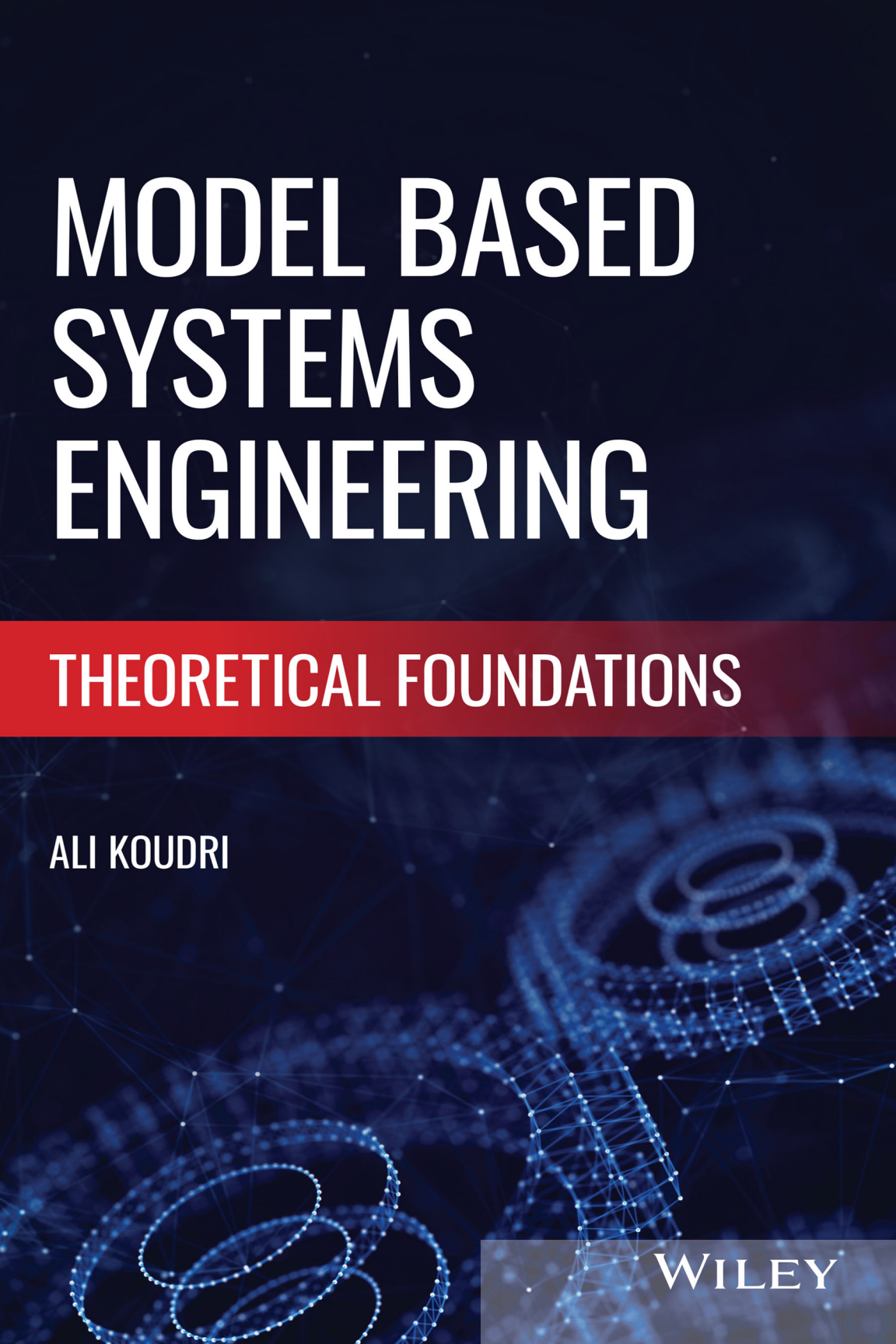
MODEL BASED SYSTEMS ENGINEERING

THEORETICAL FOUNDATIONS

ALI KOUDRI



WILEY

Model Based Systems Engineering

Model Based Systems Engineering

Theoretical Foundations

Ali Koudri

WILEY

Copyright © 2026 by John Wiley & Sons, Inc. All rights reserved, including rights for text and data mining and training of artificial intelligence technologies or similar technologies.

Published by John Wiley & Sons, Inc., Hoboken, New Jersey.
Published simultaneously in Canada.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning, or otherwise, except as permitted under Section 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 750-4470, or on the web at www.copyright.com. Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, (201) 748-6011, fax (201) 748-6008, or online at http://www.wiley.com/go/permission.

Trademarks: Wiley and the Wiley logo are trademarks or registered trademarks of John Wiley & Sons, Inc. and/or its affiliates in the United States and other countries and may not be used without written permission. All other trademarks are the property of their respective owners. John Wiley & Sons, Inc. is not associated with any product or vendor mentioned in this book.

Limit of Liability/Disclaimer of Warranty: While the publisher and the authors have used their best efforts in preparing this work, including a review of the content of the work, neither the publisher nor the authors make any representations or warranties with respect to the accuracy or completeness of the contents of this work and specifically disclaim all warranties, including without limitation any implied warranties of merchantability or fitness for a particular purpose. No warranty may be created or extended by sales representatives, written sales materials or promotional statements for this work. The fact that an organization, website, or product is referred to in this work as a citation and/or potential source of further information does not mean that the publisher and authors endorse the information or services the organization, website, or product may provide or recommendations it may make. This work is sold with the understanding that the publisher is not engaged in rendering professional services. The advice and strategies contained herein may not be suitable for your situation. You should consult with a specialist where appropriate. Further, readers should be aware that websites listed in this work may have changed or disappeared between when this work was written and when it is read. Neither the publisher nor authors shall be liable for any loss of profit or any other commercial damages, including but not limited to special, incidental, consequential, or other damages.

For general information on our other products and services or for technical support, please contact our Customer Care Department within the United States at (800) 762-2974, outside the United States at (317) 572-3993 or fax (317) 572-4002.

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print may not be available in electronic formats. For more information about Wiley products, visit our web site at www.wiley.com.

Library of Congress Cataloging-in-Publication Data

Names: Koudri, Ali author
Title: Model based systems engineering : theoretical foundations / Ali Koudri.
Description: Hoboken, New Jersey : Wiley Blackwell, [2026] | Includes bibliographical references and index.
Identifiers: LCCN 2025028334 | ISBN 9781394345076 (hardback) | ISBN 9781394345090 (adobe pdf) | ISBN 9781394345083 (epub) | ISBN 9781394345106 (Obook)
Subjects: LCSH: Systems engineering
Classification: LCC TA168 .K687 2026
LC record available at https://lccn.loc.gov/2025028334

Cover Design: Wiley
Cover Image: © Dmitry/stock.adobe.com

Contents

Preface *ix*
Foreword *xi*
Acronyms *xiii*
Introduction *xv*

1 The Start of the Journey *1*
1.1 A Difficult Start *1*
1.2 The Power of Abstraction *2*
1.3 The Rise of MBSE *6*
1.4 The Pillars of MBSE *9*
1.5 The Ingredients of MBSE *12*
1.6 Comparison with a Traditional Approach *14*
1.7 Some Success Stories *16*
1.8 Key Takeaways and Considerations *17*
References *20*

2 System Engineering Principles *23*
2.1 Preliminary Definitions *23*
2.2 System Definitions and Taxonomy *24*
2.3 Systems of Systems *28*
2.4 Ensuring System Consistency *31*
2.5 Managing System Complexity *33*
2.6 System Lifecycle *35*
2.7 Historical Perspective *37*
2.8 For a System Thinking *39*
2.9 System Engineering Definition *42*
2.10 System Engineering Issues *43*
2.11 The Role of a Systems Engineer *45*

2.12 Relationship with Requirements Engineering *49*
2.13 Key Considerations in SE *53*
2.14 Transitioning to MBSE *63*
2.15 Conclusion *64*
References *67*

3 Modeling Principles *73*
3.1 Systems, Languages, and Models *73*
3.2 Models *79*
3.3 Languages *86*
3.4 Formalization Principles *90*
3.5 The Power of Modeling *94*
3.6 Models and Semantics *97*
3.7 The Different Types of Model *99*
3.8 Models and Diagrams *102*
3.9 System and Viewpoints *105*
3.10 Interface Design and Management *109*
3.11 Modeling Standards and Tools *115*
3.12 Modeling Requirements *118*
3.13 Conclusion *120*
References *123*

4 Modeling Paradigms *127*
4.1 Structural Models *127*
4.2 Behavioral Models *137*
4.3 Agent-based Modeling *161*
4.4 Beyond Structure and Behavior *166*
4.5 Modeling the System Environment *171*
4.6 Conclusion *175*
References *178*

5 Model Verification and Validation *183*
5.1 Verification *183*
5.2 Validation *188*
5.3 The Evolution of V&V *193*
5.4 V&V Concerns *197*
5.5 Theoretical Bases *200*
5.6 Motivations for V&V Models *203*
5.7 Techniques of V&V in MBSE *205*
5.8 Standards and Good Practices *212*
5.9 Closing Design Space Exploration *215*
5.10 Reducing Costs and Delays *216*

5.11 Conclusion *219*
References *221*

6 System Architecting Principles *225*
6.1 Fundamental Principles *225*
6.2 Types of System Architectures *228*
6.3 Open Architectures *235*
6.4 Architectural Patterns and Tactics *239*
6.5 Evaluating and Updating System Architectures *241*
6.6 Linking MBSE to Architectural Concerns *245*
6.7 Conclusion *251*
References *253*

7 Methodological Bases *257*
7.1 Preliminary Considerations *257*
7.2 Methodological Foundations *262*
7.3 Formalizing Processes *267*
7.4 Modeling Processes *270*
7.5 Process Orchestration *272*
7.6 Model Transformations *274*
7.7 MBSE and Project Management *279*
7.8 Evaluating the Effectiveness of MBSE *282*
7.9 MBSE Adoption: Challenges and Limitations *285*
7.10 A Software Development Process Example *291*
7.11 Conclusion *295*
References *297*

8 Case Study *301*
8.1 Overview *301*
8.2 Project Overview and Motivation *302*
8.3 Blending "Meet-in-the-middle" and AI *304*
8.4 The Arcadia Methodology *305*
8.5 AI-powered Scenario Management *307*
8.6 Formal Checking and Simulations *311*
8.7 Decision-making and Optimization *314*
8.8 Evaluation and the Path Forward *317*
8.9 Conclusion *319*

9 Conclusion *321*

Index *323*

Preface

Model-based systems engineering (MBSE) has changed the way we practice systems engineering, affecting the design, development, and management of complex systems. For over 20 years, I have been part of the MBSE community and have seen it grow, noting the changes and new ideas that have emerged.

During my experience, I have read many books about MBSE that provide useful insights. However, I often noticed a lack of a comprehensive resource linking theory to practice, with practical examples to illustrate MBSE applications. This understanding motivated me to write the book I wished existed when I started learning about MBSE. I want to create a resource for readers at different levels, from beginners wanting to understand the basics to experienced professionals wanting to deepen their knowledge.

The goal of this first book is to make the concepts of MBSE understandable for a wide audience, including students, professionals, researchers, and decision-makers; the very practical aspects will be the subject of another book. Through this book, I aim to share complex ideas in a straightforward and engaging way, using comparisons, images, and real-life examples to facilitate understanding. It is essential to note that this book does not favor specific MBSE methodologies or tools. Instead, it is designed to help you grasp the fundamental ideas and principles that underpin these methodologies. By understanding these fundamentals, you can avoid potential mistakes and improve your creativity.

The writing of this book is a reflection of my dedication to MBSE and my aim to share the knowledge and experience I have gained over the course of my career. I truly hope that readers will find in these pages the same joy and interest that I experienced in writing them.

Foreword

Technology plays a key role in humankind's fervent and ceaseless striving to improve safety, comfort, and efficiency. This has, in turn, resulted in an accelerating progression of technological innovation, commencing with primitive hand tools all the way through to modern, highly sophisticated software-driven systems. As the capabilities of these technological marvels become increasingly more elaborate, so do our appetites for ever more complex automated support systems: autonomous vehicles, "intelligent" robots, "smart" homes, and even "smart" cities.

The cost of this technological progress is a commensurate, almost exponential increase in the complexity of our technologies – complexity that can be, and often is, overwhelming. The modern discipline of systems engineering has emerged as a means of coping with this phenomenon. Although, as the author of this volume points out, system engineering is much older than System Engineering, starting with ancient "mega projects," such as the Pyramids of Ancient Egypt, it is only within the last seven decades or so that the problem of constructing and operating complex systems has evolved into a fully-fledged technical discipline in its own right.

This book is a comprehensive and highly readable introduction to the latest and most advanced form of systems engineering – model-based systems engineering (MBSE). This variant of systems engineering is clearly distinguished by the central and essential role that models and modeling play in the design of today's complex systems.

Although one might expect that the author, himself an experienced practitioner of MBSE, would be naturally inclined to present a glowing and uncritical panegyric of his specialty, this is not the case. Instead, what we encounter in the text is a "warts and all" objective view; that is, in addition to the primary concepts, principles, and methods of MBSE, the various technical and socio-cultural

challenges and unresolved issues facing practitioners are also presented and discussed in depth. Moreover, a detailed discussion is included of the roles to be played by the most recent technological advances in machine learning and artificial intelligence.

Numerous real-world examples, taken from diverse domains, such as aerospace, automotive, and defense, are used to illustrate and make concrete many of the key aspects of MBSE. Moreover, in the spirit of Aristotle, who said that "to understand something we must learn how it came to be," the author provides the reader with a broader and deeper historical perspective on some of the key aspects of the discipline. In my view, this is extremely valuable, since it will help readers create a better-informed and more complete understanding of this complex and important subject.

I believe that this book is going to be useful not only to those who are looking for an introduction to modern systems engineering, but also to practitioners as a high-level reference manual that can provide a meaningful perspective on their work in critical phases of their work.

Ottawa, Canada
18 January 2025

Bran Selić

Acronyms

MBSE	Model-based Systems Engineering
UML	Unified Modeling Language
SysML	Systems Modeling Language
MDA	Model-driven Architecture
SEBoK	Systems Engineering Body of Knowledge
BPMN	Business Process Modeling and Notation
OMG	Object Management Group
MBSA	Model-based Safety Analysis
FSMs	Finite-state Machines
SDF	Synchronous Data Flow
KPNs	Kahn Process Networks
IVVQ	Integration, Verification, Validation, and Qualification
INCOSE	International Council on Systems Engineering
ABM	Agent-based Modeling
SoS	System of Systems
IEEE	Institute of Electrical and Electronics Engineers
SOA	Service-oriented Architecture
CIM	Computation Independent Model
PIM	Platform Independent Model
PDM	Platform Description Model
PSM	Platform Specific Model

Introduction

Imagine a world where complex systems, like spacecraft and smart cities, are made with great accuracy and effectiveness in a limited amount of time. A place where experts work together easily, using their combined skills and exchanging ideas through an approach that is both rigorous and flexible. This is what model-based systems engineering (MBSE) promises, changing the way we tackle problems related to the design and development of complex systems in a complex environment.

From what I have learned, MBSE is not just tools and methods; it is about a change in the way we think, an innovative way of seeing the world, and the role we play in it. In this book, we will take a look at this fascinating world, reviewing its main concepts, examining its benefits and drawbacks, and seeing how it can help us to deal with the challenges of today's systems engineering.

Whether you are an expert or just curious, this book will give you the insights and understanding to harness MBSE and influence the future of complex system design. So, let us start this journey together, discovering what MBSE holds and seeing how to use its strengths for the challenges ahead. The realm of complex systems is here, and with MBSE as our guide, we are prepared to begin a path of exploration, creativity, and change. The content of this book is organized as follows:

- **Chapter 1**: This chapter highlights the importance of MBSE in addressing the challenges of designing complex systems and outlines the key takeaways and considerations for successful MBSE adoption.
- **Chapter 2**: Establishing the groundwork for understanding systems engineering. This chapter explores definitions, taxonomy, and historical context. It discusses the importance of a systems approach and systems thinking in managing complexity and ensuring system consistency and coherence.

- **Chapter 3**: Delving into the core principles of modeling in the context of MBSE, this chapter examines the relationships between systems, languages, and models. It explores different types of models used in MBSE, the role of languages in modeling, the concept of viewpoints, and the benefits of formalization.
- **Chapter 4**: This chapter provides a comprehensive overview of modeling paradigms used in MBSE, including structural models, behavioral models, agent-based models, and beyond. It explores various modeling techniques and their applications in system design and analysis.
- **Chapter 5**: Focusing on the critical processes of verification and validation in MBSE, this chapter examines methodologies and techniques used to ensure that models accurately represent the intended system and meet stakeholder requirements.
- **Chapter 6**: Focusing on the principles and practices of system architecting within MBSE, this chapter examines different types of system architectures, architectural patterns and tactics, and the evaluation and evolution of system architectures over time.
- **Chapter 7**: This chapter explores the methodological foundations of MBSE, focusing on process formalization, orchestration, and model transformations. It discusses the importance of a well-defined MBSE process, the role of tools and technologies, and the integration of MBSE with project management practices.
- **Chapter 8**: The case study chapter showcases the practical application of an AI-enhanced MBSE methodology in the EVA project, aimed at designing an autonomous mobility service. It illustrates how integrating AI into MBSE can address traditional challenges and drive innovation in complex systems engineering.

By the end of this book, you will have gained a comprehensive understanding of MBSE's theoretical foundations and its potential applications. You will be equipped with the knowledge and insights to leverage the power of models, methodologies, and tools to design, develop, and manage complex systems effectively.

1

The Start of the Journey

1.1 A Difficult Start

1.1.1 The Initial Foray

My experience with model-based systems engineering (MBSE) started more than 20 years ago. I was curious about the complex interplay between design and execution in the world of embedded real-time systems. With some knowledge of Unified Modeling Language (UML), I began this journey but soon encountered a gap between the theoretical models and the practical embedded code. Moving from high-level concepts to low-level implementation was more difficult than I had expected. Indeed, I did not expect this path would lead me to question systems engineering to such an extent, and that it would change the way I think about, design, and develop complex systems.

1.1.2 A Turning Point

In 2004, I went back to modeling, focusing on the development of interface code that seamlessly links software and hardware parts. The technical project I was working on definitely opened the doors to the MBSE field. Working on model transformations to produce C code from UML models, I began to understand the capabilities and limits of MBSE tools. This realization fueled my motivation to embark on a thesis aimed at closing the gap between abstract models and actual implementation.

1.1.3 Expanding the Scope

Writing my thesis was a valuable experience, even if it started with certain limitations. My initial efforts were mainly technical, relying on ideas that later seemed to limit my work. However, this stage of focused effort was crucial, pushing me to question my beliefs and deepen my understanding of MBSE.

Digging deeper, I discovered that MBSE is actually a complex concept, with interpretations varying depending on the application domain (Estefan et al. 2007). This insight prompted me to investigate the theoretical roots of MBSE, exploring areas such as philosophy, linguistics, epistemology, mathematics, and sociology. This multifaceted approach broadened my vision, going far beyond my original technical focus and showcasing the true complexity of this fascinating field.

1.1.4 Embracing the Journey

Reflecting on my journey, I realize that my early difficulties were not failures but opportunities for growth. They taught me the value of questioning my assumptions, considering different perspectives, and recognizing the interdisciplinary nature of MBSE. These insights provided me with a solid base for understanding the complexities of system design and using the full potential of model-based approaches. Although the journey was complex, it led me to a deeper appreciation of the transformative possibilities of MBSE.

1.2 The Power of Abstraction

1.2.1 The Need for a Paradigm Shift

My initial approach to MBSE was paved with good intentions, but quickly showed a significant mistake. Indeed, I focused on using models mainly to automate implementation and did not fully realize their true potential for reasoning and decision-making. The frequent mistakes and inconsistencies that came up during code execution highlighted the flaws in my approach. It became clear that the quality of the generated code depends greatly on the quality of the underlying design and specifications. I realized that code is not just a simple translation of requirements but a complex mixture of business needs and technical factors. It is this complex relationship that makes debugging very difficult, often requiring a return to the initial design to fix problems.

1.2.2 The Role of Abstraction

This reflection led me to consider the key importance of abstraction in MBSE. Building high-level, abstract models that capture the essence of the system is crucial. These models act as a guide, ensuring that specifications correctly

represent the desired functions and behaviors, thus reducing confusion and promoting a shared understanding among stakeholders (Estefan et al. 2007; Friedenthal et al. 2014; Madni and Sievers 2018). The real challenge lies in gathering and understanding the real needs and requirements of stakeholders. Good communication and cooperation are essential in this process, and MBSE offers a framework to support this exchange by providing a common language and tools for requirements gathering, analysis, and validation.

By considering models as tools for reasoning and decision-making, MBSE encourages a more interactive and cooperative approach to systems thinking and engineering. Stakeholders are involved in the modeling process, providing feedback to refine specifications and ensure they meet their needs. This iterative process allows for the early finding and fixing of potential problems, reducing the risk of expensive rework later. The clarity and consistency within MBSE, maintained by a clear connection between models, specifications, and implementation, help to improve the understanding of the development lifecycle. The ability to track changes and refer to a single source of truth improves communication and understanding within the team.

1.2.3 The Modeling Benefits

Ultimately, my experience with MBSE showed that its real strength is not just in automation but in fostering better reasoning, decision-making, and communication throughout the systems engineering process. By emphasizing the creation of high-quality, abstract models that accurately represent stakeholder needs and expectations, MBSE enables the development of reliable, efficient, and purpose-driven systems. However, the success of MBSE relies on effective communication, collaboration, and a deep understanding of the problem. Only by focusing on these aspects can we truly realize the transformative potential of MBSE and create sustainable systems.

Figure 1.1 shows how abstraction works in MBSE with a comprehensive view of a library management system. This figure highlights how MBSE helps to understand the whole picture of complex systems, connecting abstract needs to actual implementations. The figure shows four interrelated areas: Need Space, Decision Space, Development Space, and Usage Space. These areas enable tracking the system development process from ideas to actual use.

The *Need Space* captures the requirements for the library system. This view formalizes stakeholder needs and provides a clear starting point for the whole system engineering process.

The *Decision Space* aims to analyze design choices. By using different modeling representations, such as use case diagrams, class diagrams, and state machines, engineers can assess various alternatives and choose wisely those that best meet

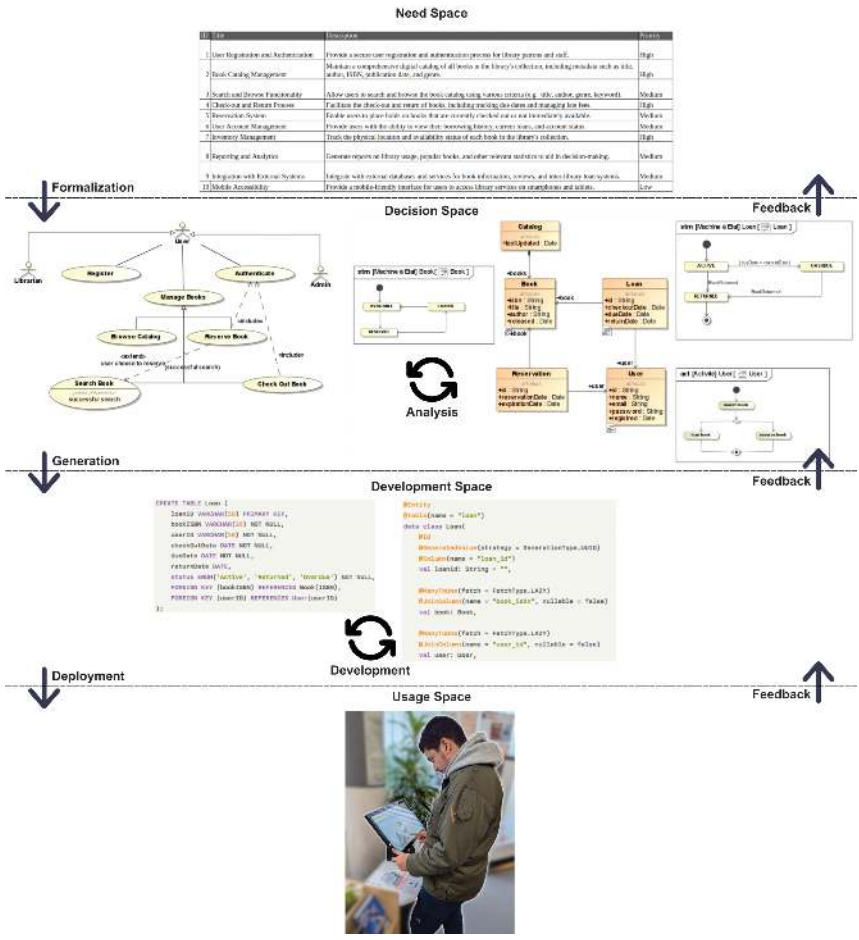


Figure 1.1 Illustration of the power of abstraction in MBSE.

the needs of stakeholders. The iterative nature of modeling allows for early detection and fixing of potential problems, reducing costly rework later.

The *Development Space* explains how abstract models are translated into actual implementations. The figure includes database designs and code segments sourced from the concept models, highlighting MBSE's ability to support model transformations. This maintains alignment between design and implementation, boosting the reliability of the system engineering.

Lastly, the *Usage Space* shows the system's operational context, emphasizing that MBSE places usage at the heart of its concerns.

The two-way arrows between these spaces highlight the active and iterative nature of MBSE. They represent the continuous flow of information and feedback, allowing continuous improvement and adjustment of the system to changing needs and context.

By providing a shared language and a set of linked models, MBSE enhances communication and teamwork among different stakeholders. It creates a mutual understanding that transcends various domains, fostering joint reasoning about the system under study. Additionally, the traceability of this model-centric approach streamlines the analysis of impacts and decisions throughout the life of the system. Changes in one space can easily be traced, and their impacts assessed in other spaces, keeping the entire system aligned.

In summary, Figure 1.1 captures how MBSE uses abstraction to facilitate better reasoning, decision-making, and communication during the systems engineering process. By emphasizing high-quality models that reflect stakeholder needs and system behavior, MBSE leads to the development of systems that are reliable, efficient, and suitable for their intended use. This in-depth model-based approach greatly improves systems engineering, increasing system quality and reliability while promoting a more effective and cooperative development environment. The effectiveness of MBSE, as shown in this example, depends strongly on good communication, teamwork, and a deep and common understanding of the problem across all areas of system development.

Exercise: Identifying and resolving inconsistencies in system models

Examine the figure of the library management system, which includes use case diagrams, class diagrams, state machines, and database schemas.

- Identify discrepancies between models. For instance, ensure that all elements in the use case diagram are represented in the class diagram, if relevant.
- Verify that transitions in the state machine align with the operations defined in the class diagram.
- Confirm that the database schema supports the entities and relationships depicted in the class diagram.

Consider two levels of alignment in this diagram: *cognitive alignment* between your understanding and what you've expressed, and *semantic alignment* between what you've expressed and the resulting code.

Propose solutions to improve the identified inconsistencies.

