribed by the practicing organization (see Appendix C of GtWR for more examples). Here the Subject refers to the part of the system corresponding to the same level as the requirement. The 'shall' keyword has been inserted above to clarify the pattern and to emphasize that "requirements are mandatory binding provisions and use 'shall"' [140]. The Action signifies that the Subject *does* something — 'shall not' is forbidden (R16 in GtWR) — and that the statement is written in the active voice (R2 in GtWR), avoiding superfluous and possibly confusing verbiage such as "be capable of" (R10 in GtWR). This pattern makes clear that every requirement must have a verifiable Constraint. Although every requirement has an associated condition of when it is active, the first pattern may be used in the high-level functional requirements when the Condition is "ubiquitous" [117]. The latter pattern may be used as a default value for a redefined 'Text' SysML Property. model-based systems engineering (MBSE) practice encourages the use of these pattern slots to reference system model elements, not just defined terms in a project glossary, while RE practice as discussed in the GtWR emphasizes the important role of textual requirement statements, especially in the presence of system model views used to enhance stakeholder understanding.

### 4.2.3 Plain-Language MBSR

Model-Based Structured Requirement (MBSR) is an adaptation of structured requirements to the MBSE paradigm developed by [126] and [127], building on the pattern concept by making the pattern slots into SysML Properties of a Structured Requirement-stereotyped element (Figure 4.2). Thus, corresponding model elements and even diagrams may slot-in as model-based supplements to the textual "shall" statement in alignment with the Information-based Needs and Requirements Definition and Management meta-model adopted by the

GtWR [116, 154] (Figure 4.3). This meta-modeling method is well-supported by SysML profiles and the [141] specification which describes how SysML requirements can be extended to define requirement types and Property-Based Requirements (Section 4.5.2). MBSR goes "beyond treating a requirement expression as an indivisible entity and allows the terms inside the requirement statement to be referenced" [147] as commonly done in SysML modeling tools.

Earlier meta-models of MBSR [127] limited the attribute types to corresponding SysML types such as Block for the Subject pattern slot, but this approach was later found to be too restrictive and the standard SysML meta-model decision to use NamedElement was adopted [126]. The MBSR meta-model works by using Generalization relationships with the existing SysML Requirement, taking advantage of the standard syntax and semantics of SysML Requirements while separating concerns of the Structured Requirement pattern slots, and an organization's conventional requirement attributes. Organization attributes are customizable and might include a secondary system V&V method, unique identifier of the associated Work Breakdown Structure unit, or a 'short text' which provides an informal requirement statement in layman's terms. Other organization attributes may be required for compliance, for syncing with the RMT in use, and for ontological coherence with the ASoT.

## 4.3 INCOSE-Derived MBSR for Digital Requirements Engineering

The MBSR extensions presented in this chapter (abridged in Figure 4.2) explore the utility of adding the 49 Attributes, 42 Rules, and 15 Characteristics described in the INCOSE GtWR with appropriate SysML standard and custom types defined. Extending the prior MBSR profile (Section 4.2) in this way provided readily-usable SysML classifiers compatible with the INCOSE GtWR ontology, and embedded specific guidance into the SysML modeling environment to aid requirements development and requirements V&V.

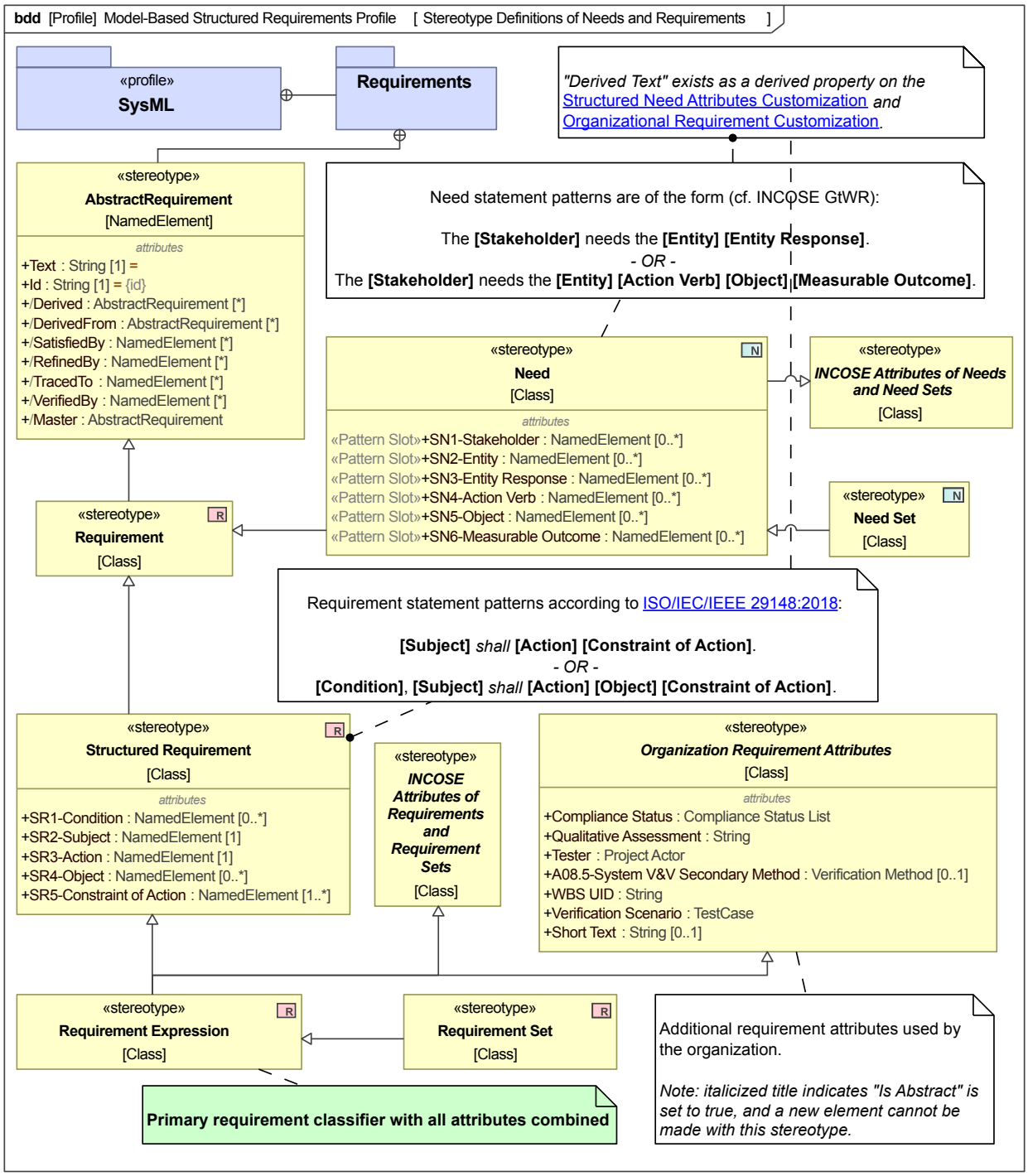



**Figure 4.2:** SysML Stereotype definitions for Requirement Expression and Requirement Set. Refer to Figure 4.5 for further detail on the INCOSE GtWR Attributes.

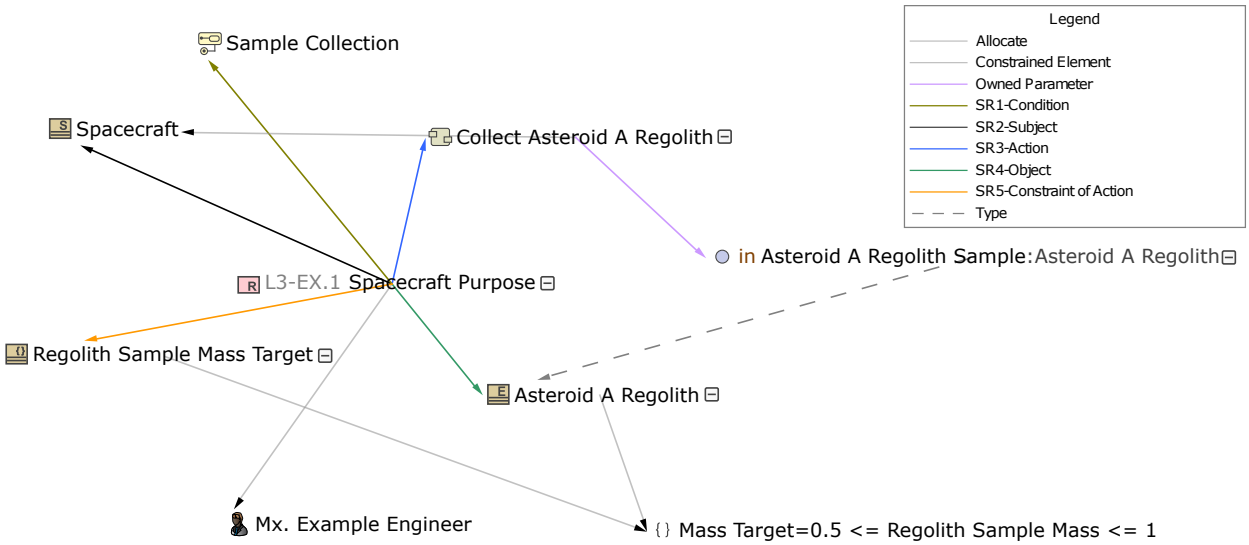



**Figure 4.3:** Relation Map of example MBSR and related system architecture SysML elements.

### 4.3.1 Meta-Model of Rules, Attributes, Characteristics

Figure 4.4 presents a SysML meta-model derived from the Entity-Relationship Diagram in Figure 4 of GtWR [117] that clearly relates these DRE terms to each other while respecting their given definitions. The "Requirement Statement" term is replaced with the MBSR "Structured Requirement" and additionally relates Patterns and Pattern Slots; and the "Requirement Expression" is emphasized as *the primary classifier for use* so as not to confuse it with the standard SysML Requirement (Section 4.2.1) in the tool's user interface. A "Requirement Set" contains Requirement Expressions and potentially Requirement Sets, and is a subclass of Requirement Expression rather than a SysML Package to maintain uniform application of the GtWR ontology in the SysML profile. UML-based multiplicity and roles were used on directed associations in Figure 4.4 to reflect the cross-reference matrices in the GtWR.

A Requirement Set (historically called requirement modules or composite requirements) is defined to distinguish from an individual Requirement Expression as in the GtWR, and provides additional querying, filtering, and meta-modeling capabilities for isolating Sets. Likewise, Needs and Need Sets are defined and inherit some of the same Attributes, with a

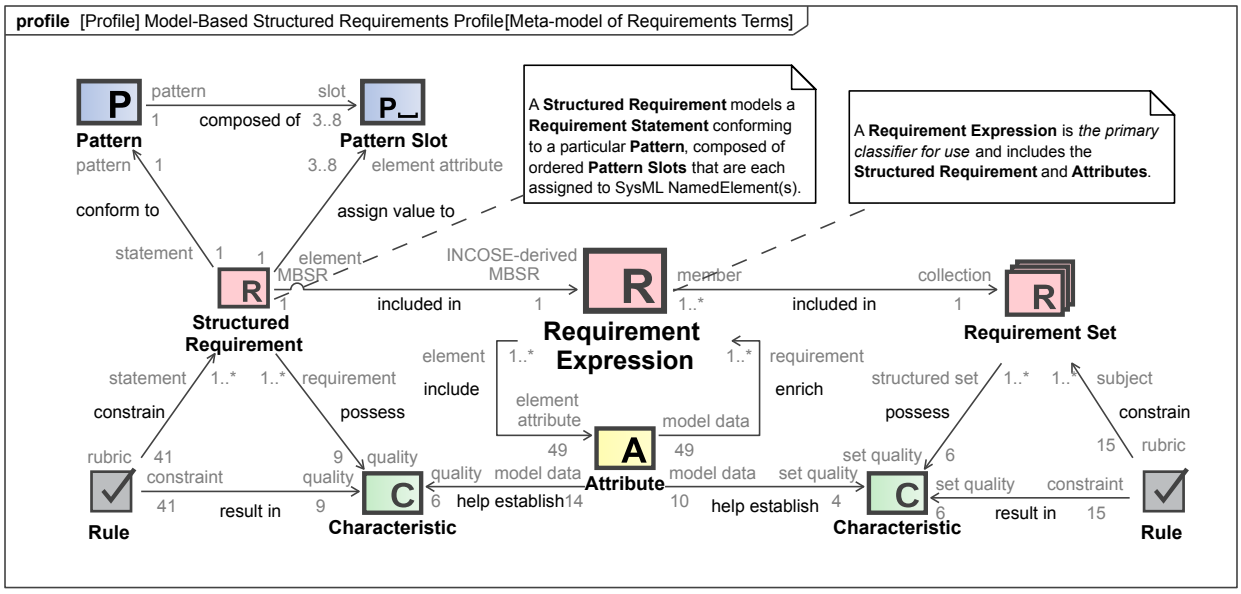



**Figure 4.4:** SysML meta-model of INCOSE-derived MBSR classifiers derived from Figure 4 of [117].

different icon to visually distinguish them from requirements, potentially addressing a major concern with standard SysML voiced in the GtWR.

Attributes help establish Characteristics, of which there are exactly 9 for Needs and Requirements, and 6 for Need Sets and Requirement Sets (see Table 4.3 in Section 4.4.3 for details). The Attributes are grouped according to their purpose and numbered to maintain order and to aid in searching; some Attributes are marked with an ending asterisk (*) to indicate membership in the minimum set of Attributes according to the GtWR. Notably, some Attributes are already defined elsewhere, such as A15 (Unique Identifier) and A16 (Unique Name), and are modeled using «Customization» derived properties that query the standard SysML properties for completeness of the INCOSE-derived MBSR profile.

Value Types such as Enumerations (e.g., Verification Method) and organization-related Stereotypes (e.g., Project Actor, Business Unit) with the Class meta-class were created and selected based on the Attribute definitions and guidance in the NRM. These Value Types and set of minimum viable Attributes should be harmonized with the organization's conventions, e.g., requirement types, statuses, and risks (Figure 4.5). The Organization Requirement Attributes shown in Figure 4.5 are representative, and their modification and use depends

on the organization's context and conventions. This list of attributes should conform to the Organizational Requirement Definition Requirements, as represented in [117] Figure 7. The presence of such attributes here highlights the flexibility of the MSBR technique while addressing potential concerns that some attributes common in the organization, such as a secondary system V&V method, are missing from the GtWR attributes.

### 4.3.2 Requirements V&V using the Profile

We emphasize the distinction between requirements V&V and system V&V, and until the release of the INCOSE Requirements Working Group technical products [116, 117], the common systems engineering parlance has been that "requirements verification" refers to system verification against the requirements. Requirements V&V refers to the verification and validation of the requirement expressions themselves, and here we present requirements V&V using the SysML modeling tool and INCOSE-derived MBSR SysML profile.

A metric suite is provided in the MBSR Profile to demonstrate how SysML-validation-based metric definitions and custom scripting can be used to compute completeness metrics on an MBSR set [125, 127]. The metric suite works by referencing SysML validation rules that check if each MBSR Pattern Slot is filled. The metric table using it shows how many MBSRs are in a given Package at a certain date and time, how many set a value for each Pattern Slot, and what percentage of the MBSRs complete the requirement Pattern. The intended usage is as follows:

1. Create a new metric table under a suitably named Package.
2. Set the "Metric Suite" under "Criteria" to "MBSR Completeness" from the MBSR Profile.
3. Click "Calculate Metrics" and select "Add New Metric with Different Parameters".
4. Ensure that "Scope" and "Type" columns are shown by selecting them from the "Columns" dropdown menu.
5. Set the Scope value of the new metric instance to the Package containing MBSRs.

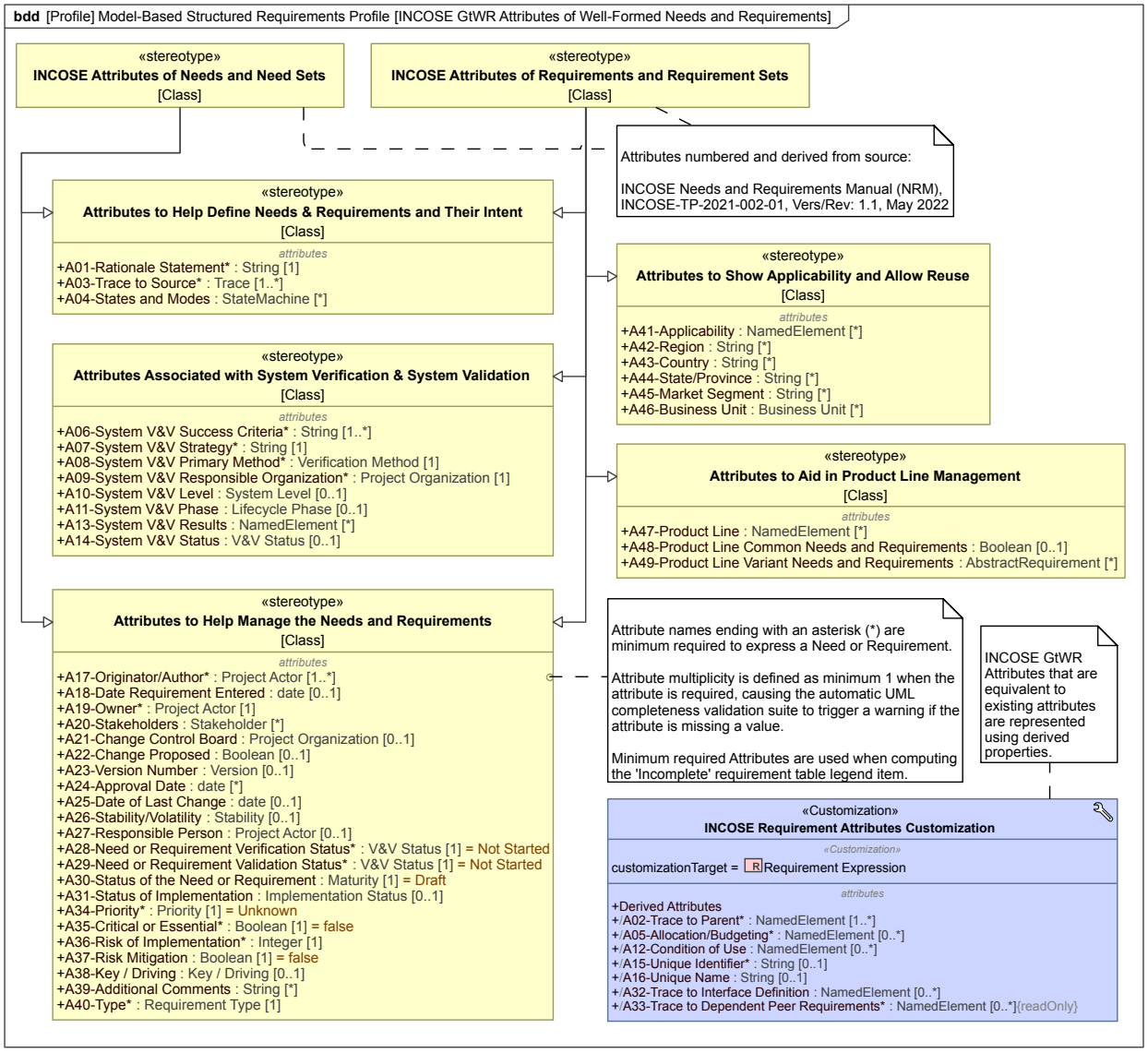



**Figure 4.5:** Model-based INCOSE GtWR Attributes of Well-Formed Needs and Requirements. Refer to Figure 4.2 for further relationship definitions.

6. Set the Type value to the "Structured Requirement" from the MBSR Profile.
7. Click "Calculate Metrics" and select "Recalculate".
8. Export the metric table to CSV or XLSX for downstream workflows, or
9. Query the metric table elements from report templates to generate custom reports.

The MBSR Completeness metric suite may be used in concert with traceability metric suites to create requirement metrics dashboards. Computing metrics in this way can be more effective than checking every requirement in a matrix, and it provides timestamped data that may be used for burndown charts or compliance audits. See Section 6.3 for a discussion of future work on MBSR metrics.

When a Requirement Expression or Requirement Set is verified and validated, a SysML «satisfy» relationship is added from each requirement to the respective Characteristic, providing model data and metrics of their well-formedness. Like Attributes, Rules help establish Characteristics of a well-formed Requirement Expression or Requirement Set, and during requirements V&V activities, a «satisfy» or new «Violate» relationship is likewise created for metrics and feedback (Figure 4.6). The Rules and Characteristics linked to each requirement may be shown in requirement tables, metric and generic tables, and rendered in custom reports.

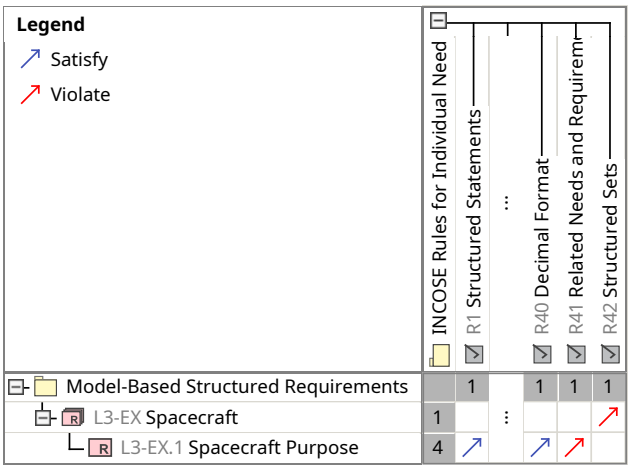



**Figure 4.6:** A Requirements Satisfaction Matrix of INCOSE GtWR Rules (abbreviated), with Violate relationships shown in red.

### 4.3.3 Application and Case Study

Having presented the INCOSE-derived MBSR profile above (Section 4.3), we now show how the SysML profile and technique are applied using an example in the space domain. The Mars Sample Return project case study is also discussed briefly to explain how the INCOSE-derived MBSR profile was used in a real-world project, starting with unverified draft requirements. Whether the requirements engineer is converting requirement statements to MBSRs or they are developing new requirements in tandem with the system architecture modeling activities, the process is much the same.

In Figure 4.7, we show how a singular INCOSE-derived MBSR may be rendered in a SysML requirements diagram. By showing and hiding compartments and attributes of interest, a custom view is rendered to meet stakeholder concerns. At a glance, a particular requirement expression is shown to be validated according to individual characteristics (Figure 4.4), and remaining rules in violation may be shown to draw attention while reviewing the requirement statement. The classical SysML requirement attributes of Id and Text are shown first, followed by categorized attributes numbered accordingly. Lastly, the base MBSR pattern slots are rendered with their associated values' icons to emphasize the model-based nature of the view. Figure 4.8 shows much of the same information as Figure 4.7 but using the standard XML-based data interchange format in support of the tool diversity present in digital engineering environments.

While Figure 4.7 is shown here more for demonstration purposes, the requirement table is a more common view due to its compactness and ability to sort columns. Figure 4.9 shows how INCOSE-derived MBSRs may be viewed as a nested tree in a requirements table, with an icon accompanying each model element to again emphasize its model-based nature. Here we also show the numbered characteristics validated, and the numbered rules violated to assist in requirements V&V activities. All structured requirement pattern slots are shown, with some being filled in with "TBD" elements to clearly identify parts of the architecture yet to be modeled and which may be easily searched and enumerated in a separate "TB(X)"

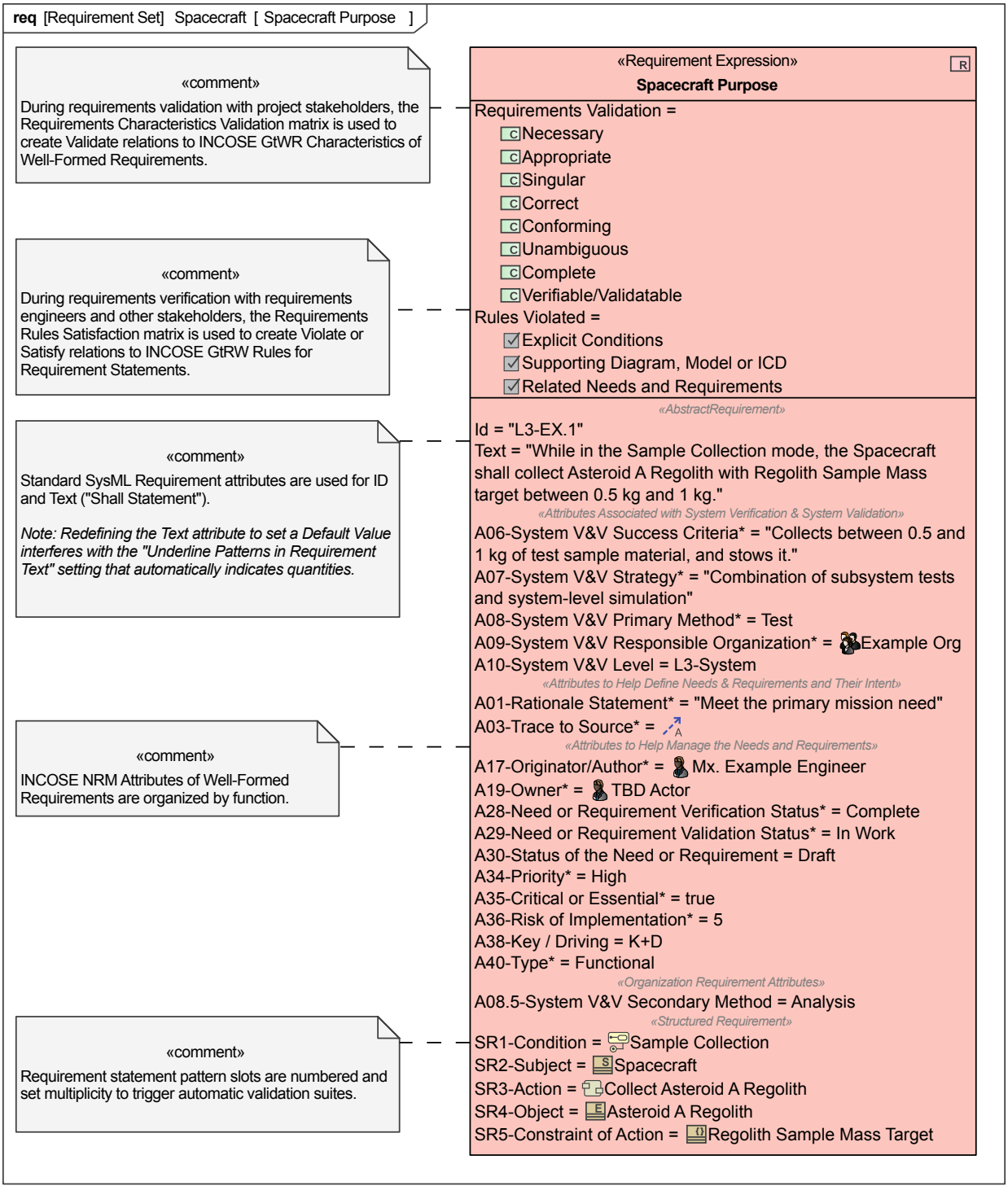



**Figure 4.7:** Single example MBSR with INCOSE GtWR Attributes, Characteristics, and Rules.

```
<Model_Based_Structured_Requirements_Profile:Requirement_Expression
xmi:id='_2022x_2_46d01c0_1707158979643_806727_21479_'
base_Class='_2022x_2_46d01c0_1707158979643_806727_21479'
Id='L3-EX.1'
Text='While in the Sample Collection mode, the Spacecraft
     shall collect Asteroid A Regolith with Regolith Sample Mass
     target between 0.5 kg and 1 kg.'
SR1_Condition='_2022x_2_46d01c0_1707158979575_932108_21295'
SR2_Subject='_2022x_2_46d01c0_1707158979604_996809_21373'
SR3_Action='_2022x_2_46d01c0_1707158979574_907885_21293'
SR4_Object='_2022x_2_46d01c0_1707158979726_396359_21900'
SR5_Constraint_of_Action='_2022x_2_46d01c0_1707158979579_344791_21322'
A01_Rationale_Statement_='Meet the primary mission need'
A08_System_V_V_Primary_Method_='Test'
A10_System_V_V_Level='L3-System'
A28_Need_or_Requirement_Verification_Status_='Complete'
A30_Status_of_the_Need_or_Requirement='Draft'
A34_Priority_='High'
A38_Key___Driving='K+D'
A40_Type_='Functional' />
```

**Figure 4.8:** UML 2.5 XMI definition of the example MBSR shown in Fig. 4.7 (newlines added & attributes reordered for readability).

table. Attribute "A02-Trace to Parent" is also shown to emphasize traceability of top-level requirements to need expressions in the model. Lastly, a model-based legend is used to clearly identify orphan requirements (red), withdrawn (but not deleted) requirements (gray), and incomplete requirements (yellow). Complete requirements according to their pattern slots are shown in the table with a clear or white background. Such a legend is easily modified using structured expressions and scripts, and, for example, may only show requirements as complete when their minimum attribute set has been filled.

The INCOSE-derived MBSR SysML profile [125] was used to develop over 300 requirements and 50 requirement sets at NASA Jet Propulsion Laboratory (JPL) for the Mars Returned Sample Handling project in Pre-Phase A [155, 156]. Drafts of requirements were received from the subsystem engineering leads in a spreadsheet, and imported into No Magic Cameo Systems Modeler 2022x [157] using the requirements table diagram and the Excel import feature. The given information, such as rationale, verification method, verification approach, and additional comments were first inserted into their respective GtWR-derived Attributes,

| # | Id | Name | Text | A02- Trace to Parent* | A40- Type* | SR1- Condition | SR2- Subject | SR3- Action | SR4- Object | SR5- Constraint of Action |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | L2-CC | Contamination Control | The Spacecraft shall satisfy the Spacecraft Cleanliness Requirements. | N1.4 Contamination Control | Science | | Spacecraft | | Spacecraft Cleanliness Requirements | CC 4-StdDev |
| 2 | L2-CC.1 | Cleanroom Sampling | While the Spacecraft in Cleanroom, the CC Technician shall Sample Spacecraft Surface Cleanliness according to the CC Sampling Daily Schedule. | N1.4 Contamination Control | Compliance | Spacecraft in Cleanroom | CC Technician | Sample Spacecraft Surface Cleanliness | Spacecraft | CC Sampling Daily Schedule |
| 3 | L3-EX | Spacecraft | The Spacecraft shall return Asteroid A Regolith within the Mission Duration. | N1 Mission Need | Operational | | Spacecraft | Collect Asteroid A Regolith (context Spacecraft) | Asteroid A Regolith | Mission Duration |
| 4 | | Orphan Requirement | [This requirement has no value for A02-Trace to Parent.] | | | | | | | |
| 5 | | Withdrawn Requirement | [This requirement has the value of A30-Status of the Need or Requirement set to Withdrawn.] | | | | | | | |
| 6 | L3-EX.1 | Spacecraft Purpose | While in the Sample Collection mode, the Spacecraft shall collect Asteroid A Regolith with Regolith Sample Mass target between 0.5 kg and 1 kg. | N1.1 Sample Collection | Functional | Sample Collection | Spacecraft | Collect Asteroid A Regolith (context Spacecraft) | Asteroid A Regolith | Regolith Sample Mass Target |
| 7 | L3-EX.2 | Imaging | While in the Target Imaging Range, the Spacecraft shall Capture Images of the Asteroid A while preserving the High Dynamic Range. | N1.2 Target Imaging | Functional | Target Imaging Range | Sample Capture Subsystem | Capture Images | Asteroid A | High Dynamic Range |
| 8 | L3-EX.3 | Return to Earth | When the Sample Collection Activity completes, the Spacecraft shall return to the Earth Landing Site within 3 years. | N1.3 Return Safely | Performance | Collect Asteroid A Regolith (context Spacecraft) | Spacecraft | TBD Activity | TBD Object | Return Travel Duration |
| 9 | L3-EX.5 | New Requirement | [Condition], [Subject] shall [Action] [Object] [Constraint of Action]. | | | TBD State | Spacecraft | TBD Activity | TBD Object | TBD Constraint |

**Figure 4.9:** INCOSE-derived MBSRs in a requirement table with model-based legend highlighting.

and continued to be revised during requirements development. Matrix diagrams were created, as in Figure 4.6, to aid in the revision of the requirement statements, providing visual feedback as a kind of checklist, and data for metrics used to triage requirements for later revision. All Rules, Attributes, and Characteristics contained the corresponding documentation from GtWR to further assist the usage of the Profile with tooltips. To model Defined Terms as described in GtWR, Cameo Systems Modeler Glossary Tables were filled with Terms and synonyms—often acronyms—active hyperlinks to the definition source, and SysML Allocate relationships to system model elements and diagrams, providing automatic underlining of Defined Terms used consistently throughout requirements statements. In addition, other meta-model elements were added to the Profile to capture the broader range of related information: Goal, Assumption, Project Actor / Role / Organization, Requirement Types relevant to NASA /JPL.