1.3 The Rise of MBSE

1.3.1 Historical Context

Efforts to create complex systems are a significant part of human history, starting with ancient societies. Great structures like the pyramids show early attempts to manage large complex projects. The construction of huge cathedrals in medieval Europe also highlights the human urge to design complex systems (Fitchen 1981). However, formal complex systems engineering only developed in the twentieth century, driven by the space race (Johnson 2006). The Systems Engineering Body of Knowledge (SEBoK)'s survey of the history of systems engineering highlights this development, showing a growing recognition of the need for a comprehensive approach to handling modern system complexities (SEBoK Editorial Board 2024).

1.3.2 Limitations of Document-centric Approaches

Historically, complex system development relied heavily on documents. Requirements, contracts, and guidelines were mostly shared through written documents. As noted by the SEBoK, this document-centric approach creates difficulties when several experts work together on a design. The lack of clarity in natural language can cause confusion and inconsistencies, making it hard for stakeholders to reach a common understanding (Grady 2013; SEBoK Editorial Board 2024). The complex nature of these systems, often related to organizational structures, increases the need for a more cautious approach to finding and fixing inconsistencies that might lead to serious issues.

1.3.3 The Emergence of Formal Models

The drawbacks of document-centric approaches have led to the use of formal models to gain a better grasp of systems and their environments. Formal models provide a solid way to record and share design alternatives, merging different expert viewpoints and allowing collective reasoning about the system (SEBoK Editorial Board 2024). By using a common modeling language and shared storage, individuals from various fields can work together more effectively, ensuring that the system design is aligned and consistent (Estefan et al. 2007; Ramos et al. 2011).

1.3.4 The Essence of MBSE

This model-based teamwork is at the heart of the MBSE approach. It promotes the use of models as the main tool for sharing information and making decisions throughout the system's lifecycle, overcoming the limits of document-centric approaches (Friedenthal et al. 2014; SEBoK Editorial Board 2024). The MBSE

approach leads to a more organized and systematic way of designing complex systems, facilitating early detection of inconsistencies, better traceability, and improved communication among stakeholders (International Council on Systems Engineering [INCOSE] 2023). This aligns well with the SEBoK's emphasis on maintaining system consistency and coherence as key objectives in systems engineering (SEBoK Editorial Board 2024).

1.3.5 The MBSE Journey

The beginnings of MBSE date back to the 1990s, when systems became more complex in many industries. Traditional document-centric approaches proved very limited in dealing with these complex systems, leading to the emergence of MBSE as a solution. MBSE aims to build models that capture various aspects of the system, including requirements, behavior, and architecture (Madni and Sievers 2018; Ramos et al. 2011). The problems posed by document-centric approaches and the importance of moving to MBSE to fully benefit from a more rigorous systems approach have been extensively discussed by the SEBoK Editorial Board (2024). The foundational concepts for MBSE were developed by early contributors such as A. Wayne Wymore, who notably coined the term "MBSE" in his influential work (Wymore 2018). The 2006 standardization of Systems Modeling Language (SysML) helped boost the use of MBSE (Friedenthal et al. 2014). In September 2007, the "MBSE 2020 Vision" by the INCOSE played a key role in disseminating and promoting the MBSE approach. This vision document, crafted by INCOSE leaders and various industry contributors, acted as a guiding light for the continuous growth of MBSE and systems modeling (INCOSE 2023). In 2014, the scope of MBSE expanded to include modeling and simulation, linking system model specifications with their corresponding software simulations (Madni and Sievers 2018). The SEBoK describes MBSE as part of digital engineering, highlighting its importance in modern systems engineering (SEBoK Editorial Board 2024).

Exercise: Checking the consistency of requirements

Consider the following list of 20 requirements for a Mars exploration rover:

1. The rover must be able to traverse rough Martian terrain.
2. The rover must have a robust structure to withstand debris impacts.
3. The rover should operate on Mars for at least five years.
4. The rover's power supply, essentially solar, must be minimal due to payload constraints.
5. The rover should be equipped with advanced scientific instruments for various geological and atmospheric studies.

(Continued)

(Continued)

6. The rover's total weight must not exceed 300kg, to respect the launcher's capabilities.
7. The rover must transmit high-resolution images and large data sets back to Earth.
8. The rover is limited by available bandwidth and the constraints of interplanetary communication systems.
9. The rover must be highly autonomous and capable of making its own navigation decisions.
10. It must also allow precise remote control from Earth should the need arise.
11. The rover must be cost-effective and within a strict budget.
12. The rover must incorporate cutting-edge technologies to meet mission objectives.
13. The rover must be ready for launch within a very short time frame, requiring rapid deployment.
14. The rover must undergo extensive testing to ensure reliability, which can take a long time.
15. The rover must be compact enough to fit within the confines of the launch vehicle.
16. Once on Mars, the rover must expand or deploy to a larger size in order to be operational.
17. The rover must be able to withstand the extreme temperature variations on Mars.
18. The rover carries sensitive instruments that require stable temperature conditions.
19. The rover must be simple in design to improve reliability and ease of repair.
20. The rover must carry out complex scientific experiments, often requiring complex, multifaceted components.

Analyze these requirements and answer the following questions:

- Do you see any inconsistent or unfeasible requirements?
- What are the different categories of requirements expressed here?
- What challenges does the analysis of these requirements pose?
- Imagine a scenario with thousands of requirements expressed in different documents. What problems might this situation create? How can MBSE help improve the management of requirements in such cases?

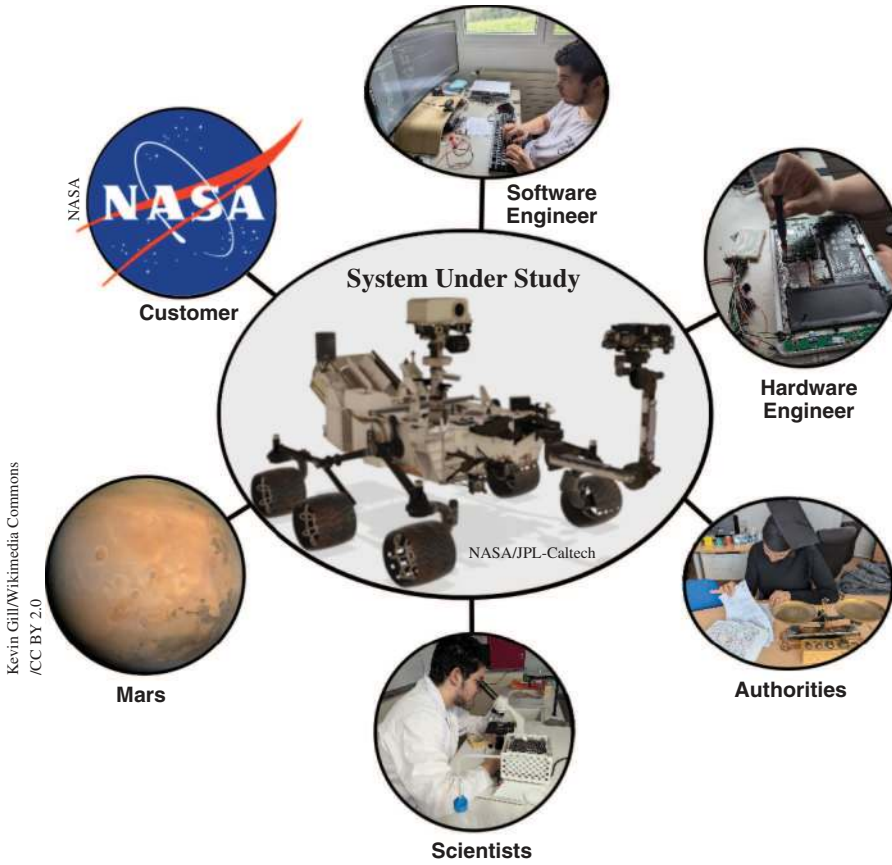


Figure 1.2 A difficult reconciliation.

1.4 The Pillars of MBSE

1.4.1 The Need for Alignment

As illustrated by Figure 1.2, the design and development of complex systems involving multiple stakeholders raise critical questions:

- How can we ensure that system requirements are clearly expressed and understood by all parties concerned?
- How can we align design decisions across diverse perspectives and expertise?
- How can we reliably and reproducibly design, produce, and maintain systems that meet the expectations of both customers and stakeholders?

Answering these questions is challenging due to the inherent differences in expectations and the use of disparate tools that create information silos, hindering effective collaboration and a shared vision.

1.4.2 The MBSE Solution

As mentioned before, a structured, model-centric approach is necessary to tackle these difficulties and promote a comprehensive understanding of the system and its environment. MBSE proves to be a suitable option, using formal models to offer a clear and shared representation of the system that goes beyond the unclear nature of natural language and connects different perspectives (Dori 2016; Estefan et al. 2007).

1.4.3 Core Tenets of MBSE

The essence of MBSE lies in its ability to facilitate collaboration, ensure consistency, and enable effective decision-making throughout the system's lifecycle. The core tenets of MBSE can be summarized as follows (INCOSE 2023):

- **Formalized Application of Modeling**: MBSE fosters the use of formal models as the primary medium for information exchange and decision-making, replacing reliance on traditional documents.
- **Shared System Model**: The system model acts as a single source of truth, integrating information from various domains and stakeholders, to ensure consistency and traceability.
- **Integrated System Views**: MBSE enables the creation of multiple, interconnected views of the system, meeting the specific needs of different stakeholders or engineering disciplines.
- **Continuous Verification and Validation (V&V)**: Models facilitate early and continuous V&V activities, minimizing the risk of errors and inconsistencies.
- **Enhanced Communication and Collaboration**: The use of a common modeling language and a shared system model fosters effective communication and collaboration among diverse teams.

1.4.4 Beyond Technicalities

Designing complex systems has challenges that go beyond just technical issues. MBSE, which is based on system thinking, understands that bringing together different viewpoints and finding trade-offs is crucial for a complete solution (Madni and Sievers 2018). It acts as an approach for combining analysis tools

such as model checking, simulation, and optimization to check the system design, ensuring that it achieves its purpose and meets the needs of stakeholders (Baier et al. 2008; Dori 2016).

1.4.5 The Power of Collaboration

Using formal modeling languages, MBSE allows for accurate and steady representations of complex systems, which helps in effective communication and teamwork among various groups (Holt and Perry 2013). Its successful use in many industries shows it can improve system quality, shorten development time and costs, and enhance stakeholder satisfaction (Madni and Sievers 2018).

Exercise: Exploring model types in your domain

In the context of your specific field or domain, consider the following questions:

1. What are the different types of models that can help you make better decisions in terms of system design or realization? These models could include, but are not limited to:
   - Scale models or physical prototypes
   - 3D models
   - Mathematical models, such as differential equations or optimization models
   - Simulation models, such as discrete-event, continuous, or hybrid simulations
   - Data-driven models, such as statistical or machine learning models
   - Graphical models, such as UML or SysML diagrams
   - Domain-specific models, such as process flow diagrams or electrical schematics
2. For each type of model identified, provide a brief description of its purpose and the key decisions it supports.
3. Discuss how these models can be integrated into an MBSE approach to enhance traceability, consistency, and collaboration among stakeholders.
4. Identify any challenges or limitations associated with using these models in your domain and propose strategies for addressing them.
5. Reflect on how the use of these models can contribute to the overall success of a project in your field, considering factors such as improved communication, risk reduction, and innovation.

By exploring these questions, you will gain a deeper understanding of the role that different types of models can play in your specific domain and how they can be leveraged within an MBSE framework to drive better decision-making and project outcomes.

1.5 The Ingredients of MBSE

1.5.1 The MBSE Tool Ecosystem

The effective implementation of MBSE hinges on a well-orchestrated set of tools and technologies that support the modeling, analysis, and transformation activities throughout the system development lifecycle. These tools, acting in concert, empower engineers to create, refine, and validate models that accurately capture the system's essence and facilitate informed decision-making. The MBSE tool ecosystem comprises several key elements:

- **Modeling Tools**: These tools allow the creation and manipulation of models that represent various aspects of the system, such as requirements, architecture, behavior, and physical components. They support different modeling languages and notations, such as SysML or UML, enabling engineers to express their design intent in a structured and standardized manner. Popular modeling tools include Cameo Systems Modeler, MagicDraw, and Enterprise Architect.
- **Constraint Languages**: These languages provide a means to express rules, conditions, and limitations that the system must adhere to. They ensure the consistency and correctness of the models by formally specifying constraints on model elements. The Object Constraint Language (OCL) is a widely used constraint language that complements UML and SysML models (Object Management Group 2014).
- **Domain-specific Language (DSL) Creation Tools**: These tools allow engineers to define specialized languages tailored to their specific domain or project needs. DSLs offer a more expressive and intuitive way to capture domain-specific knowledge and requirements, bridging the gap between technical jargon and domain expertise. The Xtext and Sirius frameworks are examples of tools that facilitate the creation of textual and graphical DSLs, respectively.
- **Transformation Tools**: These tools automate the conversion of models from one formalism to another and the generation of various artifacts from the models (documents, code, configuration). They capitalize on existing knowledge and free engineers from tedious and error-prone manual tasks.
  - Model-to-model transformations enable seamless integration between different modeling languages.
  - Model-to-text transformations generate artifacts such as documentation, code, or configuration files.

Among the most popular transformation tools, Atlas Transformation Language is widely used for model-to-model transformations, while Acceleo is widely used for text generation (code, documentation, configuration).

- **Process Orchestration Tools**: These tools streamline the management and coordination of MBSE processes, ensuring consistency and reproducibility. They enable the definition, enactment, and monitoring of engineering workflows, facilitating task allocation, progress tracking, and collaboration among team members. The Eclipse Process Framework is an example of a process orchestration tool that supports the definition and execution of engineering processes.
- **Verification Tools**: These tools assess the correctness and consistency of models against specified requirements and constraints. They help identify errors, inconsistencies, and potential issues early in the development process, preventing costly rework in later stages. Model checking and theorem proving are common verification techniques. For instance, the Alloy Analyzer is a powerful verification tool that can be used to check the consistency and correctness of UML and SysML models (Jackson 2019).
- **Simulation Tools**: These tools enable the dynamic analysis of system behavior, performance, and reliability through the execution of the models. They support various simulation paradigms, such as discrete-event, continuous, or hybrid simulation, allowing engineers to gain insights into the system's dynamics and validate its functionality under different scenarios. For instance, MathWorks Simulink is a popular simulation tool for modeling, simulating, and analyzing dynamic systems (MathWorks 2021).
- **Traceability Tools**: These tools establish and maintain relationships between different model elements, artifacts, and requirements. They enable impact analysis, change management, and coverage analysis, ensuring that the system design remains aligned with stakeholder needs and regulatory compliance. For instance, IBM Engineering Requirements Management DOORS is a widely used traceability tool that supports the creation and management of traceability links between requirements, design elements, and test cases (IBM 2021).
- **Standard Exchange Formats**: These formats facilitate interoperability and collaboration between different tools and stakeholders by providing a common concrete syntax for exchanging models and data. They ensure that information can be seamlessly shared and reused across various tools and platforms. XML Metadata Interchange and Requirements Interchange Format are examples of standard exchange formats used in the MBSE ecosystem (Object Management Group 2016).

1.5.2 Key Considerations for Tool Selection and Integration

The selection and integration of appropriate tools are crucial for successful MBSE adoption. Organizations should carefully assess their specific needs and requirements, considering factors such as:

- **Cost**: The total cost of ownership, including licensing fees, maintenance, and training.
- **Ease of Use**: The tools should be user-friendly and intuitive, minimizing the learning curve for team members.
- **Compatibility**: The tools should seamlessly integrate with existing infrastructure.
- **Scalability**: The tools should be able to handle the complexity and scale of the systems being developed.
- **Vendor Support**: The vendor should provide adequate training, documentation, and technical support.
- **Open Standards**: The tools should ideally rely on open standards to promote interoperability and avoid vendor lock-in.

By carefully considering these factors and selecting the right tools, organizations can build a robust and effective MBSE environment that supports their systems engineering endeavors.

1.6 Comparison with a Traditional Approach

1.6.1 The Document-centric Approach

The traditional document-centric approach is commonplace, but it often fails to keep up with the growing complexity of modern systems. Using documents as the main way to capture and share system information has limitations that can hamper effective systems engineering. Natural language can create confusion and misinterpretations, leading to inconsistencies among stakeholders (Grady 2013). As systems become more complex, tracing the impact of changes and managing relationships between many documents become harder and error-prone. The isolated nature of documents also makes teamwork and communication more difficult, while the tedious manual updates required to synchronize multiple documents can lead to errors.

1.6.2 The MBSE Paradigm Shift

In contrast, the model-driven approach championed by MBSE offers a transformative solution. By placing formal models at the heart of the systems engineering process, MBSE provides a precise and unambiguous representation of the system, encompassing its requirements, architecture, behavior, and physical components.

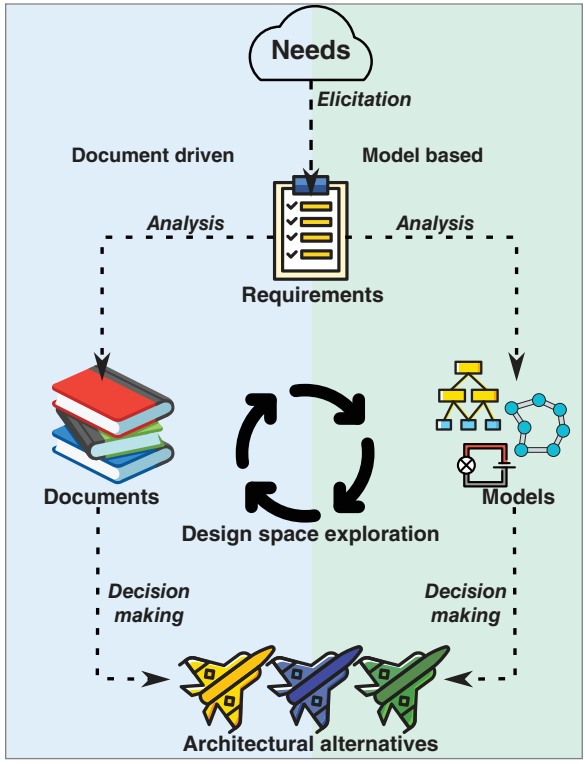


Figure 1.3 Comparison of document-centric and model-driven approaches.

The benefits of this paradigm shift are manifold. The use of formal modeling languages and tools ensures consistency and reduces ambiguity, fostering a shared understanding among stakeholders and promoting effective communication and collaboration (SEBoK Editorial Board 2024). The explicit definition and maintenance of relationships between model elements enhance traceability, facilitating impact analysis and change management, thereby contributing to overall system coherence. The ability to integrate different viewpoints and perspectives into a cohesive set of interconnected models promotes a *holistic understanding* of the system and its context.

1.6.3 Supporting Design Space Exploration

A key benefit of MBSE, shown in Figure 1.3, is that it supports iterative development and effective exploration of design alternatives. Indeed, models enable fast prototyping and testing of various design alternatives, allowing teams to quickly refine their solutions. This adaptability is essential for meeting changing

requirements and finding optimal designs, leading to stronger and more efficient systems.

Figure 1.3 shows how MBSE's emphasis on system thinking, consistency, collaboration, and reproducibility greatly enhances the systems engineering process. By taking advantage of these benefits, MBSE improves the quality and reliability of systems while creating a more efficient and cooperative development environment.

1.6.4 Analysis and Automation

Additionally, MBSE gives engineers tools for automation and analysis. Models can be used for automatic code generation, simulation, verification, and optimization, enabling problems to be identified and fixed quickly, thus improving system quality and efficiency (SEBoK Editorial Board 2024). Simulation allows for analysis of system's real-time behavior, performance, and reliability through model execution. Model checking techniques can rigorously verify that the system meets specified requirements, while trade-off analysis helps in selecting the best system architectures based on criteria such as cost, performance, or safety.

1.6.5 Navigating the Challenges

Although MBSE brings many benefits, it is important to recognize the challenges that can come with using it. Learning to use new tools and methods, along with managing changes within the organization, may create some initial difficulties. Still, the long-term Return on Investment of MBSE, such as improved system quality, reduced development time and costs, and increased stakeholder satisfaction, far outweighs the early expenses.

1.7 Some Success Stories

Here are some success stories illustrating the benefits of MBSE in different fields.

1.7.1 Aerospace Industry

The aerospace industry has been one of the early adopters of MBSE due to the inherent complexity of aircraft and spacecraft systems. Boeing has reported significant benefits from using MBSE on projects like the 787 Dreamliner (Malone et al. 2016):

- Improved quality through early defect detection and resolution
- Fewer system integration issues
- Fewer engineering change requests
- Facilitated communication and collaboration across globally distributed teams

NASA's Jet Propulsion Laboratory has also successfully applied MBSE to design complex robotic spacecraft such as the Mars Science Laboratory (Curiosity rover) (Fosse et al. 2015). MBSE enabled comprehensive modeling of the spacecraft's design, behavior, interfaces, and operating environments, leading to:

- Early identification and mitigation of risks
- Improved traceability from requirements to design
- Reusable model artifacts across future missions

1.7.2 Automotive Industry

In the automotive domain, MBSE has helped manage the increasing complexity introduced by advanced driver assistance systems, electrification, and autonomous driving features. For instance, Ford has adopted MBSE for developing vehicle features such as park assist and adaptive cruise control (Quezada Gomez 2015) with the following key benefits:

- Reduced rework by catching errors and integration issues early
- Improved communication through standardized system models
- Facilitated impact analysis of changing requirements

1.7.3 Defense Industry

The defense industry has also embraced MBSE to handle the complexity of modern weapon systems. Lockheed Martin applied MBSE for the development of the F-35 Joint Strike Fighter (West and Pyster 2015). This enabled:

- Coherent modeling of the entire air system across domains
- Early V&V through model simulation
- Streamlined integration of systems from multiple suppliers

These success stories highlight how MBSE improves system quality, reduces rework and integration issues, enhances communication and traceability, and facilitates reuse across projects, providing a strong value proposition for adopting MBSE, especially for complex, interdisciplinary systems.

1.8 Key Takeaways and Considerations

The journey of adopting and implementing MBSE has yielded valuable lessons that can guide organizations and individuals seeking to harness its potential. The

following key takeaways and considerations serve as a compass for navigating the complexities of MBSE adoption and maximizing its benefits:

- **The Importance of a Well-defined Process**: A rigorous and clearly defined MBSE process provides a structured framework for system realization, from conceptualization to operation. It ensures consistency, traceability, and effective management of system complexity through well-defined activities and responsibilities.
- **Effective Communication and Collaboration**: MBSE fosters a collaborative environment by providing a common language and tools for capturing, analyzing, and validating requirements. Active participation and open communication among stakeholders are crucial for successful MBSE adoption.
- **Careful Tool Selection and Integration**: The selection and integration of appropriate tools are paramount. Organizations must select tools that align with their specific needs, support seamless integration, and enable automation and analysis.
- **Investment in Training and Education**: MBSE requires a skilled workforce. Investing in training programs, workshops, and mentoring initiatives equips team members with the necessary skills to effectively apply MBSE principles and tools.
- **The Power of Iteration**: Starting small and iterating allows organizations to gain experience, refine processes, and demonstrate the value of MBSE before scaling to larger projects.
- **Continuous Improvement**: MBSE adoption is a continuous process. Regular assessment of processes, tools, and skills, coupled with feedback from stakeholders and benchmarking against industry best practices, ensures continuous evolution and optimization.

The future of MBSE is bright, with emerging trends such as the integration of artificial intelligence and machine learning, the rise of digital twins, and the increasing emphasis on sustainability and resilience shaping its trajectory. By embracing these key takeaways and considerations, organizations can position themselves at the forefront of systems engineering innovation, leveraging MBSE to design and develop complex systems that meet the evolving needs of stakeholders and society.

Questions

1 What does the acronym MBSE stand for?
   - **a** Model-based software engineering
   - **b** Model-based system engineering
   - **c** Modular-based systems engineering
   - **d** Modular-based software engineering

2 The author's early struggles with MBSE led to the realization that:
   - **a** MBSE is primarily a technical discipline.
   - **b** Models should only be used for implementation purposes.
   - **c** MBSE has diverse interpretations among practitioners.
   - **d** The gap between high-level models and low-level code is insurmountable.

3 The core principle of MBSE is:
   - **a** The use of documents as the primary means of information exchange.
   - **b** The automation of tedious and repetitive tasks.
   - **c** The extensive use of models for information exchange and decision-making.
   - **d** The generation of code from high-level models.

4 Which of the following is NOT a core tenet of MBSE?
   - **a** Formalized application of modeling
   - **b** Shared system model
   - **c** Integrated system views
   - **d** Elimination of document-centric artifacts

5 The challenges in designing complex systems:
   - **a** Are primarily technical in nature
   - **b** Can be easily addressed with document-centric approaches
   - **c** Extend beyond mere technical considerations
   - **d** Are limited to aligning diverse perspectives

6 Abstraction in MBSE is crucial for:
   - **a** Automating implementation tasks
   - **b** Generating code from models
   - **c** Creating high-level models that capture the essence of the system
   - **d** Debugging complex code

7 Which of the following is NOT an essential ingredient of MBSE?
   - **a** Modeling tools
   - **b** Constraint languages
   - **c** Natural language processing tools
   - **d** Transformation tools

8 The adoption of MBSE can lead to:
   - **a** Increased development time and costs
   - **b** Reduced stakeholder satisfaction
   - **c** Improved system quality and efficiency
   - **d** Siloed information and limited collaboration

(Continued)

(Continued)

9 The author's key takeaway from their MBSE journey is the importance of:
- **a** Focusing solely on technical aspects
- **b** Automating all tasks
- **c** Questioning assumptions and seeking diverse perspectives
- **d** Relying on document-centric approaches

10 Which of the following languages is most commonly used for creating system models in MBSE?
- **a** UML
- **b** SysML
- **c** BPMN
- **d** Capella

Answers

1-b; 2-c; 3-c; 4-d; 5-c; 6-c; 7-c; 8-c; 9-c; 10-b

References

Baier, C., Katoen, J.P., and Larsen, K.G. *Principles of Model Checking*. The MIT Press. MIT Press, 2008. ISBN 9780262304030. https://books.google.fr/books?id=5dvxCwAAQBAJ.

Dori, D. *Model-Based Systems Engineering with OPM and SysML*. Springer New York, 2016. ISBN 9781493932955. https://books.google.fr/books?id=dY9PDAAAQBAJ.

Estefan, J.A. Survey of model-based systems engineering (MBSE) methodologies. *INCOSE MBSE Focus Group* 2007:25(8):1–12.

Fitchen, J. *The Construction of Gothic Cathedrals: A Study of Medieval Vault Erection*. Art: Architecture. University of Chicago Press, 1981. ISBN 9780226252032. https://books.google.fr/books?id=XTHC6c_mCi0C.

Fosse, E., Devereaux, A., Harmon, C., et al. Inheriting curiosity: leveraging MBSE to build Mars2020. In *AIAA SPACE 2015 Conference and Exposition*, 2015:4617.

Friedenthal, S., Moore, A., and Steiner, R. *A Practical Guide to SysML: The Systems Modeling Language*. The MK/OMG Press. Elsevier Science, 2014. ISBN 9780128002025. https://books.google.fr/books?id=97XfoAEACAAJ.

Grady, J.O. *System Requirements Analysis*. Elsevier insights. Elsevier Science, 2013. ISBN 9780124171305. https://books.google.fr/books?id=o8kIR7lvrRQC.

Holt, J. and Perry, S. SysML for systems engineering: a model-based approach. Technical Report, IET, 2013.

IBM. IBM engineering requirements management doors. 2021. https://www.ibm.com/products/requirements-management.

INCOSE. *INCOSE Systems Engineering Handbook*. Wiley, 2023. ISBN 9781119814290. https://books.google.fr/books?id=HhjBEAAAQBAJ.

Jackson, D. Alloy: a language and tool for relational models. *IEEE Transactions on Software Engineering* 2019:45(12):1295–1307.

Johnson, S.B. *The Secret of Apollo: Systems Management in American and European Space Programs*. New Series in NASA History. Johns Hopkins University Press, 2006. ISBN 9780801885426. https://books.google.fr/books?id=HovWLPbM8ywC.

Madni, A.M. and Sievers, M. Model-based systems engineering: motivation, current status, and research opportunities. *Systems Engineering* 2018:21(3):172–190.

Malone, R., Friedland, B., Herrold, J., et al. Insights from large scale model based systems engineering at Boeing. In *INCOSE International Symposium,* 2016:26:542–555. Wiley Online Library.

MathWorks. Simulink—simulation and model-based design. https://www.mathworks.com/products/simulink.html,2021.

Object Management Group. Object constraint language (OCL), version 2.4. 2014. https://www.omg.org/spec/OCL/2.4/(accessed 24 April 2023).

Object Management Group. Requirements interchange format (ReqIF), version 1.2. 2016. https://www.omg.org/spec/ReqIF/1.2/(accessed 24 April 2023)..

Quezada Gomez, J.M. *Model-based guidelines for automotive electronic systems software development*. PhD thesis, Massachusetts Institute of Technology, 2015.

Ramos, A.L., Ferreira, J.V., and Barceló, J. Model-based systems engineering: an emerging approach for modern systems. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews*) 2011:42(1):101–111.

SEBoK Editorial Board. The Guide to the Systems Engineering Body of Knowledge (SEBoK), version 2.10. INCOSE, 2024.

West, T.D. and Pyster, A. Untangling the digital thread: the challenge and promise of model-based engineering in defense acquisition. *Insight* 2015:18(2):45–55.

Wymore, A.W. *Model-Based Systems Engineering*. Systems Engineering. CRC Press, 2018. ISBN 9781351431088. https://books.google.fr/books?id=00taDwAAQBAJ.

2

System Engineering Principles

2.1 Preliminary Definitions

2.1.1 The Essence of Engineering

The word "engineering" has a long history, starting with the Old French term "engigneor," which referred to a builder of military machinery. Over time, the definition has evolved to cover a wider range of subjects, including the detailed study of systems in all their aspects (Checkland 1999). In essence, engineering is about understanding and designing systems, using scientific, mathematical, and practical concepts to find solutions that meet societal challenges and needs. The core of engineering is its ability to turn abstract concepts into concrete and useful systems, bridging the gap between ideas and practical applications.

2.1.2 Defining a System

The term "system" comes from the Latin "systema," meaning a *coherent assembly*. It evolved into the French term "sistème" in the sixteenth century, describing a set of statements that create a *cohesive narrative*.

Today, a system is regarded as a set of interacting components organized to reach specific objectives (International Organization for Standardization 2010; ISO 2015). These components can vary widely, including hardware, software, data, people, processes, facilities, materials, and even natural elements. A defining characteristic of a system is the way its components are connected and dependent on each other, working together to create properties and behaviors that cannot be attributed to any single component.

2.1.3 The Pursuit of Coherence

The concept of "coherence" is central to systems engineering. It refers to the harmonious integration and alignment of various aspects of the system, ensuring that they work together seamlessly to achieve the desired outcomes. In the context of systems engineering, coherence encompasses three key dimensions:

- **Coherence Between the System and Its Environment**: The system must be designed to operate effectively within its intended environment, considering external factors, constraints, and interactions.
- **Coherence Between the Various Parts of the System**: The system's components and subsystems must be properly integrated and work together harmoniously to achieve the desired functionality.
- **Coherence Between the System Objectives and the Means Used to Achieve Them**: The system's design and implementation must be aligned with its intended purpose and goals, ensuring that it delivers the desired value to stakeholders.

Achieving coherence in systems engineering is a challenging endeavor, particularly when dealing with complex systems involving multiple stakeholders and diverse perspectives. The inherent ambiguity of natural language, often used in traditional document-based approaches, can lead to misunderstandings and inconsistencies, hindering the establishment of a clear and shared understanding between stakeholders.

2.2 System Definitions and Taxonomy

The Institute of Electrical and Electronics Engineers (IEEE) Standard Glossary of Software Engineering Terminology describes a system as "a collection of components organized to accomplish a specific function or set of functions" (International Organization for Standardization 2010). This definition highlights how the parts work together to achieve a common goal that goes beyond their individual capabilities.

The ISO/IEC/IEEE 15288 standard provides a broader definition, stating that a system is a "combination of interacting elements organized to achieve one or more stated purposes" (ISO 2015). This definition highlights that systems can include various elements such as hardware, software, data, people, processes, facilities, materials, and natural elements. It shows that the relationships between these elements are as important as the elements themselves.

Both definitions share a similar concept: a system is not merely a collection of parts; it is a coherent whole formed from the interactions of its parts. To understand how a system works, one cannot observe only the parts in isolation; it is the

complex interactions between them that create the system's unique characteristics and functions.

2.2.1 Emergent Properties and Behaviors

Emergent properties are qualities or behaviors that come from the interactions between system parts but cannot be attributed to any single part alone. For instance, the human body has emergent properties, such as consciousness and self-awareness, that cannot be attributed to any one organ or cell.

In engineered systems, emergent properties and behaviors can have both positive and negative effects. They can create unexpected benefits and functions that improve the system's performance and flexibility. However, they can also lead to unforeseen consequences and risks that may threaten the system's effectiveness or the integrity of the people using it.

For example, Figure 2.1 shows a basic breakdown of a Mars rover. It highlights how different parts work together and the possibility for emerging behaviors, as mentioned in the Systems Engineering Body of Knowledge (SEBoK)'s review of system contexts and integration difficulties.

Recognizing and handling emergent properties and behaviors are significant challenges in systems engineering. This requires a holistic approach that examines the separate parts of the system as well as their interactions. By predicting

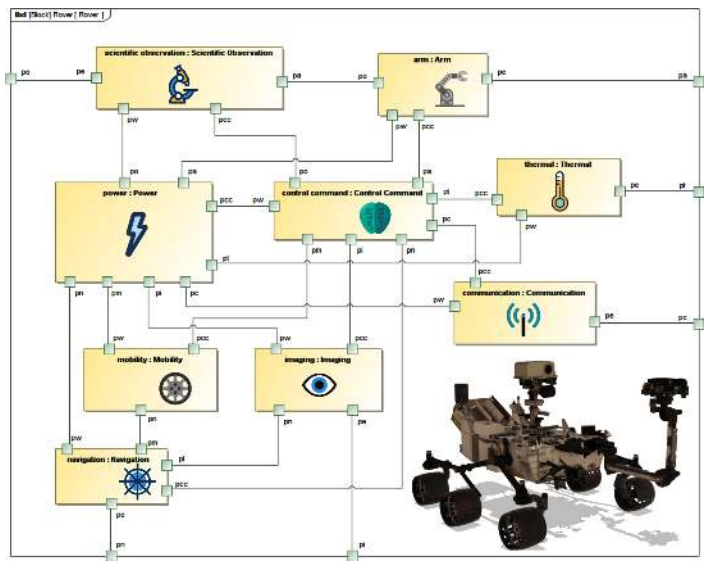


Figure 2.1 Rover subsystems. *Source*: Adapted from https://science.nasa.gov/resource/curiosity-rover-3d-model/; NASA/JPL-Caltech.

and tackling possible emergent behaviors, systems engineers can create systems that are more reliable, more resilient, and better suited to changing conditions and needs.

Exercise: Analyzing emerging behaviors in a Mars rover system

Explore potential emerging behaviors that could arise from the interactions between subsystems in the Mars rover's logical architecture. Study the logical architecture diagram of the Mars rover carefully. Note the various subsystems and their interconnections.

Consider the rover's subsystems and their interactions; for each of the following scenarios, describe a potential emerging behavior that could result from the interaction of multiple subsystems and explain your reasoning:

- The rover encounters an unexpected steep incline while navigating.
- The rover's power levels are critically low during a scientific observation.
- The thermal subsystem detects a rapid temperature increase during a communication session.
- The imaging subsystem identifies a potentially significant geological feature while the arm is deployed for sample collection.

Identify at least two additional scenarios where emerging behaviors might occur based on the interactions between subsystems shown in the diagram. Describe the potential emerging behavior and explain which subsystems would be involved.

Discuss how understanding these potential emerging behaviors could impact:

- The rover's mission planning
- Risk assessment and mitigation strategies
- The design of future Mars rover systems

Remember to consider both intended and unintended consequences that might arise from the interactions between subsystems. Your answers should demonstrate an understanding of how the interconnected nature of the rover's subsystems can lead to behaviors that may not be immediately obvious when considering each subsystem in isolation.

2.2.2 Systems Taxonomy

Mastering emergent behavior is a difficult subject, and in order to understand it more clearly, it is necessary to establish a taxonomy of systems according to their properties and behaviors. The SEBoK provides a detailed overview of system classifications,

separating natural from engineered (artificial) systems and classifying engineered systems according to their distinct qualities (SEBoK Editorial Board 2024):

- **Natural Systems**: These systems occur naturally in the world around us, encompassing ecosystems, biological systems, celestial systems, and geophysical systems. They often exhibit complex, self-organizing behaviors and serve as a source of inspiration and insight for engineered systems.
- **Engineered Systems**: These systems are human-made constructs designed to fulfill specific purposes. They can be further categorized based on their properties and behaviors, such as transportation systems, manufacturing systems, and information systems.

Within engineered systems, the SEBoK explores various taxonomies based on attributes such as:

- **Static vs. Dynamic**: Static systems remain relatively unchanged over time, while dynamic systems exhibit continuous or discrete changes in their state or behavior.
- **Linear vs. Nonlinear**: Linear systems exhibit a proportional relationship between inputs and outputs, while nonlinear systems exhibit more complex and unpredictable behaviors.
- **Time-invariant vs. Time-varying**: Time-invariant systems have constant parameters and behaviors over time, while time-varying systems exhibit changes in their parameters or behaviors as time progresses.
- **Deterministic vs. Stochastic**: Deterministic systems have predictable outcomes for given inputs, while stochastic systems involve randomness and uncertainty in their behavior.

These taxonomies help to understand the different types of systems and how they affect systems engineering practices (SEBoK Editorial Board 2024). For instance, for embedded real-time systems, A. Jantsch proposes a specific taxonomy (Figure 2.2) that examines the special characteristics and constraints of these systems (Jantsch 2004).

The emergent properties and behaviors that result from combining different systems represent issues that still challenge scientific understanding, which may result in unexpected outcomes, including serious failures. Understanding these taxonomies, and emergent properties and behaviors, can help systems engineers carefully select the right modeling and analysis methods, anticipate potential issues, and create reliable and flexible systems that are fit for purpose.

The variety of systems, made clear by these different taxonomies, creates both difficulties and opportunities for systems engineering. Model-based systems engineering (MBSE), which acknowledges this complexity, uses a wide range of theoretical foundations to deal with specific features and behaviors of various system

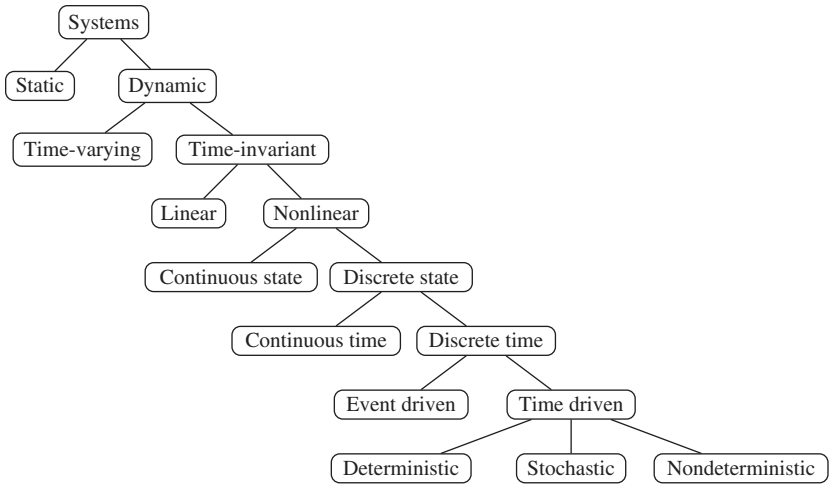


Figure 2.2 Taxonomy of real-time embedded systems.

types. The next chapters will explore these foundational theories, showing how they enable detailed and in-depth analysis of system properties in many areas. These theoretical foundations enable systems engineers to model and analyze both static and dynamic systems, linear and nonlinear behaviors, and deterministic and stochastic behaviors with more accuracy and understanding. They provide the necessary tools to tackle the complexities of modern systems, creating a robust framework for understanding and predicting their behavior.

However, it is essential to recognize the current gaps in our theoretical knowledge, especially regarding the composition of heterogeneous systems and the management of emergent behaviors. The combination of different types of systems, along with the prediction and control of emergent properties, continues to be a field of active study and growth in the MBSE community. This ongoing study highlights the changing nature of systems engineering and emphasizes the need for continuous learning and adaptation in our approaches to modeling complex systems. As our knowledge improves, so does our ability to design, analyze, and manage more sophisticated and connected systems, expanding the limits of what is achievable in systems engineering.

2.3 Systems of Systems

In the field of complex systems, the concept of a system of systems (SoS) stands out as a relatively new approach that goes beyond usual single systems. An SoS is a large, complex structure composed of several independent systems that work

together to achieve a common objective or deliver a unique capability that surpasses their individual capabilities (Jamshidi 2017; Maier 1998). The origin of the SoS concept can be traced back to the defense and aerospace fields in the mid-twentieth century, when it was necessary to combine independently created systems to develop more complex and efficient systems (Maier 1998; Sage and Cuppan 2001).

2.3.1 Key Characteristics of SoS

SoS exhibits several distinctive characteristics that set it apart from conventional systems:

- **Operational Independence**: Each constituent system within an SoS retains its autonomy and can function independently, achieving its own objectives and management strategies.
- **Managerial Independence**: The individual systems are often governed and operated by separate entities or organizations, each with their own priorities and decision-making processes.
- **Evolutionary Development**: SoS are inherently dynamic and evolve over time as constituent systems are added, removed, or modified to accommodate changing requirements or technological gaps.
- **Geographic Distribution**: The constituent systems within an SoS can be geographically dispersed, necessitating robust communication and coordination mechanisms to ensure effective collaboration.

These characteristics make SoS inherently complex and challenging to design, manage, and operate. The dynamic interactions and their resulting emergent behaviors can lead to unforeseen challenges requiring a flexible and adaptable approach to systems engineering.

2.3.2 Examples of SoS

The SoS concept finds applications in various domains, showcasing its relevance to modern, interconnected systems. Some notable examples include:

- **Integrated Air Defense Networks**: These networks comprise various radars, sensors, communication systems, and command-control centers that work together to detect, track, and intercept aerial threats.
- **Global Transportation Systems**: These systems encompass various modes of transportation, such as air, sea, and land, along with the infrastructure and logistics networks that support them, enabling the efficient movement of people and goods across vast distances.

- **Smart Cities**: These urban environments leverage interconnected infrastructure and information systems to optimize resource usage, enhance sustainability, and improve the quality of life for citizens.
- **The Internet**: A quintessential example of SoS, the Internet is a vast network of interconnected networks and systems that enable global communication and information exchange.

These examples underscore the pervasive nature of SoS and their critical role in addressing the complexities of modern society.

2.3.3 SoS vs. System of Subsystems

The terms "SoS" and "system of subsystems" are sometimes used interchangeably, but they actually mean different things that are important to understand.

In an *SoS*, the systems involved can operate independently and are often created and managed by different organizations. Their interactions are dynamic and can lead to emergent behaviors that are mostly very difficult to predict.

On the other hand, a *system of subsystems* includes subsystems that depend on each other to achieve a common objective. These subsystems are usually built and managed by one organization, making their interactions more connected and predictable. The overall system's performance depends on how well its subsystems work together.

Recognizing this difference is key for using the right systems engineering approaches and handling the specific challenges that come with each type of system.

Exercise: SoS or system with subsystems?

For each of the following systems, determine whether it represents an SoS or a system with subsystems. Justify your answer by referring to the key characteristics of each concept.

- **The Electrical Power Grid**: A vast network of power plants, transmission lines, and distribution systems that work together to deliver electricity to homes and businesses.
- **A Smartphone**: A portable device that integrates various hardware and software components, such as a processor, camera, display, and operating system, to provide communication, entertainment, and productivity functionalities.
- **The Global Air Traffic Control System**: A network of radar stations, communication systems, and air traffic control centers that coordinate the movement of aircraft across international airspace.

- **A Car**: A vehicle that combines mechanical, electrical, and electronic systems, such as the engine, transmission, brakes, and infotainment system, to provide transportation.
- **The Human Body**: A complex biological system composed of various organs, tissues, and cells that work together to sustain life and perform various functions.

Consider the following question to deepen your understanding: Can a system evolve from being a system with subsystems to an SoS, or vice versa? If so, what factors might trigger such a transition?

2.4 Ensuring System Consistency

The main objective of systems engineering is to make sure there is a coherent overall discourse throughout the system's development and its lifespan. To grasp this goal, it is important to explore what "coherent discourse" means and its linguistic roots.

In ancient Greek, the word "logos" meant "discourse;" later, it gave rise to the Latin word "lego," meaning "to put together." This background indicates that discourse is about combining different words/sentences to create meaningful communications. Likewise, "coherence" comes from the Latin "co-haereo," which means "to join parts together to create a meaningful whole." This meaning highlights the significance of how the separate parts relate and contribute to the entire communication and meaning.

From a linguistic perspective, a "coherent whole" can be categorized into three distinct types (Thibaudeau 2006), as exemplified by Figure 2.3:

1. **Universal Whole**: This type of whole explains the position of an object within a classification of objects. For example, a living being belonging to the group of dogs belongs to the group of canids and mammals.
2. **Integral Whole**: An integral whole describes the position of an object in the constitution of another object. For example, a mammal is made up of lungs, which in turn are made up of alveoli.
3. **Virtual Whole**: A virtual whole emerges from the interaction between objects, creating a new, abstract entity. A prime example of a virtual whole is an SoS, where the interaction between independent systems gives rise to new capabilities and behaviors that are not present in the individual systems.

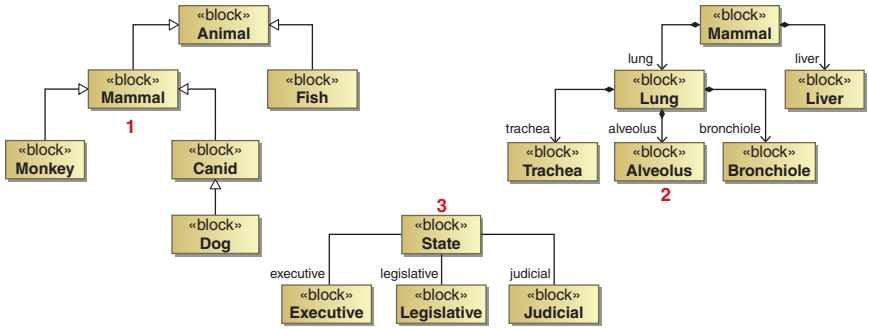


Figure 2.3 Types of coherence.

For instance, the notion of state emerges from the collaboration of the executive, legislative, and judicial powers.

In the context of systems engineering, ensuring a coherent discourse involves addressing all three types of coherence:

- **Universal Coherence**: Ensuring that the system is properly classified and positioned within the broader context of related systems and domains. This involves understanding the system's role, purpose, and relationships with other systems. For example, in the development of an autonomous vehicle, ensuring universal coherence would involve understanding how the vehicle fits into the broader transportation ecosystem, considering its interactions with other vehicles, infrastructure, and traffic management systems.
- **Integral Coherence**: Ensuring that the system's components and subsystems are properly integrated and work together seamlessly to achieve the desired functionality. This requires careful design, interface management, and verification and validation (V&V) activities. For example, in the design of a satellite communication system, ensuring integral coherence would involve ensuring that the various subsystems, such as the antenna, transceiver, and power supply, are compatible and function together seamlessly to provide reliable communication.
- **Virtual Coherence**: Ensuring that the system's interactions with other systems and its environment are well-defined and managed to achieve the desired emergent behaviors and capabilities. This involves understanding and managing the system's boundaries, interfaces, and dependencies. For example, in the development of a smart grid, ensuring virtual coherence would involve managing the interactions between the power generation,

transmission, and distribution systems, as well as the integration of renewable energy sources and demand-side management technologies, to achieve a stable and efficient power grid.

Achieving coherence in systems engineering requires a holistic approach that considers all aspects of the system, from its individual components to its interactions with other systems and the environment. This involves:

- Clearly defining the system's purpose, objectives, and requirements.
- Identifying and managing the system's stakeholders and their needs.
- Developing a comprehensive system architecture that addresses all aspects of the system.
- Ensuring effective communication and collaboration among the various disciplines and teams involved in the system's development.
- Continuously monitoring and adapting the system throughout its lifecycle to maintain coherence in the face of changing requirements and environments.

In summary, maintaining system coherence is an important goal of systems engineering. By exploring the fundamental meaning of coherence and examining the three types – universal, integral, and virtual – systems engineers can create systems that are well-connected, aligned with their goals, and capable of delivering the required capabilities.

2.5 Managing System Complexity

Complexity is an inherent characteristic of modern systems, coming from the complex connections between many parts, their interactions, and emergent behaviors. The SEBoK defines complexity as "the degree to which a system's behavior is difficult to understand due to the number of its parts, their variety, and the interactions between them."

Good complexity management is important for the success of systems engineering projects as it affects the system's performance, reliability, and adaptability. The SEBoK distinguishes between different types of complexity:

- **Structural Complexity**: This refers to the complexity arising from the arrangement and interconnections between the components or parts that constitute a system. It relates to how these parts can be combined and organized to form the overall system structure.
- **Dynamic Complexity**: This type of complexity emerges from the behavior and interactions of the system over time. It encompasses the nonlinear, unpredictable, and often counterintuitive dynamics that can arise, which may not be foreseen during the design or planning phase.

- **Socio-political Complexity**: This complexity arises from the involvement of human actors, including stakeholders, and their individual perspectives, interests, and influences on the system. It captures the complex social and political dynamics that can shape the system's behavior and evolution.

In addition to these types, the SEBoK further categorizes complexity as either disorganized or organized:

- **Disorganized Complexity**: This is found in loosely coupled systems, where the components or elements are relatively independent, and their interactions are weak or infrequent. Such systems often exhibit a lack of central control or coordination.
- **Organized Complexity**: This is found in strongly coupled systems, where the components or elements are tightly interconnected, and their interactions are frequent and interdependent. These systems often exhibit emergent behaviors and self-organization.

Moreover, the SEBoK emphasizes the crucial distinction between complexity and complicatedness. Something inherently difficult to understand due to its intrinsic nature, interconnections, and emergent properties is considered complex. In contrast, something difficult to comprehend due to a lack of clarity, organization, or cognitive limitations is considered complicated (Gribbin 2005).

2.5.1 Factors Contributing to System Complexity

Several key factors contribute to system complexity, as identified in the SEBoK:

- **Interdependence**: The degree to which the components or elements of a system are interconnected and dependent on each other (Meadows and Wright 2008).
- **Interconnections**: The intricate network of relationships, interactions, and feedback loops that exist within the system (Meadows and Wright 2008).
- **Diversity**: The variety and heterogeneity of the components, elements, or agents that constitute the system (Page 2010).
- **Adaptability**: The ability of the system or its components to adapt, learn, and evolve in response to changes in the environment or internal dynamics (Holland 1992).

2.5.2 Managing Complexity

Complexity cannot be reduced; it can only be apprehended by using the right approaches, tools, and frameworks designed to deal with complexity (Holland 2000). The SEBoK offers guidance on different ways to handle complexity, such

as systems thinking, complexity theory, agent-based modeling (ABM), and other approaches that acknowledge the nonlinear, emergent, and changing nature of complex systems (SEBoK Editorial Board 2024). These approaches match the main concepts and practices of MBSE, highlighting the need for a holistic and integrative approach to systems engineering.

Exercise: Analyzing system complexity in the Rover model

Examine the logical decomposition of the rover model in Figure 2.1, which includes various components and subsystems.

- Identify the types of complexity present in the rover model:
  - Structural Complexity: Analyze the arrangement and interconnections between components such as power supply, scientific instruments, and communication systems.
  - Dynamic Complexity: Consider how the rover's components interact over time, especially under different environmental conditions.
  - Socio-political Complexity: Reflect on the involvement of multiple stakeholders in the rover's design and mission planning.
- Propose strategies to manage each type of complexity, such as modular designs for structural complexity or stakeholder workshops for socio-political challenges.

2.6 System Lifecycle

The lifecycle of a system encompasses all stages of its life, from the initial idea to its implementation, use, and beyond, when it is no longer used. It is an evolutionary process that contributes to making, using, and eventually decommissioning the system. The lifecycle may vary depending on the nature of the system and where and how it is used, but there are stages common to many systems (ISO 2015; SEBoK Editorial Board 2024).

The ISO/IEC/IEEE 15288 standard gives a clear outline of the system lifecycle. It explains processes that cover the whole lifecycle and how they can be adjusted for different projects. Source: Neureiter and Binder (2022).

The processes, identified in Figure 2.4 can be broadly categorized into the following groups:

- **Agreement Processes**: These processes focus on establishing and managing agreements between stakeholders involved in the system's development

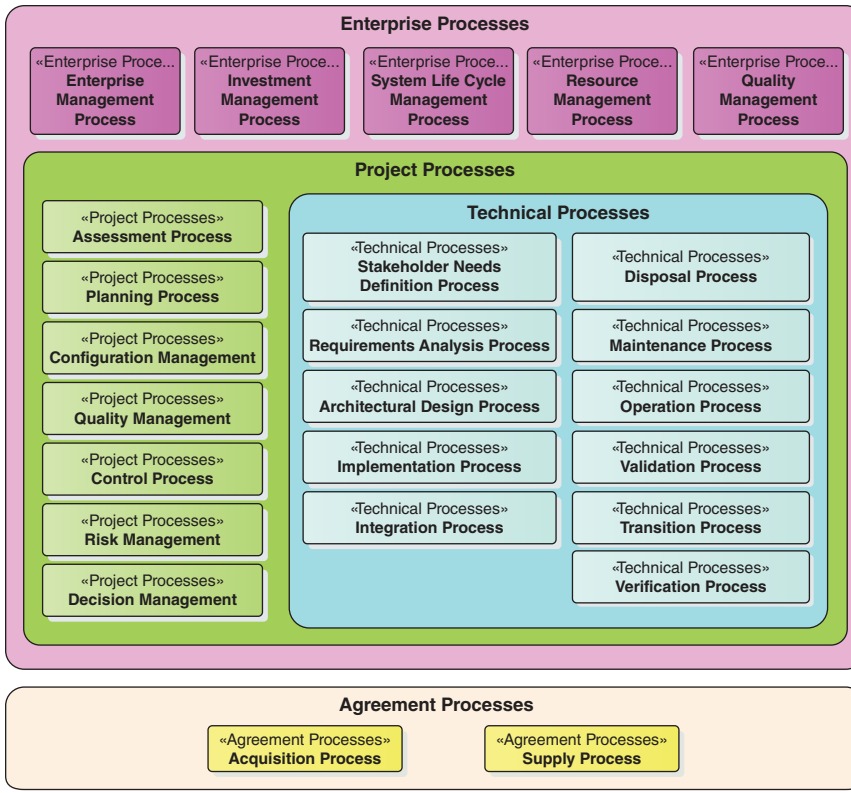


Figure 2.4 ISO-15288 overview. *Source*: Adapted from https://www.researchgate.net/figure/SO-15288-Process-Framework_fig1_359521408.

and operation. The SEBoK emphasizes the importance of clear and concise agreements to ensure a shared understanding and alignment of expectations.