## 4.4 Discussion

INCOSE-derived MBSR is a SysML language extension which is leveraged during typical requirements engineering activities, and which is compatible with existing RMT integration. However, compared to an RMT, which supports additional requirement attributes, MBSR validates not only standard data types such as dates and numbers, but also any custom elements created in an architecture model and its accompanying profile. Compared to RMTs, which would require similar customization to include the GtWR rules and characteristics, the MBSR-enabled SysML model makes requirements V&V data directly accessible, supporting a digital engineering approach of data-driven decision-making in a variety of stakeholder views.

The immediate advantage of this MBSR approach was the ability to keep architecture modeling activities confined to a single tool, and without the loss of expressiveness that would normally result from using classical SysML Requirements. Report templates were created using the full MBSR expression, and according to stakeholder expectations for engineering reviews. The system architecture model and cohabitated requirements were exported to targeted stakeholder views, including custom PowerPoint templates for an architecture overview and a Requirement Set formal review, an internal website (Web Report), and Excel spreadsheets (adding relevant table columns as needed). While this targeted approach of INCOSE-derived MBSR usage received overwhelmingly positive feedback from the team, it came with its own challenges that became evident as limitations of the tool were encountered (Section 4.4.2).

### 4.4.1 Observed Benefits

Observed benefits compared to unstructured document-based and classical SysML requirements during the NASA / ESA / JPL joint Mars Sample Return project included: improved requirement statement quality due to the use of a common requirement Pattern, Rules, and accompanying Attributes; increased accuracy and assurance of model views using generated custom reports; access to system model elements for increased traceability and specificity;

and model-assisted traceability views of Key Driving Requirements across multiple levels of the system hierarchy. Table 4.1 provides a more complete list with perceived impact on the project during Pre-Phase A.

### 4.4.2 Observed Challenges

Applying INCOSE-derived MBSR to a real project met with real challenges (Table 4.2), primarily that the SysML tool in use was slow to generate reports, render views of model data and adorning legend items, and save model updates, causing significantly delayed cycle times during MBSR meta-modeling and modeling of the system. Other challenges of this approach relate to the SysML tool's lack of RMT facilities that would automate some model updates and provide high assurance of their correctness. In this case, only one engineer was developing the requirements at the time, and formal change management had not been activated during this technology development phase. A more detailed list of the observed challenges and perceived impacts on the project are sorted in Table 4.2.

### 4.4.3 Applicability to Standard Guidance

This INCOSE-derived MBSR Profile supports the requirements engineering practices according to the NASA SE Handbook guidelines and Appendix C checklist [119]. Although the NASA SE Handbook does not present clearly defined characteristics, the glossary entry and Appendix C checklist contain language that may be mapped to all of the INCOSE GtWR Characteristics (refer to Table 4.3): (C1) 'necessary …to meet mission and system goals and objectives'; (C2) 'compliance'; (C3) 'unambiguous in meaning' and 'complies with the project's template and style rules'; (C4) 'completeness'; (C5) 'not redundant'; (C6) 'feasible to obtain'; (C7) 'verifiability/testability'; (C8) 'correct'; (C9) 'clear' and 'clarity'; (C10) 'completeness'; (C11) 'consistency' and 'not in conflict with one another'; (C12) 'technically feasible'; (C13) 'clarity' and 'adequately related with respect to terms used'; (C14) 'can be validated'; and (C15) 'correctness' and 'completeness'. By contrast, the [140] standard provides definitions for 14 out of the 15 Characteristics and clearly categorizes them for individual requirements

**Table 4.1:** Benefits and perceived impacts of using INCOSE-derived MBSR for DRE. Scale: ● High, ◐ Medium, ◔ Low.

| **Observed Benefit** | **Impact** |
|---|---|
| Access to the full system architecture model supports creation of the ASoT with rich traceability | ● |
| Combined use of a singular tool is cost-effective | ● |
| Consistent use of requirement statement patterns aids V&V of system elements | ● |
| Glossary terms are underlined throughout the model, with multiple definitions visible in the tooltip | ● |
| Requirements are exportable in custom Word / Excel / PowerPoint model-based reports | ● |
| SysML meta-modeling capabilities support extensive customization | ● |
| Full ISO 80000 units of measure and MARTE real-time SysML profiles are available to extend and use | ● |
| TBX summary table is easily made with scope, and TB[CDRN] regular expression as a filter on all fields | ● |
| Custom-query Relation Map diagrams expose MBSR relationships tailored to stakeholder concerns | ◐ |
| Requirement IDs are customizable, with predictable increments, and may be typed-in directly in diagrams | ◐ |
| Collaboration plugin supports simultaneous team use with change control and precise feedback capability | ◐ |
| Any Attribute is displayable and sortable in a table | ◐ |
| Matrices with embedded tooltip documentation and double-click entries supports efficient workflows | ◐ |
| Custom legend items adorn tables and diagrams using custom scripts or Structured Expressions | ◐ |
| Requirement tables may be imported/synchronized with Excel spreadsheets or use data connectors | ◐ |
| MBSR Attributes are quickly searchable/filterable by number, e.g. “A28” or “SR” | ◐ |
| Only used Attributes appear on symbols by default | ◐ |
| Derived Properties defined by custom scripts or Structured Expressions enhance model-based definition | ◐ |
| UML 2.5 XMI exports are complete with model element reference identifiers | ◔ |
| Copy/paste of values into multiple table cells simultaneously works in the simple case | ◔ |

**Table 4.2:** Challenges and perceived impacts of using INCOSE-derived MBSR for DRE. Scale: ● High, ◐ Medium, ◔ Low.

| Observed Challenge | Impact |
|---|---|
| Careless mistakes made in the Shared Profile may destroy system model information | ● |
| Custom report templates are time-consuming and error-prone to create | ● |
| Reports are slow to generate, increasing cycle times | ● |
| Writing custom scripts is challenging due to often inadequate documentation | ● |
| Requirements management Attributes such as 'Date of Last Change' and 'Version Number' require manual entry workflows | ◐ |
| Newly filled Attributes will appear in all affected diagrams, mangling diagrams unless compartments are manually suppressed | ◐ |
| Glossary terms sometimes do not underline, are never underlined in the Web Report, or the tooltip does not reliably appear | ◐ |
| Redefining «AbstractRequirement» Text attribute breaks automatic underlining of quantity relations | ◐ |
| Table adorning / loading can be slow | ◐ |
| Legends cannot adorn table cells, only table rows | ◐ |
| ReqIF exports require manual attribute mapping, and do not include referenced system model elements | ◐ |
| Requirement IDs are tedious to set/increase/decrease with Element Numbering dialogs | ◐ |
| Attributes with multiplicity >1 fails to copy/paste; paste chooses the first element with that name, not the same exact element copied | ◐ |
| Element naming conflicts interfere with the use of copy/paste in tables | ◐ |
| Requirement IDs may conflict and can lose their order | ◔ |
| Attributes appear under multiple groups in table column selection dialog | ◔ |
| SysML Properties may only have one (1) Owner, preventing reuse in organization-defined Attribute Sets | ◔ |
| Table scrolling performs sequential loading, temporarily revealing blank rows in large requirement tables | ◔ |
| Glossary term allocations to model elements may be duplicative of MBSR attribute values | ◔ |

**Table 4.3:** INCOSE Characteristics of well-formed sets and individual needs and requirements [117] with mapping to NASA and ISO guidance [119, 140].

| ID | Name | Applicability | Derivation | NASA? | ISO? |
|---|---|---|---|---|---|
| C1 | Necessary | Needs & Requirements | Formal Transformation | ✔ | ✔ |
| C2 | Appropriate | Needs & Requirements | Formal Transformation | ✔ | ✔ |
| C3 | Unambiguous | Needs & Requirements | Agreed-to Obligation | ✔ | ✔ |
| C4 | Complete | Needs & Requirements | Agreed-to Obligation | ✔ | ✔ |
| C5 | Singular | Needs & Requirements | Formal Transformation | ✔ | ✔ |
| C6 | Feasible | Needs & Requirements | Agreed-to Obligation | ✔ | ✔ |
| C7 | Verifiable | Needs & Requirements | Agreed-to Obligation | ✔ | ✔ |
| C8 | Correct | Needs & Requirements | Formal Transformation | ✔ | ✔ |
| C9 | Conforming | Needs & Requirements | Formal Transformation | ✔ | ✔ |
| C10 | Complete | Need Sets & Requirement Sets | Formal Transformation | ✔ | ✔ |
| C11 | Consistent | Need Sets & Requirement Sets | Formal Transformation | ✔ | ✔ |
| C12 | Feasible | Need Sets & Requirement Sets | Agreed-to Obligation | ✔ | ✔ |
| C13 | Comprehensible | Need Sets & Requirement Sets | Agreed-to Obligation | ✔ | ✔ |
| C14 | Able to be validated | Need Sets & Requirement Sets | Agreed-to Obligation | ✔ | ✔ |
| C15 | Correct | Need Sets & Requirement Sets | Formal Transformation | ✔ | |

and for requirement sets. MBSR supports the bidirectional traceability and the creation of NASA-requested artifacts such as the Requirements Allocation Sheet, TBX report, and Requirements Verification and Validation Matrices. Further discussion of the relationships among the NASA SE Handbook, the ISO 29148:2018 standard, and the INCOSE GtWR may be found in the ontology section of the NRM. While the NASA SE Handbook and INCOSE GtWR (and related guides and manuals) are complementary, the GtWR provided enumerated and defined precision amenable to SysML meta-modeling and reuse in the system architecture model.

## 4.5 Related Work

[158] presents three approaches to model-based requirements found in the literature: 1) dedicated classes and flagged models, 2) math- and property-based models of requirements, and 3) semantic extensions to model the problem space. The relationship of MBSR to template-based textual requirements is discussed in Section 4.2.2, and classical SysML requirements ("dedicated classes") are discussed in Section 4.1.3. This section provides a

brief overview of related model-based requirement approaches while noting similarities and differences with MBSR as presented in this chapter.

### 4.5.1 System Models as Requirements

[159] presents a model-based method for capturing requirements using standard SysML elements and diagrams other than the textual-based SysML «AbstractRequirement». As discussed in their chapter, this method unnecessarily constrains the solution space and is therefore considered poor requirements engineering practice. Flagged models as requirements will be unfamiliar to stakeholders and may cause significant confusion compared to "shall" statements [116]. MBSR extends the use of textual statements with deeper connectivity to the system model compared to standard SysML, without relying on SysML diagrams to model the problem space. Statement patterns are used in MBSR both to assist readability and to facilitate model definition and traceability. While SysML elements are used in MBSR slots, they may remain placeholders until design activities commence and more details of the solution space are known.

### 4.5.2 Mathematical Models of Requirements

Wymorian theory of requirements is based in set theory [160] and is not directly applicable to SysML models. This formulation of the problem space enables mathematical queries to assist in requirements V&V and may, in particular, assess completeness of a requirement set. The basic modeling construct is called a system design requirement and is defined as a sextuple including: 1) input/output requirement, 2) technology requirement, 3) performance requirement, 4) cost requirement, 5) trade-off requirement, and 6) system test requirement. Compared to Wymorian theory of requirements, MBSR is more relevant to modern MBSE practice because of its integration with SysML models. Integration of this alternative MBSE theory with SysML remains an open area of research [161], and SysML v2 may provide new research opportunities due to its strong semantic foundations compared to v1.

Property-Based Requirements (PBRs) are another mathematical formulation of system requirements based on a semilattice [148, 162]. This formulation may be applied to other modeling languages such as AADL, Modelica, and VHDL-AMS, but [162] focuses the presentation on SysML. While PBRs focus on the integration and computation of physical properties to prepare for system V&V, our work emphasizes the importance of well-formed requirement statements whose components are SysML model elements. Like MBSR, PBR extends the standard SysML requirement element by creating a PBRequirement stereotype, and like MBSR, the PBRequirement stereotype provides four primary attributes representing 1) Condition, 2) Carrier, 3) Property, and 4) Domain [162]. What is considered a well-formed PBR refers to its completion of these attributes to form a mathematical constraint, which in light of [117] is insufficient for mature requirements engineering practice. PBR does not distinguish between stakeholder needs and system requirements, whereas INCOSE-derived MBSR supports needs and need sets in addition to requirement expressions and requirement sets. Composite requirements, hierarchical nesting of requirements, and automated identification of dependencies are features of PBR that MBSR supports using SysML (e.g., Figure 4.3). [141] presents PBR as a non-normative extension with examples in Annex E, but this formulation differs from [162] in that it focuses on "quantitative specification of numerical parameters, relationships, equations and/or constraints." PBRs, as presented in the SysML v1.7 specification Annex E.8, may be adapted to the MBSR Profile to provide formal verification capabilities especially suited for numerical- and logic-focused requirements while benefitting from the INCOSE GtWR ontology and rules. [163] presents another property-based formulation of requirements using a custom object-oriented modeling tool, and discusses similar benefits to MBSR, such as enhanced traceability, consistency, completeness, maintainability, and integration with artifact and report generation.

Ontology-based requirements engineering is an active area of MBSE research with promising results that may contribute to the ASoT with reusable ontology-defined terms [93, 146, 164]. An ontology-based requirement may be composed of slots referring to Pattern Slots of the

Requirement Statement, and the Attributes contributing to the complete Requirement Expression. Through establishing ontology relationships and axioms, model-based requirements V&V may be assisted with tool automation ensuring (C10) completeness, (C11) consistency, (C15) correctness, (C13) "unambiguity", and (C14) "traceability" [146]. While the INCOSE-derived MBSR Profile may appear to implement an ontology, it lacks the formal semantics defined by ontology languages such as the Web Ontology Language (OWL) [165]. Therefore, the MBSR Profile is not currently capable of being exported to a standard ontology format, although a mapping from the meta-model to an ontology may be technically feasible. Integration with ontology tools may enhance the reasoning capability of a SysML model, but at the cost of tooling complexity, which MBSR attempts to avoid.

### 4.5.3 Semantic Extensions to Model the Problem Space

Alternatively, semantic extensions to SysML may be employed to model the problem space, known as True Model-Based Requirements (TMBR) [166]. TMBR derives from the Wymorian theory of requirements (Section 4.5.2) and so "attempts to model any type of requirement as a set of (or sets) of required input/output transformations" [158]. TMBR is not intended for modeling stakeholder needs [158], compared to INCOSE-derived MBSR, which includes Needs and Need Sets in its meta-model (Figure 4.4). The set theory foundations of TMBR prevent "formal flaws in problem formulation, such as enforcing design solutions or leaving the requirement unbounded" [158], corresponding to Characteristics (C15) Correct, (C12) Feasible, and (C14) Able to be validated. Due to the reuse of SysML features to model the problem space, stakeholders may find TMBR difficult to understand compared to traditional requirement statements [167]. TMBR and MBSR are similar in that they use SysML as the primary modeling language and make use of SysML attributes with defined quantities and constraints. As a language extension, MBSR is compatible with other SysML profiles, including SYSMOD [168].

## 4.6 Limitations

Limitations of the present work are primarily due to scoping and available SysML v1 tool capabilities. While our experiments have primarily made use of Cameo Systems Modeler, we posit that non-normative extensions to SysML v1 (such as tables, matrices, and scripting capabilities) are not restricted to certain tools, and that other tools will work in similar ways that we cannot fully describe here. Therefore, the MBSR technique is presented as a SysML extension, rather than a tool-grafted technique. Although experimental design as done by [167] to compare this MBSR approach with other tool-enabled requirements engineering methods was not conducted, qualitative validation of this MBSR approach was conducted in the context of two real-world system development projects at NASA Jet Propulsion Laboratory. Producing experimental evidence of the MBSR contribution toward satisfying the 15 Characteristics of Well-formed Requirements and Sets of Requirements (Table 4.3) is left as future work; readers are encouraged to refer to [117] and accompanying manuals available for free to INCOSE members. Contractual agreements currently limit the exposure of proprietary information, such as the roughly 500 MBSRs written, so a minimal example of an asteroid sample collection spacecraft was used for demonstration. Although included in the MBSR Profile and linked to the applicable INCOSE GtWR Attribute sets, Needs and Need Sets were not used and evaluated in the projects. We provide the MBSR SysML profile in an open-source online repository [125] so other organizations and researchers may adapt it and run experiments to further the systems engineering community's understanding of the effectiveness of this approach.

## 4.7 Conclusions

While it is too early to determine the economic value of using INCOSE-derived MBSRs compared to traditional requirements engineering methods, initial results indicate the technique had a positive impact on a real project. This chapter presented an extension of prior MBSR research using the latest INCOSE Guide to Writing Requirements and related it to

an emerging DRE paradigm. Over 300 requirements were written and revised with project stakeholder feedback to test the effectiveness of this approach in improving quality and connectedness, contributing to a valuable ASoT. The experience gained through this NASA JPL system development project was shared in brief in Section 4.4.The benefits and challenges listed in subsections 4.4.1 and 4.4.2 were gathered through experience, and while the benefits are likely transferable to other tools, the challenges may be ameliorated by improved software performance and enhanced model-based workflows for DRE activities.

We found that the development of SysML requirements in this direction has been fruitful for gaining support of MBSE in the context of requirements engineering. By encoding the primary object classes (Figure 4.4) from the INCOSE GtWR into the systems architecture modeling tool, rapid improvement in real-world requirements quality was achieved by a systems engineering student intern over a period of 6 months. The INCOSE-derived MBSR profile supports a DE approach to requirements engineering and system development that may reduce costs and improve quality if deployed at scale. Our MBSR technique is independent of a methodology, as its intent is to extend the usefulness of SysML models that include requirements, from early-phase requirements development to architecture solution development to later-phase SE activities. The technique is therefore adaptable to new and existing SE methodologies. SysML-based architecture modeling tools may facilitate the MBSR approach by supporting requirements management needs such as automatic timestamps, organization-defined collaborative workflows, extensible and rigorous requirement identifier definition, text values with embeddable model elements, and enhanced speed of the software. We believe that INCOSE-derived MBSR has the potential to leverage SysML strengths in the transition to DE while effectively satisfying stakeholder needs, but further development, testing, feedback, and experimentation are needed to further validate this claim.

## 4.8 Summary

This chapter addressed RQ3, restated as: *How do we define requirements in Seamless Digital Engineering?*

It answered this question by presenting a reusable SysML profile in the digital requirements engineering paradigm based on prior work on Model-Based Structured Requirements. The focus of this work was using SysML model elements to adequately model requirement expressions as defined in [117] from INCOSE. One of its research contributions was a formalization of the needs and requirements meta-model presented in [117]. This theoretical work was complemented by an accessible aerospace example and supporting evidence of its successful use in a real-world NASA-ESA Mars Sample Return project.

Seamless Digital Engineering is digital engineering from first principles, where authoritative data and models are used and curated throughout a system's lifecycle. Digital requirements engineering using the INCOSE-derived MBSR is the way to define needs and requirements in Seamless Digital Engineering, although future work remains to establish its parameters of use throughout the system lifecycle.

# Chapter 5

# Reference Architecture for Seamless Digital Engineering

> Computer programming is an exact science in that all the properties of a program and all the consequences of executing it in any given environment can, in principle, be found out from the text of the program itself by means of purely deductive reasoning.
>
> C. A. R. Hoare [169]

This chapter addresses RQ4, which is restated as follows: *How does the proposed Seamless Digital Engineering Reference Architecture meet the needs of "bootstrapping a trustworthy and seamless digital engineering appliance"?*

## 5.1 Introduction

While Ref. [22] set a high-level strategy with a number of goals (Section 2.3.1) and expected benefits, it is not suitable as a sole source of stakeholder needs and requirements for digital engineering. For our purposes of designing a seamless DE appliance, we must fill the gap of specifying these needs which describe the problem space of DE tooling. This problem space includes the many engineering capabilities needed to design, specify, implement, integrate, verify and validate, certify, deploy, maintain, and retire real-world systems, according to the standard Vee lifecycle model ( [74]). And rather than considering these activities separately, digital engineering demands that they are done in a model-based way which integrates all the model data into the Authoritative Source of Truth (ASoT) with active traces to/from the many digital artifacts (Figure 1.1). Given the grand challenge of Seamless DE (Table 2.4), a reference architecture model is necessary to capture the essential complexity of DE capabilities.

Rather than treating computer-aided engineering (CAE) tools as black-boxes that are procured from disparate tool vendors and integrating them to form a DEE, Seamless DE treats them as white-boxes that must be specified in sufficient detail to form a Seamless DE appliance that is correct-by-construction. And since no tool suite currently exists that is capable of specifying this level of detail in a reliably consistent, coherent, and model-based manner, the tool suite must first be bootstrapped to specify itself. This architectural technique of open reflexive specification also satisfies critical stakeholder needs related to bootstrappability, trustworthiness, conviviality, and seamlessness because the system ships with the design blueprints and analytical capabilities needed to construct a copy or derivative of itself. Such a product may be certified to conform to a reference architecture, and independently audited using its own supplied Authoritative Source of Truth (ASoT) to assure evidence-based trustworthiness.

Therefore, a Seamless DE reference architecture is the next step in addressing this grand challenge. Using SysML v2 we adopt the expressive power needed to specify such systems at the architectural levels, while more powerful formal methods are available and needed to prove correctness-by-construction. This chapter applies systems engineering processes to the definition of a reference architecture for Seamless DE, against which future system architectures may be validated. Its research contributions include (1) a needs analysis of DE guided by current and future IT capabilities, (2) description of a DE meta-language which incorporates SysML v2 and formal verification capabilities, (3) a full-source bootstrap strategy for countering Trusting Trust [170], and (4) the open-source Seamless DE Reference Architecture defined in SysML v2 which captures these analyses in a formal model.

## 5.2 Stakeholder Goals

The results of the goals analysis from Section 2.3.1 showed that despite the goals decomposition, we are left with high-level goals such as `G-DE-4.1.1` 'Develop DE IT infrastructure', `G-DE-3.1.1` 'Infuse technological innovations', and `G-DE-3.2.1` 'Make use of data'. These

goals were intended to orient organizations' change management plans rather than to supply information needed for clean-slate design of a digital engineering environment (DEE). Nevertheless, they are important sources to trace our Seamless DE mission-goals-objectives (MGOs), providing essential direction for needs analysis and later validation activities.

The Softgoal Interdependency Graph (SIG) diagrams from Section 2.3.1 provided legible aids for decomposing and understanding the original DE goals, and these were transferred into the Seamless DE Reference Architecture SysML v2 model. The goals and objectives fit naturally into the mission-goals-objectives (MGOs) taxonomy described in Ref. [116], and their hierarchy is supported by the SysML v2 language (Listing 5.1). A `MGO_goal` is defined in the Seamless DE model library as a specialization of a requirement (Listing 5.1), following the standard pattern for extending the SysML v2 language with reusable types and accompanying attributes.

**Listing 5.1:** Definition of MGO goal in the SDE Profile as a SysML v2 specialized requirement with semantic metadata

```
requirement def Goal;
requirement goals : Goal [*];

metadata def <MGO_goal> GoalMetadata :> SemanticMetadata {
    redefines baseType = goals meta SysML::RequirementUsage;

    attribute authors : String [1..*] ordered;
    attribute version : String [1];
    attribute changeHistory : String [1..*] ordered;
    attribute priority : Positive [1];
    attribute criticality : Positive [1];
    attribute source : String [1..*];
    attribute info : String [0..*];

    subsets annotatedElement : SysML::RequirementDefinition;
    subsets annotatedElement : SysML::RequirementUsage;
}
```

The five claimed benefits from Ref. [22] were reformulated as stakeholder concerns, expressed as questions, with the original expected benefit text included as an additional comment for traceability (Listing 5.2). This formulation enabled MGOs to frame those concerns, following the standard SysML v2 modeling pattern (Listing 5.3).

**Listing 5.2:** Expected DE benefit reformulated as a stakeholder concern (SysML v2)

```
concern def <'C-DE-1'> 'Informed␣decision-making' {
        doc /* How does the system support better informed decision-making? */
    comment /* Informed decision-making and/or greater insight thru increased
        transparency */
    stakeholder pm : DE_Stakeholders::'Program␣Manager';
}
```

**Listing 5.3:** Excerpt of DE goal framing stakeholder concern of Listing 5.2 (SysML v2)

```
#MGO_goal requirement def <'G-DE-1'> 'Formalize␣model␣lifecycles'
specializes DE_Supergoals::'SG-DE-A', DE_Supergoals::'SG-DE-B', DE_Supergoals
    ::'SG-DE-C' {
    doc /* The DE Enterprise should formalize the development, integration,
        and use of models. */

    authors[1] = "DoD␣Office␣of␣the␣Deputy␣Assistant␣Secretary␣of␣Defense␣for␣
        Systems␣Engineering";
    priority = 1;
    source = "DoD␣Digital␣Engineering␣Strategy␣(2018)";

    frame concern : DE_Stakeholders::DE_Concerns::'C-DE-1';
    frame concern : DE_Stakeholders::DE_Concerns::'C-DE-2';

    #MGO_goal requirement def <'G-DE-1.1'> 'Formalize␣model␣planning' {
            doc /* The DE Enterprise should formalize model planning. */

            #MGO_objective requirement def <'G-DE-1.1.1'> 'Formally␣develop␣
                plans␣to␣digitally␣represent␣SoI' {
                doc /* The DE Enterprise should formalize plans to digitally
                    represent the SoI. */
            }
    }

    // rest omitted for brevity
```

The MGOs of DE derived from Ref. [22] provide a baseline for traceability from Seamless DE MGOs that are more specific to correctness-by-construction [121, 171–173]. The Seamless DE mission (Listing 5.4) derived from the definitions of Seamless DE in Chapter 2 and Chapter 3 specializes the general DE mission, then goals and objectives are derived and decomposed to form the `SDE_MGOs` package.

**Listing 5.4:** The 'Seamless DE Mission' as a SysML v2 requirement with semantic metadata

```
#MGO_mission requirement def <'SDE-M.1'> 'Seamless␣DE␣Mission' specializes
    DE_Problem_Space::'DE-M.1' {
         doc /*
              * The mission of Seamless Digital Engineering is to develop
                  systems that are correct-by-construction
              * by employing a DE Appliance that is itself correct-by-
                  construction.
              */

     comment /*
              * Seamless Digital Engineering is a digital engineering tooling
                  paradigm that guarantees model
              * coherence and integrity by affording an elegant human-computer
                  interface for systems modeling
              * that is end-to-end formally verified down thru the computer
                  hardware.
              */
}
```

## 5.3 Quality Attributes

Rather than stating goals, objectives, and qualities as vague textual statements in a design document, as was done in traditionally architected systems, the SDE Reference Architecture captures them as specified model elements available for further and continued analysis. The quality and quality-in-use characteristics described ontologically in Chapter 3 are modeled as goals and objectives (Listing 5.5) which are amenable to further decomposition for measurement and traceability to requirements. Overall, the Seamless DE MGOs capture key quality attributes of a Seamless DE Appliance, namely seamless, elegant [44], trustworthy [90], and convivial [174, 174].

**Listing 5.5:** Excerpt of the 'Seamless DE Appliance' quality goal as a SysML v2 requirement with semantic metadata

```
#MGO_goal requirement def <'SDE-G.1'> 'Seamless␣DE␣Appliance' specializes 'SDE
    -M.1' {
        doc /* The DE Appliance should be seamless. */
        subject DE_Appliance;

        #MGO_goal requirement def <'SDE-G.1.1'> 'Seamless␣Interaction␣
            Capability' {
            doc /* The DE Appliance should have the quality of seamless
                interaction capability. */

            #MGO_objective requirement def <'SDE-G.1.1-O.1'> 'Compatibility' {
                doc /*
                    * The DE Appliance should exchange information with other
                        products,
                    * and/or to perform its required functions while sharing
                        the same common environment and resources
                    */
            }

            // rest omitted for brevity
```

The goal `SDE-G.1` (Listing 5.5) includes the lower-level goals of 'Seamless Interaction Capability', 'Seamless Integration', and 'Seamless Quality-in-Use', each decomposed into objectives corresponding to the quality (sub-)characteristics as defined in the Seamless DE Ontology (Section 3.4). These lower-level goals and objectives inherit the subject element, `DE_Appliance`, with the textual statements rewritten from the original SQuaRE definitions to make the DE Appliance subject of the sentence. One example of this formulation is the `MGO_objective` `` `SDE-G.1.3-O.3' `` ('Trustworthiness') which states, according to Ref. [90], "The DE Appliance should support user confidence that their expectations are met in a verifiable way." A trustworthy DE appliance is therefore one that does not lie to its operator, and which affords the capability of verifying its operation[7]. Research in trustworthy computing [175–180] has found it to be a grand challenge of its own [181], with essential quality characteristics of (1) intuitive, (2) controllable, (3) reliable, and (4) predictable [182].

[7]The quality claim here is of the appliance itself, not of the system models it produces. That is, it is possible to design an untrustworthy system using a trustworthy system.

In addition to seamless and trustworthy, elegance was a quality attribute identified in Section 2.3.4 that captures essential dimensions of quality for any engineered system. Like 'seamless', a commonly applied term which previously lacked engineering elucidation, 'elegant' was described in detail by Refs. [43–45, 183] as the quality of "a system that is robust in application, fully meeting specified and adumbrated intent, is well-structured, and is graceful in operation." This quality attribute was decomposed in Ref. [47] as having four sub-characteristics: (1) system efficacy, (2) system efficiency, (3) system robustness, and (4) minimizing unintended consequences. These sub-characteristics are traceable to SQuaRE quality and quality-in-use sub-characteristics, as before.

Lastly, to meet the perceived needs of stakeholders who operate the Seamless DE Appliance, qualities of conviviality are desirable. Convivial tools were first described in Ref. [184] tools which can be mastered by the operator, are developed and maintained by a community, and which afford the operator independent efficiency. These ideas later influenced the first creators of the personal computer and we would be remiss to omit them from our model. Similar 'laws of sane personal computing' have been developed [185] and are captured in the reference architecture as stakeholder expectations (Listing 5.6). Although numerous works have drawn inspiration from the concept of conviviality, Refs. [174, 186] expand it for use in engineering, calling upon engineers to design-for-conviviality and decomposing it into the dimensions: relatedness, adaptability, accessibility, bio-interaction, and appropriateness, throughout a system's lifecycle of materials, production, use, and infrastructure. These qualities, in addition to being traceable to SQuaRE, shall be used to derive needs and requirements that suitably constrain the design space and provide traceability from particular product components, features, and capabilities to high-level design intent.

**Listing 5.6:** Datskovskiy's "Seven Laws of Sane Personal Computing" [185] modeled as stakeholder expectations in SysML v2

```
package Seven_Laws_of_Sane_Personal_Computing {
    doc /* Datskovskiy's Seven Laws of Sane Personal Computing. Source: http
        ://www.loper-os.org/?p=284 */

    comment /* Although described as "laws", these statements read as
        stakeholder expectations. */

    #expectation requirement def <'ESTK-SL-SPC.1'> 'Obeys␣operator' {
        doc /* Does not interrupt tasks or activities; does not perform side-
            effects unless clearly defined */
    }

    #expectation requirement def <'ESTK-SL-SPC.2'> 'Forgives␣mistakes' {
        doc /* Infinite `undo'; clear messaging of the source of mistakes as
            well as advice on how to correct them */
    }

    #expectation requirement def <'ESTK-SL-SPC.3'> 'Retains␣knowledge' {
        doc /* With the modern capacity of RAM and disk space, computers
            should not forget past actions and data; work drafts are never
            lost */
    }

    #expectation requirement def <'ESTK-SL-SPC.4'> 'Preserves␣meaning' {
        doc /* The machine does not execute untrusted binary blobs because
            they cannot be verified to operate in accordance with secure
            semantics; the machine is an interpreter of high-level language */
    }

    #expectation requirement def <'ESTK-SL-SPC.5'> 'Survives␣disruptions' {
        doc /* Upon encounter of a run-time error, the state is preserved and
            an interface provided to the operator to recover from the error
            and continue operation */
    }

    #expectation requirement def <'ESTK-SL-SPC.6'> 'Reveals␣purpose' {
        doc /* All information held within the system is accessible,
            introspectible, and modifiable by the operator */
    }

    #expectation requirement def <'ESTK-SL-SPC.7'> 'Serves␣loyally' {
        doc /* The machine shall never tell a lie to the operator */
    }
}
```

## 5.4 Operational Concept

To gain insight into the proposed system from an operational point-of-view, HCI mock-ups were constructed to highlight key scenarios and features (Figures 5.1, 5.2, 5.3, 5.4, 5.5, 5.6, 5.7, 5.8, 5.9, and 5.10). These mock-ups may resemble the HCIs of the LISP machines of yesteryear [25–27], and this resemblance is not accidental. Those machines were precursors to the proposed DE appliance, although they lacked the safety guarantees and computational power that we expect today.

Reference [12] proposed the 'information appliance' and the Activity-Based Computing (ABC) HCI paradigm that focuses on activities broken down into tasks, actions, and operations. The idea is to provide the tools necessary for the operator's activity and no other distractions. Learning the activity is equivalent to learning how to use the information appliance which was especially designed for those tasks. Sometimes an activity is collaborative and so incorporates seamless chat and other collaboration capabilities — obviously, without the need to open one of the many messaging software applications in use today. ABC is notable for being researched for more than two decades, although other models may be relevant, especially in light of augmented and virtual reality, and verbal interface technologies.

The exposition of the operational concept is communicated by a series of figures and explanations. To start, rather than being bombarded by technical readouts and software branding, the operator should see a smooth animation with helpful messages and other indicators that communicate the state of the Seamless DE Appliance as it starts up and prepares its assets for use (Figure 5.1).

The appliance always remembers its last valid state, including the position, sizing, and other layout information of the operator's last Activity-Based Computing (ABC) session. When the startup process has completed, the operator is presented with focused options to continue their work, or start a new project (Figure 5.2). The library or knowledgeable is always available, providing structured and linked information, templates, documents, and ontologies that assist the operator in any DE task.

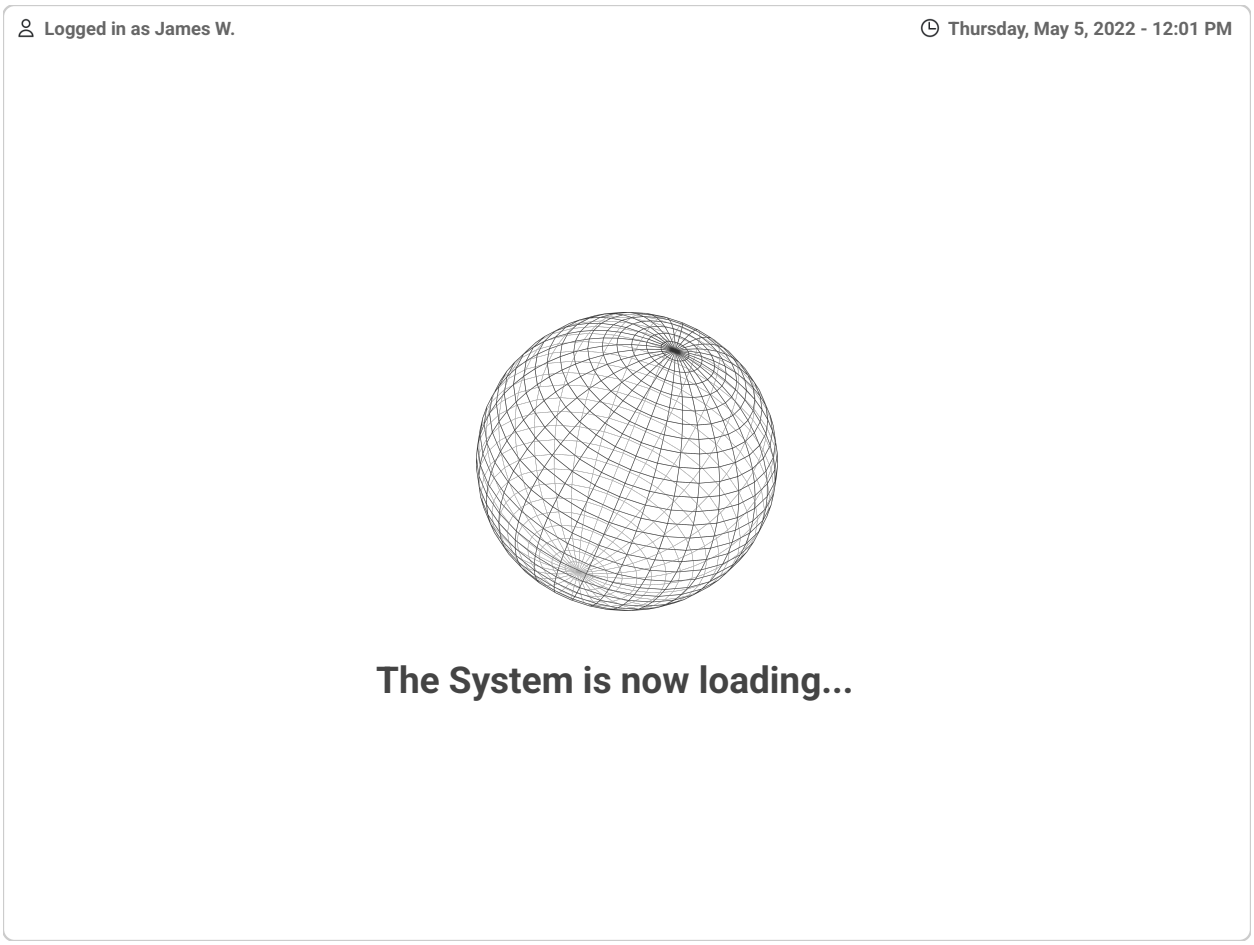



**Figure 5.1:** Booting up the Seamless DE Appliance.

When the operator chooses to start a new project, they are greeted with a familiar digital form (Figure 5.3). However, rather than leaving the operator to complete many repetitive tasks of setting up a new project, the appliance provides all the scaffolding needed for the complete project management lifecycle, including mathematical and financial models to support scheduling, resource allocation, and reporting. The appliance avoids inundating the operator with too many options in each view, instead preferring to fill in common default values, and grouping input fields appropriately (Figure 5.4).

The appliance provides methodological guidance to the operator to avoid the "blank page" syndrome and unprincipled modeling activities (Figure 5.5). Rather than abstract text descriptions common in software engineering, an elucidating and interactive figure [187] is

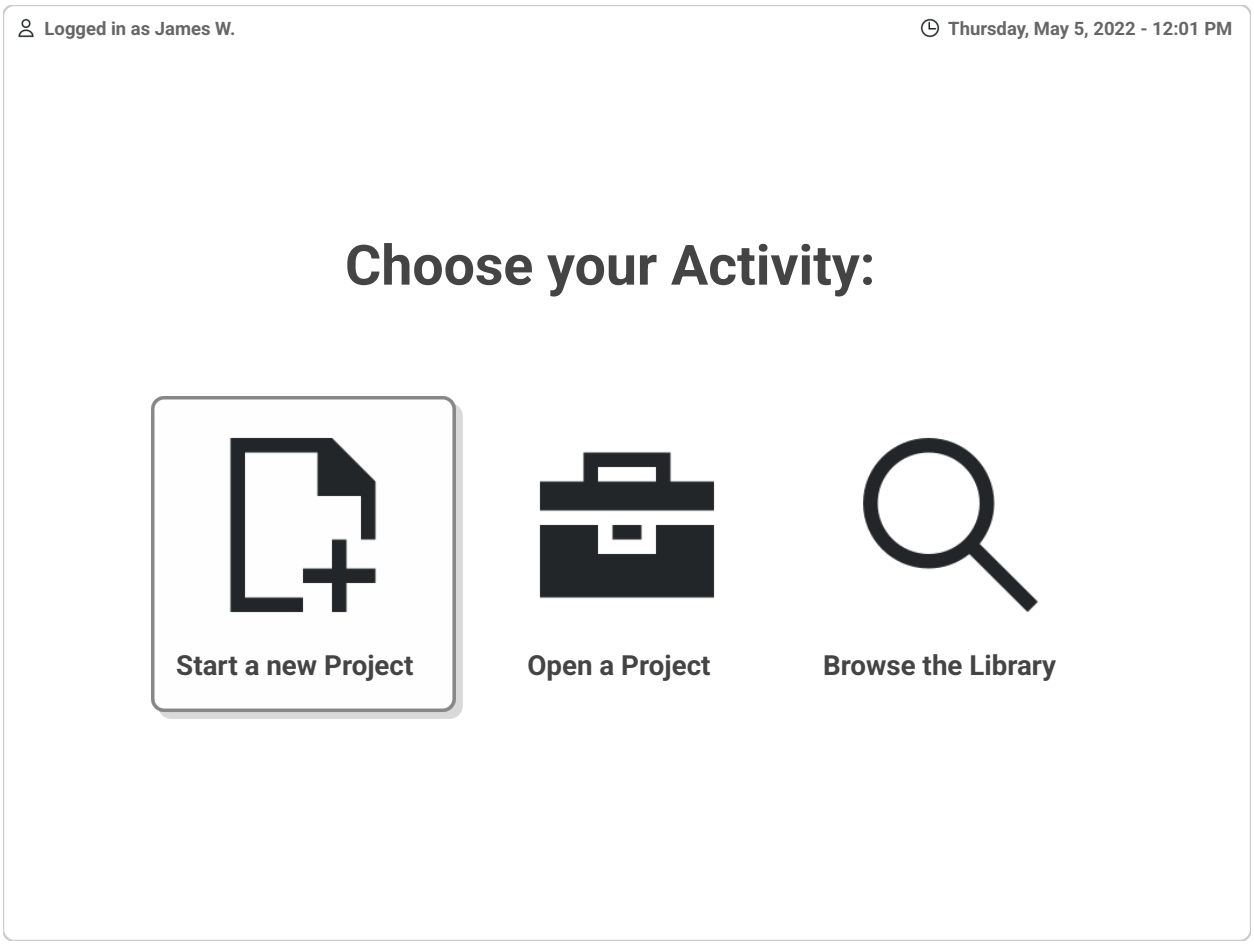



**Figure 5.2:** Choosing your first activity.

presented. Additional help is offered in this activity context, which leads to architecture modeling and systems engineering guidance in the library seen in Figure 5.2.

The "Activities" bar at the bottom, although seemingly familiar to operators in the Window-Icon-Menu-Pointer (WIMP) paradigm, actually serves different purposes. The title in the middle of the bottom bar indicates the current activity in Activity-Based Computing (ABC) [12, 188]. The "Activities" button at the bottom-left affords the ability to change activities, where the last state of each activity session is preserved for seamless resumption. The function icons at the bottom-left provide functionality to modify and configure the current activity, possibly adding remote human collaborators. The icons at the bottom right next to the current local time indicate states of the appliance, including seamless voice, text, video, and screen-share collaboration with single or multiple remote human collaborators.

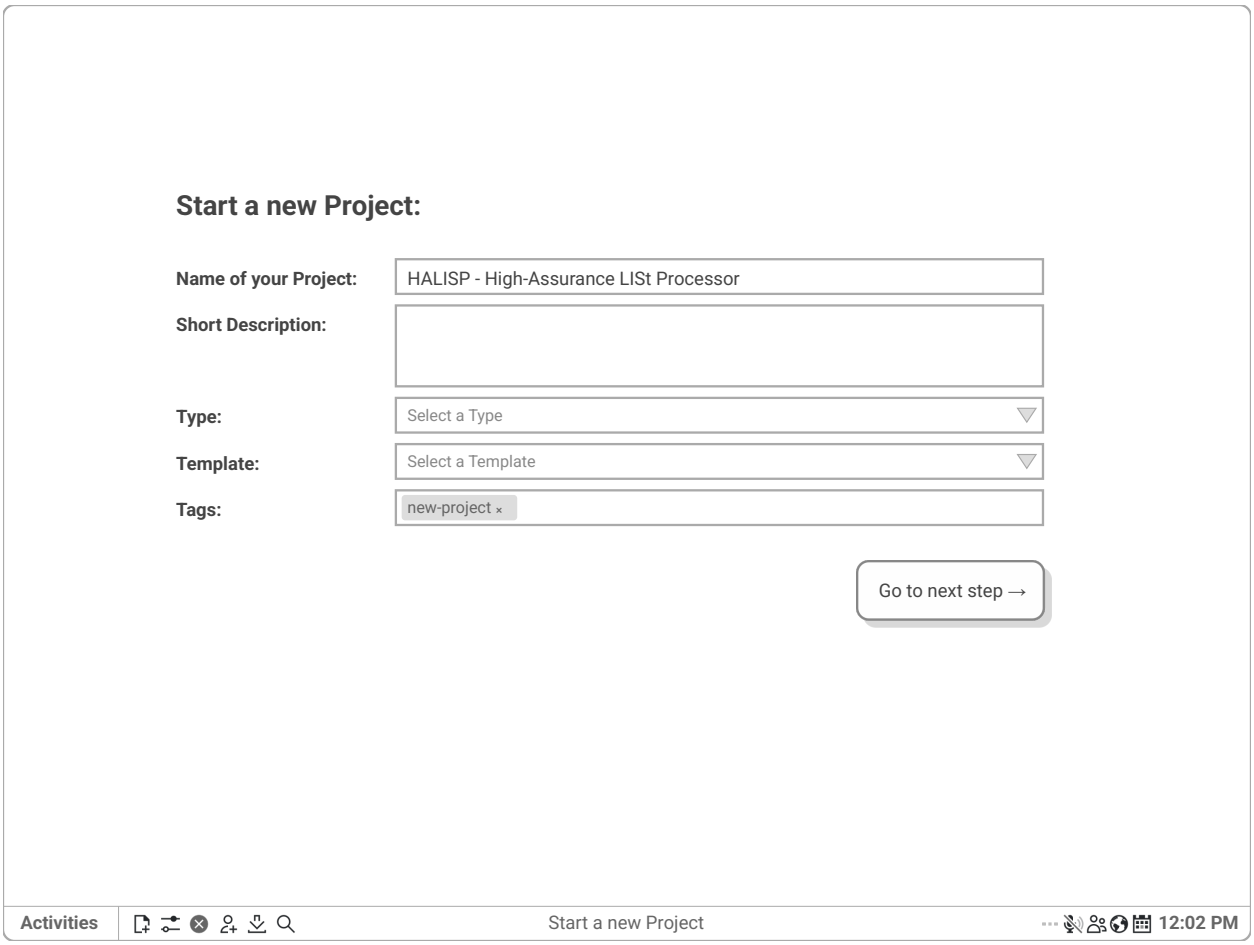



**Figure 5.3:** Starting a new project — Step 1.

This bottom "Activities" bar is of course able to be hidden, but is shown here for additional context.

The appliance 'knows' what a stakeholder or system requirement is, i.e. its metamodels and ontologies already contain the definitions so the operator does not need to configure it how to represent or use requirements. In Figure 5.6 we see a familiar tabular view, however, the appliance is applying generic "Tabular Object Editor" view to the set of requirement objects it was told to display. Here, the INCOSE-derived SysML meta-model from Chapter 4 applies. The title in the bottom bar indicates the activity is an instance of the Development activity type. A ubiquitous, seamless, and trustworthy audit log is available to every timestamped operation performed by the identified operator. The last pane in this full-screen activity shows the Document Examiner, here displaying fluidly-formatted content from the *How to*

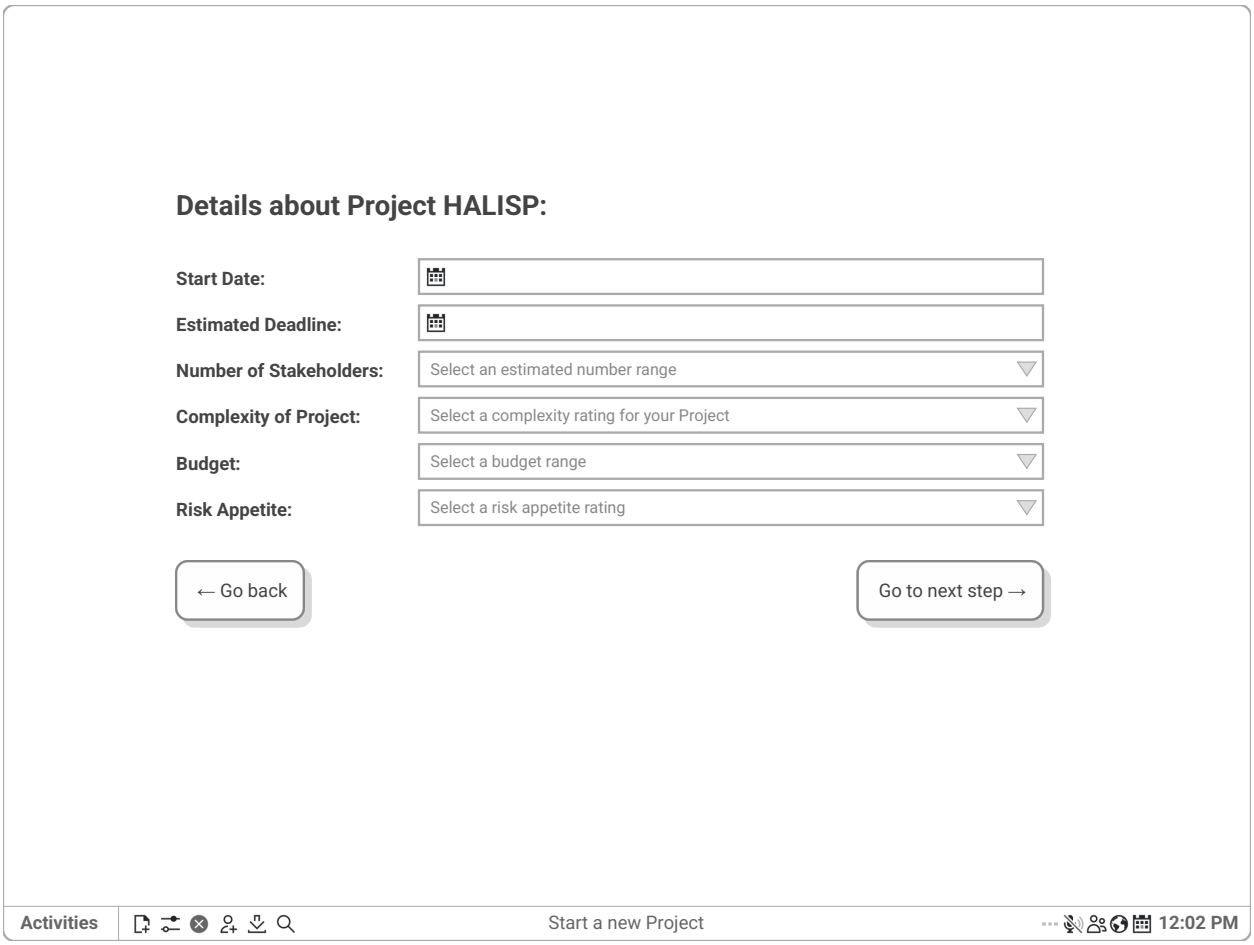



**Figure 5.4:** Starting a new project — Step 2.

*Write Requirements* document, providing important contextual help for the operator's chosen and current activity. Although no requirements are shown in the table here, the operator knows by developed intuition that any object displayed is interactive, navigable, and able to be explored in the library [28].

By navigating to the Document Examiner (Figure 5.7), the operator does not lose the state of the prior activity, and may return to it using a simple gesture or input device shortcut. Here the operator is viewing a section of a library document alongside an object graph rendering showing how sections of the document are linked to each other. This object graph view is available for any object in the system. The document also provides bookmarks linking to the sections. Again, the operator does not fear of losing their place in the document by navigating to different sections, because infinite undo is ubiquitous in the seamless DE

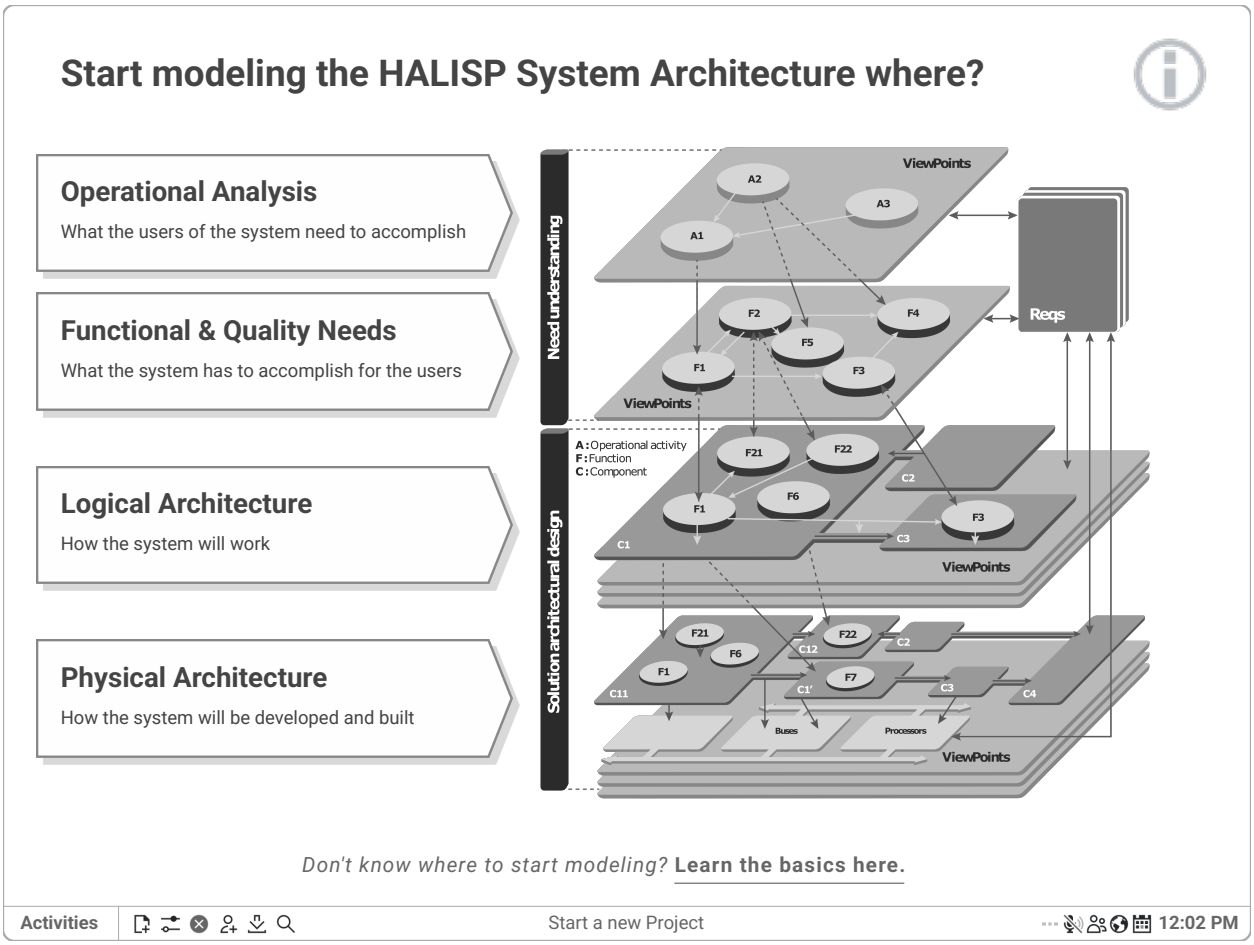



**Figure 5.5:** Starting a new project — Step 3 (graphic adapted from [187]).

appliance. The operator's document annotations are stored and displayed in a separate pane when desired.

Rather than documents in the library duplicating information and figures from other documents, or knols (knowledge elements), transclusion [189] is used to insert the object into the current document by reference (Figure 5.8). This ubiquitous and seamless functionality enables the inclusion, or rather transclusion, of international standards information into a document referencing that information. Here a figure from the ISO/IEC/IEEE 29148 [140] standard is transcluded, providing authoritative information to the operator when and where they need it, without disrupting the current activity. Bibliographic citations are also seamless, and metadata is preserved and accessible. Here we also the object graph pane rendering the graph as a familiar document outline.

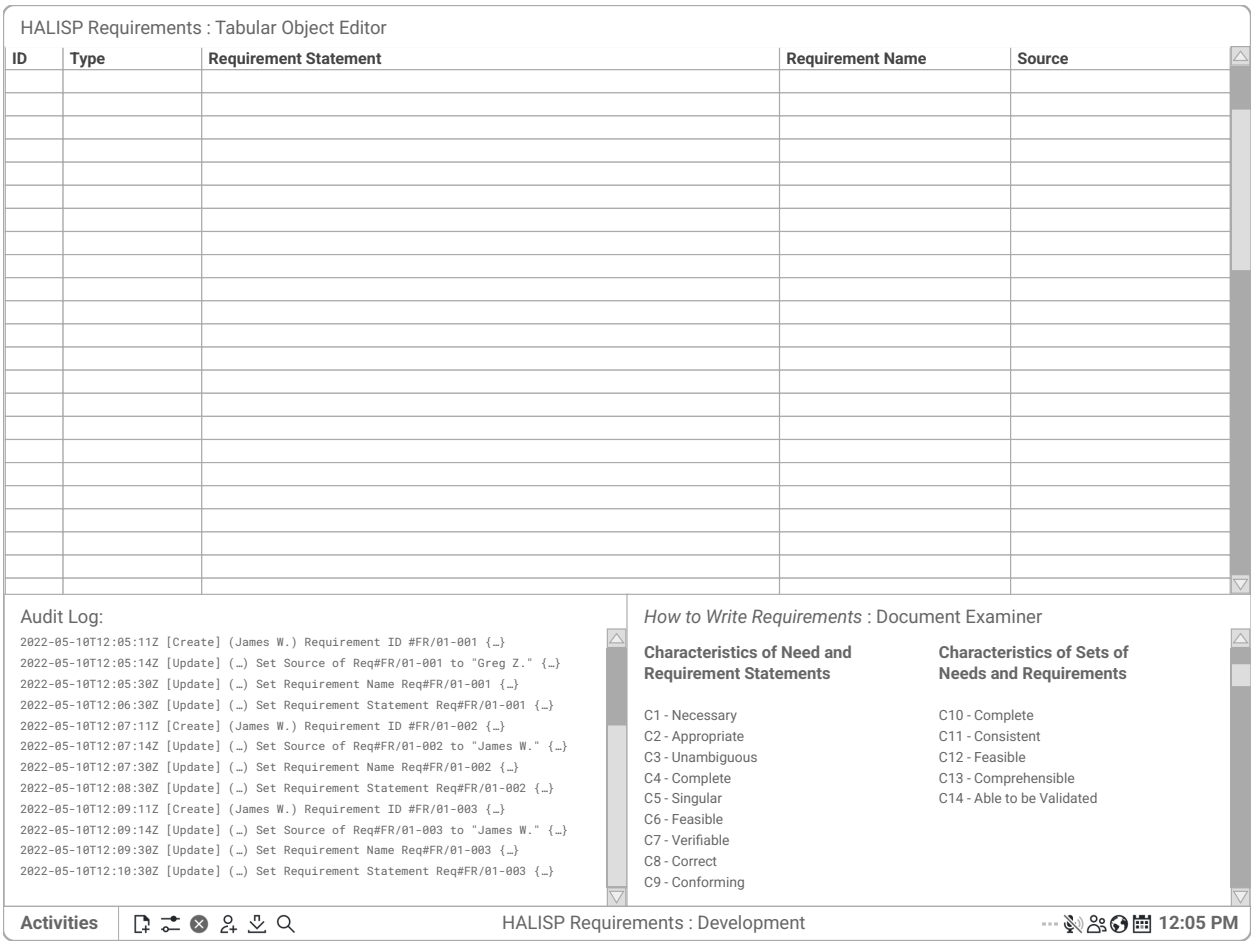



**Figure 5.6:** Writing requirements in a tabular editor.

The operator is now in the midst of specifying their system, and is developing a suitable view conforming to a viewpoint [104] for their stakeholders (Figure 5.9). The appliance always provides human- and machine-readable textual syntax for corresponding graphical elements, allowing the operator to fine-tune and automate operations that manipulate graphical elements. On the top-left and right of Figure 5.9 is the same object, represented as validated declarative text and familiar boxes-and-connectors. Any operation the operator performs on the graphical or textual representation is reflected in the other immediately, because it is the same object being manipulated and the appliance works with live objects. As before, any text or object shown here is interactive and navigable.