- **Enterprise Processes**: These processes address the organizational structures, resources, and capabilities needed to support system development projects. The SEBoK provides guidance on establishing effective project teams, managing resources, and fostering a culture of systems thinking within the organization.
- **Project Processes**: These processes deal with the planning, monitoring, and control of technical activities throughout the system lifecycle. The SEBoK provides detailed descriptions of processes such as project planning, risk management, configuration management, and measurement, highlighting their role in ensuring project success.
- **Technical Processes**: These processes focus on the specific technical activities involved in designing, developing, and verifying the system. The SEBoK

provides detailed guidance on processes such as requirements engineering, architecture definition, design definition, system analysis, implementation, integration, verification, validation, transition, operation, maintenance, and disposal.

These processes are not necessarily sequential but can be applied concurrently and iteratively throughout the system lifecycle, reflecting the dynamic and complex nature of systems engineering.

2.6.1 An Iterative and Evolutionary Journey

The system lifecycle is mostly an iterative and evolving process, requiring adjustments due to new needs, new technologies, and changing stakeholder priorities (Madni and Sievers 2018; Holt and Perry 2013; SEBoK Editorial Board 2024). This iterative nature stresses the need for a clear and repeatable process, backed by suitable tools and methods, to manage the inherent complexity and interdependencies in the system lifecycle.

2.7 Historical Perspective

2.7.1 The Ancient Roots

Systems engineering is a practice that has only recently been recognized but whose origins can be traced back to antiquity. Large-scale projects, such as pyramids and cathedrals, show early attempts to manage complex work. Although these efforts were not called systems engineering at the time, they paved the way for more organized approaches that developed later into the field we know today. The careful planning, division of tasks, and gathering of knowledge required for these projects hinted at the concepts that would shape systems engineering later on.

2.7.2 The Rise of Formalization

In the sixteenth century, systems engineering began to change significantly. Hierarchical organizations appeared, work became more contract-based, and budgets and deadlines were introduced. A greater focus was placed on reliability and peer reviews. These changes created a foundation for more methodical strategies for complex projects, which helped systems engineering to become a recognized field of study.

2.7.3 The Space Race Catalyst

The true catalyst for the formalization of systems engineering occurred in the twentieth century, driven by competition in the space and arms sectors.

The demand for more advanced systems, along with complex organizations and longer lifespan needs, required a more organized approach. The Apollo program represented this complexity, involving vast amounts of funding, many people, and numerous suppliers, along with constant communication among different stakeholders and groups. The scale of the Apollo program highlighted the need for organized management, leading to the creation of thorough standards, methods, and dedicated tools to ensure the integration and successful management of all parts.

2.7.4 The Apollo Case

The Apollo program, which spanned several years from the early 1960s to the late 1960s, illustrated the scale and complexity of these undertakings (Mindell 2011):

- The program involved tens of billions of dollars in funding, equivalent to over US$25 billion in today's dollars.
- Several hundred thousand individuals were employed across various organizations and contractors, forming a vast network of expertise and collaboration.
- Tens of thousands of suppliers contributed to the program's extensive supply chain, ensuring the availability of critical components and resources.
- Continuous negotiations and coordination were essential between different stakeholder groups, including scientists, politicians, military officers, and program managers, to align objectives and resolve conflicts.

Thomas Ronald Baron's 1966 report on the Apollo program highlighted the main problems encountered during this program (Baron 1966):

- Lack of coordination between people in positions of responsibility.
- Lack of communication between almost everyone.
- The fact that people in positions of responsibility didn't take many of the problems seriously.
- Engineers operating equipment instead of technicians.
- Many technicians don't know their job. This is partly due to the fact that they are constantly being moved from one position to another.
- People are lax about safety.
- People are lax about maintaining cleanliness levels.
- People don't keep proper work records and systems.
- NASA doesn't give engineers a sense of accomplishment.
- There is not a single procedure that has been completed without deviation, whether written or oral.
- Allowing bad practices to continue while the company is aware of them,
- And others.

The Apollo program showcased the crucial need for organized management to effectively manage such a large, complex system. It required creating formal processes based on standards, rigorous methods, and specific tools to enable the integration and successful operation of its numerous parts. This program played a key role in improving systems engineering practices and highlighted the need for a systematic approach to manage the complexity involved in large-scale initiatives.

2.7.5 Lessons Learned and the Path Forward

The challenges posed by the Apollo program and comparable major projects highlighted how crucial it is to adopt a systems approach. Problems such as managing complexity, ensuring effective communication and coordination, and reducing risks became notably clear. The lessons learned pushed for the formal establishment and wider acceptance of systems engineering as its own discipline.

The term "Systems Engineering" itself originates from Bell Telephone Laboratories in the 1940s, where pioneers such as A. Wayne Wymore laid foundational concepts for the field. Organizations like the RAND Corporation and INCOSE also played important roles in advancing and disseminating systems engineering principles and methods.

As current systems become increasingly complex and interconnected, the need for a systematic approach to systems engineering is becoming more apparent. Today's complex system challenges require a comprehensive framework covering the entire lifecycle, from initial concepts to deployment and beyond. The systems engineering approach focuses on integration, collaboration, and global coherence, providing a sound basis for designing, developing, and managing complex systems to achieve efficiency and success.

2.8 For a System Thinking

2.8.1 The Need for a Holistic Approach

The problems encountered in projects of any size have led to the adoption of a systems approach and thinking, which offer a holistic and comprehensive framework for managing complex systems (Meadows and Wright 2008; Sterman 2002). This approach considers the system as a whole, taking into account how its parts interact and depend on each other. This holistic perspective gives a better understanding of how the system works, and its properties and behavior, which cannot be reduced to the sum of its parts. The systems approach also considers the context in which the system exists, examining its relationship with the ecosystem, external boundaries, and stakeholders (Morecroft 2015). This broader perspective fosters the creation of adaptable systems that operate well in their environment,

enabling risks, opportunities, and intervention points to be identified (Forrester 1997).

When dealing with more complex systems, the systems approach provides methods for understanding and managing this complexity. It uses modeling and breakdown principles to carefully analyze complex systems, leading to a more structured approach to the challenges they pose. Tools such as system dynamics modeling, ABM, and network analysis are effective in representing and simulating complex behaviors (Boardman and Sauser 2008).

Systems engineering, grounded in the systems approach, encourages collaboration between different disciplines, such as engineering, human sciences, management, and others (Midgley 2012). This cross-fertilization allows for the integration of varied and complementary perspectives, fostering the design of innovative, high-performance systems that address diverse stakeholder needs and requirements. By bringing together expertise from several domains, systems engineering can tackle the multifaceted challenges posed by complex systems (Ackoff 1999).

Thanks to its systemic perspective, this approach facilitates decision-making by considering the overall impact of the system. It helps to predict side effects, risks, and opportunities tied to design choices. Therefore, decisions become more robust and better aligned with system objectives. System thinking encourages a long-term view that considers both short-term benefits and long-term sustainability (Capra and Luisi 2014).

In summary, the systemic approach enhances systems engineering by providing a framework for understanding systems in their complexity. It fosters more comprehensive and effective engineering practices that meet the challenges of today's complex systems.

2.8.2 The Power of Systems Thinking

Systems thinking, the key to the systemic approach, considers the system as a whole, not just its separate parts. It highlights how its constituent parts interact and connect to shape the system's behavior (Meadows and Wright 2008). With a systems-thinking perspective, we can improve our understanding and management of the complex dynamics resulting from these interactions (Sterman 2002). Systems thinking is based on the following key considerations:

- **Feedback Loops**: Systems are governed by feedback loops, where outputs influence inputs. Systems thinking analyzes these loops, whether they are reinforcing (amplifying a phenomenon) or balancing (bringing the system back toward a goal) (Senge 2010). Understanding these feedback mechanisms is crucial for comprehending and influencing system behavior. Example: A thermostat regulates room temperature by sensing the current temperature and turning the heating or cooling system on or off accordingly. The room

temperature (output) influences the thermostat's decision (input) to maintain a desired setpoint.

- **Emergent Properties**: The overall behavior of a system emerges from the interactions between its parts. These emergent properties cannot be predicted by studying the elements in isolation (Checkland 1999). For example, an ant colony exhibits complex behaviors, such as foraging for food and building nests, that emerge from the simple interactions between individual ants.
- **System Dynamics**: Systems evolve over time in nonlinear ways, exhibiting complex and often counterintuitive behaviors (Forrester 1997). Systems thinking uses tools such as stock-flow diagrams and simulation models to represent and analyze these complex dynamics. For example, a predator-prey model simulates the population dynamics of two species, showing how their populations fluctuate over time due to their interactions.
- **Mental Models**: Our perceptions and mental models of the world shape the way we interact with systems. Systems thinking encourages the clarification and questioning of these mental models to avoid blind spots and improve decision-making processes (Midgley 2012; Ulrich 1994).
- **Leverage Points**: By identifying leverage points where a small change can have amplified effects on the system as a whole, systems thinking enables effective and efficient decision-making (Ackoff 1999; Weinberg 2001).
- **System Boundaries**: Defining the boundaries of the system under study is crucial, as systems thinking analyzes the system's interactions with its external environment and the consequences of perimeter choices (Capra and Luisi 2014).

2.8.3 Benefits of Systems Thinking in Systems Engineering

Using these concepts, systems thinking offers an effective way of analyzing complexity, finding key areas to influence, and planning effective actions for technological, ecological, or social systems (Meadows and Wright 2008; Sterman 2002). It helps us to understand complex connections, predict unexpected outcomes, and create plans for sustainable and reliable systems. In the context of systems engineering, systems thinking brings several key benefits:

- **Improved Problem-solving**: By considering the entire system and its context, systems thinking helps identify root causes of problems and develop more effective and sustainable solutions.
- **Enhanced Communication and Collaboration**: Systems thinking provides a common language and framework for understanding complex systems, facilitating better communication and collaboration among stakeholders from diverse backgrounds.

- **Increased Adaptability and Resilience**: By understanding the dynamics of the system, systems thinking helps anticipate potential challenges and develop strategies for adaptation and resilience in the face of change.
- **Continuous Learning and Improvement**: The reflective nature of systems thinking promotes a culture of continuous learning and improvement, enabling organizations and individuals to evolve and thrive in complex environments.

2.9 System Engineering Definition

Building upon the foundational concepts we have explored, we can now formulate a more formal definition of systems engineering. It is a multifaceted discipline that lies at the heart of technological progress and innovation. The INCOSE defines systems engineering as follows:

> An interdisciplinary approach and means to enable the realization of successful systems. It focuses on defining customer needs and required functionality early in the development cycle, documenting requirements, then proceeding with design synthesis and system validation while considering the complete problem: operations, performance, test, manufacturing, cost and schedule, training and support, and disposal.

The SEBoK provides a complementary perspective, defining systems engineering as:

> An engineering discipline whose responsibility is creating and executing an interdisciplinary process to ensure that the customer and stakeholder's needs are satisfied in a high quality, trustworthy, cost efficient and schedule compliant manner throughout a system's entire lifecycle.

These definitions highlight several key aspects of systems engineering:

- **Interdisciplinary Approach**: Systems engineering transcends traditional disciplinary boundaries, integrating knowledge and expertise from various fields to address the complex challenges of system design and development.
- **Focus on Stakeholder Needs**: Systems engineering prioritizes understanding and fulfilling the needs and expectations of customers and stakeholders throughout the system's lifecycle.
- **Systematic Process**: Systems engineering follows a well-defined and structured process that encompasses a range of activities, from requirements elicitation and analysis to design, implementation, verification, and validation.
- **Holistic Perspective**: Systems engineering considers the complete problem, taking into account not only the technical aspects but also the operational, performance, cost, schedule, and other relevant factors.

- **Lifecycle Focus**: Systems engineering spans the entire lifecycle of a system, from concept development to disposal, ensuring that the system remains effective and adaptable throughout its operational life.

The essence of systems engineering lies in its ability to orchestrate the collaborative efforts of diverse teams, leverage specialized tools and methods, and manage the inherent complexity of large-scale systems. By adopting a systems engineering approach, organizations can improve communication, streamline processes, mitigate risks, and ultimately deliver high-quality systems that meet the changing needs of stakeholders and society.

2.10 System Engineering Issues

The execution of systems engineering projects can be fraught with many problems that can lead to cost increases, delays, and suboptimal outcomes. The effects of inefficient systems engineering can extend far and wide, affecting the project's budget and its ability to satisfy stakeholder requirements, and even the safety of people.

2.10.1 The High Cost of Poor Execution

The financial consequences of bad systems engineering practices can be considerable. Rework, delays, and cost overruns are usual problems of poorly manufactured, integrated, or tested systems. In fact, the National Defense Industrial Association (NDIA) claims that rework can take a huge 30–50% of project budgets in defense projects with deficient systems engineering (NDIA 2010).

The Standish Group's CHAOS report shows an average cost overrun of 43% for troubled IT projects, which further emphasizes the financial burden of poor systems engineering (Standish Group 2015). Additionally, poorly designed systems often face higher operating and maintenance costs over the course of their use due to increased downtime, support requirements, and upgrade challenges (Honour and Jenkins 2010).

Besides direct financial costs, poor systems engineering can result in significant opportunity costs. Missed market chances, lost income, and less competitiveness are frequent results of project delays and poor system performance. A Government Accountability Office (GAO) study on NASA projects found several cases where missions missed planetary launch times because of systems engineering problems, resulting in lost opportunities for scientific discovery (NASA GAO 2024).

A KPMG survey reported that 86% of organizations experienced revenue losses due to delays, highlighting the financial impact of poor systems engineering (Armstrong 2015). Moreover, organizations that cannot produce high-quality systems on time and within budget risk losing their competitive advantage to rivals who excel in systems engineering (INCOSE 2023).

In safety-critical fields, the results of poor systems engineering can be worse, possibly causing loss of life and harm to the environment. Medical mistakes, transportation accidents, and industrial disasters have all been linked to poor systems engineering practices. A 2016 Johns Hopkins study estimated that over 250,000 deaths per year in the United States result from medical errors, many of which are due to poorly designed health IT and medical devices (Makary and Daniel 2016).

The 2018 Lion Air and 2019 Ethiopian Airlines crashes, which sadly took 346 lives, were due to systems engineering issues in the Boeing 737 MAX's flight control system (Johnston and Harris 2019). These cases highlight the vital need for effective systems engineering to protect human lives and prevent serious incidents.

2.10.2 Understanding the Root Causes

To mitigate the risks and challenges associated with systems engineering projects, it is essential to understand the root causes of these issues. Several factors contribute to the problems observed in system engineering projects, and they can be broadly categorized as follows:

- **Stakeholder Engagement and Communication**
  - Lack of customer involvement, leading to incomplete or incorrect requirements and misaligned objectives.
  - Poor visibility into project status and risks, resulting in ill-informed decisions and unmanaged risks.
  - Silos between domains hindering effective communication and collaboration, leading to suboptimal designs and inconsistent assumptions.
- **Requirements and Design**:
  - Ambiguity of requirements leading to scope creep, incorrect implementations, and difficulties in verification.
  - Rushed requirements and design processes resulting in incomplete requirements, design flaws, and technical debt.
- **System Architecture and Integration**
  - System heterogeneity causing integration failures, performance degradation, and maintenance difficulties.
- **Technology and Lifecycle Management**
  - Obsolescence of technologies leading to unsupported components, incompatible replacements, and diminished performance.
- **Testing and Quality Assurance**
  - Insufficient testing and skipped quality checks increasing the risk of defects and compromising system quality.
  - Time-to-market pressure leading to missed market windows, damaged reputation, and increased development costs.

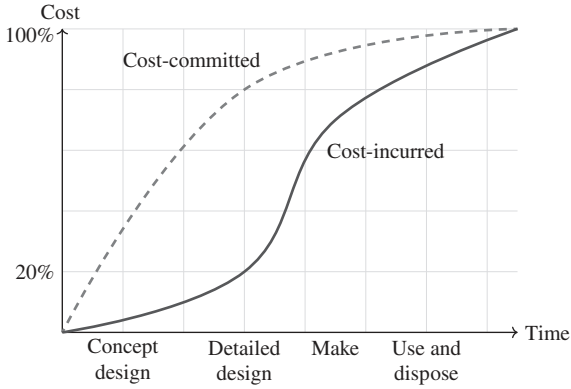


Figure 2.5 Cost commitment over system lifecycle. *Source:* Hansen and Nielsen (2015).

These failure factors are generally due to problems of communication, transparency, and the realism of project objectives. To solve these issues, organizations need to create a culture of collaboration, focus on clear requirement gathering, manage risks proactively, and take a holistic view of the entire system lifecycle.

2.10.3 The Cost of Inaction

The cost commitment curve (Figure 2.5) shows the financial effects of postponing or ignoring systems engineering activities. It shows that most costs are committed early in the design phase, but only a small fraction is actually incurred during this time. This difference clearly shows that investing in systems engineering early in the project is essential to prevent expensive fixes and delays later. By addressing problems and risks early using good systems engineering methods, organizations can reduce total costs and improve project success.

In summary, the problems encountered in systems engineering projects are complex and can lead to significant financial, operational, and even human costs. By grasping the main reasons for these issues and using a proactive, model-based strategy, organizations can manage risks, boost project results, and create complex systems that satisfy the needs of stakeholders and society, while remaining flexible and reliable when facing changes.

2.11 The Role of a Systems Engineer

The systems engineer plays an important role in making and managing complex systems. They act as planners, mediators, and problem-solvers. They combine technical skills with good communication and creative thinking, helping them to

manage the many parts of the system lifecycle (Kossiakoff et al. 2020; Hirshorn et al. 2017). Being a systems engineer is not just about knowing techniques and technologies; it also means understanding the human aspects that influence a project's success. It means being aware of the needs and concerns of all stakeholders, from users to project managers (Hitchins 2008). By listening to others and building relationships with them, the systems engineer can spot potential problems, create agreement, and lead the team toward a solution that meets the various requirements of the system (Sillitto 2014).

Creating and sharing a holistic vision for the system is at the heart of the systems engineering's work. This means understanding the big picture, seeing how parts connect, and anticipating emergent properties arising from interactions between components (Sillitto 2014), and even between people. The systems engineer needs to think broadly, consider the system as a coherent whole, and share their vision with people from different backgrounds (Hitchins 2008). To bring the shared vision to life, the systems engineer acts as a mediator to ensure smooth information flow among team members and stakeholders. They create a space that supports communication, teamwork, and innovative problem-solving (Rechtin 2017). By removing barriers and promoting teamwork, the systems engineer helps the team reach a shared understanding of the system and its objectives (Maier 1998).

2.11.1 The Systems Engineer as a Problem solver and Leader

During the system lifecycle, the systems engineer use their problem-solving skills to address challenges that may arise (Kossiakoff et al. 2020). They examine problems from different perspectives, taking into account not only the technical aspects but also the social, economic, and environmental aspects that affect the system (Hirshorn et al. 2017). By thinking differently and looking for nonstandard solutions, the systems engineer can help the team in handling the uncertainties and risks related to the development of complex systems (Rechtin 2017). To make informed decisions, the systems engineer uses their knowledge of systems thinking, modeling, and analysis (Maier 1998). They apply tools and methods such as system dynamics, ABM, and scenario planning to study how the system behaves under various situations (Sterman 2002). By simulating different design alternatives and trade-offs, the systems engineer assists the team in making decisions based on data that can improve the system's performance, reliability, and cost-efficiency (Jamshidi 2017).

The role of the systems engineer goes beyond technical duties, including management and leadership of the project (Hirshorn et al. 2017). They must create a clear project plan, set milestones, and use resources wisely (Blanchard and Blyler 2016). The systems engineer must also act early to find and reduce risks, ensuring issues are dealt with quickly during development (Kossiakoff et al. 2020).

By advising, coaching, and supporting the team, the systems engineer promotes a culture of excellence and continuous improvement (Sillitto 2014).

2.11.2 The Systems Engineer in the MBSE Era

The advent of MBSE has further empowered systems engineers to fulfill their roles more effectively. MBSE provides a powerful toolkit that supports various aspects of the systems engineer's responsibilities:

- **Developing and Communicating a System Vision**: MBSE models offer a visual and interactive representation of the system, facilitating the communication of the system vision to stakeholders from diverse backgrounds.
- **Facilitating Collaboration**: MBSE tools enable real-time collaboration and information sharing among team members, promoting a shared understanding of the system and its objectives.
- **Managing Complexity**: MBSE models capture the system's intricacies and interdependencies, enabling systems engineers to analyze and manage complexity effectively.
- **Making Informed Decisions**: MBSE supports simulation, analysis, and trade-off studies, empowering systems engineers to make data-driven decisions throughout the system lifecycle.

By leveraging MBSE, systems engineers can improve their ability to navigate the complexities of modern systems engineering, ensuring that the final product aligns with stakeholder needs and fulfills its intended purpose.

Exercise: The systems engineer's role in the Mars rover project

You are the lead systems engineer for a new Mars rover mission. Your task is to coordinate the design and development of the rover, ensuring that all subsystems work together seamlessly to achieve the mission objectives.

Part 1: Identifying Key Responsibilities

List at least five key responsibilities of the systems engineer in the Mars rover project. For each responsibility, explain how it contributes to the overall success of the mission.

Example:

- Responsibility: Developing and maintaining the system architecture
- Contribution: Ensures that all subsystems are properly integrated and aligned with mission objectives, providing a blueprint for the entire project team.

(Continued)

(Continued)

Part 2: Interfacing with Specialty Engineers

Identify five specialty engineering disciplines involved in the Mars rover project (e.g. power systems, communications, navigation, etc.). For each discipline:

1. Describe a specific challenge they might face in the rover design.
2. Explain how the systems engineer would collaborate with them to address this challenge.
3. Discuss how the systems engineer ensures that the solution aligns with the overall system requirements and constraints.

Part 3: Tools and Techniques for Building the Big Picture

Select three MBSE tools or techniques that the systems engineer could use to create and share a comprehensive view of the Mars rover system. For each tool/technique:

1. Explain its purpose and how it is used in the context of the rover project.
2. Describe the key information or insights it provides.
3. Discuss how it facilitates communication and decision-making among the project team and stakeholders.

Example:

- Tool: Systems Modeling Language (SysML) Block Definition Diagram.
- Purpose: To model the structural composition of the rover system.
- Key information: Shows the hierarchical breakdown of the rover.
- Facilitation: Helps team members understand the overall system structure and identify interfaces between subsystems.

Part 4: Ensuring System Coherence

Describe three strategies the systems engineer would use to ensure coherence between different aspects of the rover system (e.g. between requirements and design, between different subsystems, or between the rover and its operational environment). For each strategy:

1. Explain the potential inconsistency or conflict it addresses.
2. Describe how the strategy is implemented.
3. Discuss how it contributes to the overall system integrity and mission success.

2.12 Relationship with Requirements Engineering

Requirements engineering and systems engineering are related fields that work together to ensure the successful development of complex systems. Requirements engineering is about gathering, analyzing, writing, and managing system requirements, while systems engineering focuses on the overall design, development, and management of the system to make sure it meets its requirements and specifications (Young 2004; INCOSE 2023).

The main goal of requirements engineering is to create a clear, complete, and consistent set of requirements that define the problem and guide system development to meet stakeholder needs (Dick et al. 2017). To achieve this, requirements engineering involves several key activities:

- **Requirements Elicitation**: Identifying and gathering requirements from various stakeholders, including users, customers, domain experts, and regulatory bodies. This involves techniques such as interviews, workshops, surveys, and document analysis (Zowghi and Coulin 2005).
- **Requirements Analysis**: Examining the collected requirements to ensure they are clear, unambiguous, consistent, and feasible. This involves activities such as requirements classification, prioritization, and negotiation (Wiegers and Beatty 2013).
- **Requirements Specification**: Documenting the requirements in a structured and manageable format, such as a requirements specification document or a requirements database. The specification should be clear, concise, and traceable (Pohl 2016).
- **Requirements Validation**: Verifying that the documented requirements accurately reflect the stakeholders' needs and expectations. This involves techniques such as requirements reviews, prototyping, and acceptance testing (Laplante and Kassab 2022).
- **Requirements Management**: Controlling changes to the requirements throughout the system development lifecycle, ensuring that the requirements remain consistent, up-to-date, and aligned with the project objectives (Hood et al. 2007).

Systems engineering, on the other hand, is concerned with the overall design, development, and management of the system, ensuring that it meets specified requirements and delivers value to the stakeholders (Kossiakoff et al. 2020). Systems engineering activities contribute to the objectives of requirements engineering in several ways:

- **Stakeholder Needs Analysis**: Systems engineers work closely with stakeholders to understand their needs, goals, and constraints. They help elicit and

refine requirements by facilitating communication between stakeholders and the development team (Hood et al. 2007).

- **System Requirements Definition**: Systems engineers translate stakeholder needs into system-level requirements, ensuring that they are clear, consistent, and achievable. They decompose high-level requirements into more detailed, technical requirements that guide the design and development of the system (Dick et al. 2017).
- **Requirements Allocation and Traceability**: Systems engineers allocate requirements to specific system components and subsystems, ensuring that each element of the system contributes to meeting the overall requirements. They establish and maintain traceability between requirements, design elements, and test cases, enabling effective change management and impact analysis (Gotel and Finkelstein 1994).
- **System Architecture and Design**: Systems engineers develop the system architecture and design, taking into account the functional and nonfunctional requirements, as well as the system's operating environment and constraints. They ensure that the design satisfies the requirements and optimizes the system's performance, reliability, and maintainability (Maier 1998).
- **Requirements V&V**: Systems engineers plan and execute V&V activities to ensure that the system meets the specified requirements and satisfies the stakeholders' needs. They develop test plans, conduct system integration and testing, and perform acceptance testing with stakeholders (Engel 2010).
- **Requirements Change Management**: Systems engineers collaborate with requirements engineers to manage changes to the requirements throughout the system development lifecycle. They assess the impact of proposed changes on the system design, schedule, and cost, ensuring that the changes are properly documented, approved, and communicated to all relevant stakeholders (Hood et al. 2007).

2.12.1 SE/RE Interplay

The complex relationship between requirements engineering and systems engineering, especially in the context of MBSE, can be seen as a continuous feedback loop. Each domain influences the other, providing inputs during the system development lifecycle, as shown in Figure 2.6:

- **Requirements as Input**: High-level requirements are gathered and documented by the requirement engineer, serving as the initial input for the MBSE process.
- **Model Development**: High-level requirements are translated into abstract models and progressively refined into detailed models, representing the system's architecture, behavior, and functionality.