Figure 5.9 also shows an interactive session where the operator can perform non-destructive tests and exploratory operations, or programmatically commit versioned changes using the

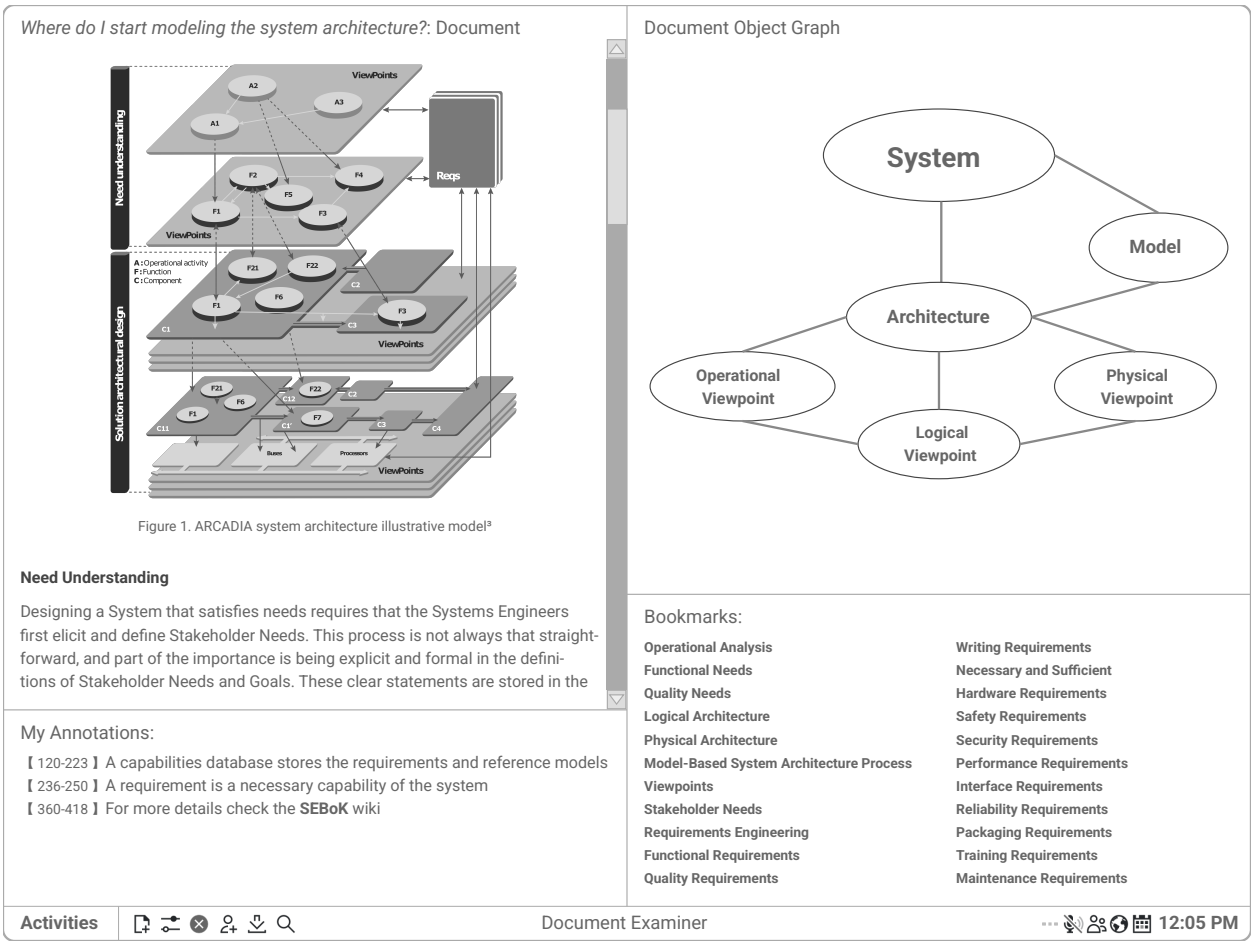



**Figure 5.7:** Researching how to model system architectures.

Seamless DE Meta-language. Although the syntax shown is LISP-like, other surface syntaxes may be available, depending on the domain-specific language (DSL) being used. Contextual actions are available in the bottom-right pane, giving the operator easy access to common actions and offering the operator a visual indicator of what is possible without having to remember the function's existence before using it.

Again, in Figure 5.10 we see the bidirectional editing of textual and graphical representations, as well as the ubiquitous availability of the editor and graph renderer. In addition to the interactive session pane described above, a group chat is afforded to the operator. With Activity-Based Computing (ABC) and the Seamless DE Appliance, collaboration is built-in. There is no need for the operator to exit the current activity to start collaborating with colleagues. The human collaborator sees the activity screen, except where the operator has

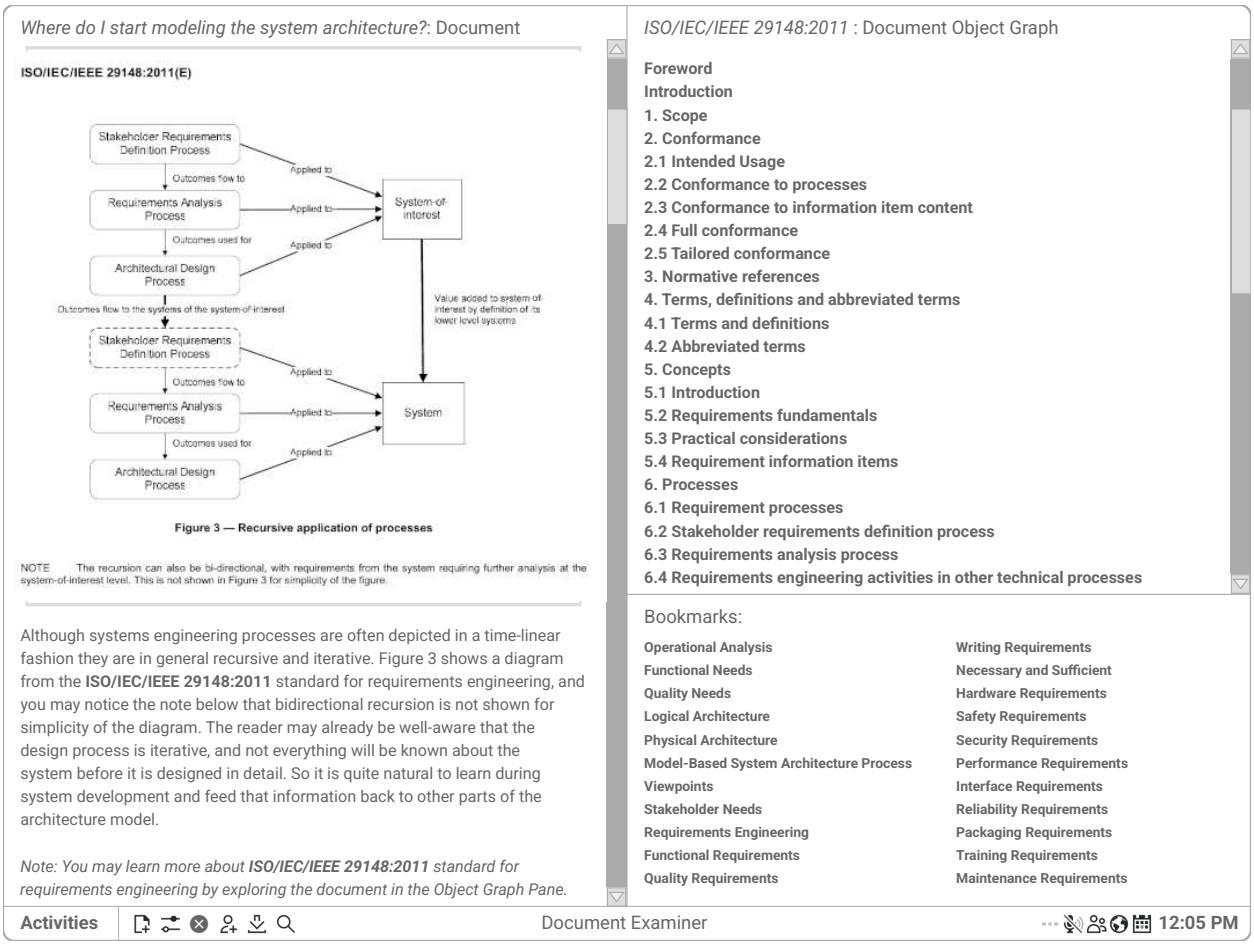



**Figure 5.8:** Using transclusion in a document.

prohibited visibility of sensitive information tracked by the appliance. The operator may also give temporary, measured control of the input devices to the collaborator when necessary to receive assistance in the current activity. This kind of collaboration is seamless, without the usual dance of "can you see my screen?" and "can you hear me now?"

The above HCI mock-ups and operational concept descriptions highlight areas where the Seamless DE Appliance noticeably differs from existing DE environments. Project management, architecture modeling, declarative and functional programming, knowledgebase exploration, and collaboration were presented as indicative of common activities in digital engineering. Other multiphysics modeling, analysis, and simulation would operate in much the same way, for example. The graph-based knowledgebase keeps track of all the objects in the appliance, and is therefore natively capable of assembling the Authoritative Source of

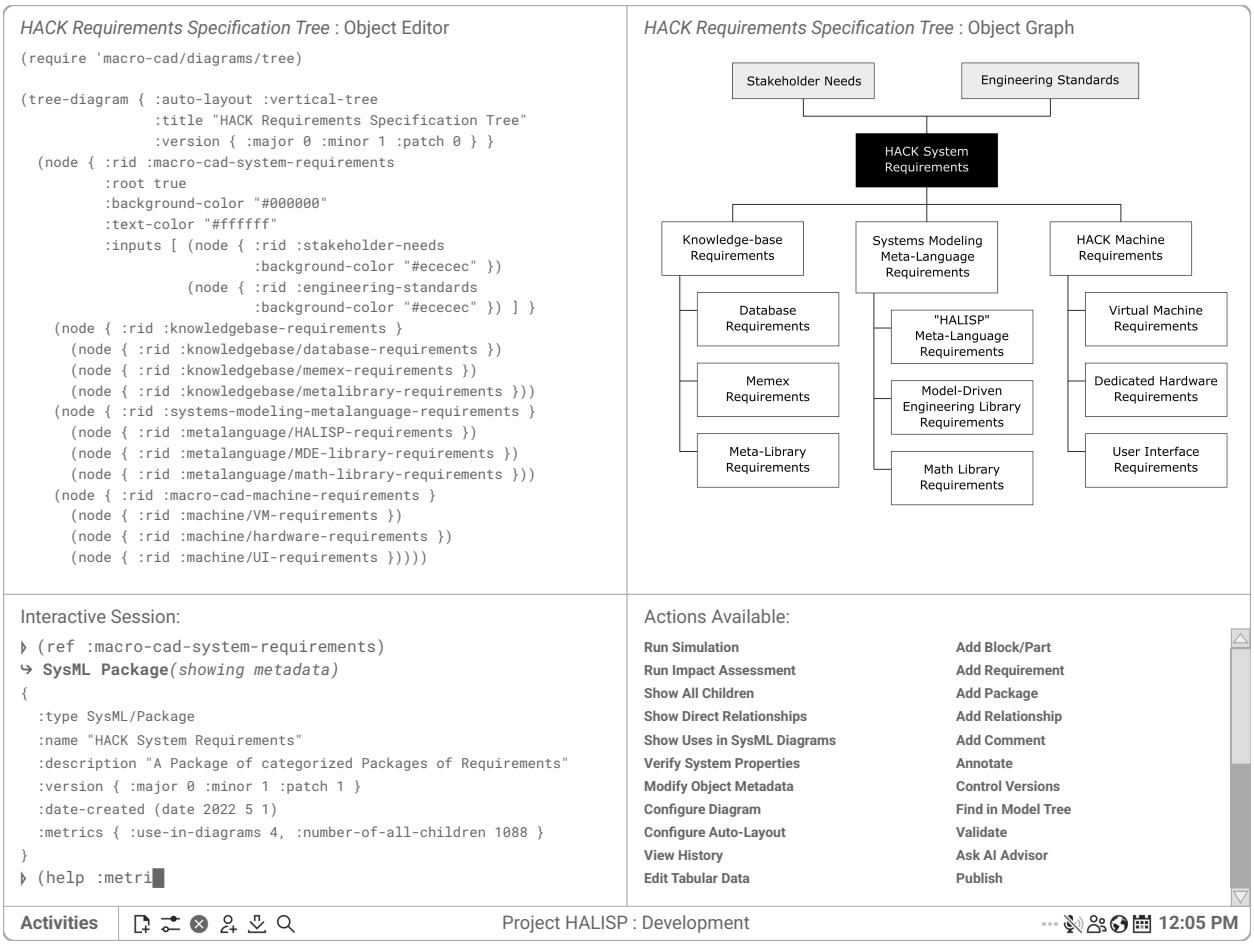



**Figure 5.9:** Textual/visual bidirectional editing.

Truth (ASoT) with full change control and audit capabilities. Therefore, the common DE tasks and artifacts fall out of the elegant design of the system, without the need for expensive integration, operator training, and remediation of errors and defects.

## 5.5 Needs Analysis

As systems engineers we are typically aware of the many expressed and adumbrated needs of stakeholders, and are ourselves stakeholders of the digital engineering system (DES). In principle, DE covers all the engineering activities needed to design, manufacture, deploy, and sustain engineered systems. These activities may include those which inform and enhance engineering activities such as business and mission analysis, project portfolio management,

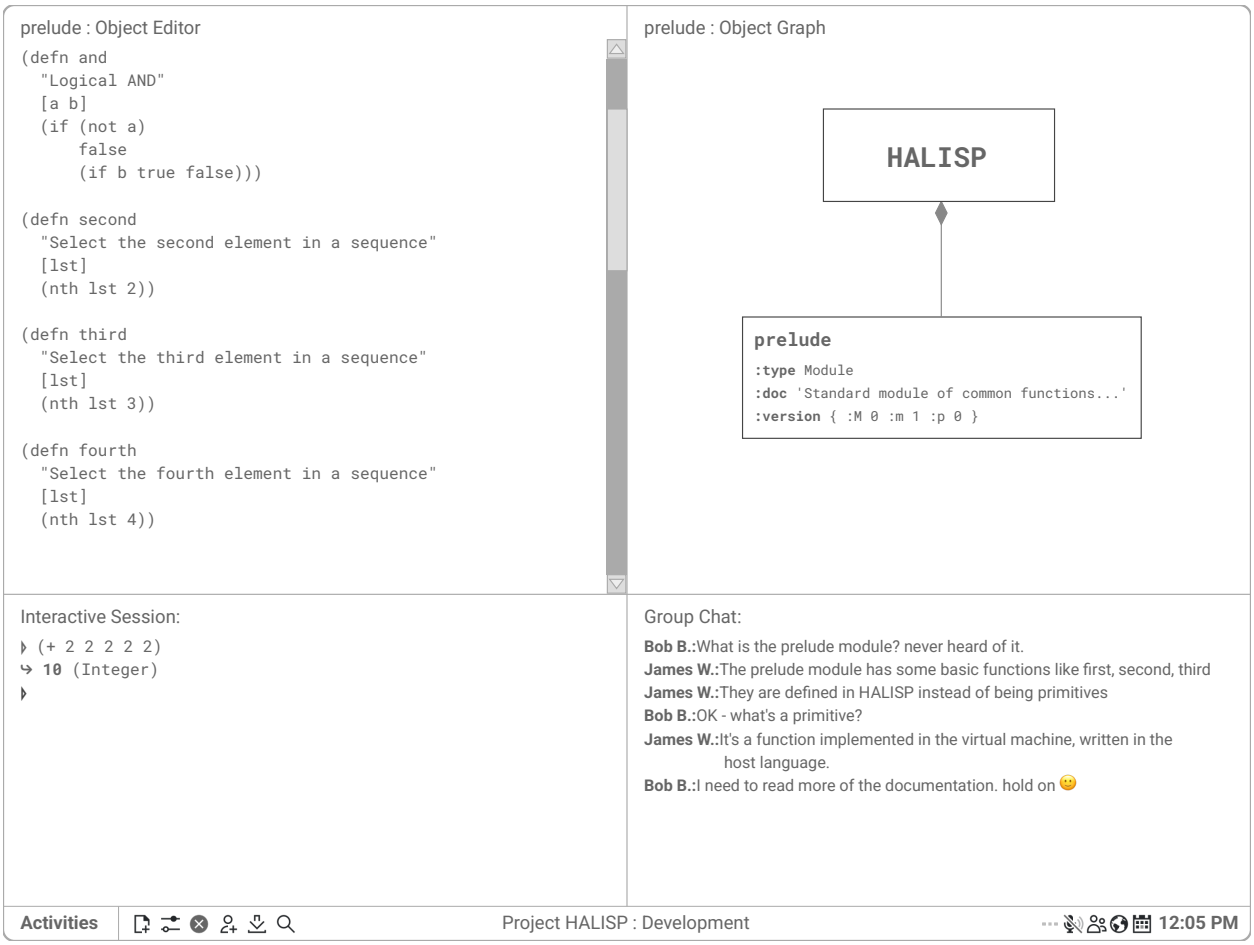



**Figure 5.10:** Unified Engineering Environment with collaboration.

knowledge management, and others as described in Refs. [74, 91]. Although most engineered systems are developed by teams large and small, history teaches us that usable and worthwhile systems are also sometimes developed by a single dedicated person. Therefore the Seamless DE Appliance shall support both individual and team-based engineering activities, without locking in the operator to workflows only available to larger enterprises typically associated with DE. Based on experience and understanding of similar systems, we can begin the needs analysis of Seamless DE, keeping in mind that stakeholder needs and requirements must iteratively undergo V&V as described in Refs. [116, 117]. Stakeholder requirements appear above or below their corresponding explanatory passages.

### 5.5.1 Essential Engineering Capabilities

As a digital engineering system (DES), the appliance must support the wide range of engineering activities as integrated model-based capabilities. These activities include those defined in many international standards such as ISO/IEC/IEEE 15288 [91], ISO/IEC/IEEE 29148 [140], and SAE ANSI/EIA-649C [190]. Such standards provide essential activity definitions useful for top-down needs analysis, supporting further requirements analysis activities that derive requirements targeting particular engineering capabilities. Some essential stakeholder requirements for engineering are summarized below in list form.

**REQUIREMENTS:**

The Seamless DE Appliance shall provide capabilities for systems engineering.

- The Seamless DE Appliance shall provide Systems Modeling Language (OMG SysML v2) as an embedded domain-specific language.
- The Seamless DE Appliance shall provide language support for modeling security countermeasures (cf. SysML-Sec [191, 192]).
- The Seamless DE Appliance shall provide Risk Analysis and Assessment Modeling Language (OMG RAAML) as an embedded domain-specific language.
- The Seamless DE Appliance shall provide Soft-goal Interdependency Graphs as an embedded domain-specific language.
- The Seamless DE Appliance shall provide decision support capabilities.

The Seamless DE Appliance shall provide capabilities for project and program management.

- The Seamless DE Appliance shall provide a project modeling language as an embedded domain-specific language.
- The Seamless DE Appliance shall provide the capability to perform Performance Evaluation and Review Technique (PERT) analysis on a system project model.
- The Seamless DE Appliance shall provide the capability to produce Gantt charts of a system project model.

- The Seamless DE Appliance shall provide the capability to produce a Work Breakdown Structure chart of a system project model.

The Seamless DE Appliance shall provide capabilities for configuration management.

- The Seamless DE Appliance shall provide model version-control capabilities.
- The Seamless DE Appliance shall provide configuration management workflows.

The Seamless DE Appliance shall provide requirements engineering capabilities.

- The Seamless DE Appliance shall manage requirements.
- The Seamless DE Appliance shall capture requirements trace.
- The Seamless DE Appliance shall generate requirements metrics.
- The Seamless DE Appliance shall manage requirements changes.
- The Seamless DE Appliance shall provide change-impact analysis.
- The Seamless DE Appliance shall control access to requirements changes.
- The Seamless DE Appliance shall enable reuse of requirements.

The Seamless DE Appliance shall provide specialty discipline engineering capabilities.

- The Seamless DE Appliance shall provide electrical engineering capabilities.
- The Seamless DE Appliance shall provide mechanical engineering capabilities.
- The Seamless DE Appliance shall provide chemical engineering capabilities.
- The Seamless DE Appliance shall provide industrial engineering capabilities.
- The Seamless DE Appliance shall provide physical sciences capabilities.
- The Seamless DE Appliance shall provide mathematical capabilities.

### 5.5.2 Knowledgebase System

Every software engineer and user of technical computing software can relate to the dreadful feeling of searching the Web for questions and answers regarding a specific and sometimes unique problem. Internet access is often assumed such that operating manuals are no longer delivered with computers. Hypertext and hypermedia once held the promise of navigable knowledge elements following the operator's intention and curiosity. Today, documentation

is frequently out-of-date and incomplete, owing in part due to its definition being separate from the artifacts being documented. Complex graphs of knowledge may attend a system model so stakeholders of various perspectives and generations may understand it. Ontologies have been identified as a critical aid for communication in MBSE [71, 92], wherein concepts are formally defined and explicitly related in a graph that may be referenced anywhere in the ASoT.

Scientific computing notebooks have provided researchers partially-reproducible and partially-interactive documents with the embedded software code used for their scientific calculations. This type of distribution format has been a boon to scientific reproducibility [193]. Conceivably, textbooks may also be authored in this way so students and researchers can interact with scientific and mathematical models directly or transclude them in their own documents with embedded citations. Transclusion is a concept discussed in early hypertext and hypermedia research that specifies how hypermedia objects can be included in documents without copying their contents [189]. Linked objects may be updated, and then those updates will reflect in other objects or documents that transclude the updated object. Transcluded objects may be browsed so that additional context and information can be found without losing the path taken [194], similar to how HTML pages are linked together and navigated by the Web browser.

The Seamless DE Appliance shall provide all the documentation necessary to operate it, without network access. Systems engineering standard documents such as the Concept of Operations, Operational Concept, and System Requirements Document, among other specifications and architecture frameworks, shall be provided to the operator with embedded support (tutorial or wizard) for guiding the operator in tailoring and using the document template for their own systems. Reports and documents may be seamlessly generated from models with the help of semantic document standards [195, 196]. Templates may be modified and versioned without losing access to original revisions. And since it is a tool for DE, it shall provide textbook material, executable formulas, and interactive diagrams and charts as

reference and learning material for the system operators (Figure 5.7). Many textbooks and reference materials are now published under open licenses, thereby easing their inclusion into the knowledgebase. The Web-based wikis of today cannot compare to the power achievable by a fully object- and model-based approach with a unified HCI.

**REQUIREMENTS:**

The Seamless DE Appliance provide a hypermedia library.

- The Seamless DE Appliance shall provide a standards-based ontology.
- The Seamless DE Appliance shall provide a mathematics hypermedia library.
- The Seamless DE Appliance shall provide a sciences hypermedia library.
- The Seamless DE Appliance shall provide an engineering hypermedia library.
- The Seamless DE Appliance shall provide a hypermedia library editor.
- The Seamless DE Appliance shall provide the capability to link any model element to a hypermedia knowledge-element.
- The Seamless DE Appliance shall provide engineering document templates.
- The Seamless DE Appliance shall provide a dictionary.
- The Seamless DE Appliance shall provide international standards documents as accessible hypermedia.

The Seamless DE Appliance shall provide a content publisher for producing various model artifacts for use by other systems.

- The Seamless DE Appliance shall produce beautifully[8]-typeset mathematical and scientific documents.
- The Seamless DE Appliance shall produce data visualizations (charts, plots, graphs, tables, etc.).

[8]LaTeX documents are widely considered to be beautiful, especially for scientific and mathematical documents where it has enjoyed long use by the academic community. This quality is not necessarily subjective, as there has been much development time invested in tuning the typesetting algorithms that adjust spacing between glyphs, lines, paragraphs, boxes, etc. [197, 198], while adopting more traditional typesetting techniques such as ligatures. In particular, mathematical notation relies on intricate layouts not found in typical prose.

- The Seamless DE Appliance shall produce diagrams that are semantically-valid for their respective domain-specific language.
- The Seamless DE Appliance shall produce accessible documentation of the system-under-development.
- The Seamless DE Appliance shall provide documentation of the current activity functions in less than or equal to two HCI operations.
- The Seamless DE Appliance shall provide a spellchecker.
- The Seamless DE Appliance shall provide a grammar checker.
- The Seamless DE Appliance shall produce hypermedia documents that are linked to the system model and metamodel for interactive use, e.g. code snippets, simulate-able system model diagrams, data visualizations, and mathematical calculations.
- The hypermedia documents shall support object transclusion.

### 5.5.3 Parallel, Concurrency, and Real-Time

The central processing unit (CPU) is a historical and practical bottleneck [199] to computing architectures. A CPU is difficult to secure due to the sharing of caches and buffers across otherwise independent, isolated computing processes [200]. Recent cybersecurity research has revealed multiple microarchitectural flaws related to speculative execution, a method for improving the execution speed of sequential programs, such as Meltdown [52] and Spectre [53] that cannot be patched in microcode or software.

Interaction Nets [201] and Propagation Networks [202] may be appropriate theories of computation that support concurrency and parallelism, and deserve further research and trade studies in the context of this reference architecture. Likewise, the transputer [203] is a historical architecture worth studying in light of recent advancements in semiconductor manufacturing. The Reduceron [204] is an alternative to a traditional CPU by being specially designed to execute functional programs.

The Seamless DE Appliance shall be capable of specifying real-time systems — itself a real-time system. Today, this capability usually requires special tools and knowledge unknown to many software engineers. However, there have been some high-profile successes, such as Amazon's use of Temporal Logic of Actions (TLA$^+$) [205]. More broadly, the Modeling and Analysis of Real Time and Embedded systems (MARTE) profile [206–208] and Timed Abstract State Machines [209] have seen adoption by MBSE practitioners. The Seamless DE Appliance shall use hardware architecture designed for parallelism and concurrency to avoid the von Neumann bottleneck [199].

**REQUIREMENT:**

The Seamless DE Appliance shall use a parallel and concurrent computing architecture.

### 5.5.4 Next-Generation Networking

The Internet is showing its age and its high complexity with some architectural and security flaws that are difficult-to-impossible to patch [210–213]. Research efforts for a next-generation Internet have been underway, notable Named-Data Networking (NDN) [214] supported by the National Science Foundation (NSF), and the Scalability, Control, and Isolation On Next-Generation Networks (SCION) architecture [215]. As with other advanced research, their benefits are best realized by integration into a cohesive system.

Alternative Internet architectures may still rely on cryptographic certificates issued by central authorities, however, introducing a vulnerable central point of failure. Mesh networks, which use a distributed topology, are designed to sustain damage to network nodes, and spread the usage load across the network while making data widely available. The Seamless DE Appliance shall use mesh networking to satisfy the design principle of conviviality [186] and to support the rapid configuration of private and public networks in support of collaborative engineering activities.

**REQUIREMENT:**

The Seamless DE Appliance shall support next-generation, secure mesh networking topologies and technologies.

### 5.5.5 Hardware Acceleration

In contrast to the aging CPU-focused computing paradigm, we now have access to hardware accelerators such as graphics processing units (GPUs) and field-programmable gate arrays (FPGAs), as well as application-specific integrated circuits (ASICs) for digital signal processing and machine learning. So, in addition to being focused on parallel- and concurrent-computing, the Seamless DE Appliance shall use computing hardware to accelerate common workload processing. The Seamless DE Appliance shall send common workloads to FPGAs on-the-fly, without involving FPGA engineers and another workflow. While we are familiar with GPUs for accelerating video and artificial intelligence workloads, text rendering is ubiquitous and necessary, and yet currently lacks universal hardware support: a text-processing ASIC shall obviate millions of lines of code and redundant software libraries.

Cryptography is now ubiquitous, but its implementation often leads to security vulnerabilities, and some can be devastating [216]. The Seamless DE Appliance shall provide formally-verified hardware-accelerated cryptography capabilities. However, translators of high-level code defined using the Seamless DE Meta-language to hardware accelerator microcode shall be formally-verified such that the translated output can be proven to conform to its high-level specification. Lastly, hardware acceleration for Satisfiability Modulo Theories (SMT) solving shall used for efficient formal verification activities. Other hardware acceleration capabilities are conceivable and desirable for common processing tasks.

**REQUIREMENTS:**

The Seamless DE Appliance shall provide hardware acceleration for common system operations.

- The Seamless DE Appliance shall provide hardware acceleration for text rendering.

- The Seamless DE Appliance shall provide hardware acceleration for common cryptographic functions.
- The Seamless DE Appliance shall provide hardware acceleration for matrix calculations.
- The Seamless DE Appliance shall provide hardware acceleration for vector calculations.
- The Seamless DE Appliance shall provide hardware acceleration for tensor calculations.
- The Seamless DE Appliance shall provide hardware acceleration for Satisfiability Modulo Theories (SMT) solvers.
- The Seamless DE Appliance shall provide field-programmable gate arrays (FPGAs) that are transparently reprogrammed according to activity-specific loads.

### 5.5.6 Digital Identity and Access Control

Many digital identity practices still in use today have been rendered obsolete by National Institute of Standards and Technology (NIST) Digital Identity Guidelines [217]. Passwords, or at least password hashes, are routinely stolen from corporate databases and sold in black markets along with other personally identifiable information (PII). In addition, social engineering attacks can target personal questions whose answers can be used to recover accounts illegally. A clean-slate computing system has the opportunity to learn from security decisions made decades ago when devices everywhere were not Internet-connected. In particular, the operator of the Seamless DE Appliance shall use a HSM that securely holds private keys in a tamper-proof enclosure.

Object-capabilities (OCAPs) are a powerful alternative [218] to problematic access-control list (ACL) and role-based access control (RBAC) based systems. They provide granular control over access to system objects and functions, including over the network where objects may reside on remote machines. Object-capabilities (OCAPs) provide the best affordances when designed into the system architecture rather than being added later. Object-capabilities (OCAPs) thus provide the needed security for enterprise- and government-grade engineering

programs that may nevertheless need to share certain versioned and redacted information to third parties such as suppliers and contractors.

**REQUIREMENTS:**

- The Seamless DE Appliance shall provide object-capabilities (OCAPs).
- The Seamless DE Appliance shall conform to NIST Special Publication 800-63-3 Digital Identity Guidelines. [217]
- The Seamless DE Appliance shall provide a removable hardware security module (HSM) for the operator.

### 5.5.7 Multilingual and Accessible

The use of engineering jargon in system documentation can be difficult to curb without clear guidelines and automated checkers. Simplified Technical English [145] is already widely used in the aviation industry, where system model artifacts are shared among people of various nationalities and native languages and of varying proficiency levels in written English. The Seamless DE Appliance shall include the hypermedia specification and automated checkers for ASD-STE100 [145] so that system designers can quickly validate their documented models against the standard. This facility aids in the development of systems that can be shared, modified, used, and reused throughout the world.

As with the above multi-lingual requirement, a newly-designed computing system must support disabled or impaired operators. Specially-designed HCI devices, elements, and functions appropriate for the disabled have existed for decades. The Seamless DE Appliance incorporates these requirements to ensure they fit into system architecture and benefit from the system synergies.

**REQUIREMENTS:**

The Seamless DE Appliance shall be translate-able to any written human language.

The Seamless DE Appliance shall be accessible to operators with disabilities.

- The Seamless DE Appliance shall provide text-to-speech assistance of all HCI elements to a vision-impaired operator.
- The Seamless DE Appliance shall provide keyboard navigational assistance of all HCI elements to a vision-impaired operator.
- The Seamless DE Appliance shall provide speech-to-text assistance of all sound-enabled HCI elements to a hearing-impaired operator.
- The Seamless DE Appliance shall provide cognitive assistance to a mentally-impaired operator.
- The Seamless DE Appliance shall provide alternative HCI interfaces to a physically-impaired operator.

### 5.5.8 Performance Profiling and Run-Time Analysis

Today's software-intensive systems must be explicitly (re-)engineered with the attendant subsystems and tools to be profiled for performance characteristics. It is unacceptable to deploy a software component without having measured and understood how it will perform during typical and intensive use. The Seamless DE Appliance shall support run-time analysis of memory and compute usage that the operator can switch on when needed, aiding a comprehensive understanding of the system performance for later modeling, simulation, and analysis.

**REQUIREMENTS:**

The Seamless DE Appliance shall provide a tool for performance self-monitoring.

The Seamless DE Appliance shall provide a tool for performance monitoring of the system-of-interest.