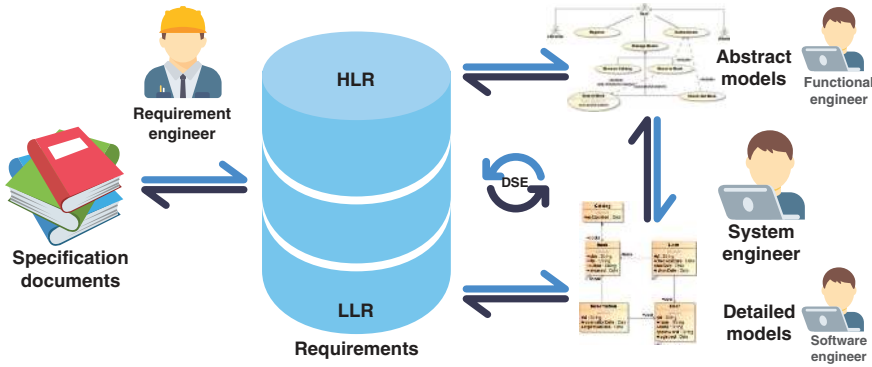


Figure 2.6 System engineering-requirement engineering interplay.

- **Design Space Exploration (DSE)**: The circular arrow labeled DSE visually captures the iterative and exploratory nature of design within MBSE. It means that models keep evolving through continuous refinement and evaluation.
- **Feedback and Refinement**: The bidirectional arrows between all elements highlight how insights gained during the modeling and analysis phases can lead to adjustments in requirements or the system design.

Collaboration between requirements engineering and systems engineering is important for developing complex systems successfully. Requirements engineering makes sure system requirements are clear, consistent, and meet stakeholders expectations. Meanwhile, systems engineering focuses on creating, developing, and delivering the system to meet these requirements (Broy 2006).

2.12.2 Challenges and Strategies for Integration

While the collaboration between requirements engineering and systems engineering is essential, it is not without its challenges. Some of the common hurdles include:

- **Communication Gaps**: Misunderstandings and misinterpretations can arise due to differences in terminology, perspectives, and priorities between requirements engineers and systems engineers.
- **Conflicting Requirements**: Stakeholders may have conflicting or competing requirements that need to be reconciled and prioritized.
- **Changing Requirements**: Requirements may evolve or change throughout the system development lifecycle, requiring flexibility and adaptability in both requirements engineering and systems engineering processes.

To overcome these challenges, organizations can adopt several strategies:

- **Define Clear Roles and Responsibilities**: Clearly define the roles and responsibilities of requirements engineers and systems engineers, ensuring that there is no overlap or ambiguity in their tasks.
- **Foster Open Communication and Collaboration**: Encourage open communication and collaboration between the two disciplines using tools and techniques that facilitate information sharing and feedback.
- **Adopt MBSE**: Leverage MBSE techniques to create a unified, consistent, and traceable representation of the system requirements, design, and behavior. This can help bridge the gap between the two disciplines and ensure that everyone is working from the same information.
- **Implement Iterative and incremental Development**: Embrace agile and iterative development processes that allow for continuous refinement and validation of requirements and system design, ensuring that the system remains aligned with stakeholder needs throughout its development.

2.12.3 The Rover Example

Here are the lessons we can learn from the rover's requirements presented in Chapter 1:

- **The Inevitability of Trade-offs**: It is essential to understand that trade-offs are an integral part of engineering decisions. Reconciling conflicting requirements often means prioritizing some aspects over others, depending on the project's objectives and constraints.
- **The Importance of Stakeholder Commitment**: Early and continuous commitment with all stakeholders is essential. This helps to understand the relative importance of each requirement and where compromises are acceptable.
- **Need for Flexibility and Creativity**: Engineering solutions often require creative thinking to meet conflicting requirements. Flexibility in approach and design can lead to innovative solutions that meet multiple constraints.
- **Value of Iterative Development and Testing**: An iterative approach, where design, testing, and evaluation can be repeated several times, allows the system to be progressively refined to meet as many requirements as possible.
- **Systems Thinking**: This example highlights the need for a systems approach, where the focus is on how different components and requirements interact and impact each other, rather than looking at them in isolation.

- **Risk Management**: It is essential to identify and manage the risks related to each requirement and the trade-offs made. It is essential to understand the potential impact of each decision on the overall success of the project.
- **Clear Communication and Documentation**: Effective communication between team members and clear documentation of decisions and their rationale are essential. This ensures that everyone understands the trade-offs and the rationale behind each decision.
- **Prioritization and Focus on Key Objectives**: Prioritizing requirements according to the project's key objectives enables informed decisions to be made. Focusing on what is most important to the success of the project makes it easier to resolve conflicts.
- **Leveraging Technology and Expertise**: Using the latest technologies and drawing on the expertise of team members and external experts can offer new ways of addressing and reconciling conflicting requirements.
- **Ethical and Sustainable Considerations**: It is increasingly important to strike a balance between technical requirements and ethical and sustainable considerations. The aim is to ensure that the project not only achieves immediate objectives but is also sustainable in the long term.

2.12.4 Summary

Requirements engineering and systems engineering are important fields that work together in developing complex systems successfully. They enable system requirements to be clearly defined, ensure they are consistent and feasible, and guarantee that the system is created and delivered in a way that meets those requirements and fulfills stakeholder expectations. Good integration of requirements and systems engineering practices, with the help of MBSE, is key for managing complexity, reducing risks, and delivering quality systems that benefit users and society.

2.13 Key Considerations in SE

When designing complex systems, there are important problems to consider throughout the process. Considering these problems helps to ensure that the system is reliable, flexible, and operates well in the long term. The SEBoK gives useful tips on these main points, offering help for dealing with the various systems engineering issues (SEBoK Editorial Board 2024).

2.13.1 Managing Change and Risk

The changing nature of complex systems requires active management of changes and risks. The SEBoK emphasizes the need to adapt to changes in the system environment, which include organization, rules, standards, stakeholder needs, and new technologies (Jarratt et al. 2011). This involves:

- **Risk and Opportunity Analysis**: Identifying and assessing potential risks and opportunities related to obsolescence, skills, maintenance, and acquisition.
- **Mitigation and Capitalization Strategies**: Developing and implementing strategies to mitigate risks and capitalize on opportunities.
- **Continuous Monitoring and Adaptation**: Continuously monitoring the system's environment and adapting the system design and processes as needed to ensure its continuous viability and effectiveness.

2.13.2 Capitalization and Reuse

Designing efficient and cost-effective system design often hinges on the ability to capitalize on and reuse existing assets and knowledge. The SEBoK advocates for leveraging design models, patterns, and intellectual properties from legacy systems and Commercial Off-The-Shelf. This can be achieved through:

- **Contract-based and Platform-based Design**: Adopting design approaches that promote modularity, interoperability, and reuse of components across different systems or product lines.
- **Product Line Engineering and Process Engineering**: Applying systematic approaches to develop families of related products or streamline engineering processes to maximize reuse and efficiency.
- **Designing for Adaptability and Extensibility**: Creating systems that can readily accommodate future changes and enhancements, reducing the need for costly redesigns or replacements.

2.13.3 Design Space Exploration

DSE is a critical aspect of complex system design, involving the systematic evaluation and optimization of design alternatives. The SEBoK highlights the importance of exploring the design space to identify optimal solutions that balance various trade-offs. Key activities in DSE include:

- **Requirements Refinement and Integration**: Incrementally refining and integrating requirements to ensure a clear and consistent understanding of the system's purpose and constraints.

- **Virtual Integration and System Composition**: Using modeling and simulation techniques to virtually integrate and assess the behavior of system components and subsystems before physical implementation.
- **Component Characterization and Interaction Analysis**: Analyzing the characteristics and interactions of system components to identify potential conflicts, dependencies, and performance bottlenecks.
- **Semi-automatic Partitioning**: Applying algorithms and heuristics to partition the system into manageable subsystems, optimizing for modularity, cohesion, and coupling.
- **Performance Evaluation and Trade-off Analysis**: Using simulation, analysis, and optimization techniques to evaluate system performance, identify trade-offs between different design alternatives, and make informed decisions based on multiple criteria.
- **Architecture Selection and Co-simulation**: Comparing alternative architectures and selecting the most suitable one based on the evaluation results. Conducting co-simulation to assess the interactions and emergent behaviors of the selected architectures.

2.13.4 Integration, Verification, Validation, and Qualification

Integration, verification, validation, and qualification (IVVQ) is an important stage in the system lifecycle. It helps to make sure that the system meets its requirements, operates correctly, and delivers the required performance. The SEBoK stresses that it is crucial to carry out IVVQ tasks throughout the entire system's lifecycle, using testing, simulation, and formal methods to check how the system behaves and detect any problem or defect (Engel 2010). Good IVVQ requires teamwork between systems engineers, domain experts, and stakeholders to ensure the system is properly tested and validated based on its requirements and operational constraints.

2.13.5 Human Factors and Ergonomics

Human factors and ergonomics are very important in the design of complex systems, especially those involving human–machine interactions. The SEBoK stresses the importance of taking into account human needs and constraints throughout the design phase to ensure that the system is user-friendly, safe, and efficient (Wickens et al. 2021). Key considerations include:

- **User Interface Design**: Creating intuitive, efficient, and error-resistant user interfaces that facilitate effective interaction between humans and the system.
- **Human-system Integration**: Ensuring that the system is compatible with human cognitive and physical capabilities, minimizing the potential for human error and fatigue.

- **Task Allocation and Workflow Optimization**: Analyzing and optimizing the allocation of tasks between humans and machines, and designing workflows that promote efficiency and productivity.
- **Performance, Safety, and Well-being**: Considering the impact of the system on human performance, safety, and well-being, and incorporating design features that promote a positive user experience.

2.13.6 Sustainability and Lifecycle Management

Sustainability and lifecycle management are important in the design of complex systems, especially those with long lifetimes or significant environmental impacts. The SEBoK supports a comprehensive approach that considers the entire system's lifecycle, from raw material to final disposal (Fiksel and Fiksel 2015). Key aspects include:

- **Design for Maintainability, Upgradability, and Disposability**: Creating systems that are easy to maintain, upgrade, and dispose of responsibly at the end of their lifecycle.
- **Material and Component Selection**: Choosing materials and components that have minimal environmental impact and promote sustainability.
- **Energy Efficiency and Resource Optimization**: Designing systems that are energy-efficient and optimize the use of resources throughout their lifecycle.
- **End-of-Life and Disposal Requirements**: Considering the system's end-of-life and disposal requirements to minimize environmental impact and promote responsible recycling or reuse.
- **Lifecycle Cost Analysis and Environmental Impact Assessment**: Conducting comprehensive analyses to evaluate the system's economic and environmental performance throughout its lifecycle.

2.13.7 Cybersecurity and Resilience

As systems become increasingly connected and dependent on digital tools, cybersecurity and resilience are essential to their design and operation. The SEBoK highlights the need to proactively identify and address potential cybersecurity risks while creating systems capable of withstanding attacks and failures (Bodeau et al. 2011). Key aspects include:

- **Vulnerability and Threat Assessment**: Identifying and assessing potential cybersecurity vulnerabilities and threats that could compromise the system's confidentiality, integrity, or availability.

- **Resilient Architecture Design**: Designing the system architecture to be inherently resilient against attacks and failures, incorporating redundancy, fault tolerance, and self-healing capabilities.
- **Secure Communication and Access Control**: Implementing secure communication protocols and access control mechanisms to protect sensitive data and prevent unauthorized access.
- **Security Audits and Penetration Testing**: Conducting regular security audits and penetration testing to identify and address vulnerabilities before they can be exploited.
- **Incident Response and Recovery**: Developing and rehearsing incident response and recovery plans to minimize the impact of security breaches and ensure rapid restoration of system functionality.

2.13.8 Safety Management

Safety is a key issue in systems engineering, especially for systems likely to affect human life, the environment, or critical infrastructures. The SEBoK stresses the need to treat safety as a vital property that must be considered throughout the whole lifecycle of the system. Safety management includes a variety of tasks aimed at finding, assessing, and reducing potential hazards and risks related to the system's design, development, operation, and disposal. Key aspects include:

- **Hazard Analysis and Risk Assessment**: Systematically identifying and evaluating potential hazards that could lead to accidents, injuries, or damage. This involves analyzing the system's components, interactions, and operating environment to identify potential failure modes and their consequences (Ericson 2015).
- **Safety Requirements Definition**: Translating safety-related concerns into clear and verifiable requirements that guide the system's design and development. These requirements should address both the prevention of hazards and the mitigation of their consequences (Leveson 2016).
- **Safety Design and Engineering**: Incorporating safety principles and practices into the system's architecture, design, and implementation. This includes the use of safety-critical components, redundancy, fault tolerance, and other design strategies to minimize the risk of accidents and failures (Storey 1996).
- **Safety V&V**: Conducting rigorous testing and analysis to ensure that the system meets its safety requirements and operates safely under various conditions. This involves techniques such as hazard analysis, fault tree analysis, and safety case development (Leveson 2016).

- **Safety Assurance and Monitoring**: Establishing and maintaining processes for continuous safety monitoring and assessment throughout the system's lifecycle. This includes collecting and analyzing safety data, conducting periodic audits, and implementing corrective actions as needed (INCOSE 2023).

2.13.9 Complexity Management

Managing complexity is a major challenge in designing large, intricate systems. The SEBoK notes that complexity can come from various factors, including the number and variety of system parts, the complexity of their interactions, and the uncertainty of the system's operating environment. Key strategies for managing complexity include:

- **Modular and Hierarchical Design**: Decomposing the system into manageable subsystems with well-defined interfaces and responsibilities, promoting clarity and maintainability.
- **Abstraction and Information Hiding**: Encapsulating complexity within components and subsystems to reduce coupling, while promoting independent development and evolution.
- **Standardization**: Relying on standardized interfaces, protocols, and design patterns to facilitate integration and interoperability between system components.
- **Modeling and Simulation**: Leveraging modeling and simulation techniques to analyze and predict system behavior, identify potential issues, and optimize design choices.
- **Agile and Iterative Development**: Adopting flexible and iterative development approaches that allow for incremental refinement and adaptation of the system design in response to feedback and changing requirements.

2.13.10 System Sizing

The concept of "system sizing" includes estimating the appropriate scale and capacity of a system to satisfy its functional and extra-functional needs (Law 2024). It considers several factors, such as performance, throughput, resource use, and cost, to find the best size and arrangement for the system's components and infrastructure. System sizing is crucial in systems engineering, as it greatly affects the system's performance, cost-efficiency, and overall effectiveness. The complexity of today's systems usually requires the involvement of experts from different fields, each providing their unique perspectives and knowledge. The challenge is to integrate these viewpoints and expertise into a solution that balances various priorities and constraints.

The system sizing process requires careful consideration, weighing the insights and priorities of different experts and stakeholders. Key considerations include:

- **Performance Requirements**: The system's performance requirements, as envisioned by stakeholders and translated into technical specifications by engineers, directly influence the sizing of its components and infrastructure.
- **Resource Constraints**: The available resources, as assessed by project managers and technical experts, impose limitations on the system's size and configuration.
- **Scalability**: The system's ability to accommodate future growth, as anticipated by business analysts and architects, may influence the choice of architectural patterns and technologies.
- **Cost-effectiveness**: Balancing the desired performance and functionality with cost considerations, as evaluated by financial analysts and engineers, is crucial for achieving a viable solution.
- **Reliability and Availability**: The system's reliability and availability requirements, as defined by safety engineers and operational experts, dictate the sizing and redundancy mechanisms needed.
- **Maintainability**: Ensuring the system's maintainability, as emphasized by maintenance engineers and support teams, influences its size and complexity.

System sizing isn't just a technical task; it's a collaborative effort that combines various skills and perspectives.

System sizing illustration

In commercial aircraft design, landing gear configuration is a critical systems engineering challenge that involves complex trade-offs between weight, stability, performance, and operational flexibility.

Let's compare two potential landing gear configurations for a large commercial aircraft: a 2-gears configuration with 6 wheels per gear versus a 4-gears configuration with 4 wheels per gear.

Configuration A: 2 Gears with 6 Wheels Each

- Advantages:
  - Fewer mechanical components
  - Reduced complexity in landing gear system
  - Lower manufacturing and maintenance costs
  - Less weight overall
  - Simplified hydraulic and structural systems

(Continued)

(Continued)

- Drawbacks:
  - Extreme concentration of weight on two primary points
  - Higher ground pressure
  - Limited stability during landing
  - Reduced redundancy
  - Potentially more challenging ground maneuverability
  - Higher risk during single-gear failure

Configuration B: 4 Gears with 4 Wheels Each

- Advantages:
  - Superior weight distribution
  - Enhanced stability during landing and ground operations
  - Better shock absorption
  - Increased safety through gear redundancy
  - More flexible airport compatibility
  - Reduced stress on individual wheel structures
  - Improved lateral stability
- Drawbacks:
  - More complex mechanical system
  - Higher manufacturing costs
  - Increased maintenance requirements
  - Additional structural weight
  - More intricate hydraulic and control systems

For large commercial planes, the 4-gear setup with 4 wheels gives better overall performance, safety, and flexibility. The slight increase in complexity is less important than the big stability and safety advantages.

2.13.11 System Environment

The system environment comprises diverse factors that can significantly influence a system. These can be broadly categorized into four main types:

- **Physical Environment**: The tangible, external conditions in which a system operates. This includes factors such as temperature, humidity, electromagnetic interference, vibration, shock, atmospheric conditions, and radiation levels. Those factors require meticulous consideration during design and modeling (National Aeronautics and Space Administration 2000).

- **Operational Environment**: The conditions and contexts in which the system is used and interacts with users or other systems. Key aspects include user behaviors, operational scenarios, workload patterns, system demand, and interfacing systems. Modeling the operational environment helps to understand system performance under different usage conditions and identify potential issues (Buede and Miller 2016).
- **Regulatory Environment**: The legal, standards-based, and policy-driven factors that constrain or guide system development and operation. This includes industry standards, government regulations, safety and security protocols, and environmental laws. The regulatory environment heavily influences system requirements and design (ISO 2015).
- **Technological Environment**: The broader ecosystem of technologies, platforms, and infrastructures that a system must operate within. This encompasses existing and emerging technologies, interoperability standards, technology lifecycles, obsolescence, and the cybersecurity landscape. Modeling this environment ensures systems can effectively integrate with existing infrastructure and adapt to technological advancements (Madni et al. 2019).

The environment greatly influences system behavior, performance, and effectiveness. Environmental factors can manifest as external stimuli and constraints, leading to:

- **Performance Variations**: Environmental conditions can significantly affect system performance. For instance, temperature and altitude can impact the efficiency of aerospace systems (National Aeronautics and Space Administration 2000), while ambient conditions can influence processing speed and reliability in computer systems (Dumas 2016).
- **Emergent Behaviors**: The interplay between a system and its environment can lead to emergent behaviors/complex patterns not easily predicted from individual components (Madni et al. 2019). These behaviors can be beneficial (e.g. self-organization) or detrimental (e.g. unexpected failures).
- **Reliability and Durability Issues**: Harsh environments can accelerate component degradation and lead to premature system failures (Rausand and Hoyland 2003). Modeling these impacts is crucial for predicting system lifespan and planning maintenance.
- **Operational Constraints**: The environment often imposes operational constraints. For instance, autonomous vehicles must adapt to road conditions, weather, and traffic (Wickens et al. 2021), while communication systems adjust to network congestion and signal interference.
- **Safety and Security Implications**: Environmental factors impact system safety and security. In safety-critical systems, conditions can affect risk levels and necessitate specific safety measures (Leveson 2016).

2.13.12 Managing Interactions with Subcontractors and Partners

Creating complex systems often involves working with external organizations, such as subcontractors and partners, who bring their own skills or parts for the system. The SEBoK stresses that managing these relationships well is critical for the success of the project and to ensuring the final system fulfills its purpose and meets its requirements. Systems engineering, which focuses on a holistic and integrative method, provides a useful framework for dealing with the challenges of collaborating with subcontractors and partners. Key aspects include:

- **Requirements Allocation and Traceability**: The foundation of successful collaboration lies in clear and unambiguous requirements. Systems engineering facilitates the decomposition of high-level system requirements into more specific and measurable requirements that can be allocated to subcontractors and partners. The establishment of traceability links between system-level requirements and the deliverables of external entities ensures that their contributions align with the overall system objectives and can be effectively verified and validated.
- **Interface Definition and Management**: The SEBoK underscores the importance of well-defined interfaces for enabling seamless integration and communication between different system components. Systems engineering plays a crucial role in defining and managing the interfaces between the main system and the components or subsystems developed by subcontractors and partners. This includes specifying data exchange formats, communication protocols, and performance requirements, ensuring that the various parts of the system can interact effectively and efficiently.
- **Communication and Collaboration**: Effective communication and collaboration are essential for successful project execution, especially when working with external entities. MBSE fosters a collaborative environment by providing a common language and framework for communication, enabling stakeholders to share information, exchange concepts, and resolve issues in a timely and efficient manner. The use of models and collaborative platforms can further enhance communication and collaboration by providing a visual and interactive representation of the system and its interfaces.
- **Risk Management and Mitigation**: Relying on external entities introduces inherent risks, such as schedule delays, quality issues, or communication breakdowns. Systems engineering helps in proactively identifying and assessing these risks, developing mitigation strategies, and establishing contingency plans to ensure project success. Regular reviews and audits of subcontractor and partner deliverables, coupled with clear communication and performance monitoring, can help mitigate risks and ensure that their contributions meet the required standards.

- **Contractual and Legal Considerations**: The SEBoK highlights the importance of considering contractual and legal aspects in systems engineering. When working with subcontractors and partners, it is crucial to ensure that contractual agreements and legal obligations are aligned with the system requirements and technical specifications. Systems engineering plays an essential role in defining these specifications, reviewing contractual documents, and ensuring that the deliverables of external entities comply with the agreed-upon terms and conditions.

By focusing on these important points, systems engineering offers a clear and organized way of managing interactions with subcontractors and partners. It creates a cooperative environment, ensures clear communication and shared goals, and actively reduces risks, promoting the creation of complex systems that meet the requirements of stakeholders and society.

2.14 Transitioning to MBSE

Using a systems approach represents an important advance in managing the difficulties of modern systems. Nevertheless, as these systems become more complex and large, it is clear that older document-based approaches are not enough to achieve truly comprehensive and integrative solutions (Holt and Perry 2013). The issues observed in different projects, mentioned before, demonstrate the shortcomings of document-based techniques in managing the wealth of information and connections in complex systems (Estefan 2007). Problems such as ambiguity, inconsistency, and traceability tied to documents can result in costly errors and extra work, making it difficult to effectively develop complex systems.

2.14.1 The MBSE Paradigm Shift

To overcome these issues and reap the full benefits of a systems approach, there is growing recognition of the need for a more structured, model-based approach in systems engineering (Weilkiens 2011). MBSE has emerged as a critical change, using the power of models to represent, analyze, and share various system aspects (Jamshidi 2017). The heart of MBSE is its ability to give a detailed and holistic perspective of the system, covering multiple levels of abstraction and domains (Friedenthal et al. 2014). By formally representing system architecture, behavior, and requirements through models, MBSE supports a more thorough and methodological approach to design and analysis (Estefan 2007).

Moving from document-based to model-based approaches presents its own difficulties. It requires major changes in thinking, processes, and tools. Organizations

need to allocate resources for training, infrastructure, and support to help their teams efficiently adopt and use MBSE practices.

2.14.2 Advantages of MBSE

The benefits of MBSE, however, far outweigh the challenges. By embracing a model-based approach, organizations can:

- **Enhance Communication and Collaboration**: MBSE provides a common language and visual representation of the system, fostering effective communication and collaboration among diverse teams (Holt and Perry 2013).
- **Improve Agility and Responsiveness**: MBSE enables a more agile and iterative development process, allowing for the rapid exploration of design alternatives and the ability to respond to changing requirements and stakeholder needs (Madni and Sievers 2018).
- **Enable Early Issue Detection and Resolution**: Models allow for the early identification and resolution of system inconsistencies, conflicts, and gaps, reducing the risk of costly errors and rework later in the lifecycle.
- **Facilitate Advanced Analysis and Simulation**: MBSE enables the application of advanced analysis and simulation techniques, allowing engineers to explore the behavior of the system under various scenarios and conditions (Madni and Sievers 2018). This capability is crucial for understanding the emergent properties and interactions that arise in complex systems, facilitating the design of more robust and resilient solutions (Weilkiens 2011).

2.14.3 MBSE: A Natural Evolution

As we explore the concepts and approaches of MBSE in the following chapters, it becomes clear that this approach is a natural step forward in systems thinking. It gives the tools and means to meet the challenges of complex systems engineering (Estefan 2007). By using MBSE, organizations can access all the benefits of the systems approach, allowing them to create, develop, and oversee systems that are more reliable, flexible, and better able to adjust to the changing demands of stakeholders and the environment (Friedenthal et al. 2014).

2.15 Conclusion

This chapter has presented the basic concepts of systems engineering, explaining the complex parts of this important field. We began by examining key concepts such as “discourse” and “coherence,” stressing how crucial clear communication is in systems engineering. Then, we examined the formal meanings of systems,

noting their complex nature and the different ways we can classify them. We introduced the concept of SoS, showing its peculiarities and difficulties, as well as how MBSE plays a key role in its development and management.

We also stressed the need for keeping systems coherent. We discussed the three types of coherence – universal, integral, and virtual – and their significance for systems engineering work. We addressed the issues of managing system complexity, highlighting the importance of a holistic and comprehensive approach that uses systems thinking, complexity theory, and MBSE to deal with the complex nature of modern systems. The system lifecycle, which is a changing and repetitive process that guides a system from inception to completion, was examined, emphasizing the need for flexibility and continuous improvement.

Moreover, we followed the historical development of systems engineering from ancient times to its establishment as a distinct field in the twentieth century. We discussed the crucial role of the systems engineer, the close link between systems engineering and requirements engineering, and the main factors influencing the systems engineering process. We recognized the shortcomings in traditional document-based approaches, paving the way for the concept of MBSE as a relatively new approach using models to improve communication, teamwork, and decision-making.

As we move to the next chapter on "Modeling Principles," we will dig deeper into the main concepts and approaches that support MBSE. We will examine the links between systems, languages, and models, and check the different types of models used in MBSE. We will also discuss how languages are used in modeling, the concept of viewpoints, and the advantages of formalization, laying a solid foundation for grasping the strength and possibilities of model-based approaches in systems engineering.

Questions

1 What is the primary goal of systems engineering?
   - **a** To develop isolated components
   - **b** To ensure a coherent overall discourse throughout the system's lifecycle
   - **c** To focus solely on software development
   - **d** To eliminate the need for stakeholder involvement

2 Which of the following best describes a "system" according to the ISO/IEC/IEEE 15288 standard?
   - **a** A collection of independent elements with no specific purpose
   - **b** A combination of interacting elements organized to achieve one or more stated purposes

(Continued)

(Continued)

c A single element designed to perform multiple functions
d An unorganized set of unrelated components

3 What are emergent properties in the context of systems engineering?
a Characteristics that are present in every individual component
b Properties that arise from the interactions between system elements
c Features that are irrelevant to system functionality
d Traits that are easily predicted from individual components

4 Which of the following is a key characteristic of an SoS?
a Operational independence of constituent systems
b Centralized control over all components
c Lack of interaction between systems
d Uniformity in system design and operation

5 What is the role of MBSE in systems engineering?
a To replace all traditional engineering practices
b To use models to enhance communication, collaboration, and decision-making
c To eliminate the need for requirements documentation
d To focus solely on hardware components

6 How does systems engineering address system complexity?
a By ignoring emergent behaviors
b By using a holistic approach that considers interactions and interdependencies
c By focusing only on individual components
d By simplifying systems to the point of losing functionality

7 What is the significance of the historical context in the evolution of systems engineering?
a It shows that systems engineering has always been a static discipline.
b It highlights the development of structured approaches to complex projects.
c It proves that systems engineering is unnecessary in modern projects.
d It emphasizes the superiority of ancient engineering approaches.

8 Which of the following is a challenge associated with managing an SoS?
a Ensuring all systems are geographically co-located
b Managing dynamic interactions and emergent behaviors
c Reducing the number of constituent systems
d Eliminating all forms of independence among systems

9 What is the purpose of a system taxonomy in systems engineering?
- **a** To categorize systems based on their characteristics and behaviors
- **b** To eliminate the need for system classification
- **c** To focus solely on natural systems
- **d** To reduce the complexity of all systems to a single type

10 How does MBSE contribute to ensuring system consistency and coherence?
- **a** By using formal models to reduce ambiguity and ensure consistency
- **b** By eliminating the need for stakeholder communication
- **c** By focusing only on the technical aspects of systems
- **d** By simplifying all systems to a single model

Answers

1-b; 2-b; 3-b; 4-a; 5-b; 6-b; 7-b; 8-b; 9-a; 10-a

References

Ackoff, R.L. *Ackoff's Best: His Classic Writings on Management*. Wiley, 1999. ISBN 9780471316343. https://books.google.fr/books?id=QO1mCgAAQBAJ.

Armstrong, G. Global construction survey 2015: climbing the curve. *KPMG International*, 2015.

Baron, T.R. Apollo accident report: Apollo 204 review board. Technical Report, NASA, 1966.

Blanchard, B.S. and Blyler, J.E. *System Engineering Management. Wiley Series in Systems Engineering and Management*. Wiley, 2016. ISBN 9781119047827. https://books.google.fr/books?id=WPVmCgAAQBAJ.

Boardman, J. and Sauser, B. *Systems Thinking: Coping with 21st Century Problems*. Systems Innovation Book Series. CRC Press, 2008. ISBN 9781420054927. https://books.google.fr/books?id=UIoXeMC4f6EC.

Bodeau, D., Graubart, R., Picciotto, J., et al. Cyber resiliency engineering framework. *MTR110237, MITRECorporation*, 2011.

Broy, M. Requirements engineering as a key to holistic software quality. In *International Symposium on Computer and Information Sciences*, 2006:24–34. Springer.

Buede, D.M. and Miller, W.D. *The Engineering Design of Systems: Models and Methods*. Wiley Series in Systems Engineering and Management. Wiley, 2016. ISBN 9781119027904. https://books.google.fr/books?id=RxeRCgAAQBAJ.

Capra, F. and Luisi, P.L. *The Systems View of Life: A Unifying Vision*. Cambridge University Press, 2014. ISBN 9781107011366. https://books.google.fr/books?id=iEwHAwAAQBAJ.

Checkland, P. *Systems Thinking, Systems Practice*. John Wiley, 1999. ISBN 9780471986058. https://books.google.fr/books?id=R2dQAAAAMAAJ.

Dick, J., Hull, E., and Jackson, K. *Requirements Engineering*. Springer International Publishing, 2017. ISBN 9783319610726. https://books.google.fr/books?id=5lc2swEACAAJ.

Dumas, J.D. *Computer Architecture: Fundamentals and Principles of Computer Design,* 2e. CRC Press, 2016. ISBN 9781498772747. https://books.google.fr/books?id=TZ6VDQAAQBAJ.

Engel, A. *Verification, Validation, and Testing of Engineered Systems*. Wiley Series in Systems Engineering and Management. Wiley, 2010. ISBN 9780470527511. https://books.google.fr/books?id=H6N2CgAAQBAJ.

Ericson, C.A. *Hazard Analysis Techniques for System Safety*. Wiley, 2015. ISBN 9781118940389. https://books.google.fr/books?id=cTikBgAAQBAJ.

Estefan, J.A. Survey of model-based systems engineering (MBSE) methodologies. *INCOSE MBSE Focus Group* 2007:25(8):1–12.

Fiksel, J. and Fiksel, J.R. *Resilient by Design: Creating Businesses That Adapt and Flourish in a Changing World*. Island Press, 2015. ISBN 9781610915878. https://books.google.fr/books?id=GAWICgAAQBAJ.

Forrester, J.W. Industrial dynamics. *Journal of the Operational Research Society* 1997:48(10):1037–1041.

Friedenthal, S., Moore, A., and Steiner, R. *A Practical Guide to SysML: The Systems Modeling Language*. The MK/OMG Press. Elsevier Science, 2014. ISBN 9780128002025. https://books.google.fr/books?id=97XfoAEACAAJ.

Gotel, O.C. and Finkelstein, C.W. An analysis of the requirements traceability problem. In *Proceedings of IEEE International Conference on Requirements Engineering*, 94–101. IEEE, 1994.

Gribbin, J. *Deep Simplicity: Bringing Order to Chaos and Complexity*. Random House Publishing Group, 2005. ISBN 9781400062560. https://books.google.fr/books?id=df6x_T0yDioC.

Halliday, M.A.K. and Matthiessen, C.M.I.M. *Halliday's Introduction to Functional Grammar*. LSE International Studies. Taylor & Francis, 2013. ISBN 9781135983413. https://books.google.fr/books?id=odUqAAAAQBAJ.

Hansen, P.K. and Nielsen, L.M. Modularization management and network configuration. In *DS 80-3 Proceedings of the 20th International Conference on Engineering Design (ICED* 15) Vol 3: Organisation and Management, Milan, Italy, 27-30.07.15, 2015:21–30.

Hirshorn, S.R., Voss, L.D., and Bromley, L.K. *NASA Systems Engineering Handbook*. Technical Report, 2017.

Hitchins, D.K. *Systems Engineering: A 21st Century Systems Methodology*. Wiley Series in Systems Engineering and Management. Wiley, 2008. ISBN 9780470518755. https://books.google.fr/books?id=tdZod1zaIeQC.

Holland, J.H. Complex adaptive systems. *Daedalus* 1992:121(1):17–30.

Holland, J.H. *Emergence: From Chaos to Order*. Popular science/Oxford University Press. Oxford University Press, 2000. ISBN 9780192862112. https://books.google.fr/books?id=VjKtpujRGuAC.

Holt, J. and Perry, S. SysML for systems engineering: a model-based approach. Technical Report, IET, 2013.

Honour, E. 11.4.2 systems engineering return on investment. *INCOSE International Symposium* 2010:20:(1)1422–1439. https://doi.org/10.1002/j.2334-5837.2010.tb01150.x.

Hood, C., Wiedemann, S., Fichtinger, S., et al. *Requirements Management: The Interface Between Requirements Development and All Other Systems Engineering Processes*. SpringerLink: Springer e-Books. Springer Berlin Heidelberg, 2007. ISBN 9783540684763. https://books.google.fr/books?id=8bAjMdORQv0C.

INCOSE. *INCOSE Systems Engineering Handbook*. Wiley, 2023. ISBN 9781119814290. https://books.google.fr/books?id=HhjBEAAAQBAJ.

International Organization for Standardization, International Electrotechnical Commission, Institute of Electrical, Electronics Engineers, and IEEE-SA Standards Board. *2476 - ISO/IEC/IEEE International Standard—Systems and Software Engineering—Vocabulary*. IEEE Std. IEEE, 2010. ISBN 9780738162058. https://books.google.fr/books?id=d5c8zQEACAAJ.

ISO. ISO/IEC/IEEE 15288: 2015. *Systems and software engineering-content of systems and software life cycle process information products (Documentation). International Organization for Standardization/International Electrotechnical Commission: Geneva, Switzerland*, 2015.

Jamshidi, M. *Systems of Systems Engineering: Principles and Applications*. System of Systems Engineering. CRC Press, 2017. ISBN 9781420065893. https://books.google.fr/books?id=YvxUon2vAfUC.

Jantsch, A. *Modeling Embedded Systems and SoC's: Concurrency and Time in Models of Computation*. Electronics & Electrical. Elsevier Science, 2004. ISBN 9781558609259. https://books.google.fr/books?id=GMamoSg9A00C.

Jarratt, T.A.W., Eckert, C.M., Caldwell, N.H.M., et al. Engineering change: an overview and perspective on the literature. *Research in Engineering Design* 2011:22:103–124.

Johnston, P. and Harris, R. The Boeing 737 MAX Saga: lessons for software organizations. *Software Quality Professional* 2019:21(3):4–12.

Kossiakoff, A., Biemer, S.M., Seymour, S.J., et al. *Systems Engineering Principles and Practice*. Wiley Series in Systems Engineering and Management. Wiley, 2020. ISBN 9781119516668. https://books.google.fr/books?id=gGzoDwAAQBAJ.

Laplante, P.A. and Kassab, M. *Requirements Engineering for Software and Systems*. Applied Software Engineering Series. CRC Press, 2022. ISBN 9781000593792. https://books.google.fr/books?id=HEpmEAAAQBAJ.

Law, A.M. *Simulation Modeling and Analysis, 6e*. McGraw-Hill Series in Industrial Engineering and Management Science. McGraw-Hill Education, 2024. ISBN 9781264268245. https://books.google.fr/books?id=cXu6zwEACAAJ.

Leveson, N.G. *Engineering a Safer World: Systems Thinking Applied to Safety*. Engineering Systems. MIT Press, 2016. ISBN 9780262533690. https://books.google.fr/books?id=fA38DwAAQBAJ.

Madni, A.M., Madni, C.C., and Lucero, S.D. Leveraging digital twin technology in model-based systems engineering. *Systems* 2019:7(1):7.

Madni, A.M. and Sievers, M. Model-based systems engineering: motivation, current status, and research opportunities. *Systems Engineering* 2018:21(3):172–190.

Maier, M.W. Architecting principles for systems-of-systems. *Systems Engineering: The Journal of the International Council on Systems Engineering* 1998:1(4):267–284.

Makary, M.A. and Daniel, M. Medical error—the third leading cause of death in the us. *BMJ* 2016:353.

Meadows, D.H. and Wright, D. *Thinking in Systems: A Primer*. Chelsea Green Publishing, 2008. ISBN 9781603580557. https://books.google.fr/books?id=CpbLAgAAQBAJ.

Micouin, P. *Model Based Systems Engineering: Fundamentals and Methods*. FOCUS Series. Wiley, 2014. ISBN 9781118579596. https://books.google.fr/books?id=6lCEBAAAQBAJ.

Midgley, G. *Systemic Intervention: Philosophy, Methodology, and Practice*. Contemporary Systems Thinking. Springer US, 2012. ISBN 9781461542018. https://books.google.fr/books?id=Wp_xBwAAQBAJ.

Mindell, D.A. *Digital Apollo: Human and Machine in Spaceflight*. The MIT Press. MIT Press, 2011. ISBN 9780262266680. https://books.google.fr/books?id=gXYItzQARVoC.

Morecroft, J.D.W. *Strategic Modelling and Business Dynamics, + Website: A feedback systems approach*. Wiley, 2015. ISBN 9781118844687. https://books.google.fr/books?id=T6pwCQAAQBAJ.

NASA GAO. Assessments of major projects. 2024. https://www.gao.gov/products/gao-24-106767.

National Aeronautics and Space Administration. Report on project management in NASA. Technical report, National Aeronautics and Space Administration, 2000. https://www.dcs.gla.ac.uk/*johnson/Mars/MCO_MIB_Report.pdf.

NDIA-National Defense Industrial Association. Top systems engineering issues in US Defense Industry. *Systems Engineering Division Task Group Report*, http://www.ndia.org/Divisions/Divisions/SystemsEngineering/Documents/Studies/Top%20SE%20Issues,202010, 2010.

Neureiter C. and Binder C. A domain-specific, model based systems engineering approach for cyber-physical systems. *Systems* 2022:10:42. https://doi.org/10.3390/systems10020042.

Page, S.E. *Diversity and Complexity*. Primers in Complex Systems. Princeton University Press, 2010. ISBN 9781400835140. https://books.google.fr/books?id=Mi6zkXss14IC.

Pohl, K. *Requirements Engineering Fundamentals: A Study Guide for the Certified Professional for Requirements Engineering Exam - Foundation Level - IREB compliant*, 2e. Rocky Nook, 2016. ISBN 9781937538842. https://books.google.fr/books?id=1VsUDgAAQBAJ.

Rausand, M. and Hoyland, A. *System Reliability Theory: Models, Statistical Methods, and Applications*. Wiley Series in Probability and Statistics - Applied Probability and Statistics Section. Wiley, 2003. ISBN 9780471471332. https://books.google.fr/books?id=OtvWDwAAQBAJ.

Rechtin, E. *Systems Architecting of Organizations: Why Eagles Can't Swim*. Systems Engineering. CRC Press, 2017. ISBN 9781351411943. https://books.google.fr/books?id=Ars6DwAAQBAJ.

Sage, A.P. and Cuppan, C.D. On the systems engineering and management of systems of systems and federations of systems. *Information Knowledge Systems Management* 2001:2(4):325–345.

SEBoK Editorial Board. *The Guide to the Systems Engineering Body of Knowledge (SEBoK)*. INCOSE, 2024.

Senge, P.M. *The Fifth Discipline: The Art and Practice of the Learning Organization*, 2e. Random House, 2010. ISBN 9781407060002. https://books.google.fr/books?id=0V2zW09DDx4C.

Sillitto, H. *Architecting Systems: Concepts, Principles and Practice*. Systems (London, England). College Publications, 2014. ISBN 9781848901544. https://books.google.fr/books?id=1fhFrgEACAAJ.

Standish Group. Chaos report 2015. https://www.standishgroup.com/sample_research_files/CHAOSReport2015-Final.pdf,

Sterman, J. System dynamics: systems thinking and modeling for a complex world. 2002.

Storey, N. *Safety-critical Computer Systems*. Addison–Wesley, 1996. ISBN 9780201427875. https://books.google.fr/books?id=i-AeAQAAIAAJ.

Thibaudeau, V. *Principes de logique: définition, énonciation, raisonnement*. Collection Zêtêsis. Presses de l'Université Laval, 2006. ISBN 9782763784052. https://books.google.fr/books?id=mVd5ySgzWGYC.

Ulrich, W. *Critical Heuristics of Social Planning: A New Approach to Practical Philosophy*. Schriftenreihe des Management-Zentrums St. Gallen. Wiley, 1994. ISBN 9780471953456. https://books.google.fr/books?id=148oAAAAMAAJ.

Weilkiens, T. *Systems Engineering with SysML/UML: Modeling, Analysis, Design*. The MK/OMG Press. Elsevier Science, 2011. ISBN 9780080558318. https://books.google.fr/books?id=LKC9F5gQt3AC.

Weinberg, G.M. *An Introduction to General Systems Thinking*. Dorset House, 2001. ISBN 9780932633491. https://books.google.fr/books?id=eU9gDxt9X0wC.

Wickens, C.D., Helton, W.S., Hollands, J.G., et al. *Engineering Psychology and Human Performance*. Routledge, 2021. ISBN 9781032011738. https://books.google.fr/books?id=YNQ2zgEACAAJ.

Wiegers, K.E. and Beatty, J. *Software Requirements*. Developer Best Practices. Pearson Education, 2013. ISBN 9780735679627. https://books.google.fr/books?id=nbpCAwAAQBAJ.

Young, R.R. *The Requirements Engineering Handbook*. Artech House Professional Development and Technology Management Library. Artech House, 2004. ISBN 9781580536189. https://books.google.fr/books?id=Rkulpi4N3JsC.

Zowghi, D. and Coulin, C. Requirements elicitation: a survey of techniques, approaches, and tools. *Engineering and Managing Software Requirements*, 2005:19–46.

3

Modeling Principles

3.1 Systems, Languages, and Models

3.1.1 The Historical Significance of Models and Languages

Models are an integral part of human history, perhaps even predating the invention of languages. People have always lived with representations of the world, which have had a significant impact on the evolution of the world, or its simulation, as some may think. Languages have enabled the sharing and transmission of knowledge over time. As uses have evolved, technology has developed, and wars have happened; languages have grown, specialized, or blended, often making interpretation more difficult.

The rest of this section will focus on understanding the roles and subtleties in the relationships between systems, formal languages, and models (Figure 3.1).

A model represents a simplified version of a system, ignoring certain details to focus on the features of interest. Thus, a model is never totally accurate, but it can still make fairly good predictions about the system being studied. For example, Newton's laws predict how planets move. While these equations do not fully account for all the complexities of planets and their interactions, they provide sufficiently accurate predictions in many cases.

A model is expressed using one or more languages. At first, natural language may suffice to describe a model, but as mentioned earlier, natural languages pose tricky problems. These difficulties can be alleviated by using a language with a limited vocabulary that meets the needs of its users in terms of representation. With a small set of words and strict grammar, ideas about the system can be shared more clearly. These languages are known as formal languages.

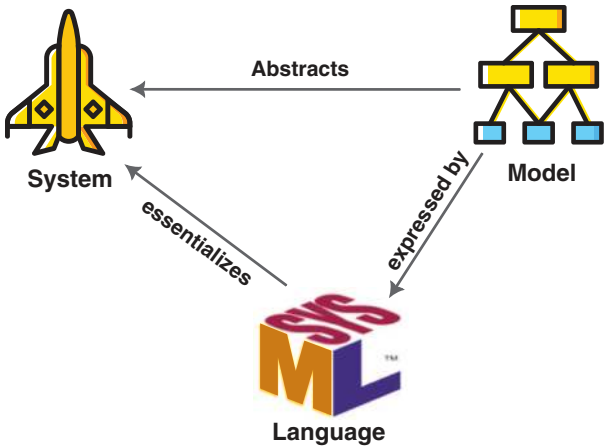


Figure 3.1 Relationship between system, language, and model.

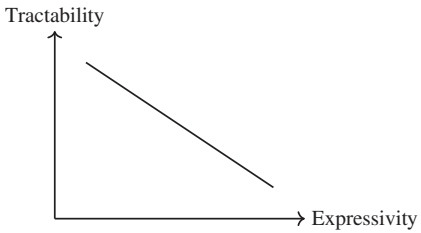


Figure 3.2 Relationship between expressivity and tractability.

3.1.2 The Trade-off Between Expressivity and Tractability

A formal language comprises a limited set of related concepts and usage rules. It can be defined through formal grammar, such as Backus–Naur form or a metamodel. It is important to note that the more expressive a language is, the less decidable or tractable (e.g. decidable in a bounded time) it becomes, as shown in Figure 3.2. In other words, the more expressive a language is, the greater the number of concepts and relationships to manage is, resulting in greater combinatorial complexity.

The choice of language for representing a system's viewpoints depends on the intended purpose, such as communication, documentation, analysis, or artifact creation. For instance, Unified Modeling Language (UML) (Object Management Group [OMG] 2011) offers great expressiveness but is not very tractable, which makes it good for communication and limited for analysis. But it can be combined with other modeling languages that are more focused on overcoming its limitations.

3.1.3 Computational Models and Abstraction Levels

The expressiveness of a language is given by the number of concepts, relations, and constraints that define it. The closer the manipulated concepts are to the machine, the less abstract the language is. Conversely, the closer the concepts are to the business domain, the more abstract the language is.

Each language is characterized by a specific representation of the world based on certain axioms regarding data representation, time, execution logic, and information exchange. For example, assembly language works with very low-level concepts such as registers and stacks, while the Java language deals with higher level concepts like objects and pointers, which are still somewhat technical. Other types of languages, such as functional languages, can move away from machine details to focus solely on business concepts such as functions. The levels of representation specific to each language define their computational models.

A. Jantsch introduces a metamodel called Rugby to classify languages based on their representation of computation, communication, data, and time, as shown in Figure 3.3. This metamodel provides orthogonalization criteria for establishing a relevant taxonomy of computational language models (Jantsch 2004).

The characterization of languages is crucial because, in the process of developing a system, from concept to execution, various models are expressed using different languages at varying levels of abstraction. Then, the choice of a particular language at a given point in the process corresponds to particular objectives in terms of representation and analysis. The rules of refinement and the articulation between these different models define the design and analysis methodology of the system under study.

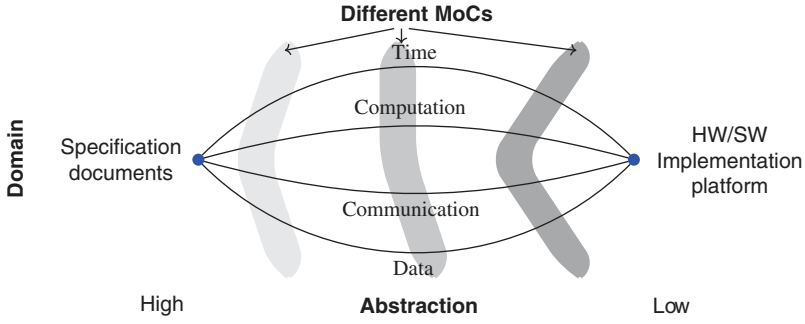


Figure 3.3 Rugby metamodel for characterizing computation models.

Here is a brief description of each domain:

- **Computation Domain**: This domain, also referred to as function or behavior, establishes relationships between system inputs and outputs. These relationships can take various forms, such as mathematical functions, boolean equations, sequences of machine instructions (assembly code), or algorithms.
- **Data Domain**: This domain deals with the representation of physical or logical data. At the system level, physical data (e.g. 20 kg, 36 km/h, 3 ms) is more commonly used, while at the logical level, abstract data types (e.g. integers, 32-bit or 64-bit reals) are used. At the hardware level, bit vectors are used to represent the number of wires needed for transmitting commands and data.
- **Time Domain**: This domain focuses on the representation of time, a crucial aspect of real-time embedded systems. The simplest form of time is causal time, which establishes precedence relationships between events. Discrete time (clocked time) is another representation where system behaviors are linked to common reference clocks (synchronous) or not (asynchronous), measuring the progression of time at regular intervals. Some languages, such as the MARTE UML profile or MATLAB, abstract continuous time as dense discrete time.
- **Communication Domain**: This domain deals with the mechanisms that enable interactions between entities. The form of communication varies depending on the level of abstraction. At the software level, communications take the form of method calls, while at the hardware level, they are realized through communication buses characterized by physical properties (length, energy dissipation) and often complex protocols requiring arbitration policies.

The characterization of languages and their related computational models extends beyond the domains previously discussed. Two additional crucial aspects merit consideration:

- **Configuration Domain**: Languages can vary in their capacity to represent different configurations of the same system. This ability is particularly valuable in model-based systems engineering (MBSE) as it allows for the exploration of design alternatives and the adaptation of systems to various operational contexts. A language with robust configuration capabilities enables engineers to model system variants efficiently, facilitating decision-making, design space exploration, and trade-off analysis.
- **Extra-functional Properties Domain**: The ability of a language to represent extra-functional properties, such as performance, dependability, or security, is another critical dimension. In complex systems engineering, these extra-functional aspects often play a critical role in determining a system's overall effectiveness and acceptability. Languages that can explicitly capture

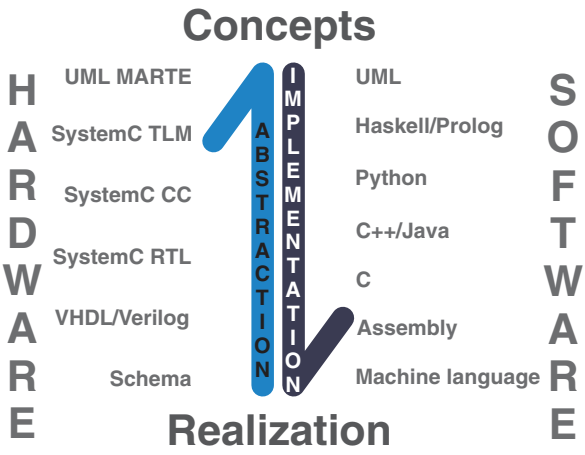


Figure 3.4 Abstractions in HW/SW co-designs.

and analyze these properties enable a more comprehensive evaluation of system designs, ensuring that both functional and extra-functional requirements are adequately addressed.

By examining these other domains, we can gain a better understanding of the power of a language and how it can be adapted for different MBSE activities. This broader view helps us to select the right modeling languages and tools for the particular requirements of a project, thus improving system modeling and analysis.

3.1.4 Illustration of Abstraction Benefits

Abstraction is important for managing the complexity of system design. It helps us focus on the key details and put aside those that are less important. This principle is illustrated in Figure 3.4, which shows a spectrum of languages and modeling techniques used in hardware/software (HW/SW) co-design, arranged along a vertical axis representing the level of abstraction.

This figure highlights the following key points:

- **Navigating the Trade-offs**: As we move down the abstraction axis, the number of manipulated objects increases, and our ability to make informed design decisions may diminish. This is because lower-level abstractions often involve intricate details and complexities that can obscure the overall system vision and hinder effective decision-making. The goal is to avoid a snowball effect, where design flaws or inconsistencies propagate and amplify as we progress to lower levels of abstraction, leading to costly rework and delays. It's far easier and more efficient to detect and rectify issues at higher levels of abstraction,

where the focus is on core concepts and relationships rather than intricate implementation details.

- **High-level Abstraction**: At the top of the spectrum, we can find languages such as UML MARTE and standard UML. These are ideal for capturing system requirements, defining system architecture, and facilitating communication between stakeholders. They offer a broad, conceptual view of the system, emphasizing its purpose, functionality, and interactions with the environment.
- **Mid-level Abstraction**: As we move down the axis, we encounter languages such as SystemC Transaction-Level Modeling and SystemC Cycle Callable. These bridge the gap between high-level concepts and their implementation, enabling the modeling and analysis of system behavior and communication at a more detailed level. They are often used for HW/SW co-design and system-level performance analysis.
- **Low-level Abstraction**: Toward the bottom of the spectrum, we can find languages such as VHDL, Verilog, and assembly language. These are closely tied to the physical implementation of the system, capturing the intricate details of hardware design and software execution. They are essential for the detailed design, verification, and validation of system components.

This figure shows how important it is to choose the right level of detail for each stage of the system development. In the early stages, general models help people understand and discuss ideas, allowing stakeholders to make key design choices with a good grasp of the system's main objectives and limitations. As the design progresses, more specific models are used to capture finer details of how the system works and to confirm its accuracy and performance.

By carefully moving through different levels of detail and understanding the benefits and drawbacks, MBSE helps engineers manage complexity, make smart design choices, and create systems that work well, are robust, flexible, and meet the requirements of stakeholders.

3.1.5 Abstraction and Design Space Exploration

The main idea behind MBSE is to drive the design process using different levels of abstraction, starting with simpler languages that are easy to handle and moving toward more complex ones that deal with much more detail. This approach makes it possible to close the design space smartly, as shown in Figure 3.5. Closure is achieved through an iterative process integrating more and more requirements, from functional requirements to sizing requirements. This allows design choices to be validated in short cycles before moving on to the next level of abstraction. At each stage, models that do not meet the requirements are discarded, keeping the design focused on the system's objectives.