- The Seamless DE Appliance shall provide a configuration switch for system performance instrumentation.
- The Seamless DE Appliance shall provide a configuration switch for usage analytics.

### 5.5.9 Integration with Existing Systems and Data

In order to deal with the plethora of data formats and protocols, a kind of universal translator component shall be developed. Being formally-verified, the component will convert data formats [219] into a native format and back again if necessary. Insecure protocols will not be supported. Likewise, other programming languages may be parsed, but their direct translation into the Seamless DE Meta-language is untenable. Such an intelligent capability may be useful to aid in the migration of software packages and algorithms.

As part of the transition period, it may be desirable to emulate software designed for extant x86/ARM/RISC-V or other ISAs. However, as Spectre-related microarchitecture vulnerabilities have shown, these CPU architectures cannot be safely used outside of a strict quarantine or sandbox. Such a sandbox must be hardware-enforced. However, the desirability of such a capability is worth further study before effort is expended on its design and implementation.

**REQUIREMENT:**

The Seamless DE Appliance shall interoperate with qualified external cyber-systems.

### 5.5.10 Units and Dimensional Analysis

Engineers must use quantities and dimensional analysis to perform analyses and design systems. Strangely, programming languages often only provide a numerical tower and are unaware of units and quantities. This scoping decision leaves this facility to be implemented by programming language users and published as unverified independent libraries, software packages [220], or even a new special-purpose programming language [221]. However, if the Seamless DE Meta-language provided a verified library of units and conversions for dimensional analysis, it would be more appropriate for engineering activities. ISO 80000 [222] may be used as a SysML profile, and as a starting point, NIST Special Publications 811 [223] and 330 [224] provide advice on how to use metric units.

**REQUIREMENTS:**

- The Seamless DE Appliance shall provide the ISO 80000 library of units of measurement.
- The Seamless DE Appliance shall support type-safe dimensional analysis.
- The Seamless DE Appliance shall provide an arbitrary-precision numeric tower.
- The Seamless DE Appliance shall provide common scientific and mathematical numerical constants.
- The Seamless DE Appliance shall provide numerical domains that are formally-defined in hierarchical structure.

### 5.5.11 Adoption of Existing Standards

Not all international standards are created equal. Standards such as the C programming language (ISO/IEC 9899 [225]) and POSIX [66] were created *ex post facto*: attempting to write down and codify what already existed in the wild. Other standards were born of industry consortia to satisfy their needs of interoperability. In particular, standards such as USB and Bluetooth are so lengthy that they defy auditability. The modern Web, too, is a mess of complex standards that often duplicate functionality commonly available using other software libraries.

Many of these standards are unnecessary or obsolete under a new computing paradigm, as they work around design flaws and accrete complexity. Additionally, industry standards that are not accessible to the general public violate principles of conviviality and trustworthiness, preventing the operator from independently audit their system's conformance to those standards. The Seamless DE Appliance shall provide decision support models to assist in these complex design decisions.

**REQUIREMENT:**

The Seamless DE Appliance shall provide architectural decision support of standards adoption.

### 5.5.12 Open Governance

Governance of the Seamless DE Reference Architecture will become a significant challenge as the project grows. Existing governance models including those used in open-source software projects are unlikely to be sufficient, due to the specialized knowledge in systems engineering and DE needed by the governing stakeholders. Therefore, an open systems engineering practice shall be developed to meet the grand challenge, leveraging the capabilities of the prototype appliance as it is developed.

**REQUIREMENTS:**

- The Seamless DE Appliance architecture shall be governed transparently.
- The Seamless DE Appliance architecture shall be free and open-source forever.
- The Seamless DE Appliance shall be specified using Simplified Technical English (ASD-STE100).

## 5.6 Seamless Digital Engineering Meta-Language

**REQUIREMENT:**

The Seamless DE Appliance shall provide the Seamless DE Meta-language.

The STEELMAN high-level programming language requirements [226] are now over 40 years old, but many programming languages have ignored its explicit goals and requirements. Rather, programming languages are often born of the creator's personal needs and curiosities, who are generally computer scientists, not systems engineers. No programming language exists that has been originally designed using SysML and systems engineering processes. Additionally, programming language designs would greatly benefit from input from human factors engineering researchers who can properly assess the tradeoffs and affordances that affect software engineers. Programming languages are often considered in isolation, neglecting the suite of necessary tools that constitute a programming system [227] such as those for documentation, testing, profiling, and static analysis. The transdisciplinary systems

engineering approach used in Seamless DE necessitates a fresh outlook on programming systems design.

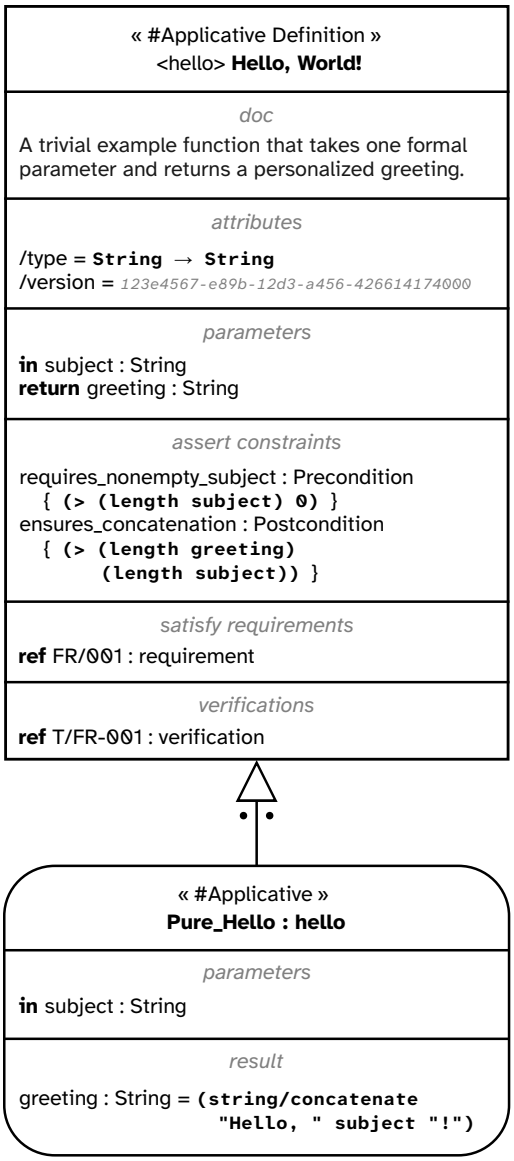
One major flaw with most programming language implementations is that not only are comments discarded, but the contextual information accumulated through parsing, semantics, and type checks is thrown away after each use. Comments are written by the engineer to help explain algorithms, naming choices, to clarify difficult sections, and often to document the functions and modules themselves. Software engineers commonly rely on external tools that parse source code separate from the interpreter/compiler and pull out structured data from comments for use in auto-documentation systems. These external documentation systems have their limitations, but perhaps most importantly, they are foreign to the language specification. Many software engineers believe the source code should be its own documentation. Since the interpreter/compiler, which is by definition the most sophisticated reader of the source code, discards that critical information, it therefore lacks the capability to bidirectionally link with other documentation or models.

There is no technical reason why a programming system cannot afford the engineer the ability to assign SysML and project model information to functions, variables, and modules. Although this information may not be used in the interpretation of the computer program, it nevertheless aids in its design and implementation. Therefore, the Seamless DE Meta-language embeds SysML v2 semantics into its specification facilities such that the SysML model need not exist separately from the software design it specifies (Figure 5.11). Further, as a meta-language with its own HCI (Section 5.4), model information may be contextually hidden, and source code need not be text line-based, instead everything being represented as Abstract Syntax Trees.

**REQUIREMENT:**

The Seamless DE Meta-language shall specify program units using SysML v2.

Rather than relying on disparate tools for formal verification, the Seamless DE Meta-language builds in these facilities for the best mathematical guarantees possible. As with the




```
;; SysML v2 "Calc Definition"
(define:applicative <hello> "Hello, World!"

  :doc    (document "A trivial example function
            that takes one formal parameter and
            returns a personalized greeting.")


  :type      [ String → String ]


  :in        [ subject ]
  :return    greeting

  ;; Hoare triple pre-condition
  :requires (> (length subject) 0)

  ;; Hoare triple post-condition
  :ensures   (> (length out)
                (length subject))

  :satisfies [ (ref :FR/001 :requirement) ]


  :verifications [ (ref :T/FR-001 :verification) ])
```

```
;; SysML v2 "Calc Usage"
(Pure_Hello : hello

  (string/concatenate
       "Hello, " subject "!"))
```

**Figure 5.11:** The correspondence between SysML v2 graphical syntax and LISP-like definitions of function specification and implementation.

'seamless' definitions from Section 3.4, seamless integration of formal verification capabilities not only supports their functional correctness and reliability, but also supports their efficiency use by the system operator. When the operator does not waste time installing, updating, and configuring tools, they can more easily focus on the work at hand, and thus supporting convivial mastery and efficiency. Since the appliance provides the knowledgebase which describes its efficient use, the knowledge gap of the proper application of formal verification techniques is more easily closed. The facilities that support formal verification also support formal logic and mathematics, and possibly support other domains such as systems engineering architecture.

**REQUIREMENTS:**

The Seamless DE Meta-language shall provide affordances for formal verification.

- The Seamless DE Appliance shall provide an automated theorem prover (ATP).
- The Seamless DE Appliance shall provide an interactive theorem prover (ITP).
- The Seamless DE Appliance shall provide a proof checker for archiving, verifying, and studying mathematical proofs.

Software compilation is popularly implemented as a critical process prior to shipping completed software packages. However, this process is designed to support efficient execution of the software, rather than safety for the operator. The difficulty and resource costs of the compilation and build processes further widens the gulf between programmers and operators, forcing the latter group to depend on the former group to satisfy their computing needs. In Seamless DE, the operator has the facilities already available to customize or design systems. Rather than being a convoluted, expensive, specialized process, compilation is transparently performed as a caching step for speeding up execution. The compilation process must itself be trustworthy and its outputs fully traceable to its inputs.

**REQUIREMENTS:**

The Seamless DE Meta-language Programs shall be transparently compiled to human- and machine-auditable microcode.

- The Seamless DE Appliance microcode shall provide back-references to Seamless DE Meta-Language code.
- The Seamless DE Appliance microcode shall carry proofs (proof-carrying) of correct algorithmic execution.

Figure 5.12 shows the high-level parts of the meta-language, including the knowledgebase with ontologies, engineering embedded domain-specific languages, standard library, and HCI, as also shown in Figure 5.13. Decomposition continues such that a model-based specification of the language exists to support standards development, change control, and evolution.

**REQUIREMENT:**

The Seamless DE Meta-language shall provide the capability of employing domain-specific languages.

The STEELMAN requirement 3A [226] states, "The language shall be strongly typed." While dynamic typing may facilitate prototyping, experimentation, brief studies, and personal automation, a strong type system is necessary for industrial-grade software. An expressive, strong type system coupled with computational logic(s) affords static analysis of the system design, ensuring preconditions, postconditions, and invariants hold where specified. There are many kinds of type systems, but more powerful systems capable of precisely expressing the problem domain and corresponding functional solution have recently been deployed in such general-purpose functional programming languages as FStar [228] and Idris [229, 230]. Dependent type systems afford detailed specification of higher-order state, side-effects, and input/output computations that reside at the boundary of the software system, known as effect types. When the core functionality of a software system is composed of pure (idempotent) functions that reliably transform data structures, the system is easier to analyze

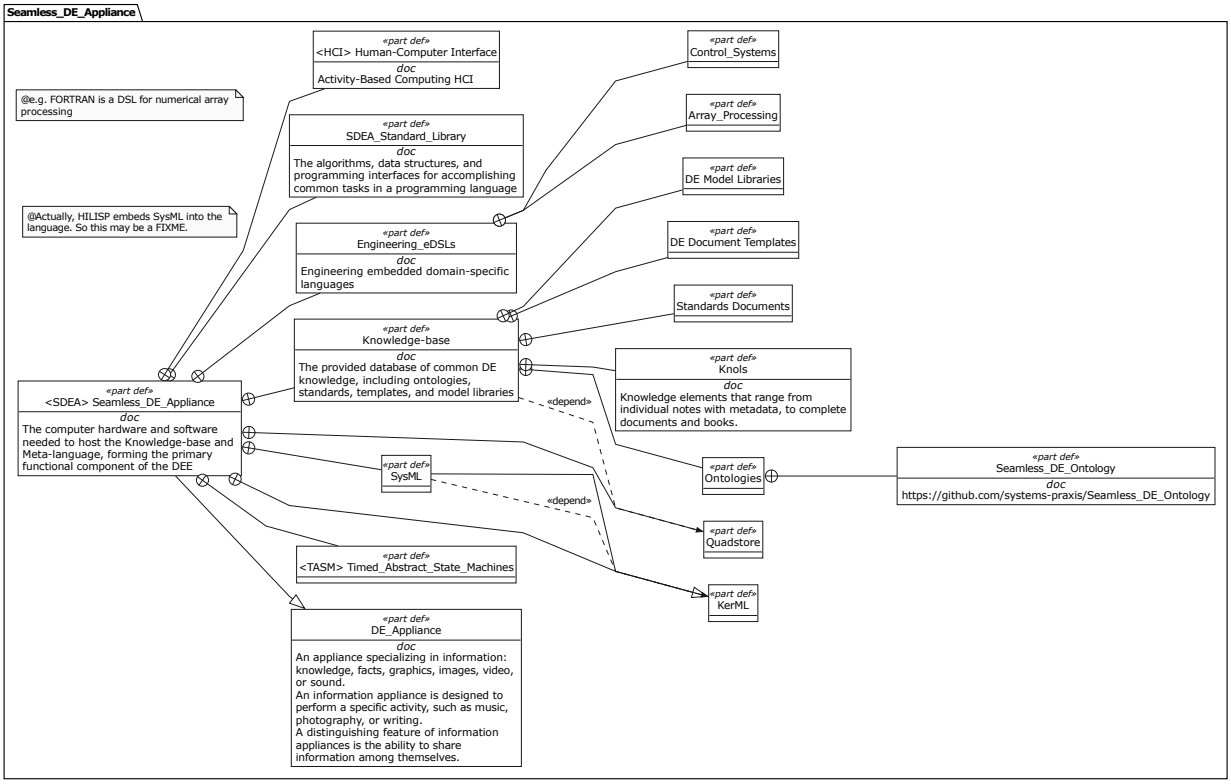



**Figure 5.12:** Major parts decomposition of the Seamless DE Appliance (cf. Figure 5.13).

and reason about. When used throughout a complex software-intensive system, the powerful type systems afford high reliability, safety, and expressive detail of specifications.

Dependent types allow type specifications to depend on values, affording the system designer a way to further specify functional behavior. The FStar language researchers have used dependent types with multi-monadic effect types and refinement types [231] to formally verify an industrial-grade cryptography system [232], a distributed system realizing Communicating Sequential Processes (CSP) concurrency [233], a high-performance key-value store [234], a cryptographic protocol parser [219], and a hardened memory allocator [235]. These use cases are known to be difficult to implement correctly due to the stringent security and safety requirements, but with formal verification, the engineers have confidence that the system will not fail or otherwise violate its security and safety guarantees. When certain verification conditions expressed in the type specifications were not automatically discharged by the Satisfiability Modulo Theories (SMT) solver, FStar researchers turned

to meta-programming for automation [236]. The latest computational type theories offer full computational interpretation [237] and are suitable for type-theoretic foundations of mathematics [238]. Strong type theories supply the system designer with sound, static verification methods that encodes the problem-solution domain and prevents errant states that may cause real-world harm.

**REQUIREMENTS:**

The Seamless DE Meta-language shall be strongly typed.

- The Seamless DE Meta-language type system shall provide gradual typing.
- The Seamless DE Meta-language type system shall provide dependent types.
- The Seamless DE Meta-language type system shall provide effect types.
- The Seamless DE Meta-language type system shall provide refinement types.
- The Seamless DE Meta-language type system shall provide abstract data types.
- The Seamless DE Meta-language type system shall provide subtypes.

## 5.7 Full-Source Bootstrap Strategy

The threat of computer backdoors is not to be taken lightly in DE, since they can circumvent stringent data confidentiality protocols. The Trusting Trust attack in particular affects software compilers and translators with a self-replicating Trojan horse, first published in [239] and then described in [240]. This attack was widely believed to be only a theoretical attack or even impossible to counter, and was largely ignored until Ref. [170] formally proved that with Diverse Double-Compiling, a compiler can be shown to some degree of confidence to be unaffected by the Trojan horse. A subsequent grassroots effort [241] has been developed to establish a full-source bootstrap of a practical computer system [242] to ensure other components are not affected by the Trusting Trust attack or other subversions. Therefore, the Seamless DE Appliance shall have a strictly verifiable and independently auditable bootstrap path.

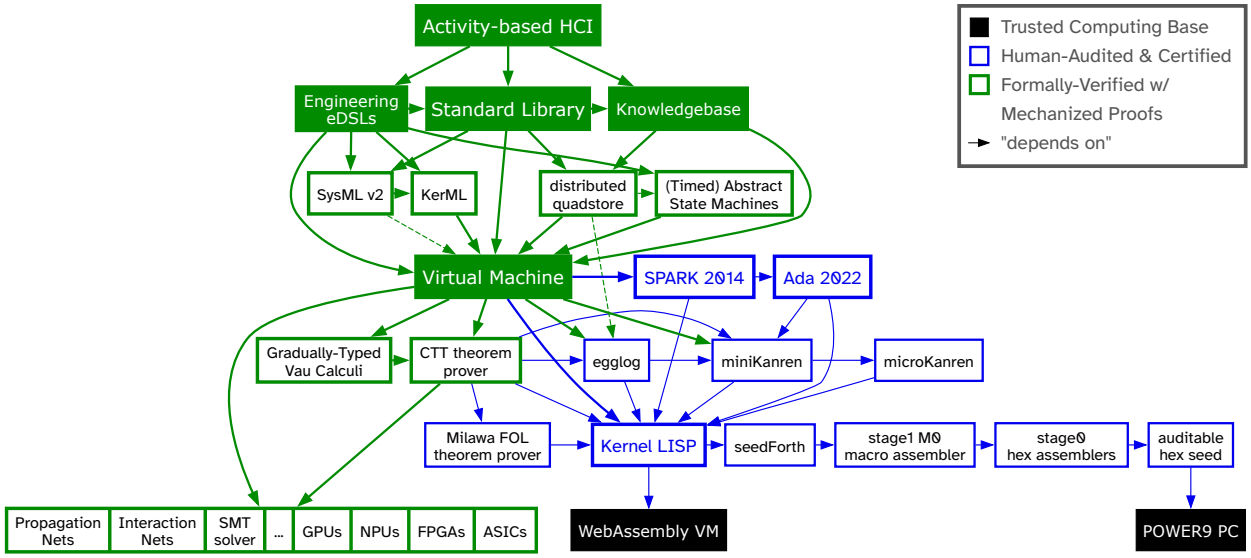



**Figure 5.13:** Reference full-source minimized bootstrap paths to Seamless DE Meta-language (Section 5.6) and HCI.

**REQUIREMENTS:**

The Seamless DE Appliance shall have a clearly-defined and human-auditable bootstrapping process.

- The Seamless DE Appliance shall be bootstrapped from a tiny (<512-byte) binary seed.
- The Seamless DE Appliance shall be bootstrapped from a self-verifying (modulo the social proof) theorem prover.
- The Seamless DE Appliance shall document its bootstrapping process and required knowledge for total reproduction by an unknown party displaced in space and time.

Figure 5.13 depicts the high-level full-source bootstrap strategy, including multiple bootstrap paths to further reduce the risk of Trusting Trust attacks. Alternate bootstrap paths also ensure that the target language remains implementable according to its specification. Diversity of implementations is one of the main factors described by Ref. [170] that fully counters Trusting Trust. Modeling of these bootstrap paths was included in the Seamless DE Reference Architecture defined in SysML v2, and shown in Figures 5.14, 5.15, 5.16, 5.17, 5.18, 5.19, 5.20, 5.21, and 5.22.

The languages used for each bootstrap component were modeled as requirement sets, and their translator implementations modeled as parts that satisfy those requirements. The bootstrap paths were organized using SysML packages named after the traditional numerical stages. The way a bootstrap part is shown to depend on the implementation of a lower part is with the SysML dependency relation. Separate items representing the source files are not shown in the following figures. Figure 5.14 shows the overall bootstrap paths of the Seamless DE Appliance, and its detail is shown in the subsequent figures.

The first bootstrap path (Figure 5.15) is based on CPU-architecture computers, which must execute only machine code specified by their corresponding instruction set architecture (ISA). This bootstrap path may benefit from existing full-source bootstraps that we can execute on the computers we already have. It consists of four stages before reaching the stage of implementing the Seamless DE Meta-language.

Starting with a POWER9-based computer (Figure 5.16) at stage0, we benefit from its open-source firmware that affords a lower-level audit compared to most commodity computers. With a suitable input device and display, the most minimally useful program may be manually entered as machine code using a paper reference, for example. This program affords the operator the ability to input programs using ASCII-encoded hexadecimal values of machine instructions and data.

The next stage (Figure 5.17) builds successively more powerful assemblers with the final capability of programming with ISA mnemonics as in common assembly languages. Each part is as small as possible to afford auditability, while providing the most functionality at that sub-stage to be useful enough to implement the next sub-stage.

Stage2 (Figure 5.18) of the bootstrap path depicted in Figure 5.15 shows how we can construct successively more powerful Forth interpreters. The Forth stack programming language, although unsafe, can be implemented in a very small footprint, and then readily bootstrap the rest of itself. This small footprint and its expressive power make it a suitable bootstrap language. Being able to bootstrap to standard ANS Forth [243] means we benefit

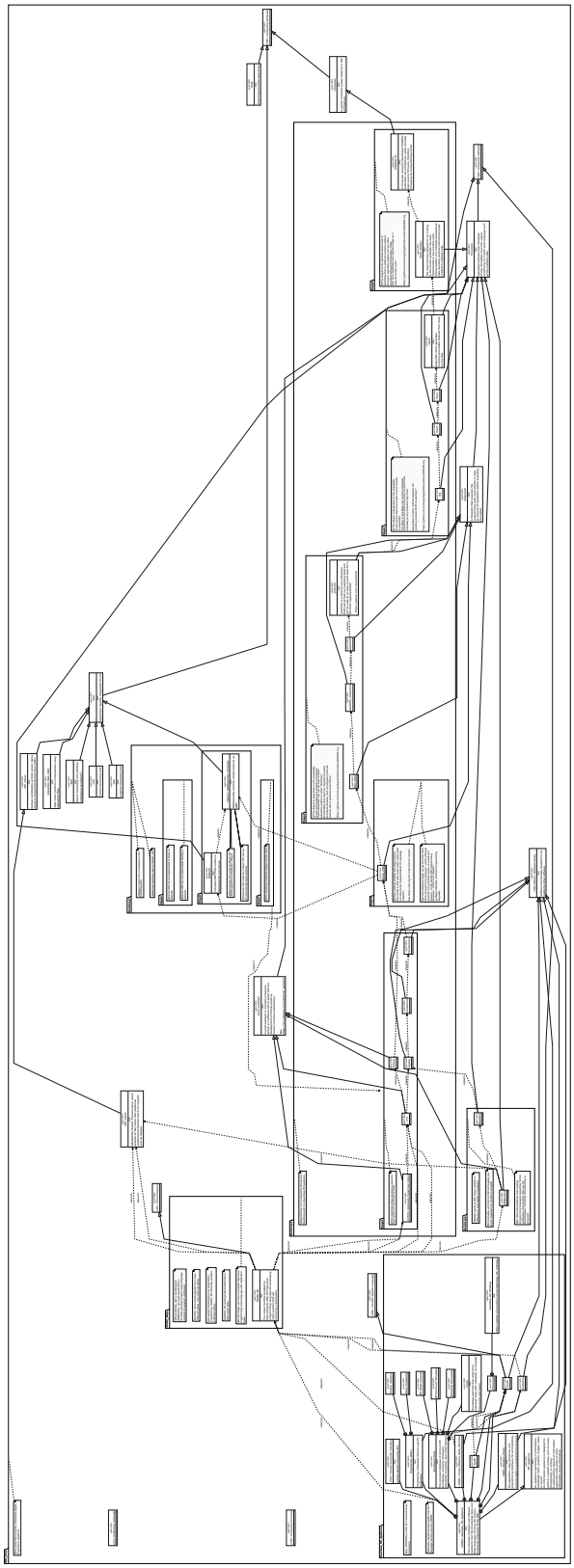


**Figure 5.14:** Overall view of Seamless DE Appliance bootstrap with 3 alternate bootstrap paths (Figs. 5.15, 5.21, 5.22)

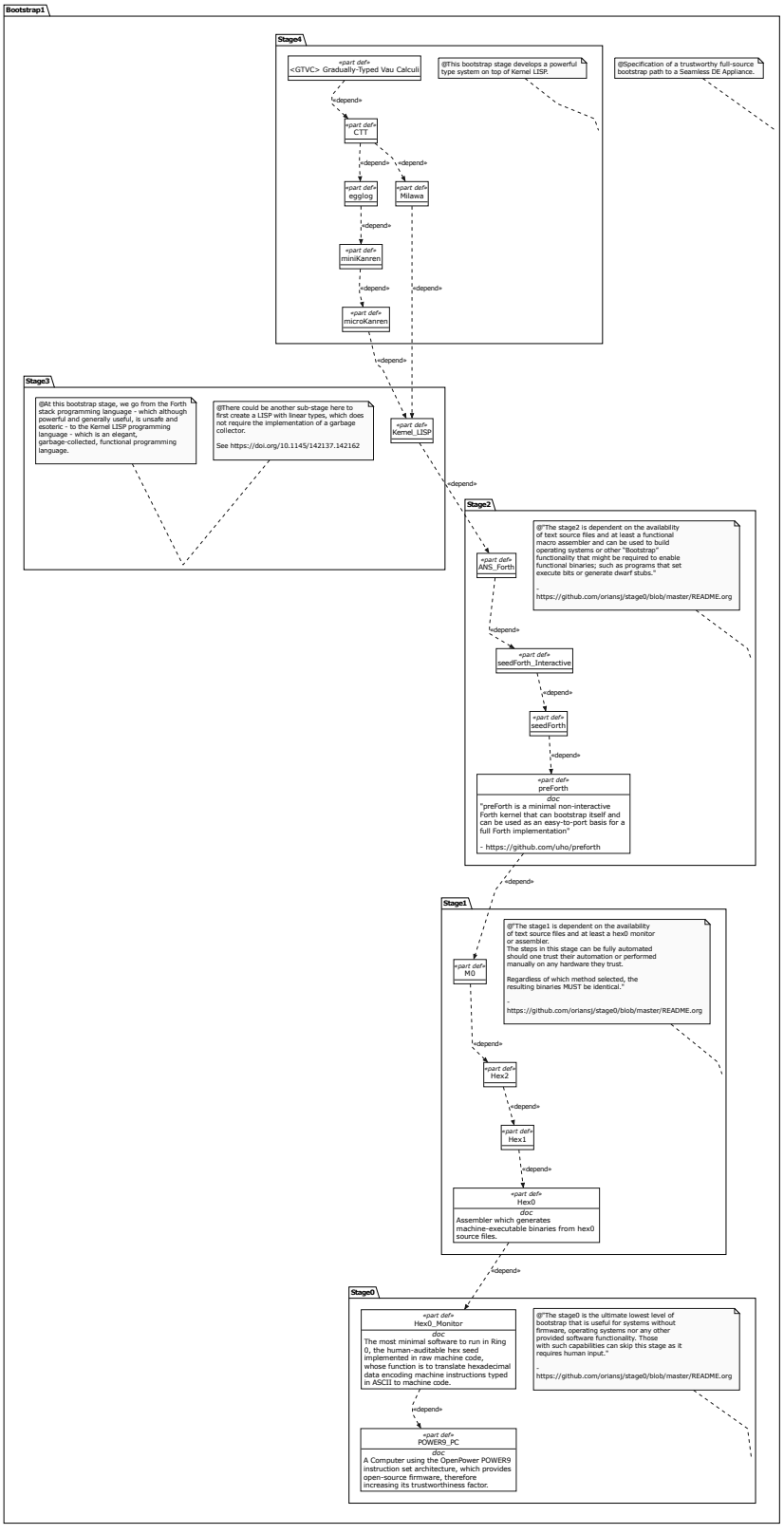



**Figure 5.15:** Overall view of bootstrap path #1 starting with the availability of a POWER9-based computer

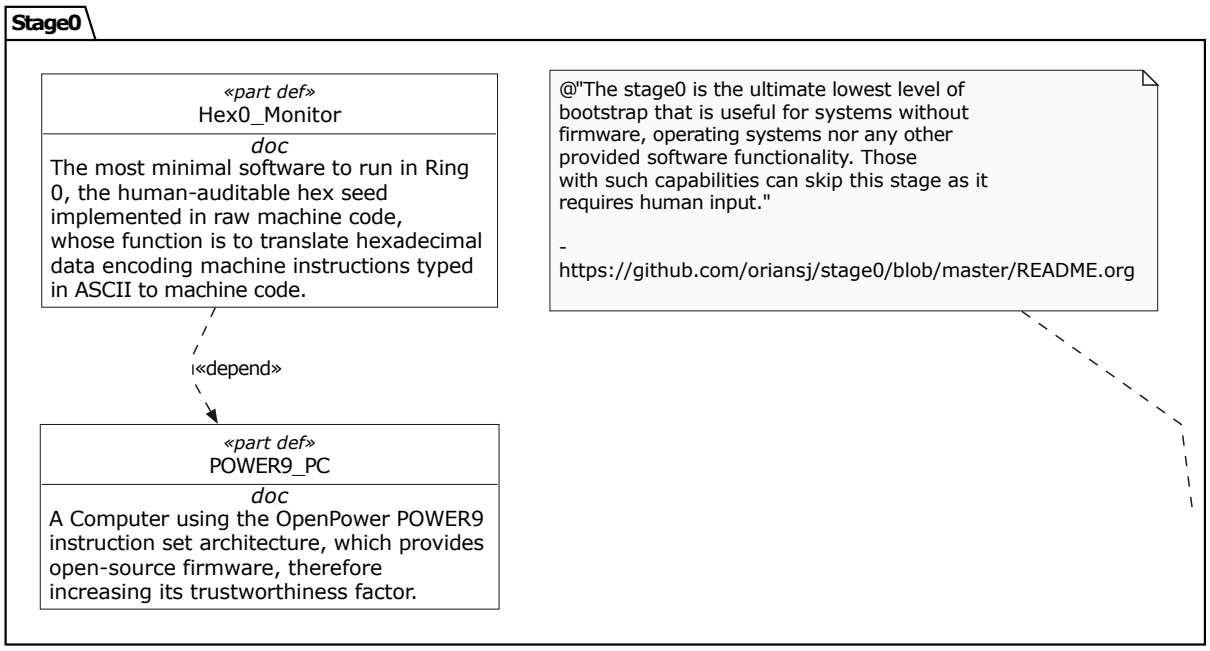



**Figure 5.16:** Stage0 of bootstrap path #1 (Figure 5.15).

from the diversity of ANS Forth compilers, which can verify the output of any bootstrapped interpreters and compilers.