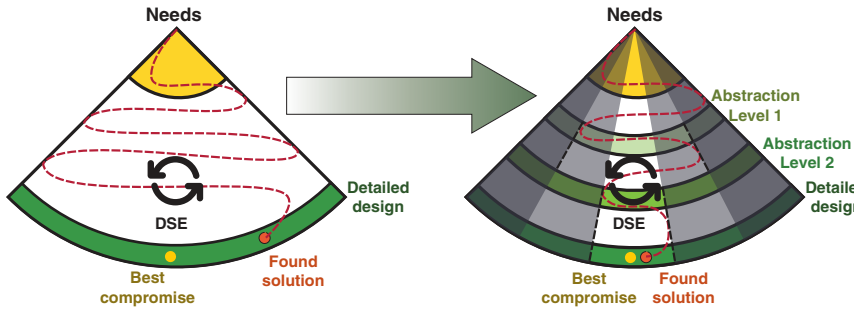


Figure 3.5 Design space exploration closure.

Trying to make quick decisions by combining all requirements into one single model is not realistic and may hinder MBSE use. What is needed is an appropriate methodology that smartly applies languages according to the iterative resolution of specific issues. The aforementioned Rugby metamodel enables us to grasp the different levels of detail and the languages tied to them. A leap in abstraction here means an increase in detail in at least one of the four axes mentioned earlier: computation, communication, data, or time.

For example, representing how a system behaves can start with simple, high-level **Petri Nets**, which show interactions between system functions and check for problems such as deadlocks. As the design evolves, the model can be enriched with temporal aspects using **Timed Petri Nets** to verify timing-related properties. Further development into **Colored Petri Nets** brings more control and checks additional properties. Finally, **Stochastic Colored Petri Nets** can be implemented to study nondeterministic behaviors. This approach addresses the gaps created by abstraction leaps between high-level business models and detailed models that cover the many design choices during the system's lifecycle. It enables early and iterative checks, preventing costly and severe issues later on. Each level of abstraction focuses on a certain family of problems to be solved, such as functional correctness, performance, security, or resilience.

3.2 Models

As Edseger W. Dijkstra once remarked, "As a slow-witted human being, I have a very small head and I had better learn to live with it and to respect my limitations and give them full credit, rather than try to ignore them, for the latter vain effort will be punished by failure" (Dijkstra 1972).

Humans do have natural cognitive limits, particularly when understanding and creating large, complex systems. We can try to predict behavior based on a few

parts, given their detail and complexity, but after a certain point, it gets difficult to predict all possible behaviors. It's like trying to guess all the moves halfway through a game of chess. So, we need to share our thoughts to think together about the same object and use a tool that helps us reason collectively on the same subject.

The Institute of Electrical and Electronics Engineers (IEEE) defines a model as:

> An approximation, representation, or idealization of selected aspects of the structure, behavior, operation, or other characteristics of a real-world process, concept, or system (International Organization for Standardization 2010).

Models are important for the design and creation of complex systems, as they bring together the viewpoints of customers, experts, and authorities (Estefan et al. 2007). They facilitate communication and teamwork, help with system design and analysis, support management of requirements, assist in system integration and validation, and constitute a type of documentation (Friedenthal et al. 2014).

According to Bran Selic, models should exhibit the following properties (Selic 2003):

- Highlight the characteristics of a system and their suitability for its requirements, given its complexity.
- Enable communication of design ideas to the various parties involved in the system.
- Support abstraction.
- Be expressed in a complete form.
- Be precise.
- Enable accurate predictions.
- Be more economical to build and test.
- Enable risk management based on various analyses, such as functional complexity and correctness, schedulability, performance, security, and reliability.

Models help to examine design alternatives and identify problems early in the development phase. Techniques such as simulation, formal verification, and model checking are useful for analyzing system behavior, performance, and reliability (Clarke et al. 2018). This minimizes the risk of costly modifications later on.

Requirements management is another key area where models play a beneficial role. They provide a means for capturing, organizing, and linking requirements to different system components, facilitating traceability and impact analysis (Wiegers and Beatty 2013). Models can also help identify and resolve conflicting or ambiguous requirements from multiple viewpoints.

In system integration and validation, models are used to generate test cases and scenarios. They assist in verifying that the various system components work effectively together, thus promoting seamless integration (Bruegge and Dutoit 2010).

Everything is a model

Since the dawn of mankind, humans have never ceased to shape the world around them. Plato, in his famous *Allegory of the Cave*, emphasized that humans only know reality through their senses and beliefs. We exist within models of the world that reflect our beliefs about what surrounds us. These models are formed from birth by family, social, and societal structures in which we grow up, and they can change over time due to factors such as rumors, indoctrination, politics, and media.

To trace the origins of modeling, we need to go back to the Paleolithic era and the cognitive revolution around 70000 BCE. Cave paintings from this time provide interesting clues; 90% of these cave paintings show animals. Although the true purpose of these artworks is still a matter of discussion, the main theories suggest practical reasons, such as hunting preparation or religious objectives such as animist worship. Whatever the case, the modeling had a clear purpose, either to share a plan for hunting or to build/reinforce a belief system that structured the group.

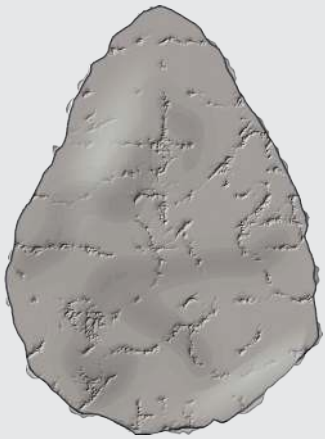


Mankind's earliest models. *Source*: Eline13Viki+ / Wikimedia Commons / Public domain.

During the Paleolithic period, *Homo sapiens* started to improve themselves with new inventions that helped them expand their abilities to dominate the world. These inventions included flint tools, boats, oil lamps, and bows. They were based on their understanding of nature and logical thinking, which enabled *Homo sapiens* to be more dominant than other species, including other Homo species. Notably, even these early tools show a preference for symmetry and appealing shapes, such as flint tools, raising interesting questions about the relationship between aesthetics and function.

(Continued)

(Continued)



Function and aesthetics.

Indeed, these visual representations, along with the mental model of the world, helped them make tools and inventions that enabled them to survive better and stay on top. This early modeling laid the foundation for the increasingly sophisticated representations and models that humans would develop over the millennia to come.

Over time, humans have used models to make sense of and engage with their environments. In the early days, when humans lived closely with nature, these models mainly described what was observed and understood. But around 10000 BCE, with the agricultural revolution, things changed dramatically, as humans began to take control of nature and modify their environment.

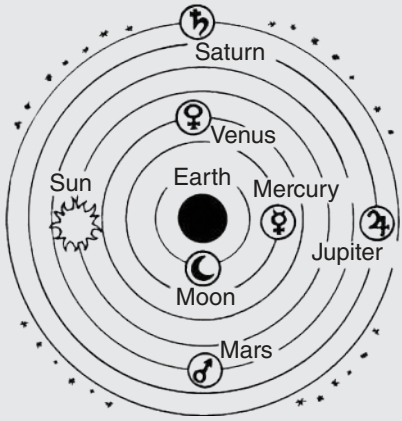
As humans settled down, they began to impose their models on the natural world. This change affected all parts of society, including the way cities were built, the laws that were made, and the way economic and religious systems developed.

One clear example is money, which Yuval Noah Harari discusses in his book "Sapiens: A Brief History of Humankind." Money acts as a shared belief that underpins the modern economy. In its simplest form, a banknote is just a piece of paper or a digital token on a computer. It is common agreement among people to give value to these simple items that gives them importance. This belief leads people to work for long hours, often because they believe that obtaining certain things, such as an expensive car, can improve their social status and provide fulfillment.

Models are built on axioms, which are accepted beliefs within a group, and rules for developing new beliefs from these axioms. The legal system also follows a model, with laws serving as beliefs that influence court decisions. Interestingly, the idea that everything is based on belief is itself a belief, showing the recursive nature of belief systems.

Belief systems appear as models in various fields, from economics and mathematics to physics and even astrology. Science, which focuses on evidence and careful methods, is not free from this phenomenon. Like any other belief system, science has its own established ideas and structures that resemble those of religious groups. As we learn more about the world, scientific models must also evolve to explain and predict natural events.

For instance, history shows that our knowledge of the universe evolved through many cosmological models. Each model built on, or sometimes opposed, earlier ideas. This change shows that scientific knowledge is constantly evolving as new discoveries and theories improve or replace old beliefs. An important early model was created by Claudius Ptolemy, a Greco-Roman astronomer, in the second century CE. Ptolemy's geocentric model put Earth at the center of the universe, with the Sun, Moon, and planets moving around it. This model was the main way to understand the universe for more than 1300 years. However, this model had its shortcomings. For instance, it could not clearly explain how some celestial objects moved backward, an issue fixed by adding epicycles, smaller circles inside the planet's wider trajectories.



Ptolemy's cosmic model. *Source*: Pearson Scott Foresman / wikimedia commons / https://fr.m.wikipedia.org/wiki/Fichier:Ptolemaic_system_(PSF).png.

(Continued)

(Continued)

The Ptolemaic model began to lose its importance in the sixteenth century when the Polish astronomer Nicolaus Copernicus suggested a heliocentric model that placed the Sun at the center of the universe. This new idea met with strong resistance, as it went against the common scientific, religious, and philosophical beliefs. Despite this initial resistance, the Copernican model gained followers among later astronomers such as Johannes Kepler and Galileo Galilei, who improved the model and backed it up with observations.

The next significant progress in understanding of the cosmos came from Isaac Newton, whose laws of motion and universal gravitation offered a mathematical basis for the movement of celestial objects. Newton's contributions confirmed the heliocentric model and paved the way for much of classical physics. However, even Newton's theory had its shortcomings, especially regarding the explanation of light and the nature of space and time. These gaps were filled in the early twentieth century by Albert Einstein's theory of relativity, which changed how we understand gravity, space, and time. Einstein's theories have been essential in explaining different astronomical occurrences and have even led to technological developments such as the Global Positioning System.

Einstein's model has proved its worth, but many questions remain in cosmology, such as the nature of dark matter and dark energy, the reasons for the accelerating expansion of the universe and the formation of galaxies. New models have been proposed to tackle these issues, some of which challenge existing scientific beliefs. One such model is the Janus cosmological model, introduced by Andrei Sakharov and developed further by Jean-Pierre Petit. This model includes ideas of negative mass and negative energy. Although the simulation of this model explains certain cosmic events, it has met with resistance from scientists because of its unusual ideas.

The history of cosmological models shows that scientific progress often involves questioning and modifying long-standing beliefs. As new evidence and theories emerge, it is important that scientists remain open to new ideas while checking them closely. Only through this continuous process of refinement and change can we hope to understand the mysteries of the universe. Throughout history, the acceptance or rejection of scientific models has often depended on factors other than their ability to explain things or be based on observations. Social, political, and religious factors have had a major influence on which models have been accepted and which ones have been ignored or silenced.

An example of this can be seen in medicine, where the germ theory, led by Louis Pasteur in the nineteenth century, became the main way to understand

and treat illnesses. Pasteur's idea suggested that the human body is normally free of bacteria and that the presence of microbes can explain diseases, which helped shape modern allopathic medicine. Yet, Pasteur's idea met with some opposition. An important rival to Pasteur was Antoine Béchamp, who introduced a different idea based on endosymbiosis. Béchamp believed that the human body is a complex ecosystem with diverse microbes and that disease arises from an imbalance in this microbial ecosystem rather than from the mere presence of harmful pathogens.

Despite the possible advantages of Béchamp's idea, Pasteur's germ theory won out in the end. This dominance can be partly explained by the political and social environment of the time. In Napoleon III's France, which was preparing for war, the vision of the body as a fortress under attack from microbes corresponded to a strong militaristic and nationalistic sentiment. Moreover, Béchamp's theory suggested that microbes could appear at random under certain conditions, which ran counter to the Catholic Church's beliefs about the divine origins of life.

In contrast, Pasteur's idea of microbes as external threats was more in line with the religious and conservative views of nineteenth-century France. As new information comes to light, and we learn more about the human microbiome, the flaws in Pasteur's idea are becoming clearer. This highlights the need to keep reviewing and improving scientific ideas based on new facts and understandings.

The impact of social factors on the acceptance of scientific ideas is obviously not just limited to the fields of medicine or cosmology. Similar patterns can be found in many fields, such as physics, mathematics, economics, and psychology.

Models are important tools for understanding the world, but they are always temporary and subject to change as our knowledge grows. As Socrates once said, "All I know is that I know nothing." This simple recognition of the limits of what we know reminds us of the importance of continual critical evaluation and improvement of our ideas and beliefs. To advance our understanding of the world and tackle the complex issues our society faces, we must be willing to ask questions and stay receptive to fresh ideas.

In summary, models are important for understanding the world and guiding our actions. They act as guides, allowing us to assess the possible outcomes of suggested changes and new ideas. Nevertheless, we must remember that models can be updated or replaced as new facts and ideas emerge. Keeping a critical and open eye on the models we use will help us keep making advancements in science, technology, and society in general.

3.3 Languages

All models are expressed in languages, and we can say that a model is actually an instance of one or more languages. Then, languages are key elements in defining, describing, and examining systems in the field of MBSE. They enable the abstract representation and modeling of systems, the statement of requirements and constraints, the implementation of communication protocols and interfaces, the execution of functions, the verification and confirmation of system properties, the documentation and discussion of systems, and the support of system evolution and maintenance (Kleppe et al. 2003). Choosing and using the right languages is important to successfully create complex systems.

3.3.1 The Spectrum of Languages

The world of MBSE encompasses a wide range of languages, each serving distinct purposes and meeting specific needs:

- **Modeling Languages**: Modeling languages, such as UML and Systems Modeling Language (SysML), enable the creation of visual and textual models that capture the structure, behavior, and requirements of a system. These models serve as a blueprint for system development and facilitate communication among stakeholders (Rumbaugh 2005). The graphical nature of UML and SysML diagrams, coupled with their standardized notations and semantics, promotes a shared understanding of the system and its complexities.
- **Formal Specification Languages**: Formal specification languages, such as Z and Vienna Development Method, allow for the rigorous description of system properties, behaviors, and invariants. They help capture functional and extra-functional requirements precisely and unambiguously (Spivey 1992). The mathematical rigor of these languages enables formal analysis and verification of system properties, ensuring consistency and correctness.
- **Formal Verification Languages**: Formal verification languages, such as temporal logic and process algebras, enable the specification and checking of system properties, such as safety, liveness, and security. Model checking and theorem-proving techniques rely on these languages to verify the correctness and consistency of system models (Baier et al. 2008). The application of formal verification languages helps identify and address potential design flaws and vulnerabilities early in the development process.

3.3.2 The Importance of Language Selection

Selecting and using the right languages is very important for creating complex systems. System engineers and architects need to consider the balance between expressiveness, precision, and analysis capability when selecting modeling languages. They also need to ensure that the languages they choose meet the specific needs of the system they are developing.

By grasping how languages, models, and systems are related and using various languages effectively, system engineers and architects can create strong, clear, and maintainable representations of the system. This promotes good communication, analysis, and implementation. Smart use of languages in MBSE helps stakeholders to share their viewpoints, work well together, and make good decisions during the system development process.

3.3.3 The Challenge of Heterogeneous Systems

Building complex systems usually involves a multitude of disciplines, each having its own specific languages, tools, and perspectives. This diversity can cause major problems in communication and teamwork, which may result in misunderstandings, mistakes, and costly reworks. The biblical allegory of the Tower of Babel highlights the dangers of poor communication in large projects. In systems engineering, this means that projects can fail when different domains cannot grasp each other's language and needs.

Good communication and mutual understanding are important for decision-making and project success. While sharing documents, code, or designs is common, keeping these different data sources clear and consistent can be difficult and lead to mistakes. This highlights the importance of having a common language or framework to help stakeholders communicate and collaborate effectively.

However, using a common language has its own difficulties. It is indeed difficult to strike a balance between being detailed in specific areas and having a unified way of expressing information. To achieve this balance, particular attention needs to be paid to the complexity of the system, the diversity of stakeholders, and the specific objectives of the modeling process.

Evolution of languages

The evolution of languages is an interesting subject that has drawn the attention of various fields, such as linguistics, anthropology, and cognitive science. To understand how languages arose and evolved, we need to think about the social and environmental context in which early humans lived and spoke.

(Continued)

(Continued)

This section examines the development of languages through the prism of hunter-gatherer groups, emphasizing the mental processes involved in sharing knowledge and creating abstract ideas.

Long ago, when humans were nomadic hunter-gatherers, their main concern was to stay alive. These small groups moved with the seasons in search of food and resources while avoiding the threats of nature. In this situation, the ability to communicate well was vital to the group's survival. Imagine a scenario where a parent says to their child, "Be careful, kids, don't eat those berries; they could be harmful! You might get hurt." This simple sentence involves several complex cognitive mechanisms, including *naming, association, and abstraction*.

Naming uses words, composed of sounds, to label physical items. These words get linked to the mental images created by those who hear these sounds. These images are general ideas, since a child knows that their parent is not talking about just one berry that grew in one year in one place. Rather, the parent is sharing the idea of a berry in a broader sense, covering its key properties, such as color, shape, and the type of plant it comes from.

The word "berry" actually refers to a group of similar berries with common properties, showing how humans can organize and generalize concepts. This ability to create general ideas is a key part of human thinking, enabling ideas to be passed from one generation to the next. In hunter-gatherer groups, language probably developed because of the need to share information important for survival, such as the location of the food, the dangers posed by animals and plants, as well as their appearance. As these groups became more complex, with more social and commercial interactions, language evolved to meet new communication needs. This highlights one of the main roles of language: to identify objects, events, and experiences using words. This naming process enables humans to talk about the world around them, covering both physical items and more abstract ideas, such as feelings and beliefs.

The cognitive revolution, brought about by new tools and technologies, is important for the evolution of languages and human thinking. When people made objects that did not exist in their natural state, such as stone tools, arrows, and boats, they had to create new words to describe and explain these creations. This new way of using language helped humans to share knowledge about these technologies, facilitating their dissemination and improvement.

The ability to express the intensity or degree of an experience is an important aspect of language. By using adverbs and other words that change meanings, people can express subtle details about feelings such as hunger

or thirst. This improves communication and decision-making within a group, as people can share their needs and feelings more accurately. In the same way, assembling words into sentences is a key step in sharing information through language. By organizing words, people can share complex thoughts, recount events, and disseminate knowledge. This ability to formulate useful statements is important for the development of human thinking and the gathering of cultural knowledge. The way knowledge is shared through language has also been shaped by storytelling and myths. These stories enable complex ideas, beliefs, and values to be presented in a way that is easy to remember and share. The use of pictures and abstract ideas in stories has probably led to more advanced linguistic structures and better thinking abilities.

The emergence of farming in the Neolithic Revolution was a key moment in human history, leading to sedentary societies and complex civilizations. When people began to farm and raise animals, they encountered new problems related to ownership, laws, and the transmission of knowledge. These issues required better ways of communicating, which ultimately led to writing. The first known writing system appeared in ancient Mesopotamia around 3500 BCE, primarily serving accounting purposes. By recording information about land ownership and trade, writing helped to resolve conflicts and maintain order in society. As societies became more complex, writing extended to laws, historical accounts, and religious writings, allowing cultural knowledge to be preserved and shared.



First writing – Mesopotamia. *Source*: Osama Shukir Muhammed Amin FRCP(Glasg) / Wikimedia Commons / CC BY-SA 4.0.

As human societies developed and evolved, so did their understanding of the world. By observing nature and inventing stories, humans began to think about complex ideas that went beyond what they could see and feel. This

(Continued)

(Continued)

development of deeper thinking required a language capable of expressing these complex ideas.

In science and technology, new tools have been invented, and knowledge has become more and more specialized, leading to the development of specific vocabularies within languages. Such diversity in language allowed experts in different fields to communicate clearly, promoting the sharing of ideas and the development of human knowledge.

In summary, languages have played an essential role in shaping human thinking and culture. They allow us to name objects, describe experiences, and share knowledge through stories, enabling the transmission of cultural knowledge from generation to generation. As societies develop and face new issues, the evolution and development of languages, including computer languages, will be crucial for our cognitive and cultural evolution.

3.4 Formalization Principles

La matematica è l'alfabeto in ciu Dio ha scritto l'Universo.

Galileo Galilei

The Systems Engineering Body of Knowledge (SEBoK) emphasizes the importance of using formal models and languages in systems engineering to overcome the limitations of natural language and achieve greater precision, consistency, and analyzability (SEBoK Editorial Board 2024). To illustrate the challenges posed by natural language, consider the following two text excerpts describing the formation of the Beatles:

> The Beatles' story begins in Liverpool in 1957. A teenager named John Winston Lennon founded a band, "The Quarry Men" … in the company of a certain James Paul McCartney. Soon came George Harrison. Two years later, the obscure group seized a great opportunity, performing on a television set. An astonishing feat for such an inexperienced band. The year was 1960. Renamed "Johnny and the Moondogs" and then "The Beatles," our rockers were beginning to enjoy great success in the Liverpool area. A fourth member now officiates as drummer, a handsome young man called Pete Best.

> The story of the legendary group The Beatles began in an English town in 1957, when 17-year-old John Lennon founded The Quarry Men with Paul McCartney, two years his junior. Accompanied by George Harrison, the Quarry Men managed to make their first TV appearance in 1960, although none of them had any real musical experience. Two years later, drummer Ringo Starr joined them to form the group that would soon become The Beatles, a group recognized and adored throughout the region.

These excerpts, while understandable to humans, highlight several problems regarding the information conveyed:

- **Inconsistency**: The excerpts provide conflicting information about the drummer's identity (Pete Best vs. Ringo Starr) and the band's name changes.
- **Incompleteness**: The first excerpt lacks information about the band members' ages and musical experience, while the second excerpt omits the specific location (Liverpool).
- **Uncertainty**: The phrase "soon came George Harrison" in the first excerpt is vague and lacks a precise timeframe.
- **Overlap**: Both excerpts convey similar information about the band's formation and early years, but with slightly different wording and emphasis.

The following list summarizes the main difficulties encountered in interpreting natural language, which largely justify the use of more formal languages:

- **Homonymy**: When words have the same spelling/pronunciation but different meanings. Example: "He used a bat to hit the bat flying in the cave" (bat: sports equipment vs. animal).
- **Polysemy**: When a word has multiple related meanings. Example: "bank" → financial institution, river bank, to bank (verb).
- **Metonymy**: Using an associated element to represent the whole. Example: "The White House made a statement" (building represents the administration).
- **Metaphor**: Figurative language comparing unlike things. Example: "He's a night owl" (person with nocturnal habits compared to an owl).
- **Anaphora**: Ambiguous references to previously mentioned elements. Example: "Mary told Susan that she was promoted" (who got promoted – Mary or Susan?).
- **Pragmatics**: Context-dependent meaning. Example: "He's my right hand" (meaning trusted assistant, not literal body part).

- **Sentiment Analysis**: Detecting emotion, especially in nonliteral expressions. Example: "Great job! You broke it again..." (sarcasm where positive words convey negative sentiment).

These issues can lead to misinterpretations, miscommunications, and ultimately errors in the system design process. The SEBoK explores various approaches for addressing these challenges, including the use of formal languages and modeling techniques (SEBoK Editorial Board 2024). While machine learning (ML) offers potential solutions, it still faces limitations in handling linguistic nuances, data volume and heterogeneity, and explainability.

3.4.1 The Essence of Formalization

Formalization, as discussed in the SEBoK, involves moving from unstructured information (natural language) to structured information based on sound mathematical theories and formal languages (Halmos 2017). This transition enables the precise and unambiguous representation of system requirements, specifications, and models, facilitating consistent understanding and analysis across different stakeholders and tools.

Formalization draws upon various mathematical and logical foundations, including:

- **Predicate Logic** (Mendelsohn 2012): Enables the expression of propositions and relationships between entities in a precise and unambiguous manner.
- **Set Theory** (Halmos 2017): Provides a framework for defining and manipulating collections of objects with specific properties.
- **Relational Algebra** (Codd 1970): Enables the manipulation of data organized in tables with defined relationships between them.
- **Graph Theory** (Diestel 2017): Provides a framework for representing and analyzing relationships between entities in a network structure.

Formalization also draws upon other disciplines, such as linguistics (Chomsky 2002) and philosophy (Quine 2000) to establish a rigorous and well-defined foundation for representing and reasoning about systems.

3.4.2 Benefits of Formalization

By formalizing the information, we gain several key benefits:

- **Precision and Consistency**: Formal languages eliminate ambiguity and ensure that system requirements, specifications, and models are interpreted consistently by all stakeholders.

- **Constraint Verification**: Formalization allows us to express and verify constraints, ensuring that the system design adheres to specified rules and constraints.
- **Enhanced Reasoning and Querying**: Formal models enable logical reasoning, inference, and querying, facilitating the analysis and validation of system properties and behaviors.

3.4.3 Illustrative Example

Figure 3.6 demonstrates how the information about The Beatles can be formalized using predicate logic, sets, graphs and relational algebra, enabling clearer identification of inconsistencies and facilitating logical reasoning.

There are several points to be made about this figure, compared to the informal approach:

- Information is more quickly understood.
- Inconsistencies can be quickly detected on the basis of axioms (for example, a group can only have one date of birth).
- We can reason at different levels: "The Beatles" can be seen as a rock band or as a group of musicians.
- Information can be navigated more easily.
- Information is easier to query.

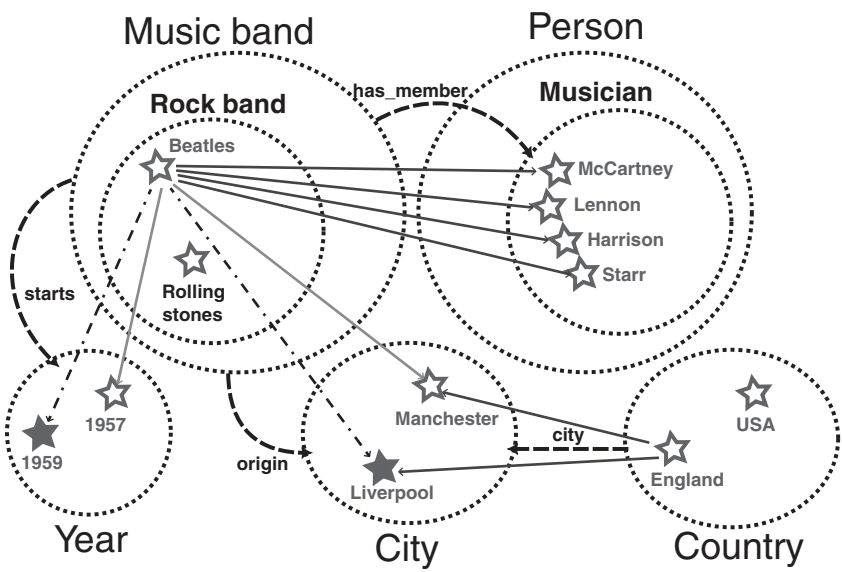


Figure 3.6 Formalization principles.