In Stage3 (Figure 5.19), the Kernel LISP language [244] is bootstrapped from the ANS Forth compiler. That is, a Kernel interpreter is written in ANS Forth. This small-but-powerful functional programming language offers a leap in expressiveness and safety, and will be used to bootstrap the next stage (Figure 5.20) which requires the construction of a theorem prover, constraint solvers, and other computational logic libraries (Section 5.6).

Stage4 (Figure 5.20) bootstraps a computational type theory from minimal components. This powerful type theory can then be used to formally specify the Seamless DE Meta-language according to its reference specification. In this way, powerful formal verification capabilities are bootstrapped early to support formal verification of the remaining system components. This method contrasts with existing formal verification facilities that are built on complex dependency networks of non-formally-verified software components [245, 246]. End-to-end formal verification of hardware and software components was successfully demonstrated by [247–249], and the same principles apply to the Seamless DE Appliance.

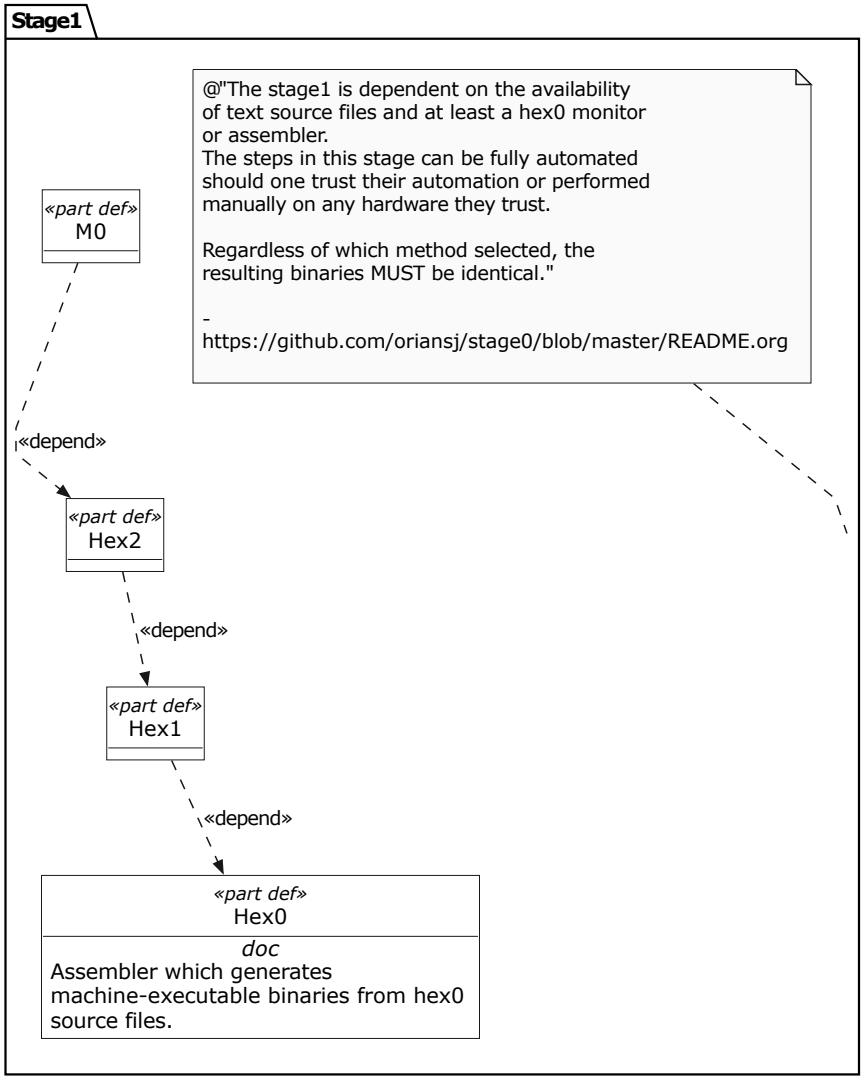



**Figure 5.17:** Stage1 of bootstrap path #1 (Figure 5.15).

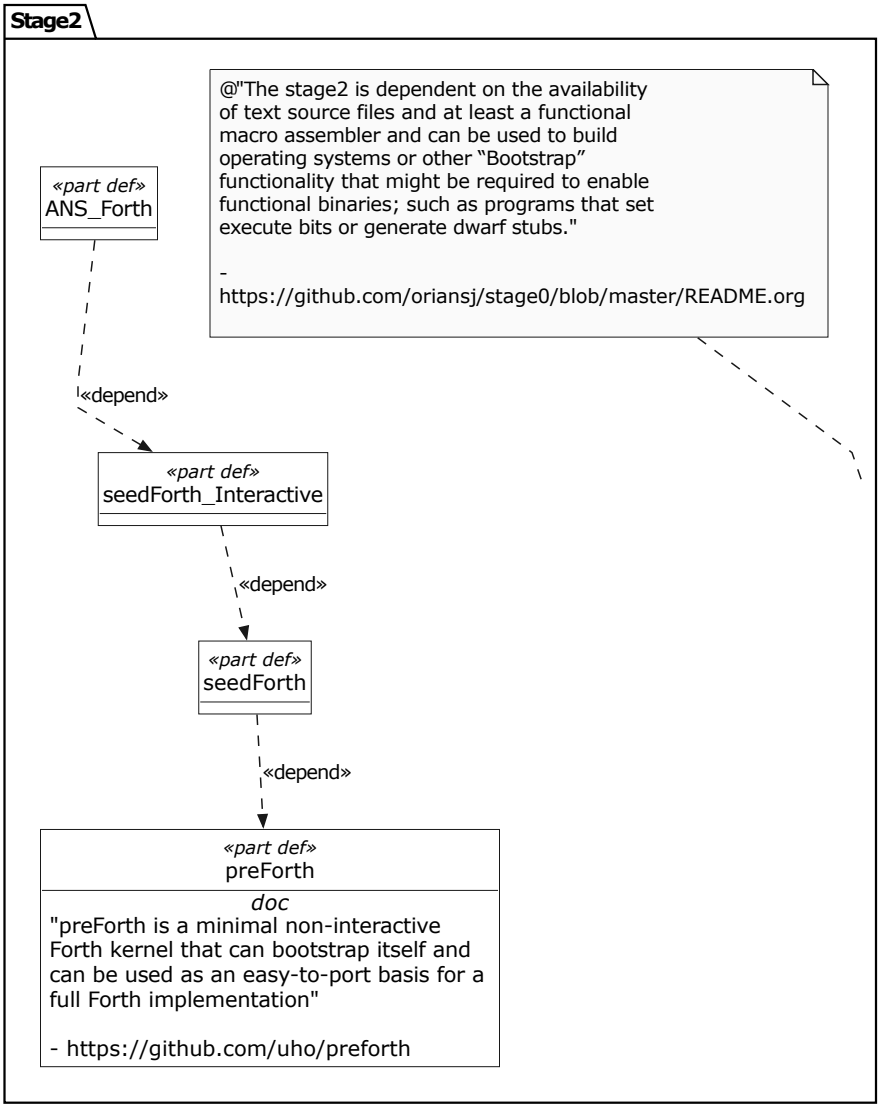



**Figure 5.18:** Stage2 of bootstrap path #1 (Figure 5.15).

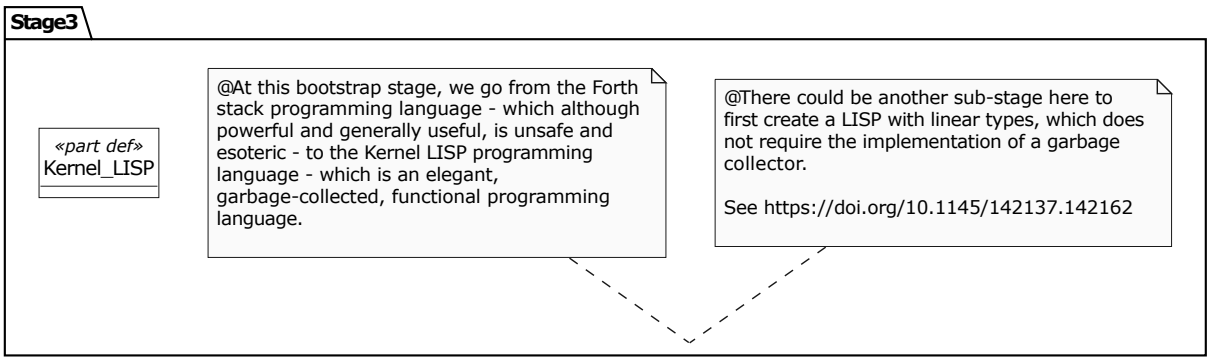



**Figure 5.19:** Stage3 of bootstrap path #1 (Figure 5.15).

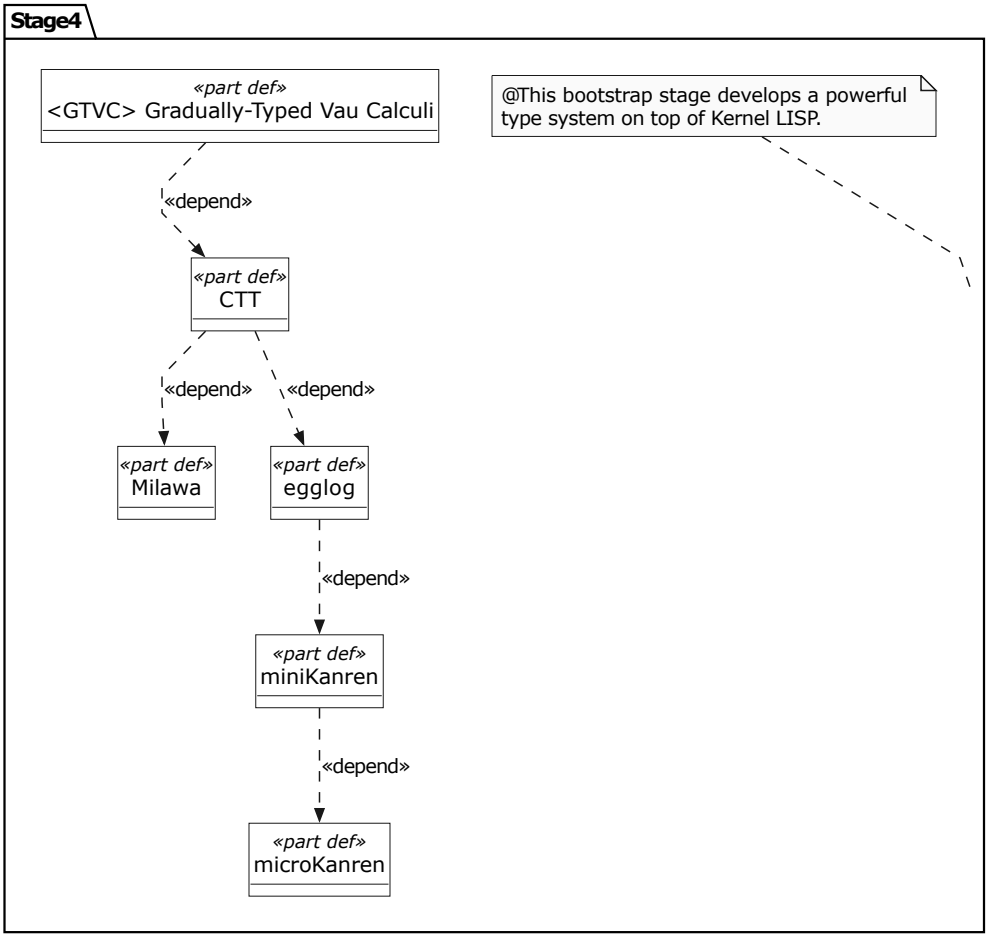



**Figure 5.20:** Stage4 of bootstrap path #1 (Figure 5.15).

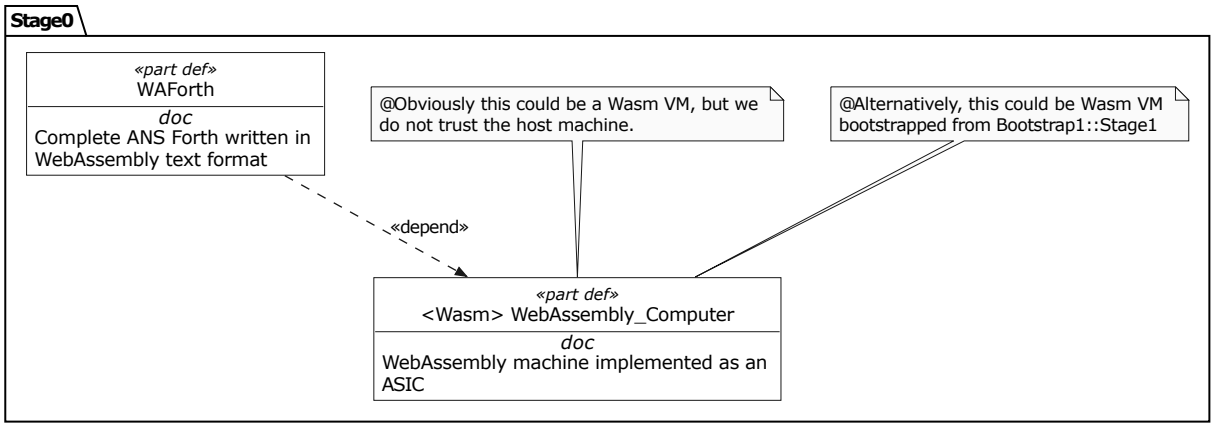



**Figure 5.21:** Stage0 of alternate bootstrap path #2 which depends on a WebAssembly VM or ASIC.

Alternatively, the WebAssembly [250–252] offers a powerful, formally-specified bootstrapping language. Multiple efficient WebAssembly virtual machines exist, satisfying the implementation diversity metric for fully countering Trusting Trust. And although the Kernel language shown in Stage3 (Figure 5.19) may be implemented directly using the WebAssembly stack machine with LISP-like textual syntax, there already exists an ANS Forth-conformant compiler written in WebAssembly, WAForth [253]. Therefore, this alternate bootstrap path can continue to first bootstrap stage 3. As noted, a WebAssembly machine may also be implemented as an ASIC or even a printed circuit board with logic chips, thus dramatically reducing the Trusted Computing Base of the bootstrap machine and therefore improving its assurance of trustworthiness.

Other higher-level bootstraps are possible, and although their value is reduced due to high complexity, they are nevertheless accessible to operators without special equipment. The Guile Scheme programming language, used in Refs. [242, 254], is full-source bootstrapped from [241] and therefore has a higher trustworthiness factor, despite its complicated implementation and dependency graph. Alternatively, Ada and SPARK may be used (Figure 5.22) to construct the Seamless DE Meta-language interpreter. Although they currently lack full-source bootstrap paths, SPARK provides formal verification facilities that can be used to produce a formally-verified, high-assurance interpreter. However, the open-source compiler machinery used in

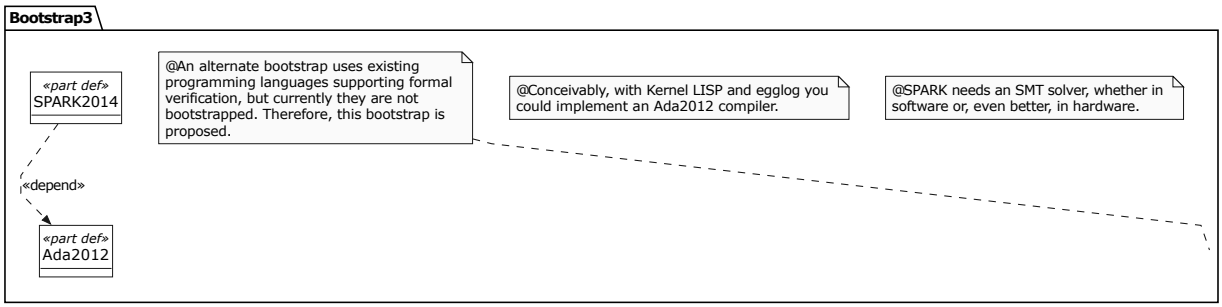



**Figure 5.22:** Stage5 of alternate bootstrap path #2 (dependencies to other stages not shown).

both of these language implementations is very complicated, representing millions of lines of unverified, changing code, and is therefore not an ideal bootstrap path.

## 5.8 Conclusions

By following a transdisciplinary systems engineering approach [63, 255] to this mess [1] of problems in modern computing systems, we can apply greater rigor to the process of understanding problem space, attain important insights not mentioned in Ref. [22], synthesize approaches from adjacent domains, and specify the solutions space for digital engineering using the methods it aims to support. Systems engineers understand not to skip processes at the beginning of the lifecycle, to avoid greater costs, risks, and schedule delays further in the lifecycle. Understanding the mission, goals, and objectives (Section 5.2) is critical to developing suitable system lifecycle concepts, and deriving needs and requirements with an understanding of current drivers and constraints.

The unique characteristics of each of these bootstrap paths and parts (Section 5.7) change according to the particular hardware used, although the same basic approach verified by Diverse Double-Compiling [170] applies. Depending on the assurance level needed, some or all of the bootstrap stages may be automated with accompanying scripts without diminishing the value of the full-source bootstrap. However, for a root-of-trust machine in a DE environment, manual entry with minimal peripherals and computer components may be desirable to achieve high assurance of defense against backdoors imitating Trusting Trust. The detailed

model-based documentation provided by the Seamless DE Reference Architecture supports the repeatability and analyzability of the bootstrap process(es).

## 5.9 Summary

This chapter addressed RQ4, restated as follows: *How does the proposed Seamless Digital Engineering Reference Architecture meet the needs of "bootstrapping a trustworthy and seamless digital engineering appliance"?*

It answered this research question by addressing its parts, and referring to the results of previous chapters. Although we cannot comprehensively yet say that the set of goals, objectives, and needs are fully verified and validated, constituting an Integrated Set of Needs [116], they are written to answer the parts of this question, namely 'bootstrapping', 'trustworthy', 'seamless', and 'digital engineering appliance'. Reference architectures are designed to be updated as we learn more about the problem space and solutions space (Figure 2.1), especially after successful systems have been fielded. The open-source Seamless DE Reference Architecture [256] will continue to improve its specificity and completeness to support multiple competing implementations of a Seamless DE system or appliance.

# Chapter 6

# Conclusions and Future Work

> When a mess, which is a system of problems, is taken apart, it loses its essential properties and so does each of its parts. The behavior of a mess depends more on how the treatment of its parts interact than on how they act independently of each other. A partial solution to a whole system of problems is better than whole solutions of each of its parts taken separately.
>
> Russell L. Ackoff [257, 258]

## 6.1 Summary

Through the course of this research and dissertation, key research questions in digital engineering were addressed.

RQ1 asked, *What is Seamless Digital Engineering and what distinguishes it from Digital Engineering?* While we often hear the word 'seamless' in the context of digital engineering, it is considered aspirational and lacks sufficient definition to be measured or compared. Chapter 2 considers this notion seriously, and by drawing from the literature in MBSE, computer science, and human-computer interaction research, identifies Seamless DE as a grand challenge in digital engineering research. The grand challenge criteria and rationale are documented in Table 2.4. Section 2.4 defines Seamless DE as a high-integrity DE tooling paradigm oriented toward total system elegance. The complete definition is repeated here:

> **Seamless Digital Engineering** is a digital engineering tooling paradigm that guarantees model coherence and integrity by affording an elegant human-computer interface for systems modeling that is end-to-end formally verified down through the computer hardware.

Where DE presupposes the complex integration of heterogeneous IT systems to achieve its goals, Seamless DE applies MBSE and DE to itself, using rigorous systems and software engineering processes to develop a purpose-built system that meets the stakeholder needs. Seamless DE rejects the notion that we can achieve seamless integration without the rigorous application of SE processes, and highlights the extreme risks and costs of attempting the DE transition without the requisite preparation and assessment. Section 2.3 makes the case that a clean-slate development approach is viable, whereas continued investment in fragile computing systems is foolhardy. The grand challenge described in Section 2.5 invites other researchers to engage this topic and contribute to its development.

RQ2 asked, *How do we precisely define Seamless Digital Engineering and its related concepts?* While Chapter 2 provided a natural language definition of Seamless DE, integration requirements and design pattern, and architecture tenets, precise definitions of Seamless DE concepts remained. Chapter 3 used a rigorous ontological framework and formal language to develop those precise definitions, and related them to existing concepts in SE and DE. The research drew from a corpus of international standards and other canonical sources, and identified key sources of concept definitions that were used to develop the ontological definitions in Seamless DE. These sources included the SQuaRE product quality series [89, 90, 94, 95], systems and software assurance series [96–98], and the evaluation criteria ("Common Criteria") series [106, 107]. The Seamless DE Ontology is published open-source at [88].

These concepts, in addition to those in the ISO/IEC/IEEE 15288 standard on system life cycle processes, were defined within the framework of BFO top-level ontology [85] and CCO mid-level ontology [70] and thus greatly benefited from additional standardization efforts made by expert ontology engineers. Essential concepts in DE were developed with feedback and coordination from the INCOSE DEIX Ontology WG [71]. This diversity of sources (Table 3.2) is summarized in a proposed domain ontology import hierarchy Figure 3.4, suggesting that future standardization efforts can improve the rigor and harmonization of concepts through the use of formal ontology engineering.

The essential Seamless DE concepts are summarized in Figure 3.5 and include 'Seamless Digital Engineering Paradigm' (Listing 3.1), Correct-by-Construction (Listing 3.2), 'Seamless Digital Engineering System' (Listing 3.3), 'Seamless Quality Claim' (Listing 3.4), 'Seamless Integration' (Listing 3.5), 'Seamless Interface' (Listing 3.6), 'Seamless Interaction Capability' (Listing 3.7), 'Seamless Quality-in-Use' (Listing 3.8), and Trustworthiness (Listing 3.9). Of these, two concepts stand out: Correct-by-Construction and Trustworthiness. While trustworthiness is already included in the SQuaRE Quality-in-Use standard [90] under the Acceptability quality-in-use characteristic, it is defined ontologically using concepts in systems assurance [96] prescribe that trustworthiness is evidence-based. The concept of correct-by-construction has a long history in computer science [173, 259–261] and is now a central idea, defined as a subclass of assurance goal, in Seamless DE. Finally, to summarize one key outcome of this ontological analysis, 'seamless' was disambiguated into two quality characteristics and one quality-in-use characteristic, which rely on the definition of Seamless Interface (Listing 3.6) introduced earlier in Figure 2.13.

RQ3 asked, *How do we define requirements in Seamless Digital Engineering?* In Chapter 4, this question was addressed by developing a SysML meta-model based on the latest INCOSE guidance on writing requirements [116,117]. That guidance identifies and defines 49 Attributes, 42 Rules, and 15 Characteristics of well-formed requirements. This meta-model was extended from the Model-Based Structured Requirement (MBSR) SysML profile [127] and published open-source at [125]. The structured patterns of requirement statements as recommended in [117] fit naturally with MBSE principles and contribute to a rich Authoritative Source of Truth (ASoT) for needs and requirements. This research was applied on two real-world projects at NASA JPL [155, 156] where we informally assessed its benefits Table 4.1, while noting its challenges Table 4.2. In short, the tool integration with the SysML profile provided the basis from which to rapidly improve the quality of the requirement sets compared to informal document-based requirements. The INCOSE-derived MBSR profile and modeling

technique are therefore essential to developing the high-quality requirements needed for Seamless DE.

RQ4 asked, *How does the proposed Seamless Digital Engineering Reference Architecture meet the needs of "bootstrapping a trustworthy and seamless digital engineering appliance"?* To address this question, the reference architecture proposed in Chapter 2 was developed using SysML v2, incorporating research results from Chapter 3 and Chapter 4. Chapter 5 presents the reference architecture starting from DE goals analyzed from Ref. [22] (Figures 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8), to operational concept, stakeholder needs, system requirements and specification of a systems modeling meta-language. Critically, reflecting the ontological definitions from the Seamless DE Ontology Annex A: Bootstrappable, the Seamless DE Reference Architecture provided multiple alternatives for a full-source bootstrap of a Seamless DE information appliance (Section 5.7). A full-source bootstrap achieves critical security properties identified and proven by Refs. [170, 262] in response to the backdoor vulnerability disclosure by Refs. [239, 240]. Such provable resistance to this high-severity vulnerability is unachievable in today's digital engineering environments (DEEs) due to the preponderance of unaudited source code and severed bootstrap paths. In order for a seamless DE appliance to achieve *trustworthy correct-by-construction*, multiple full-source bootstrap paths must be completed to satisfy independent human auditors. Therefore, the Evaluation Assurance Level (EAL) 7+ certification goal that the resulting system is free of security backdoors and correctly implements its detailed specification is achieved through evidence-based, consistent analysis, without the need for unauditable artifacts generated from extrinsic systems.

## 6.2 Research Contributions

The research contributions of this dissertation are summarized below, in order of the research questions from Section 1.2.

In addressing RQ1, Chapter 2 clarified the definition of *Seamless Digital Engineering* in contrast with how DE is currently being implemented, and identified it as a grand challenge

in DE research. It characterized the problem space using SysML v2 depictions and provided essential requirements derived from relevant literature. The seamless integration design pattern (Figure 2.13) presented an *elegant system* [43,44,46] approach compared to commonly available integration patterns in DE. The architecture tenets from Section 2.4.4 provide guidance for future research in this area, and the criteria and rationale provided in Table 2.4 establish baseline rationale for continued research in this area. For researchers of digital engineering, this reframing of DE as a transdisciplinary systems engineering problem may lead to elegant system solutions for DE stakeholders. For DE industry leaders, this focus on elegant system design with formally-verified interfaces may lead to new acquisition strategies that obsolete some IT-focused procurement processes.

In addressing RQ2, Chapter 3 provided an ontological definition of 'seamless' and Seamless DE in the context of systems engineering standards and based on established top- and mid-level ontologies. The resulting Seamless DE Ontology assimilated ISO/IEC/IEEE standards information, including the ISO/IEC 25000-series on the SQuaRE product quality model into an ontology used to define these seamless DE concepts. The resulting ontology is an open-source, machine-readable Semantic Web standards-based ontology for DE and MBSE applications. Stakeholders of DE may appreciate the clarity that such ontology-based definitions bring to DE acquisition discussions, while practicing engineers and researchers may adapt and extend the ontology to continue modeling domains of interest while maintaining ontological coherence. As DE becomes operationalized in organizations, the use of ontologies to support the definition of ASoTs becomes necessary; as an open-source ontology, the Seamless DE Ontology can be a critical starting point or a complementary domain ontology.

In addressing RQ3, Chapter 4 presented a complete open-source SysML v1 profile for digital requirements engineering (DRE) based on structured need/requirement statements. It demonstrated the application of the profile using an example model in the aerospace domain. Finally, the INCOSE-derived MBSR approach was validated in a real-world NASA-ESA Mars Sample Return project in Pre-Phase A [119], with documented observed benefits

and challenges (Tables Table 4.1 and Table 4.2). Model-based systems engineering and requirements engineering researchers may use the open-source SysML profile as the basis for researching MBSRs, contributing extensions, refinements, examples, and evidence to the body of knowledge. Practitioners may use the profile as-is for lightweight requirements management work, or explore how it can fit into existing RMT-based workflows.

In addressing RQ4, Chapter 5 presents and provides an open-source Seamless DE Reference Architecture [256], integrating concepts from earlier chapters into a SysML v2 model ready for continued development. Where explanatory prose falls short in describing architectures, the architecture model benefits from language-based validation, contextual annotations, and unlimited, differentiated detail when necessary. The model of the full-source bootstrap process conforming to Ref. [170] affords unprecedented detail to describe this complicated process that often goes ignored in programming language development projects. For computer science researchers and software engineers, the bootstrap model can serve as a teaching aid and implementation reference model. For systems engineering researchers and practitioners, detail of the bootstrap process highlights the systems security implications of this otherwise hidden and potentially dangerous aspect of digital systems. For DE researchers and practitioners, the reference architecture may help answer important questions in the acquisition and sustainment phases of the DEE, or it may be tailored and extended for organization-specific DE implementation plans.

## 6.3 Future Work

As a grand challenge, Seamless DE calls for significant future work to explore its implications. Comprehensive research is needed in eliciting stakeholder needs, verifying and validating the needs to form an Integrated Set of Needs [117], formally transforming those needs into a verified and validated set of Design Input Requirements [117], developing a reference architecture with SysML v2 that satisfies the needs and requirements, and constructing prototypes for early evaluation. SysML v2 has been chosen to position this work in

line with the next-generation of MBSE tools. The open-source reference architecture model provides the concrete basis for further feedback and collaboration. By enacting research and development of a clean-slate design, the solution space grows, potentially leading to elegant solutions that could not exist within the constraints of existing IT systems.

Ontology development in the systems engineering and digital engineering knowledge domains is ongoing internally among multiple groups, including INCOSE DEIX WG, AIAA Digital Engineering Integration Committee, INCOSE GfSE Collaborative Artifact, Specification, Context and Data Exchange (CASCaDE), OMG Model-Based Acquisition Working Group, and other entities, varying in ontological frameworks used and level of rigor. Final results from those working groups are forthcoming. Definition of concepts in Digital Engineering is ongoing, and this ontology was developed in tandem with the smaller-scoped INCOSE DEIX Ontology to be published soon. Future work includes a definition of object properties, especially those in the systems engineering domain, which was deferred to focus on the class hierarchy and use of existing BFO and CCO object properties.

Ontological definitions of Seamless DE concepts are offered based on existing and up-to-date international standards on product quality, but work remains to further define all of the quality characteristics (Figures 3.3 and 3.2) using additional concepts from the domains pictured in Figure 3.4. As the ontology grows, it will split into those harmonized domain ontologies. Within this ontological framework, sub-disciplines of DE, such as digital requirements engineering (DRE) and digital quality engineering, may be defined and distinguished from their traditional counterparts, potentially furthering our understanding and development of Digital Engineering. The open availability of the Seamless DE Ontology invites researchers and practitioners to collaborate on its definition and use cases. The standards harmonization effort involved in the ontology highlights the growing need for formal model-based standards development, which is especially important to the whole-integrated nature of DE.

Future work will continue to explore the feasibility and effectiveness of INCOSE-derived MBSR. Future work remains to make full use of SysML modeling tool capabilities in conjunc-

tion with other tools such as MATLAB, Modelica, and Python to exploit the SysML-based traceability that the MBSR technique provides. Some INCOSE GtWR Rules, such as those disallowing certain words and phrases, may be automatically checked using simple string matching, achieving a similar capability to existing requirements quality management tools. Such a capability may be encoded in automated validation suites using custom scripts similar to the ones we wrote for metrics and legends, although the potential performance impact is a concern addressable by future research. Macros may be created that employ automated validation suite results to add and remove «Satisfy» and «Violate» relationships for Rules amenable to automation. Many other model validation rules may be added that further enhance the model-based requirements V&V automation afforded by MBSR, such as checking that architecture decomposition levels match across MBSR-referenced model elements, or checking for conflicting or redundant requirements.

Automatically-generated requirement statements, as proposed by [127], are provided as a read-only derived attribute, but future work is needed to make this 'Derived Requirement Statement' attribute more readable while not interfering with model element naming conventions: such as by adding a stereotype to MBSR-linked elements that provides text attributes for defining the requirement statement fragment depending on its intended pattern slot. Names of the Pattern Slots and their element values could be used with Large Language Models or other advanced natural language processing (NLP) models to generate natural language sentences with proper syntax and grammar. The Condition pattern slot may be further decomposed following [150], which categorizes conditions into event-driven, unwanted behavior, state-driven, and optional feature; or it may be decomposed according to a hierarchical ontology, as shown in [151] and the GtWR Appendix C. The capacity of an ASoT with MBSRs to answer targeted mission programmatic questions is an area of future research that depends on the high maturity of both the system architecture model and requirements engineering organizational processes.

The revised semantics in SysML v2 present an opportunity to adapt the MBSR Profile to the new language and to research the change impact and potential benefits. SysML v2 uniformly applies the modeling language design pattern of *definition* and *usage* to requirements; it defines requirements necessarily as constraints with boolean satisfiability; stakeholder concerns are more explicitly modeled compared to SysML v1; and metadata elements are used instead of stereotypes. Furthermore, SysML v2 specifies a textual syntax in addition to the graphical syntax, potentially making model representations such as Figure 4.8 more readable and amenable to automation. SysML v2 also provides a standardized API for accessing the system architecture model, which may ease future research in integrating MBSR with NLP-based and formal requirements engineering tools. This shift toward more model-based requirements presents new opportunities to enrich the ASoT with metadata and automated model validation derived from the INCOSE GtWR.

Future work will continue through research involving the Seamless DE Ontology and Reference Architecture, full formal specification of the Seamless DE Meta-language, and developing a prototype to eventually fully specify itself is a high priority. The verified full-source bootstrap processes constitute years of future detailed effort. The same principles apply to the computer hardware, since electronic design automation software also constitutes a kind of complex compiler. How can we trust their design outputs? Once an ASIC is manufactured, how can we trust that it conforms 100% to its formal specification? New technologies and techniques must be developed for an operator to independently verify their specific hardware affordably.

There are numerous research paths in the Seamless DE grand challenge that are foreseeable: open specification of a reference architecture of DE lifecycle(s); computational design of product-line architecture instantiations for particular form-factors of Seamless DE appliances; computational type theory-based investigations of seamless system architectures building from SysML v2 semantics, proving an architecture seamless and trustworthy at the type level; comprehensive analysis of DE system bootstrap paths, size and effort; technical debt

comparisons and type-level analysis of extant computer system architectures. In the short term, the Seamless DE Ontology will continue to grow, and the Reference Architecture continue to add detail and feedback from stakeholders. As a grand challenge, decades of future work awaits for those interested and determined.

# Bibliography


[1] R. L. Ackoff, *Redesigning the Future: A Systems Approach to Societal Problems*, ser. A Wiley-Interscience Publication. Wiley, 1974, p. 21.

[2] D. Seal, "The system engineering 'V'–is it still relevant in the digital age?" *Boeing Company, Global Product Data Interoperability Summit, Presentation*, 2018.

[3] D. Vaughan, *The Challenger Launch Decision: Risky Technology, Culture, and Deviance at NASA*. University of Chicago Press, 2016.

[4] F. P. Brooks, Jr., *The Mythical Man-Month: Essays on Software Engineering*. Pearson Education, 1995.

[5] CVE Project, "CVE list V5," 2025, retrieved: December 11, 2022. url: https://github.com/CVEProject/cvelistV5.

[6] C. Booth, H. Rosten, W. Mertens, J.-P. Calderone, A. Nelson, and S. Eyeoglu, "Nix dependency visualizer," 2024, retrieved: July 6, 2025. url: https://github.com/craigmbooth/nix-visualize.

[7] T. P. Hughes, *Networks of Power: Electrification in Western Society, 1880-1930*. Johns Hopkins University Press, Mar. 1993.

[8] O. Dedehayir and S. J. Mäkineif, "Dynamics of reverse salience as technological performance gap: an empirical study of the personal computertechnology system," *Journal of Technology Management & Innovation*, vol. 3, no. 3, pp. 55–66, 2008, doi: 10.4067/S0718-27242008000100006.

[9] DARPA, "Clean-slate design of resilient, adaptive, secure hosts (CRASH)," Defense Advanced Research Projects Agency, 3701 North Fairfax Drive, Arlington, VA, 22203-1714, Broad Agency Announcement DARPA-BAA-10-70, Jun. 1, 2010.

[10] ——, "META-II," Defense Advanced Research Projects Agency, Tactical Technology Office (TTO), 3701 North Fairfax Drive, Arlington, VA, 22203-1714, Broad Agency Announcement DARPA-BAA-10-59, Apr. 28, 2010.

[11] ——, "Circuit realization at faster timescales (CRAFT)," Defense Advanced Research Projects Agency, Microsystems Technology Office (TTO), 675 North Randolph Street, Arlington, VA, 22203-2114, Broad Agency Announcement DARPA-BAA-15-55, Aug. 17, 2015.

[12] D. A. Norman, *The Invisible Computer: Why Good Products Can Fail, the Personal Computer is So Complex, and Information Appliances are the Solution*. MIT Press, 1999.

[13] J. S. Wheaton and D. R. Herber, "Seamless digital engineering: A grand challenge driven by needs," in *AIAA SCITECH 2024 Forum*, no. AIAA 2024-1053. Orlando, FL, USA: American Institute of Aeronautics and Astronautics, Jan. 2024, doi: 10.2514/6.2024-1053.

[14] J. S. Moore, *A Grand Challenge Proposal for Formal Methods: A Verified Stack*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 161–172, doi: 10.1007/978-3-540-40007-3_11.

[15] R. W. Shumaker, K. R. Walkup, and B. B. Beck, *Animal tool behavior: the use and manufacture of tools by animals*. John Hopkins University Press, May 2011.

[16] J.-J. Salomon, "What is technology? the issue of its origins and definitions," *History and Technology, an International Journal*, vol. 1, no. 2, pp. 113–156, 1984, doi: 10.1080/07341518408581618.

[17] E. B. Skolnikoff, *The Elusive Transformation: Science, Technology, and the Evolution of International Politics*. Princeton University Press, 1993, doi: 10.1515/9781400820924.

[18] Engineers' Council Professional for Development, *Canons of Ethics for Engineers*. New York: Engineers' Council for Professional Development, 1947.

[19] INCOSE, "Systems engineering and system definitions," International Council on Systems Engineering, Technical Product INCOSE-TP-2020-002-06, Jul. 22, 2019.

[20] ISO, "Systems and software engineering — vocabulary," International Organization for Standardization, Geneva, CH, ISO/IEC/IEEE Standard 24765:2017, 09 2017.

[21] INCOSE, "INCOSE systems engineering vision 2020," International Council on Systems Engineering, Technical Product INCOSE-TP-2004-004-02, 09 2007.

[22] M. D. Griffin, K. Baldwin, J. Stanley, R. H. Kewley, Jr., and W. Bray, "Digital engineering strategy," Deputy Assistant Secretary of Defense, Systems Engineering, 3030 Defense Pentagon, 3C167, Washington, D.C., USA 20301-3030, Jun. 2018.

[23] R. A. Noguchi, M. J. Wheaton, and J. N. Martin, "Digital engineering strategy to enable enterprise systems engineering," in *INCOSE International Symposium*, vol. 30, no. 1, Sep. 2020, pp. 1727–1741, doi: 10.1002/j.2334-5837.2020.00815.x.

[24] U.S. Department of Defense Defense Acquisition University, "Glossary of defense acquisition acronyms and terms," 2025, retrieved May 15, 2025. url: https://www.dau.edu/glossary.

[25] L. A. Lombardi, "Lisp as the language for an incremental computer," Sloan School of Management, Massachusetts Institute of Technology, 50 Memorial Drive, Cambridge, MA, USA, Tech. Rep. 51-64, Mar. 1964, url: http://hdl.handle.net/1721.1/48305.

[26] A. Bawden, R. Greenblatt, J. Holloway, T. Knight *et al.*, "Lisp machine progress report." Artificial Intelligence Lab, Massachusetts Institute of Technology, Cambridge, MA, USA, Tech. Rep., 1977, url: https://apps.dtic.mil/sti/citations/ADA062178.

[27] J. H. Walker, D. A. Moon, D. L. Weinreb, and M. McMahon, "The symbolics genera programming environment," *IEEE Software*, vol. 4, no. 6, pp. 36–45, Nov. 1987, doi: 10.1109/MS.1987.232087.

[28] D. H. H. Ingalls, "Design principles behind Smalltalk," *BYTE Magazine*, vol. 6, no. 8, pp. 286–298, Aug. 1981.

[29] B. W. Boehm, M. H. Penedo, E. D. Stuckle, R. D. Williams, and A. B. Pyster, "A software development environment for improving productivity," *Computer*, vol. 17, no. 06, pp. 30–44, Jun. 1984, doi: 10.1109/MC.1984.1659160.

[30] I. Thomas and B. A. Nejmeh, "Definitions of tool integration for environments," *IEEE Software*, vol. 9, no. 2, pp. 29–35, Mar. 1992, doi: 10.1109/52.120599.

[31] M. Broy, "Seamless model driven systems engineering based on formal models," in *Formal Methods and Software Engineering*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 1–19, doi: 10.1007/978-3-642-10373-5_1.

[32] M. Broy, M. Feilkas, M. Herrmannsdoerfer, S. Merenda, and D. Ratiu, "Seamless model-based development: From isolated tools to integrated model engineering environments," *Proceedings of the IEEE*, vol. 98, no. 4, pp. 526–545, Apr. 2010, doi: 10.1109/JPROC.2009.2037771.

[33] M. Broy, "Seamless model-based system development: Foundations," in *Engineering Trustworthy Software Systems*. Springer, Cham, 2020, pp. 1–9, doi: 10.1007/978-3-030-55089-9_1.

[34] A. Perzylo, I. Kessler, S. Profanter, and M. Rickert, "Toward a knowledge-based data backbone for seamless digital engineering in smart factories," in *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1, Sep. 2020, pp. 164–171, doi: 10.1109/ETFA46521.2020.9211943.

[35] L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos, *Non-functional requirements in software engineering*. Springer Science & Business Media, 2012, vol. 5, doi: 10.1007/978-1-4615-5269-7.

[36] AIAA Digital Engineering Integration Committee, "Digital Twin: Definition & Value," American Institute of Aeronautics and Astronautics, AIAA and AIA Position Paper, Dec. 2020.

[37] ——, "Digital Twin: Reference Model, Realizations & Recommendations," American Institute of Aeronautics and Astronautics, AIAA, AIA, and NAFEMS Implementation Paper, Jan. 2023.

[38] R. Karban, F. G. Dekens, S. Herzig, M. Elaasar, and N. Jankevičius, "Creating system engineering products with executable models in a model-based engineering environment," in *Modeling, Systems Engineering, and Project Management for Astronomy VII*, vol. 9911. SPIE, 2016, pp. 96–111, doi: 10.1117/12.2232785.

[39] C. Delp, "Open model-based engineering environments," Apr. 2, 2019, url: https://www.nist.gov/system/files/documents/2019/04/05/14_delp.pdf.

[40] M. Bajaj, S. Friedenthal, and E. Seidewitz, "Systems modeling language (SysML v2) support for digital engineering," *Insight*, vol. 25, no. 1, pp. 19–24, 2022, doi: 10.1002/inst.12367.

[41] R. G. Enticknap and E. F. Schuster, "Sage data system considerations," *Transactions of the American Institute of Electrical Engineers, Part I: Communication and Electronics*, vol. 77, no. 6, pp. 824–832, Jan. 1959, doi: 10.1109/TCE.1959.6372899.

[42] Princeton University. (2011, Jun.) About wordnet. url: https://wordnet.princeton.edu/.

[43] M. Watson, B. Mesmer, and P. Farrington, "Engineering elegant systems: The practice of systems engineering," National Aeronautics and Space Administration,

Marshall Space Flight Center, Huntsville, Alabama, US 35812, Tech. Rep., Jun. 2020, url: https://ntrs.nasa.gov/api/citations/20205003646/downloads/NASA_TP_20205003646_interactive.pdf.

[44] M. D. Watson, B. Mesmer, and P. Farrington, "Engineering elegant systems: Postulates, principles, and hypotheses of systems engineering," in *Systems Engineering in Context*, S. Adams, P. A. Beling, J. H. Lambert, W. T. Scherer, and C. H. Fleming, Eds. Cham: Springer International Publishing, Jun. 2019, pp. 495–513, doi: 10.1007/978-3-030-00114-8_40.

[45] M. D. Watson, M. Griffin, P. A. Farrington, L. Burns, W. Colley, P. Collopy, J. Doty, S. B. Johnson, R. Malak, J. Shelton *et al.*, "Building a path to elegant design," in *Proceedings of the International Annual Conference of the American Society for Engineering Management*. Virginia Beach, VA, USA: American Society for Engineering Management (ASEM), Oct. 2014.

[46] A. M. Madni, "Elegant systems design: Creative fusion of simplicity and power," *Systems Engineering*, vol. 15, no. 3, pp. 347–354, Jul. 10, 2012, doi: 10.1002/sys.21209.

[47] M. D. Griffin, "How do we fix systems engineering," in *61st International Astronautical Congress*, vol. 27, Prague, Czech Republic, 2010, september 27—October 1.

[48] W. Whitney and B. Smith, *The Century Dictionary and Cyclopedia: Dictionary*, ser. The Century Dictionary and Cyclopedia. Century Company, 1906, vol. 7, url: http://www.global-language.com/CENTURY/.

[49] J. D. Claxton, C. Cavoli, and C. Johnson, "Test and evaluation management guide," Defence Acquisition University, Fort Belvoir, Virginia, USA, Accession Number ADA436591, Nov. 2005, url: https://apps.dtic.mil/sti/pdfs/ADA436591.pdf.

[50] P. H. Hochschild, P. Turner, J. C. Mogul, R. Govindaraju, P. Ranganathan, D. E. Culler, and A. Vahdat, "Cores that don't count," in *Proceedings of the Workshop on*

*Hot Topics in Operating Systems*, ser. HotOS '21. New York, NY, USA: Association for Computing Machinery, Jun. 2021, pp. 9–16, doi: 10.1145/3458336.3465297.

[51] T. R. Halfhill, "An error in a lookup table created the infamous bug in intel's latest processor," *BYTE*, Mar. 3, 1995.

[52] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom *et al.*, "Meltdown: Reading kernel memory from user space," *Communications of the ACM*, vol. 63, no. 6, pp. 46–56, 2020, doi: 10.1145/3357033.

[53] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher *et al.*, "Spectre attacks: Exploiting speculative execution," *Communications of the ACM*, vol. 63, no. 7, pp. 93–101, 2020.

[54] O. Sibert, P. A. Porras, and R. Lindell, "The Intel 80×86 processor architecture: Pitfalls for secure systems," in *Proceedings of the 1995 IEEE Symposium on Security and Privacy*, 1995, pp. 211–222, doi: 10.1109/SECPRI.1995.398934.

[55] S. Chiricescu, A. DeHon, D. Demange, S. Iyer, A. Kliger, G. Morrisett, B. C. Pierce, H. Reubenstein, J. M. Smith, G. T. Sullivan, A. Thomas, J. Tov, C. M. White, and D. Wittenberg, "SAFE: A clean-slate architecture for secure systems," in *2013 IEEE International Conference on Technologies for Homeland Security (HST)*, Nov. 2013, pp. 570–576, doi: 10.1109/THS.2013.6699066.

[56] H. Reubenstein, T. Giannakopoulos, S. Chiricescu, A. Strnad, and J. Fahey, "Semantically aware foundation environment (SAFE) for clean-slate design of resilient, adaptive secure hosts (CRASH)," BAE Systems, Burlington, Mass., USA, Tech. Rep., Feb. 2016, url: https://apps.dtic.mil/sti/pdfs/AD1007956.pdf.

[57] A. von Gernler, "Towards a clean slate: Attempting to preserve civil liberty in the post-Snowden age," *it-Information Technology*, vol. 57, no. 3, pp. 203–207, 2015, doi: 10.1515/itit-2015-0008.

[58] E. Snowden, *Permanent Record*. Henry Holt and Company, 2019.

[59] J. Easterly, "CISA director easterly remarks at Carnegie Mellon University," Feb. 27, 2023, url: https://www.cisa.gov/cisa-director-easterly-remarks-carnegie-mellon-university.

[60] E. Hapeman, W. Zeuch, J. Crandall, S. Carioti, S. Barclay, and C. Underkoffler, "Telecom glossary 2000–american national standard t1. 523-2001." 2001, url: https://glossary.atis.org/glossary/data-integrity/.

[61] A. C. Kay, "A personal computer for children of all ages," in *Proceedings of the ACM Annual Conference - Volume 1*, ser. ACM '72. New York, NY, USA: Association for Computing Machinery, 2011. article 1, doi: 10.1145/800193.1971922.

[62] T. Hoare, "The verifying compiler: A grand challenge for computing research," in *International Conference on Compiler Construction*. Springer, 2003, pp. 262–272, doi: 10.1145/602382.602403.

[63] A. M. Madni, "Transdisciplinary systems engineering: Exploiting disciplinary convergence to address grand challenges," *IEEE Systems, Man, and Cybernetics Magazine*, vol. 5, no. 2, pp. 6–11, April 2019, doi: 10.1109/MSMC.2019.2899957.

[64] R. L. Ackoff, "Systems, messes and interactive planning," *The Societal Engagement of Social Science*, vol. 3, no. 1997, pp. 417–438, 1997, doi: 10.9783/9781512819069-021.

[65] IFSR, "The systems praxis framework, developed as a joint project of INCOSE and ISSS," International Federation for Systems Research (IFSR), Vienna, Austria, Tech. Rep., 2012, url: http://systemspraxis.org/framework.pdf.

[66] S. R. Walli, "The POSIX family of standards," *StandardView*, vol. 3, no. 1, pp. 11–17, Mar. 1995, doi: 10.1145/210308.210315.

[67] J. S. Wheaton and D. R. Herber, "Ontological definition of seamless digital engineering based on ISO/IEC 25000-series SQuaRE product quality model," in *INCOSE International Symposium*, Ottawa, Ontario, Canada, Jul. 2025.

[68] B. Stroustrup, *The C++ Programming Language*, 4th ed. Pearson Education, 2013, p. v.

[69] ISO, "Information technology — Top-level ontologies (TLO) — Part 2: Basic Formal Ontology (BFO)," International Organization for Standardization, Geneva, CH, ISO/IEC Standard 21838-2:2021, Nov. 2021.

[70] M. Jensen, G. De Colle, S. Kindya, C. More, A. P. Cox, and J. Beverley, "The common core ontologies," 2024, doi: 10.48550/ARXIV.2404.17758.

[71] J. Gregory, J. S. Wheaton, C. Moreland, and C. Tseng, "Towards a digital engineering ontology to support information exchange," in *INCOSE International Symposium*, Ottawa, Ontario, Canada, Jul. 2025.

[72] AIAA Digital Engineering Integration Committee, "Digital thread: Definition, value, and reference model," American Institute of Aeronautics and Astronautics, AIAA, AIA, and NAFEMS Implementation Paper, Jun. 2023.

[73] D. L. Allison, M. W. Cribb, T. McCarthy, and R. LaRowe, "Authoritative sources of truth and consistency in digital engineering," in *AIAA SCITECH 2023 Forum*. American Institute of Aeronautics and Astronautics, Jan. 2023, doi: 10.2514/6.2023-1404.

[74] INCOSE, Ed., *INCOSE Systems Engineering Handbook*, 5th ed., ser. Technical Reports. John Wiley & Sons, 2023, no. INCOSE-TP-2003-002-005.

[75] D. Dunbar, T. Hagedorn, M. Blackburn, J. Dzielski, S. Hespelt, B. Kruse, D. Verma, and Z. Yu, "Driving digital engineering integration and interoperability through semantic

integration of models with ontologies," *Systems Engineering*, vol. 26, no. 4, pp. 365–378, Mar. 2023, doi: 10.1002/sys.21662.

[76] H. Krasner, "The cost of poor software quality in the US: A 2022 report," Consortium for Information & Software Quality (CISQ), Tech. Rep., Dec. 2022, url: https://www.it-cisq.org/the-cost-of-poor-quality-software-in-the-us-a-2022-report/.

[77] C. Becker, *Insolvent: How to Reorient Computing for Just Sustainability*. MIT Press, 2023.

[78] A. Gómez-Pérez, M. Fernandez-Lopez, and O. Corcho, *Ontological Engineering: with examples from the areas of Knowledge Management, e-Commerce and the Semantic Web*, 1st ed., ser. Advanced Information and Knowledge Processing. Springer London, 2006, doi: 10.1007/b97353.

[79] M. A. Musen *et al.*, "The Protégé project: A look back and a look forward," *AI Matters*, vol. 1, no. 4, pp. 4–12, Jun. 2015, doi: 10.1145/2757001.2757003.

[80] World Wide Web Consortium, "OWL 2 Web Ontology Language primer," World Wide Web Consortium, W3C Recommendation, Dec. 11, 2012, retrieved May 15, 2025. url: https://www.w3.org/TR/owl2-primer/.

[81] B. C. Grau, I. Horrocks, B. Motik, B. Parsia, P. Patel-Schneider, and U. Sattler, "OWL 2: The next step for OWL," *Journal of Web Semantics*, vol. 6, no. 4, pp. 309–322, Nov. 2008, doi: 10.1016/j.websem.2008.05.001.

[82] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, Eds., *The Description Logic Handbook: Theory, Implementation and Applications*, 2nd ed. Cambridge University Press, Aug. 2007, doi: 10.1017/cbo9780511711787.

[83] F. Baader, I. Horrocks, and U. Sattler, "Description logics," in *Handbook of Knowledge Representation*, 1st ed., ser. Foundations of Artificial Intelligence, F. van Harmelen,

V. Lifschitz, and B. Porter, Eds. Elsevier, 2008, vol. 3, ch. 3, pp. 135–179, doi: 10.1016/S1574-6526(07)03003-9.

[84] M. Horridge and P. F. Patel-Schneider, "OWL 2 Web Ontology Language Manchester syntax," World Wide Web Consortium, W3C Working Group Note, Dec. 11, 2012, retrieved May 15, 2025. url: https://www.w3.org/TR/owl2-manchester-syntax/.

[85] J. N. Otte, J. Beverley, and A. Ruttenberg, "BFO: Basic formal ontology," *Applied Ontology*, vol. 17, no. 1, pp. 17–43, Mar. 2022, doi: 10.3233/ao-220262.

[86] N. F. Noy and D. L. McGuinness, "Ontology development 101: A guide to creating your first ontology," Stanford University, Tech. Rep. KSL-01-05, 2001.

[87] B. Motik, B. C. Grau, and U. Sattler, "Structured objects in OWL: Representation and reasoning," in *Proceedings of the 17th International Conference on World Wide Web*. Beijing, China: Association for Computing Machinery, Apr. 2008, pp. 555–564, doi: 10.1145/1367497.1367573.

[88] J. S. Wheaton, "Seamless digital engineering ontology," 2025, url: https://github.com/systems-praxis/seamless-digital-engineering-ontology.

[89] ISO, "Systems and software engineering — systems and software quality requirements and evaluation (SQuaRE) — product quality model," International Organization for Standardization, Geneva, CH, ISO/IEC/IEEE Standard 25010:2023, Nov. 2023.

[90] ——, "Systems and software engineering — systems and software quality requirements and evaluation (SQuaRE) — quality-in-use model," International Organization for Standardization, Geneva, CH, ISO/IEC/IEEE Standard 25019:2023, Nov. 2023.

[91] ——, "Systems and software engineering: System life cycle processes," International Organization for Standardization, Geneva, CH, ISO/IEC/IEEE Standard 15288:2023, May 2023.

[92] J. Gregory and A. Salado, "Towards a systems engineering ontology stack," in *INCOSE International Symposium*, vol. 34, no. 1, 09 2024, pp. 1304–1318, doi: 10.1002/iis2.13210.

[93] L. Yang, K. Cormican, and M. Yu, "Ontology-based systems engineering: A state-of-the-art review," *Computers in Industry*, vol. 111, pp. 148–171, 2019, doi: 10.1016/j.compind.2019.05.003.

[94] ISO, "Systems and software engineering — systems and software quality requirements and evaluation (square) — guide to SQuaRE," International Organization for Standardization, Geneva, CH, ISO/IEC/IEEE Standard 25000:2014, Mar. 2014.

[95] ——, "Systems and software engineering — systems and software quality requirements and evaluation (SQuaRE) — quality model overview and usage," International Organization for Standardization, Geneva, CH, ISO/IEC/IEEE Standard 25002:2024, Mar. 2024.

[96] ——, "Systems and software engineering — systems and software assurance — part 1: Concepts and vocabulary," International Organization for Standardization, Geneva, CH, ISO/IEC/IEEE Standard 15026-1:2019, Mar. 2019.

[97] ——, "Systems and software engineering — systems and software assurance — part 2: Assurance case," International Organization for Standardization, Geneva, CH, ISO/IEC/IEEE Standard 15026-2:2022, Nov. 2022.

[98] ——, "Systems and software engineering — systems and software assurance — part 4: Assurance in the life cycle," International Organization for Standardization, Geneva, CH, ISO/IEC/IEEE Standard 15026-4:2021, May 2021.

[99] ——, "Systems and software engineering — life cycle management — part 1: Guidelines for life cycle management," International Organization for Standardization, Geneva, CH, ISO/IEC/IEEE Standard 24748-1:2024, Mar. 2024.

[100] ——, "Systems and software engineering — life cycle management — part 6: System and software integration," International Organization for Standardization, Geneva, CH, ISO/IEC/IEEE Standard 24748-6:2024, Jul. 2023.

[101] ——, "Systems and software engineering — taxonomy of systems of systems," International Organization for Standardization, Geneva, CH, ISO/IEC/IEEE Standard 21841:2019, Sep. 2019.

[102] ——, "Systems and software engineering — methods and tools for model-based systems and software engineering," International Organization for Standardization, Geneva, CH, ISO/IEC/IEEE Standard 24641:2023, May 2023.

[103] ——, "Systems and software engineering — design and development of information for users," International Organization for Standardization, Geneva, CH, ISO/IEC/IEEE Standard 26514:2022, Jan. 2022.

[104] ——, "Software, systems and enterprise — architecture description," International Organization for Standardization, Geneva, CH, ISO/IEC/IEEE Standard 42010:2022, Nov. 2022.

[105] ——, "Software, systems and enterprise — architecture processes," International Organization for Standardization, Geneva, CH, ISO/IEC/IEEE Standard 42020:2019, Jul. 2019.

[106] ——, "Information security, cybersecurity and privacy protection — evaluation criteria for IT security — part 1: Introduction and general model," International Organization for Standardization, Geneva, CH, ISO/IEC/IEEE Standard 15408-1:2022, Aug. 2022.

[107] ——, "Information security, cybersecurity and privacy protection — evaluation criteria for IT security — part 5: Pre-defined packages of security requirements," International Organization for Standardization, Geneva, CH, ISO/IEC/IEEE Standard 15408-5:2022, Aug. 2022.

[108] ——, "Software engineering — life cycle processes — software acquisition," International Organization for Standardization, Geneva, CH, ISO/IEC/IEEE Standard 41062:2022, Oct. 2024.

[109] ——, "Innovation management — fundamentals and vocabulary," International Organization for Standardization, Geneva, CH, ISO Standard 56000:2025, Jan. 2025.

[110] ——, "Quality management systems — fundamentals and vocabulary," International Organization for Standardization, Geneva, CH, ISO Standard 9000:2015, Sep. 2015.

[111] ——, "Information and documentation — foundation and vocabulary," International Organization for Standardization, Geneva, CH, ISO Standard 5127:2017, May 2017.

[112] ——, "Information technology — governance of IT for the organization," International Organization for Standardization, Geneva, CH, ISO/IEC Standard 38500:2024, Feb. 2024.

[113] ——, "Systems and software engineering — software life cycle processes," International Organization for Standardization, Geneva, CH, ISO/IEC/IEEE Standard 12207:2019, Nov. 2017.

[114] ——, "Space systems — programme management and quality — vocabulary," International Organization for Standardization, Geneva, CH, ISO Standard 10795:2019, Jul. 2019.

[115] ——, "Industrial automation systems and integration — product data representation and exchange — part 2: Vocabulary," International Organization for Standardization, Geneva, CH, ISO Standard 10303-2:2024, Feb. 2024.

[116] L. Wheatcraft, T. Katz, M. Ryan, and R. B. Wolfgang, "Needs and Requirements Manual," International Council on Systems Engineering, Tech. Rep. INCOSE-TP-2021-002-01, 2022.

[117] L. Wheatcraft and M. Ryan, "Guide to Writing Requirements," International Council on Systems Engineering, Tech. Rep. INCOSE-TP-2010-006-04, 2023.

[118] SEBoK contributors, "The guide to the systems engineering body of knowledge (SEBoK)," International Council on Systems Engineering, IEEE Systems Council, and The Systems Engineering Research System, Guide v2.11, Nov. 2024, url: https://sebokwiki.org/w/index.php?title=Guide_to_the_Systems_Engineering_Body_of_Knowledge_(SEBoK)&oldid=73360.

[119] S. R. Hirshorn, L. D. Voss, and L. K. Bromley, "NASA systems engineering handbook," NASA Technical Reports Server, National Aeronautics and Space Administration, Tech. Rep. NASA/SP-20166105 Rev 2, 2017.

[120] DAMA International, *DAMA-DMBOK: Data Management Body of Knowledge*, 2nd ed., D. Henderson and S. Earley, Eds. Technics Publications, Jul. 2017.

[121] M. Broy, H. Daembkes, and J. Sztipanovits, "Editorial to the theme section on model-based design of cyber-physical systems," *Software & Systems Modeling*, vol. 18, no. 3, pp. 1575–1576, 03 2018, doi: 10.1007/s10270-018-0670-9.

[122] J. Sztipanovits, X. Koutsoukos, G. Karsai, N. Kottenstette, P. Antsaklis, V. Gupta, B. Goodwine, J. Baras, and S. Wang, "Toward a science of cyber–physical system integration," *Proceedings of the IEEE*, vol. 100, no. 1, pp. 29–44, 2012, doi: 10.1109/JPROC.2011.2161529.

[123] J. S. Wheaton and D. R. Herber, "Digital requirements engineering with an INCOSE-derived SysML meta-model," in *The Proceedings of the 2024 Conference on Systems Engineering Research*. Springer Nature Switzerland, Mar. 2024, pp. 15–26, doi: 10.1007/978-3-031-62554-1_2.

[124] E. Ullman, *Life in Code: A Personal History of Technology*. Farrar, Straus and Giroux, 2017, p. 47.

[125] The Authors, "Model-based Structured Requirements," 2024, retrieved: May 20, 2025. url: https://github.com/danielrherber/model-based-structured-requirements.

[126] D. R. Herber and K. Eftekhari-Shahroudi, "Building a requirements digital thread from concept to testing using model-based structured requirements applied to thrust reverser actuation system development," in *Recent Advances in Aerospace Actuation Systems and Components*, Toulouse, France, Sep. 2023.

[127] D. R. Herber, J. B. Narsinghani, and K. Eftekhari-Shahroudi, "Model-based structured requirements in SysML," in *2022 IEEE International Systems Conference (SysCon)*. IEEE, 2022, pp. 1–8, doi: 10.1109/SysCon53536.2022.9773813.

[128] INCOSE, "INCOSE Systems Engineering Vision 2025," International Council on Systems Engineering, Technical Product, Jul. 2014.

[129] ——, "INCOSE Systems Engineering Vision 2035," International Council on Systems Engineering, Technical Product, 2021.

[130] E. B. Rogers III and S. W. Mitchell, "MBSE delivers significant return on investment in evolutionary development of complex SoS," *Systems Engineering*, vol. 24, no. 6, pp. 385–408, 2021, doi: 10.1002/sys.21592.

[131] P. J. Younse, J. E. Cameron, and T. H. Bradley, "Comparative analysis of a model-based systems engineering approach to a traditional systems engineering approach for architecting a robotic space system through knowledge categorization," *Systems Engineering*, vol. 24, no. 3, pp. 177–199, 2021, doi: 10.1002/sys.21573.

[132] A. M. Madni and S. Purohit, "Economic analysis of model-based systems engineering," *Systems*, vol. 7, no. 1, 2019, doi: 10.3390/systems7010012.

[133] E. R. Carroll and R. J. Malins, "Systematic literature review: How is model-based systems engineering justified?" Sandia National Lab, Tech. Rep. SAND-2016-2607, Mar. 2016, doi: 10.2172/1561164.

[134] V. Sundararajan, "Understanding NASA-ESA mars sample return (MSR) campaign concept by model-based systems engineering (MBSE) design and analysis," in *AIAA SCITECH 2022 Forum*, 2022, pp. 21–34, doi: 10.2514/6.2022-2134.

[135] O. Figueroa, S. Kearns, N. Boll, and J. Elbel, "Mars sample (MSR) independent review board-2 final report," 2023, retrieved: November 13, 2023. url: https://www.nasa.gov/wp-content/uploads/2023/09/msr-irb-report-final-copy-v3.pdf.

[136] J. M. Marlowe, C. L. Haymes, and P. L. Murphy, "NASA enterprise digital transformation initiative strategic framework & implementation approach," National Aeronautics and Space Administration, Technical Memorandum NASA/TM-20220018538, 2022, url: https://ntrs.nasa.gov/citations/20220018538.

[137] K. J. Weiland, "Future model-based systems engineering vision and strategy bridge for NASA," National Aeronautics and Space Administration, Tech. Rep. NASA/TM-20210014025, 2021, url: https://ntrs.nasa.gov/citations/20210014025.

[138] Office of the NASA Chief Engineer, "NASA digital engineering acquisition framework handbook," National Aeronautics and Space Administration, NASA Handbook NASA-HDBK-1004, 2020.

[139] J. Duprez, P. Paper, A. Fraj, L. Royer, and B. Petteys, "An approach to integrated digital requirements engineering," in *INCOSE International Symposium*, vol. 33, 2023, pp. 133–149, doi: 10.1002/iis2.13013.

[140] ISO, "Systems and software engineering — life cycle processes — requirements engineering," International Organization for Standardization, Geneva, CH, ISO/IEC/IEEE Standard 29148:2018, Nov. 2018.

[141] OMG, "OMG SysML version 1.7 beta 1," Object Management Group, Tech. Rep. formal/2022-08-02, 2022, retrieved: February 20, 2024. url: https://www.omg.org/spec/SysML/1.7/Beta1.

[142] M. Bajaj, S. Friedenthal, and E. Seidewitz, "Systems modeling language (SysML v2) support for digital engineering," *INSIGHT*, vol. 25, no. 1, pp. 19–24, 2022, doi: 10.1002/inst.12367.

[143] D. R. Call and D. R. Herber, "Applicability of the diffusion of innovation theory to accelerate model-based systems engineering adoption," *Systems Engineering*, vol. 25, no. 6, pp. 574–583, Nov. 2022, doi: 10.1002/sys.21638.

[144] OMG, "OMG SysML version 2.0 beta 2, specification – language," Object Management Group, Tech. Rep. ptc/24-02-03, 2024, retrieved: July 16, 2024. url: https://www.omg.org/spec/SysML.

[145] AeroSpace and Defence Industries Association of Europe, "ASD simplified technical english — international specification for the preparation of technical documentation in a controlled language," AeroSpace and Defence Industries Association of Europe, Brussels, Belgium, Specification ASD-STE100, Apr. 2022.

[146] T. V. Avdeenko and N. V. Pustovalova, "The ontology-based approach to support the requirements engineering process," in *2016 13th International Scientific-Technical Conference on Actual Problems of Electronics Instrument Engineering (APEIE)*, vol. 02, Oct. 2016, pp. 513–518, doi: 10.1109/APEIE.2016.7806406.

[147] J. Dick, L. Wheatcraft, D. Long, M. Ryan, J. Llorens, R. Zinni, C. Svensson, and S. D. Materiel, "Integrating requirement expressions with system models," in *INCOSE UK Annual Systems Engineering Conference*, University of Warwick, Coventry, England, United Kingdom, Nov. 2017.

[148] Y. Bernard, "Requirements management within a full model-based engineering approach," *Systems Engineering*, vol. 15, no. 2, pp. 119–139, 2012, doi: 10.1002/sys.20198.

[149] T. Gilb, "The use of planguage to improve requirement specifications," in *INCOSE International Symposium*, vol. 14, Jun. 2004, pp. 1604–1614, doi: 10.1002/j.2334-5837.2004.tb00598.x.

[150] A. Mavin and P. Wilkinson, "Big EARS (the return of "easy approach to requirements engineering")," in *2010 18th IEEE International Requirements Engineering Conference*. IEEE, 2010, pp. 277–282, doi: 10.1109/RE.2010.39.

[151] R. S. Carson, "Implementing structured requirements to improve requirements quality," in *INCOSE International Symposium*, vol. 25, 2015, pp. 54–67, doi: 10.1002/j.2334-5837.2015.00048.x.

[152] R. Carson, "Developing complete and validated requirements," INCOSE Seattle-Metropolitan Chapter Monthly Meeting, Jun. 2021, doi: 10.13140/RG.2.2.28526.74561.

[153] J. Dick, E. Hull, and K. Jackson, *Requirements Engineering*. Springer International Publishing, 2017.

[154] L. Wheatcraft, M. Ryan, J. Llorens, and J. Dick, "The need for an information-based approach for requirement development and management," in *INCOSE International Symposium*, vol. 29, Jul. 2019, pp. 1140–1157, doi: 10.1002/j.2334-5837.2019.00658.x.

[155] J. S. Wheaton and P. J. Younse, "Architecting the mars returned sample handling system-of-systems using agile MBSE," in *2025 IEEE Aerospace Conference*, Big Sky, MT, USA, Mar. 2025.

[156] P. J. Younse, J. Chesin, S. Gerdts, P. Phelps, O. R. Perez, J. Munger, J. S. Wheaton, H. Kelman, T. Kim, A. Kakarlapudi, and N. Hofer, "MSR returned sample handling and sample removal technology development," in *2025 IEEE Aerospace Conference*, Mar. 2025.

[157] No Magic, "Cameo Systems Modeler 2022x," Nov. 2023, retrieved: November 13, 2023. url: https://docs.nomagic.com/display/CSM2022xR2/2022x+Refresh2+Version+News.

[158] A. Salado, "Model-based requirements," in *Handbook of Model-Based Systems Engineering*. Springer, 2023, pp. 349–377, doi: 10.1007/978-3-030-93582-5_19.

[159] P. Wach and A. Salado, "The need for semantic extension of SysML to model the problem space," in *Recent Trends and Advances in Model Based Systems Engineering*. Springer, 2022, pp. 279–289, doi: 10.1007/978-3-030-82083-1_24.

[160] A. W. Wymore, *Model-Based Systems Engineering*, ser. Systems Engineering. Taylor & Francis, 1993, doi: 10.1201/9780203746936.

[161] P. Wach and A. Salado, "Can Wymore's mathematical framework underpin SysML? an initial investigation of state machines," *Procedia Computer Science*, vol. 153, pp. 242–249, 2019, doi: 10.1016/j.procs.2019.05.076.

[162] P. Micouin, "Toward a property based requirements theory: System requirements structured as a semilattice," *Systems Engineering*, vol. 11, no. 3, pp. 235–245, 2008, doi: 10.1002/sys.20097.

[163] C.-W. Lu, C.-H. Chang, W. C. Chu, Y.-W. Cheng, and H.-C. Chang, "A requirement tool to support model-based requirement engineering," in *2008 32nd Annual IEEE International Computer Software and Applications Conference*. IEEE, 2008, pp. 712–717, doi: 10.1109/COMPSAC.2008.232.

[164] R. Lorch, B. Meng, K. Siu, A. Moitra, M. Durling, S. Paul, S. C. Varanasi, and C. Mcmillan, "Formal methods in requirements engineering: Survey and future directions," in *Proceedings of the 2024 IEEE/ACM 12th International Conference on Formal Methods in Software Engineering (FormaliSE)*, 2024, pp. 88–99, doi: 10.1145/3644033.3644373.

[165] World Wide Web Consortium, "OWL 2 Web Ontology Language document overview," Dec. 2012, retrieved: June 20, 2024. url: https://www.w3.org/TR/owl2-overview/.

[166] A. Salado and P. Wach, "Constructing true model-based requirements in SysML," *Systems*, vol. 7, no. 2, p. 19, 2019, doi: 10.3390/systems7020019.

[167] A. Salado and N. Shadab, "A comparative experiment between textual requirements and model-based requirements on proxies for contractual safety," *Systems Engineering*, vol. 27, no. 3, pp. 556–569, 2024, doi: 10.1002/sys.21738.

[168] T. Weilkiens, *SYSMOD - The Systems Modeling Toolbox: Pragmatic MBSE with SysML*. MBSE4U, January 2020.

[169] C. A. R. Hoare, "An axiomatic basis for computer programming," *Communications of the ACM*, vol. 12, no. 10, pp. 576–580, Oct. 1969, doi: 10.1145/363235.363259.

[170] D. A. Wheeler, "Fully countering trusting trust through diverse double-compiling," Ph.D. dissertation, George Mason University, Dec. 02, 2009, doi: 10.13021/MARS/5014.

[171] D. G. Kourie and B. W. Watson, *The Correctness-by-Construction Approach to Programming*. Springer Berlin Heidelberg, 2012.

[172] R. Chapman, "Correctness by construction: a manifesto for high integrity software," in *Proceedings of the 10th Australian Workshop on Safety Critical Systems and Software - Volume 55*, ser. SCS '05. AUS: Australian Computer Society, Inc., 2006, pp. 43—46.

[173] K. Berkling, "Computer architecture for correct programming," in *Proceedings of the 5th Annual Symposium on Computer Architecture*, ser. ISCA '78. New York, NY, USA: Association for Computing Machinery, 1978, p. 78–84, doi: 10.1145/800094.803031.

[174] A. Vetter, "The matrix of convivial technology – assessing technologies for degrowth," *Journal of Cleaner Production*, vol. 197, pp. 1778–1786, 2018, technology and Degrowth. doi: 10.1016/j.jclepro.2017.02.195.

[175] L. M. Hively, F. T. Sheldon, and A. Squicciarini, "A vision for scalable trustworthy computing," *IEEE Security and Privacy*, 2010.

[176] C. Mundie, P. de Vries, P. Haynes, and M. Corwine, "Trustworthy computing," Microsoft, Tech. Rep., Oct. 2002.

[177] S. Lipner, "The trustworthy computing security development lifecycle," in *20th Annual Computer Security Applications Conference*. IEEE, 2004, pp. 2–13.

[178] M. Sahinoglu, *Trustworthy computing: Analytical and quantitative engineering evaluation*. John Wiley & Sons, 2007.

[179] J. J. Rajendran, O. Sinanoglu, and R. Karri, "Building trustworthy systems using untrusted components: A high-level synthesis approach," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 9, pp. 2946–2959, Oct. 2016, doi: 10.1109/TVLSI.2016.2530092.

[180] X. Cui, X. Zhang, H. Yan, L. Zhang, K. Cheng, Y. Wu, and K. Wu, "Toward building and optimizing trustworthy systems using untrusted components: A graph-theoretic perspective," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 5, pp. 1386–1399, May 2022, doi: 10.1109/TCAD.2021.3086765.

[181] E. Spafford, "Exploring grand challenges in trustworthy computing," 03 2004, url: https://nitrd.gov/Subcommittee/lsn/material/20040316_lsn_spafford.pdf.

[182] Computing Research Association *et al.*, "Four grand challenges in trustworthy computing," in *Proceedings of Second Conferences on Grand Research Challenges in Computer Science and Engineering*, 1100 17th Street, NW, Suite 507, Washington, D.C., USA 20036-4632, Nov. 2003, url: https://archive.cra.org/reports/trustworthy.computing.pdf.

[183] M. D. Watson, "Engineering elegant systems: Design at the system level," in *Penn State University Graduate Seminar*, no. M17-6300, 2017.

[184] I. Illich, *Tools for Conviviality*, ser. Open Forum. Harper & Row, 1973.

[185] S. Datskovskiy. (2010, Aug.) Seven laws of sane personal computing. Retrieved: June 29, 2025. url: http://www.loper-os.org/?p=284.

[186] C. Voinea, "Designing for conviviality," *Technology in Society*, vol. 52, pp. 70–78, 2018, technology and the Good Society. doi: 10.1016/j.techsoc.2017.07.002.

[187] J.-L. Voirin, *Model-based System and Architecture Engineering with the Arcadia Method*. ISTE Press - Elsevier, 2017.

[188] J. E. Bardram, S. Jeuris, and S. Houben, "Activity-based computing: computational management of activities reflecting human intention," *AI Magazine*, vol. 36, no. 2, pp. 63–72, 2015, doi: 10.1609/aimag.v36i2.2585.

[189] T. H. Nelson, "The heart of connection: Hypermedia unified by transclusion," *Commun. ACM*, vol. 38, no. 8, pp. 31–33, Aug. 1995, doi: 10.1145/208344.208353.

[190] Electronic Industries Alliance, "Configuration management standard," Electronic Industries Alliance, Standard SAE ANSI/EIA-649C, Feb. 7, 2019.

[191] L. Apvrille and Y. Roudier, "SysML-Sec: A SysML environment for the design and development of secure embedded systems," *Asia-Pacific Council on Systems Engineering (APCOSEC 2013)*, pp. 8–11, Aug. 2013, url: https://hal.telecom-paris.fr/hal-02288385.

[192] B. Sultan, L. Apvrille, and P. Jaillon, "Safety, security and performance assessment of security countermeasures with SysML-Sec," in *10th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2022)*, Feb. 2022, url: https://hal-emse.ccsd.cnrs.fr/emse-03559558.

[193] J. F. Pimentel, L. Murta, V. Braganholo, and J. Freire, "A large-scale study about quality and reproducibility of jupyter notebooks," in *2019 IEEE/ACM 16th international*

*conference on mining software repositories (MSR)*. IEEE, 2019, pp. 507–517, doi: 10.1109/MSR.2019.00077.

[194] R. Revere and J. Blustein, "Transhierarchy: A stable tree view with transclusion for hypertext navigation," in *Proceedings of the 3rd Workshop on Human Factors in Hypertext*, ser. HUMAN'20. New York, NY, USA: Association for Computing Machinery, 2020. article 6, doi: 10.1145/3406853.3434771.

[195] L. Bellamy, M. Carey, and J. Schlotfeldt, *DITA Best Practices: A Roadmap for Writing, Editing, and Architecting in DITA*, ser. IBM Press Series. IBM Press, 2012.

[196] N. Walsh and R. L. Hamilton, *DocBook 5: The Definitive Guide: The Official Documentation for DocBook*. O'Reilly Media, 2010.

[197] D. Verna, "Similarity problems in paragraph justification: An extension to the knuth-plass algorithm," in *Proceedings of the ACM Symposium on Document Engineering 2024*, ser. DocEng '24. New York, NY, USA: Association for Computing Machinery, 2024. article 17 (4 pages), doi: 10.1145/3685650.3685666.

[198] D. E. Knuth and M. F. Plass, "Breaking paragraphs into lines," *Software: Practice and Experience*, vol. 11, no. 11, pp. 1119–1184, 1981, doi: https://doi.org/10.1002/spe.4380111102.

[199] J. Backus, "Can programming be liberated from the von Neumann style? a functional style and its algebra of programs," *Communications of the ACM*, vol. 21, no. 8, pp. 613–641, 1978, doi: 10.1145/359576.359579.

[200] D. Chisnall, "C is not a low-level language: Your computer is not a fast PDP-11." *Queue*, vol. 16, no. 2, pp. 18–30, 2018, doi: 10.1145/3212477.3212479.

[201] Y. Lafont, "Interaction nets," in *Proceedings of the 17th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, Dec. 1989, pp. 95–108, doi: 10.1145/96709.96718.

[202] A. Radul, "Propagation networks: A flexible and expressive substrate for computation," Ph.D. dissertation, Massachusetts Institute of Technology, Nov. 3, 2009, url: http://hdl.handle.net/1721.1/49525.

[203] C. Whitby-Strevens, "The transputer," *ACM SIGARCH Computer Architecture News*, vol. 13, no. 3, pp. 292–300, Jun. 1985, doi: 10.1145/327070.327269.

[204] M. Naylor and C. Runciman, "The reduceron reconfigured and re-evaluated," *Journal of Functional Programming*, vol. 22, no. 4-5, pp. 574–613, 2012, doi: 10.1017/S0956796812000214.

[205] C. Newcombe, T. Rath, F. Zhang, B. Munteanu, M. Brooker, and M. Deardeuff, "How Amazon web services uses formal methods," *Commun. ACM*, vol. 58, no. 4, pp. 66–73, Mar. 2015, doi: 10.1145/2699417.

[206] OMG, "A UML profile for MARTE: Modeling and analysis of real-time embedded systems, beta 2," Object Management Group, Tech. Rep. ptc/2008-06-09, 2008, retrieved: July 18, 2025. url: https://www.omg.org/omgmarte/.

[207] M. Faugere, T. Bourbeau, R. De Simone, and S. Gerard, "MARTE: Also an UML profile for modeling AADL applications," in *12th IEEE International Conference on Engineering Complex Computer Systems (ICECCS 2007)*. IEEE, 2007, pp. 359–364, doi: 10.1109/ICECCS.2007.29.

[208] C. André, F. Mallet, and R. de Simone, "Modeling time(s)," in *Model Driven Engineering Languages and Systems*, G. Engels, B. Opdyke, D. C. Schmidt, and F. Weil, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 559–573.

[209] M. Ouimet and K. Lundqvist, "The timed abstract state machine language: Abstract state machines for real-time system engineering," *J.UCS (Annual print and CD-ROM archive ed.)*, vol. 14, no. 12, pp. 2007–2033, 2008, doi: 10.3217/jucs-014-12-2007.

[210] G. Papastergiou, G. Fairhurst, D. Ros, A. Brunstrom, K.-J. Grinnemo, P. Hurtig, N. Khademi, M. Tüxen, M. Welzl, D. Damjanovic *et al.*, "De-ossifying the internet transport layer: A survey and future perspectives," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 1, pp. 619–639, 2016, doi: 10.1109/COMST.2016.2626780.

[211] M. Ammar, "ex uno pluria: The service-infrastructure cycle, ossification, and the fragmentation of the internet," *ACM SIGCOMM Computer Communication Review*, vol. 48, no. 1, pp. 56–63, 2018, doi: 10.1145/3211852.3211861.

[212] H. Hammood, "An investigation to cybersecurity countermeasures for global internet infrastructure." Ph.D. dissertation, Bournemouth University, 2021, url: https://eprints.bournemouth.ac.uk/36020/.

[213] F. Douzet, L. Pétiniaud, L. Salamatian, K. Limonier, K. Salamatian, and T. Alchus, "Measuring the fragmentation of the internet: the case of the border gateway protocol (bgp) during the ukrainian crisis," in *2020 12th International Conference on Cyber Conflict (CyCon)*, vol. 1300. IEEE, 2020, pp. 157–182, doi: 10.23919/CyCon49761.2020.9131726.

[214] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, k. claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang, "Named data networking," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 66–73, Jul. 2014, doi: 10.1145/2656877.2656887.

[215] D. Barrera, L. Chuat, A. Perrig, R. M. Reischuk, and P. Szalachowski, "The SCION internet architecture," *Communications of the ACM*, vol. 60, no. 6, pp. 56–65, May 2017, doi: 10.1145/3085591.

[216] Z. Durumeric, F. Li, J. Kasten, J. Amann, J. Beekman, M. Payer, N. Weaver, D. Adrian, V. Paxson, M. Bailey *et al.*, "The matter of heartbleed," in *Proceedings of the 2014 conference on internet measurement conference*, 2014, pp. 475–488, doi: 10.1145/2663716.2663755.

[217] P. A. Grassi, M. E. Garcia, and J. L. Fenton, "Digital identity guidelines," *NIST Special Publication*, vol. 800, pp. 63–3, Jun. 2017, doi: 10.6028/NIST.SP.800-63-3.

[218] J. A. Rees, "A security kernel based on the lambda-calculus," Ph.D. dissertation, Massachusetts Institute of Technology, 1995, url: http://hdl.handle.net/1721.1/5944.

[219] T. Wallez, J. Protzenko, and K. Bhargavan, "Comparse: Provably secure formats for cryptographic protocols," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 564–578, doi: 10.1145/3576915.3623201.

[220] A. Mariano. (2016, May) Units. url: https://www.gnu.org/software/units/units.html.

[221] A. Eliasen, *Frink*, Dec. 2022, url: https://frinklang.org/.

[222] ISO, "Quantities and units," International Organization for Standardization, Geneva, CH, ISO/IEC/IEEE Standard 80000:2022, Dec. 2022.

[223] B. N. Taylor and A. Thompson, "Guide for the use of the international system of units (si)," US Department of Commerce, National Institute of Standards and Technology, Gaithersburg, MD 20899, Special Publication 811, Mar. 2008, url: https://www.nist.gov/pml/special-publication-811.

[224] D. B. Newell, E. Tiesinga *et al.*, "The international system of units (si)," US Department of Commerce, National Institute of Standards and Technology, Physical Measurement Laboratory, Gaithersburg, MD 20899, Special Publication 330, Aug. 2019, doi: 10.6028/NIST.SP.330-2019.

[225] ISO, "Programming languages — c," International Organization for Standardization, Geneva, CH, ISO/IEC Standard 9899:1999, Dec. 1999.

[226] High Order Language Working Group *et al.*, "Department of defense requirements for high order computer programming languages "STEELMAN"," US Department of Defense, Tech. Rep. AD-A059 444, Jun. 1978.

[227] J. Jakubovic, J. Edwards, and T. Petricek, "Technical dimensions of programming systems," *The Art, Science, and Engineering of Programming*, vol. 7, no. 3, Feb. 2023, doi: 10.22152/programming-journal.org/2023/7/13.

[228] N. Swamy, J. Weinberger, C. Schlesinger, J. Chen, and B. Livshits, "Verifying higher-order programs with the Dijkstra monad," in *Proceedings of the 34th annual ACM SIGPLAN conference on Programming Language Design and Implementation*, ser. PLDI '13, 2013, pp. 387–398, doi: 10.1145/2499370.2491978.

[229] E. C. Brady, "IDRIS —: systems programming meets full dependent types," in *Proceedings of the 5th ACM workshop on Programming languages meets program verification*, Jan. 2011, pp. 43–54, doi: 10.1145/1929529.1929536.

[230] E. Brady, "Idris, a general-purpose dependently typed programming language: Design and implementation," *Journal of functional programming*, vol. 23, no. 5, pp. 552–593, Sep. 2013, doi: 10.1017/S095679681300018X.

[231] N. Swamy, C. Hritcu, C. Keller, A. Rastogi, A. Delignat-Lavaud, S. Forest, K. Bhargavan, C. Fournet, P.-Y. Strub, M. Kohlweiss, J.-K. Zinzindohoué, and S. Zanella-Béguelin, "Dependent types and multi-monadic effects in F*," in *43rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*. ACM, Jan. 2016, pp. 256–270, doi: 10.1145/2837614.2837655.

[232] K. Bhargavan, B. Bond, A. Delignat-Lavaud, C. Fournet, C. Hawblitzel, C. Hritcu, S. Ishtiaq, M. Kohlweiss, R. Leino, J. Lorch, K. Maillard, J. Pang, B. Parno, J. Protzenko, T. Ramananandro, A. Rane, A. Rastogi, N. Swamy, L. Thompson, P. Wang, S. Zanella-Béguelin, and J.-K. Zinzindohoué, "Everest: Towards a verified, drop-in replacement

of HTTPS," in *2nd Summit on Advances in Programming Languages*, May 2017, doi: 10.4230/LIPIcs.SNAPL.2017.1.

[233] A. Fromherz, A. Rastogi, N. Swamy, S. Gibson, G. Martínez, D. Merigoux, and T. Ramananandro, "Steel: Proof-oriented programming in a dependently typed concurrent separation logic," in *25th ACM SIGPLAN International Conference on Functional Programming (ICFP)*, Aug. 2021, doi: 10.1145/3462300.

[234] A. Arasu, B. Chandramouli, J. Gehrke, E. Ghosh, D. Kossmann, J. Protzenko, R. Ramamurthy, T. Ramananandro, A. Rastogi, S. Setty, N. Swamy, A. van Renen, and M. Xu, "Fastver: Making data integrity a commodity," in *Proceedings of the 2021 International Conference on Management of Data*, ser. SIGMOD '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 89–101, doi: 10.1145/3448016.3457312.

[235] A. Reitz, A. Fromherz, and J. Protzenko, "Starmalloc: Verifying a modern, hardened memory allocator," *Proc. ACM Program. Lang.*, vol. 8, no. OOPSLA2, article 333 (30 pages), Oct. 2024, doi: 10.1145/3689773.

[236] G. Martínez, D. Ahman, V. Dumitrescu, N. Giannarakis, C. Hawblitzel, C. Hritcu, M. Narasimhamurthy, Z. Paraskevopoulou, C. Pit-Claudel, J. Protzenko, T. Ramananandro, A. Rastogi, and N. Swamy, "Meta-F*: Proof automation with SMT, tactics, and metaprograms," in *28th European Symposium on Programming (ESOP)*. Springer, Apr. 2019, pp. 30–59, doi: 10.1007/978-3-030-17184-1_2.

[237] C. Angiuli, K.-B. Hou (Favonia), and R. Harper, "Cartesian cubical computational type theory: Constructive reasoning with paths and equalities," in *27th EACSL Annual Conference on Computer Science Logic (CSL 2018)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), D. Ghica and A. Jung, Eds., vol. 119. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018, pp. 6:1–6:17, doi: 10.4230/LIPIcs.CSL.2018.6.

[238] T. Univalent Foundations Program, *Homotopy Type Theory: Univalent Foundations of Mathematics*. Institute for Advanced Study: https://homotopytypetheory.org/book, 2013.

[239] P. A. Karger and R. R. Schell, "Multics security evaluation: Vulnerability analysis," in *Proceedings of the 18th Annual Computer Security Applications Conference, 2002*. IEEE, 2002, pp. 127–146, doi: 10.1109/CSAC.2002.1176286.

[240] K. Thompson, "Reflections on trusting trust," *Communications of the ACM*, vol. 27, no. 8, pp. 761–763, 1984, doi: 10.1145/358198.358210.

[241] J. Nieuwenhuizen, "GNU Mes - full source bootstrap," 02 6–7, 2021, url: https://github.com/oriansj/talk-notes/raw/master/fosdem_2021/gnu_mes_fosdem21.pdf.

[242] L. Courtès, "Building a secure software supply chain with GNU Guix," *The Art, Science, and Engineering of Programming*, vol. 7, no. 1, 06 15, 2022, doi: 10.22152/programming-journal.org/2023/7/1.

[243] ANSI Technical Committee X3J14, "Programming languages: Forth," American National Standards Institute, Inc., Standard X3.215-1994, Mar. 24, 1994.

[244] J. N. Shutt, "Fexprs as the basis of lisp function application or $vau: the ultimate abstraction," Ph.D. dissertation, Worcester Polytechnic Institute, 2010.

[245] J. C. Davis, "A self-verifying theorem prover," Ph.D. dissertation, The University of Texas at Austin, 2009.

[246] J. Davis and M. O. Myreen, "The reflective milawa theorem prover is sound (down to the machine code that runs it)," *Journal of Automated Reasoning*, vol. 55, no. 2, p. 117–183, Jun. 2015, doi: 10.1007/s10817-015-9324-6.

[247] J. S. Moore, *The Correctness of Piton on FM9001*. Dordrecht: Springer Netherlands, 1996, pp. 79–95, doi: 10.1007/978-0-585-33654-1_6.

[248] B. C. Brock and W. A. Hunt, Jr., "An overview of the formal specification and verification of the FM9001 microprocessor," Computational Logic, Inc., Tech. Rep., 1994.

[249] B. C. Brock, W. A. Hunt, and M. Kaufmann, "The FM9001 microprocessor proof," Computational Logic Inc., Tech. Rep., 1994.

[250] A. Haas, A. Rossberg, D. L. Schuff, B. L. Titzer, M. Holman, D. Gohman, L. Wagner, A. Zakai, and J. Bastien, "Bringing the web up to speed with WebAssembly," *SIGPLAN Not.*, vol. 52, no. 6, p. 185–200, Jun. 2017, doi: 10.1145/3140587.3062363.

[251] World Wide Web Consortium, "WebAssembly specification," 06 16, 2025, retrieved: June 18, 2025. url: https://webassembly.github.io/spec/core/.

[252] K. Bhargavan, J. Protzenko, A. Rossberg, and D. Stefan, "Foundations of webassembly (dagstuhl seminar 23101)," *Dagstuhl Reports*, vol. 13, no. 3, pp. 1–16, 2023, doi: 10.4230/DagRep.13.3.1.

[253] R. Tronçon, "WAForth," 09 2024, retrieved: June 18: 2025. url: https://github.com/remko/waforth.

[254] N. Vallet, D. Michonneau, and S. Tournier, "Toward practical transparent verifiable and long-term reproducible research using Guix," *Scientific Data*, vol. 9, no. 1, p. 597, 2022, doi: 10.1038/s41597-022-01720-9.

[255] B. Mesmer, D. Mckinney, M. Watson, and A. M. Madni, "Transdisciplinary systems engineering approaches," in *Recent Trends and Advances in Model Based Systems Engineering*, A. M. Madni, B. Boehm, D. Erwin, M. Moghaddam, M. Sievers, and M. Wheaton, Eds. Cham: Springer International Publishing, 2022, pp. 579–590, doi: 10.1007/978-3-030-82083-1_49.

[256] J. S. Wheaton, "Seamless digital engineering reference architecture," 2025, url: https://github.com/systems-praxis/seamless-digital-engineering-reference-architecture.

[257] R. L. Ackoff, "The future of operational research is past," *Journal of the Operational Research Society*, vol. 30, no. 2, pp. 93–104, 1979, doi: 10.1057/jors.1979.22.

[258] C. Alpaslan and I. I. Mitroff, *Swans, Swine, and Swindlers: Coping with the Growing Threat of Mega-Crises and Mega-Messes*, ser. High Reliability and Crisis Management. Stanford University Press, 2011, p. 16.

[259] R. Constable, "Computational type theory," *Scholarpedia*, vol. 4, no. 2, p. 7618, 2009, doi: 10.4249/scholarpedia.7618.

[260] M. Bickford and R. L. Constable, "A logic of events," Cornell University, Tech. Rep. TR2003-1893, Mar. 7, 2003.

[261] X. Liu, C. Kreitz, R. van Renesse, J. Hickey, M. Hayden, K. Birman, and R. Constable, "Building reliable, high-performance communication systems from components," in *Proceedings of the Seventeenth ACM Symposium on Operating Systems Principles*, ser. SOSP '99. New York, NY, USA: Association for Computing Machinery, 1999, p. 80–92, doi: 10.1145/319151.319157.

[262] D. A. Wheeler, "Countering trusting trust through diverse double-compiling," in *21st Annual Computer Security Applications Conference (ACSAC'05)*. IEEE, Dec. 2005, pp. 33–48, doi: 10.1109/CSAC.2005.17.