3.4.4 Selecting the Right Tools

The SEBoK highlights the need to select the right modeling languages and tools to represent the different parts of the system, such as SysML and UML. These provide standard ways of creating and understanding system models (SEBoK Editorial Board 2024; Friedenthal et al. 2014). In addition, MBSE tools and methods, such as model checking and formal verification, support the use of formal languages and modeling notations, which makes system design and analysis more reliable.

3.4.5 Balancing Formalization and Abstraction

Although formalization has many advantages, it is important to avoid making things too simple and to ensure that the models properly represent the key properties and behaviors of the system. Choosing and using formal languages and modeling techniques should come from a deep understanding of the system, its environment, and the precise objectives of the modeling activities.

3.5 The Power of Modeling

Modeling is a key element of system engineering. It enables us not only to apprehend carefully complex systems but also to gain new insights into difficult problems. By changing our perspective, we can often find simple solutions that may not be obvious. This idea is clear in modeling, where a different representation can lead to surprising ideas and important findings.

3.5.1 A Simple Yet Powerful Example

Consider the problem of finding a general formula for the sum of the n first odd numbers, expressed as $1+3+5+\ldots$. At first glance, this might seem like a daunting task requiring extensive calculations. However, by using a different representation (model), we can reveal a remarkably simple solution.

As illustrated by Figure 3.7, the key insight comes from recognizing that the sum of the first n odd numbers is equal to n^2. This elegant formula allows us to bypass the need for manual addition and provides a quick way to calculate the sum for any given number of odd terms. The proof for this formula can be derived using the arithmetic progression:

$$S_n = \frac{n}{2}[2a + (n-1)d]$$

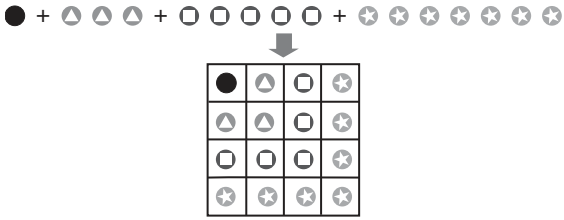


Figure 3.7 Illustration of a change of perspective in summing odd numbers.

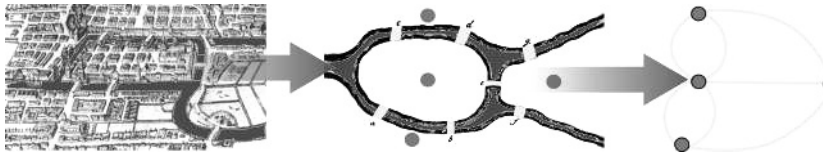


Figure 3.8 Seven bridges of Königsberg. *Source*: Wikimedia.

where $a = 1$ (first term), $d = 2$ (common difference), and the last term $l = 2n - 1$. Substituting these values, we get:

$$S_n = \frac{n}{2}[2 + (2n - 2)] = \frac{n}{2}(2n) = n^2$$

This example beautifully illustrates how a change in perspective – in this case, viewing the sum as a squared value rather than a linear addition – can lead to a powerful and generalizable solution. It demonstrates the value of exploring alternative models and representations in system engineering, as they can often "simplify" complex problems and reveal underlying patterns that might not be apparent at first glance.

3.5.2 The Seven Bridges of Königsberg

The problem of the Seven Bridges of Königsberg, which Leonhard Euler solved in 1736, was a key moment in the history of modeling. It gave birth to graph theory and contributed to the development of topology (Wikipedia Contributors 2023). The question was whether one could walk in Königsberg crossing all seven bridges exactly one time. This led Euler to create a new way of thinking beyond regular geometry.

Euler simplified the city's layout into a model using landmasses (vertices) and bridges (edges), thereby laying the foundation for graph theory (Figure 3.8). This simplification showed the value of focusing on the main relationships between elements instead of their physical characteristics, which is an important idea in topology. Euler's work introduced key notions, such as vertex degree and what we

refer to as Eulerian paths and circuits. His proof that such a walk was impossible in Königsberg not only answered the specific question but also established general rules useful for many types of network problems.

3.5.3 A Continuous Quest for New Perspectives

The development of modeling over the course of history shows a repetitive pattern of problems that have led to new solutions applicable in different fields, such as graph theory. Scientists have used specialized models and different viewpoints to address complex problems that appeared impossible. By changing the way we think about problems through these models, we have not only solved particular problems, but we have also created new fields of study and generic methods.

3.5.4 The Power of Modeling in MBSE

In MBSE, it is important to change the way we tackle problems and use models to grasp and manage the complexities of system design. By abstracting system parts and the way they work together into formal models, engineers can examine system behavior, predict performance, and spot design issues or possible improvements. This facilitates decision-making and helps create strong, efficient, and reliable systems.

Exercise: Exploring alternative representations

Consider a simple system, such as a traffic intersection. Explore different ways to represent this system using various modeling techniques, such as:

- A state machine diagram to capture the different states of the traffic lights and their transitions.
- A Petri Net to model the flow of vehicles through the intersection.
- A mathematical equation to calculate the optimal timing of the traffic lights based on traffic flow and congestion.

Discuss the advantages and limitations of each representation, how they can contribute to different aspects of system design and analysis, and how they can be confronted.

3.5.5 Summary

The history of modeling shows how the ongoing difficulty of grasping complex systems has pushed scientists to look for new views and ways to express ideas. This continuous effort to abstract and change points of view has not only helped

to solve particular issues but has also broadened our ability to study and understand our environment, resulting in major advances in various fields of research and real-world use. In the following chapter, we will present an overview of the most recognized theoretical models, examining how changes in point of view and formal modeling are still influencing our understanding of complex systems and driving innovation in system engineering.

3.6 Models and Semantics

As we mentioned before, the use of models instead of traditional documents is important in systems engineering. This reduces the risks of misunderstandings and imprecise meanings that can occur with natural language. For models to work well, they need clear semantics. This raises the question: how do we give a clear and exact meaning to a modeling language? Different types of semantics exist in the literature for modeling languages, each with unique characteristics and uses.

3.6.1 Informal Semantics

Informal semantics uses natural language to express meaning. It is common in glossaries, system requirements, and other documents where accuracy might not be critical. The advantage of informal semantics is its flexibility and ease; it can be understood by many people without special skills. However, the casual nature of natural language can cause unclear meanings and misunderstandings, which can hinder effective communication between stakeholders.

3.6.2 Operational Semantics

Operational semantics describes how a system behaves by detailing operations that can change its state. It is often used to explain how modeling languages work, with rules guiding how the state changes. The intuitive aspect of operational semantics matches the way modelers think about model execution. However, as systems become larger and more complex, managing operational semantics can become complex and difficult.

3.6.3 Denotational Semantics

Denotational semantics provides a mathematical basis for understanding modeling languages by linking language parts to mathematical objects. This method offers a clear and defined meaning of the language, facilitating reasoning, analysis, and optimization. However, using this method often requires significant mathematical knowledge, which could limit its use for some practitioners.

3.6.4 Axiomatic Semantics

Axiomatic semantics examines the logical characteristics of models and helps to prove their correctness. It establishes rules and axioms about the behavior of system parts. This method enables formal proofs and verifications, guaranteeing that models comply with the required properties and constraints. However, axiomatic semantics may not work well for all parts of the model, and it can be difficult for those lacking a solid formal logic background.

3.6.5 Pragmatic Semantics

Pragmatic semantics focuses on how context affects meaning in language use. It goes beyond the literal meaning and takes into account elements such as modeler intention and context to better understand meaning. While pragmatic semantics gives a deeper understanding of language, its focus on context can make interpretation tricky, especially when the context isn't clearly defined or shared by all involved.

3.6.6 The Train and Gateway Example

Far from being mutually exclusive, these various semantics can complement each other, each serving different purposes. For example, informal semantics help with human communication, operational semantics enable the creation of interpreters for simulation, and axiomatic semantics facilitate formal proofs of system properties.

To illustrate how different semantics can be applied in MBSE, let's look at an example from UPPAAL models. Imagine a railway system with several trains using a crossing that is managed by a gateway. The main objective is to guarantee the safe and effective operation of this system. Hence, several properties shall be verified:.

- **Safety**: Preventing collisions and unsafe conditions is paramount. State machines, with their explicit management of transitions, can be used to model and verify the system's behavior, ensuring that unsafe states, such as a train entering the crossing when it's not clear, are avoided.
- **Coordination**: The interaction between trains and the gateway requires precise synchronization. State machines can capture this coordination, ensuring that only one train occupies the crossing at a time.
- **Efficiency**: Efficient management of states and transitions optimizes train flow and minimizes waiting times, enhancing overall system throughput.

Figure 3.9 depicts this scenario, showcasing how state machines can abstract the behavior of trains and the gateway at a high level. This representation allows us to predict system behavior and proactively identify potential safety hazards, provided

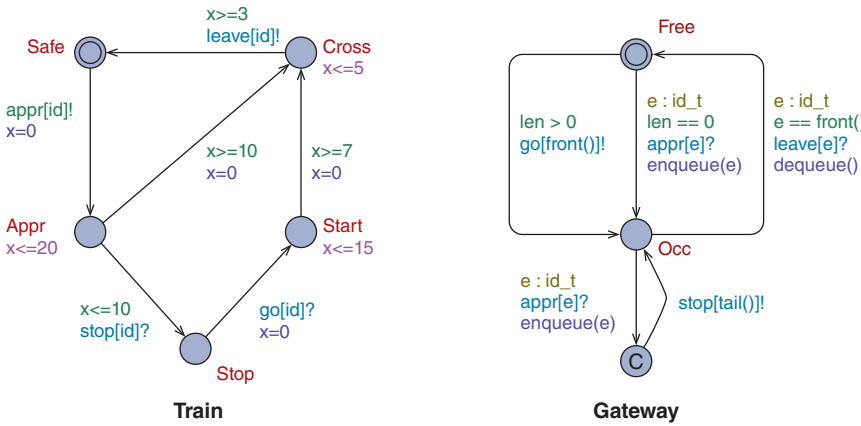


Figure 3.9 The train/gateway problem.

the semantics of the model are clear and unambiguous. Three types of semantics can be associated with these state machines:

- An *informal semantics* explaining to humans how these state machines should be interpreted.
- An *operational semantics*, telling the simulation engine how these state machines should be executed.
- An *axiomatic semantics* telling the verification tool how to interpret these state machines symbolically, thus enabling an exhaustive exploration of all possible execution paths (State Space Exploration).

3.7 The Different Types of Model

We have noticed that models act as simplified representations of systems, showing their main features to aid in requirements, design, analysis, verification, and validation. In MBSE, knowledge of the different types of models is crucial for good system representation, analysis, and decision-making.

3.7.1 Descriptive vs. Prescriptive Models

Models can be broadly classified into two main categories based on their purpose and function:

- **Descriptive Models**: These models are designed to represent or describe the current state or behavior of a system, process, or phenomenon. They use statistical analysis techniques to extract meaningful information from data,

such as mean, standard deviation, distribution, and correlations. Descriptive models provide an overview of past and present conditions, helping to identify patterns and trends in data behavior. They are often used for exploratory analysis and to gain an in-depth understanding of the characteristics of the data. In essence, descriptive models focus on summarizing and understanding the system as it exists, without necessarily predicting future states or prescribing actions.

- **Prescriptive Models**: In contrast, prescriptive models are designed to provide recommendations or guidance on what actions to take based on the available data. These models use complex algorithms to process data and generate suggestions about possible actions to take, attempting to quantify the effect of future actions to recommend possible outcomes before they are actually executed. For example, a prescriptive model can suggest the best pricing strategy to maximize a company's profits, considering various factors such as cost, demand, and competition. Thus, prescriptive models aim to optimize decision-making by suggesting specific decisions that should be taken to achieve desired outcomes.

3.7.2 Static vs. Dynamic Models

Another important distinction in models is between static and dynamic models:

- **Static Models**: These models represent a system at a particular point in time. They provide a snapshot of the system's structure or state, without considering changes over time. Static models are often used for representing physical objects, system architectures, or data structures. Examples of static models include 3D CAD models, UML class diagrams, and entity-relationship diagrams.
- **Dynamic Models**: On the other hand, dynamic models capture the behavior and evolution of a system over time. They incorporate the temporal aspects and interactions within the system, allowing for the analysis of system dynamics and performance. Dynamic models are particularly useful for studying complex systems with time-dependent behaviors. Examples of dynamic models include simulation models, state machines, and differential equations.

3.7.3 Models Across the Representation Spectrum

Beyond the descriptive/prescriptive and static/dynamic classifications, the spectrum of models used in MBSE encompasses various other types, each characterized by unique attributes and applications:

- **Mathematical Models**: These models use mathematical equations, formulas, or algorithms to describe, analyze, and predict the behavior of systems. They are widely used in various fields, such as physics, engineering, economics, and computer science, to simulate and optimize system performance, make predictions, and support decision-making.
- **Physical Models**: These models are tangible, scaled representations of objects, systems, or processes. They provide a hands-on, tactile experience of a system or product, aiding in visualization, communication, and testing. Physical models are often used for prototyping, demonstrating functionality, and evaluating design concepts.
- **Digital Models**: These models are computer-based representations of systems, processes, or objects using software tools. They enable the creation, simulation, and analysis of virtual representations of systems, facilitating design exploration, testing, and optimization before physical implementation. Digital models are widely used in various industries, such as automotive, aerospace, and manufacturing.
- **Formal Models**: These models are precise, mathematical representations of systems or processes using formal languages, notations, or logics. They are used to specify, verify, and analyze the behavior and properties of systems in a rigorous and unambiguous manner. Formal models are commonly used in fields such as computer science, software engineering, and formal methods to ensure the correctness, reliability, and safety of critical systems.

3.7.4 The Importance of Model Selection

Selecting the right model depends on the context, objective, and needs of the project. Using the right models helps organizations to represent, analyze, communicate, and make smart decisions about systems, processes, and phenomena (Estefan et al. 2007). The challenge is to find and fix any inconsistencies by aligning viewpoints.

A bit of history

In the year 1418, the Arte della Lana held a competition for designing the dome of Florence Cathedral, using earlier designs by Neri di Fioravanti. Brunelleschi won the competition to design and construct the dome. His genius lay in figuring out how to build the ambitious dome on the cathedral's enlarged octagonal space.

(Continued)

(Continued)



Florence Cathedral. *Source*: Fitamant / Wikimedia Commons / CC BY SA 3.0.

Brunelleschi did use a model, which enabled him to win the competition to build the dome of the Florence Cathedral in 1418. His design was considered an "architectural impossibility" as it aimed to build the largest brick dome without any external supports. The model likely demonstrated the practicality of his innovative approach. He impressed the judges with his concepts and won the contract, even though he was not a typical candidate due to his background as a goldsmith and sculptor, not as an architect or engineer.

3.8 Models and Diagrams

Diagrams in MBSE are often mistakenly regarded as the models themselves. However, this is not the case. Diagrams are merely visual representations of system models, serving as a means to communicate, analyze, and manage the complexity of the system.

To illustrate this distinction, consider the analogy of tables and views in a database:

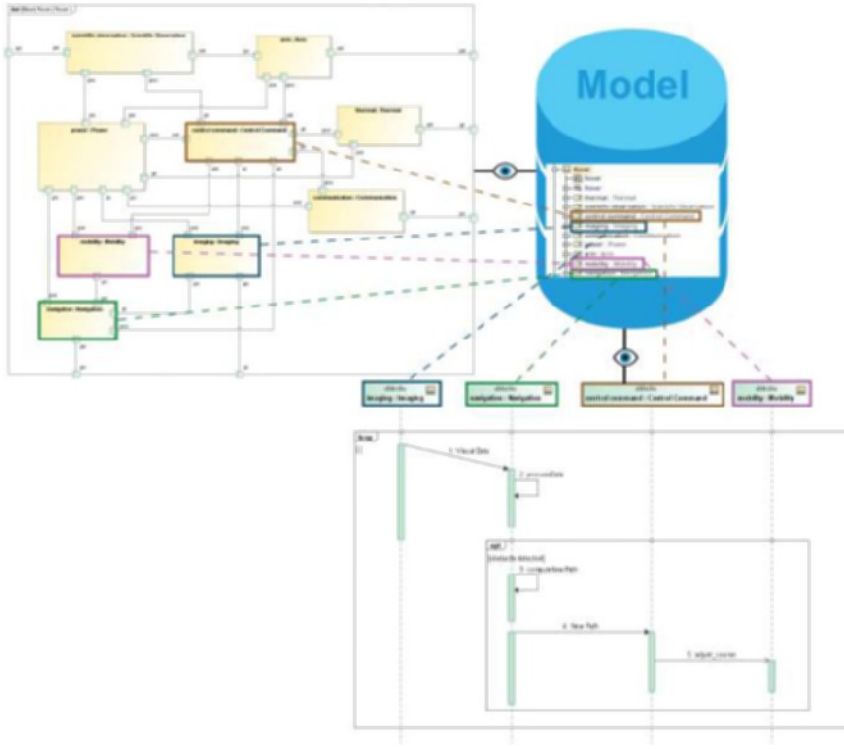


Figure 3.10 Models vs. diagrams.

- **Tables (Model)**: A table in a database stores the actual data, analogous to the model itself containing structured information about the system. The model is the core repository of system knowledge, capturing all relevant details and relationships.
- **Views (Diagrams)**: A view presents a selected subset or tailored representation of the data from a table, analogous to how diagrams show specific aspects or viewpoints of the underlying model. Diagrams are derived from the model and provide a focused perspective on particular aspects of the system.

3.8.1 One Model, Multiple Views

As illustrated in Figure 3.10, the same model element can be visualized in multiple diagrams, with possibly different shapes, depending on the viewpoint or aspect being highlighted. This multiplicity of views is essential for capturing the various facets of a complex system.

This example shows how the constituents of a component can appear in a block definition diagram (structural view) and how they can also appear in a sequence diagram (interaction view).

3.8.2 Beyond the Visual

It is crucial to emphasize that models are not just visual diagrams. Textual notations and specifications, such as formal requirements languages, also represent parts of the overall model. These textual representations might not always have a corresponding diagram but are integral to the model's completeness and accuracy.

3.8.3 Benefits of the Diagram-model Distinction in MBSE

Here are the main benefits of making the clear distinction between models and diagrams:

- Consistency and Traceability:
  - By clearly separating diagrams and models, MBSE promotes consistency in design and enhances traceability between different viewpoints. This consistency ensures that changes to the model are reflected accurately across all relevant diagrams, linking back to requirement traceability discussed in Chapter 7.
  - For instance, if a component's properties are updated in the model, all diagrams that include this component will reflect these changes, maintaining consistency and reducing errors.
- Facilitating Communication:
  - Diagrams help in understanding the complex relationships within the model by presenting tailored views for specific stakeholders or analysis purposes. This fosters clearer communication and facilitates collaboration among team members.
  - Different stakeholders can focus on the diagrams that are most relevant to their concerns, ensuring that everyone has a clear understanding of the system from their respective viewpoints.
- Evolution and Change Management:
  - The underlying model remains a consistent reference point, allowing for flexibility and adaptability in system design. This flexibility is emphasized in Chapter 2, where modifications and evolution to the system's design can be managed without having to entirely recreate diagrams from scratch.
  - Changes to the system can be reflected in the model, and subsequently, the relevant diagrams can be updated to reflect these changes, ensuring that the visual representations remain aligned with the evolving system design.

3.8.4 Summary

The main point of this section is the need to build clear and organized models that truly reflect the system being examined. Diagrams help to show, analyze, and manage complexity, but their value and accuracy come from the model itself.

By keeping a clear difference between models and diagrams, practitioners of MBSE can maintain consistency, traceability, and good communication throughout the system development process. This approach leads to the development of robust, flexible, and context-sensitive systems that satisfy stakeholder and user requirements. This difference is crucial for applying MBSE effectively in the creation of complex systems.

3.9 System and Viewpoints

In a systems engineering project, many experts share their perspectives on the system at different levels of abstraction. This variety of viewpoints raises the issue of how to combine these different viewpoints, each in its own language and suited to a certain abstraction level and purpose. The model, at any abstraction level, can be seen as an aggregation of these viewpoints, which need to be merged and integrated for a clear and complete understanding of the system (International Council on Systems Engineering [INCOSE] 2023). The relationships between systems, languages, models, and viewpoints are intricate and interconnected. Let's take a closer look at these links (Figure 3.11).

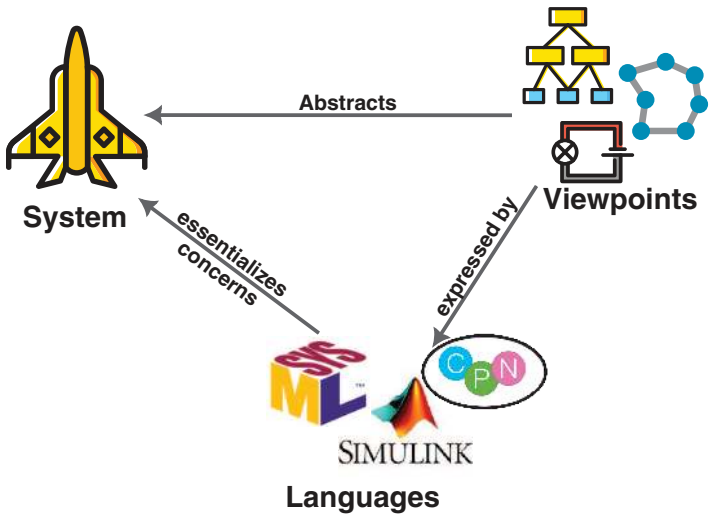


Figure 3.11 System, languages, and viewpoints.

3.9.1 The Significance of Viewpoints

Systems can be considered from various viewpoints or perspectives. Each viewpoint highlights certain parts or issues of the system, such as functionality, performance, security, or user experience (ISO/IEC/IEEE 2011). Different groups, such as customers, domain experts, or system engineers, may have different views on the system based on their goals or interests. The ISO 42010 standard seeks to streamline architecture description practices by defining important ideas such as architecture views and viewpoints. A viewpoint defines the rules for creating a view that addresses specific stakeholder concerns in an architecture description (ISO/IEC/IEEE 2011).

3.9.2 Tailoring Languages and Models to Viewpoints

Languages can be customized to emphasize different points of view about a system. Domain-specific languages (DSLs) are made to capture ideas, terms, and rules specific to a certain domain or perspective (Voelter and Benz 2013). DSLs allow stakeholders to state system requirements, constraints, and features using language and concepts that matter most to their view. Using several languages, each tailored to a specific viewpoint, allows for a better and more detailed representation of the system.

Each viewpoint aims to capture the important aspects and features of the system from a certain perspective. For instance, a functional model shows how the system behaves and interacts, while a performance model highlights its performance properties and resource use (Clements et al. 2010). Models created from different viewpoints offer a comprehensive understanding of the system, enabling stakeholders to confront their points of view and reach a compromise.

3.9.3 The City of Paris Analogy

To illustrate viewpoints, think of the city of Paris. It is a complex place with big roads, small streets, different buildings, and various types of people. The way we represent Paris depends on what we want to achieve. Are we trying to improve roads, fight poverty and inequality, reduce pollution, help tourists, or examine the city's growth policies? Each of these goals needs different ways to represent information that helps us to better understand those challenges, enabling us to create appropriate solutions. The different images in Figure 3.12 highlight the various issues that can be addressed. However, this figure does not show **mutual influences** between these representations, such as how tourism can affect pollution or changes in population density, which are also very important to consider.

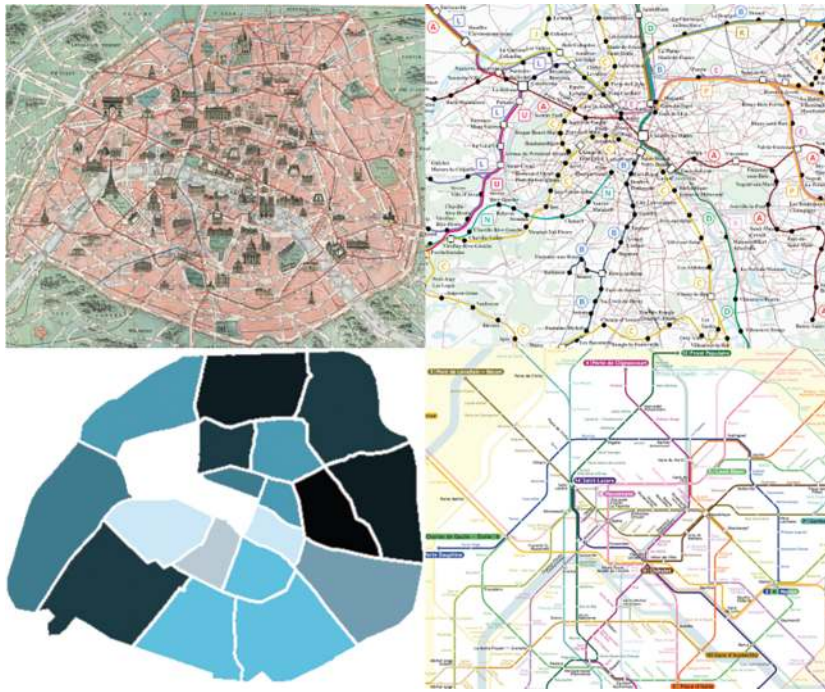


Figure 3.12 Illustration of viewpoints. *Source*: Wikimedia.

Indeed, decisions made from one perspective will impact the structure and organization of others, requiring constant adjustment until agreement is reached. For instance, the installation of a metro station in an area can change the social and economic environment, which can either bring risks (higher costs, poverty) or opportunities (economic growth).

3.9.4 The Dynamic Interplay

The links between systems, languages, models, and perspectives are changing and iterative. Languages build models that show systems from various perspectives. These models help in understanding, analyzing, and discussing the system. Different perspectives can lead to changes or improvements in the models and the system. Good system design and engineering require understanding these links and using them well. By using proper languages to create models that reflect different viewpoints, system engineers can handle complexity, maintain consistency, and support effective communication and teamwork among stakeholders (Friedenthal et al. 2014).

Figure 3.11 shows that models result from a mix of different expert viewpoints, possibly using different tools and languages. This highlights the difficulty of combining various languages with different paradigms, which is still a research area in MBSE.

It is essential to remember that all these models, providing a specific perspective on the system, are linked; they develop together, and changes in one viewpoint can lead to changes in several other viewpoints.

3.9.5 Summary

Knowing how systems, languages, models, and viewpoints relate is important for good system design and engineering. Using the right languages and models to represent different viewpoints helps system engineers manage complexity, keep consistency, and improve teamwork between stakeholders.

In a way, the UML language can be thought of as a mixture of sublanguages aimed at solving different issues. Although this language does not address all concerns, the composition of its sublanguages, while helping to ensure consistency, remains a subject of ongoing research. Globally speaking, issues related to language composition and viewpoint integration are still under investigation, but MBSE presents a valuable framework to tackle these challenges and support the effective development of complex systems.

Exercise: The human body and its viewpoints

The human body, much like the city of Paris, is a complex system that can be viewed and understood from various perspectives. Consider the following viewpoints and the types of models or representations that might be associated with each:

- **The Anatomist's View**: Focuses on the structural organization of the body, its organs, tissues, and cells.
- **The Physiologist's View**: Examines the functions and processes of the body, such as respiration, circulation, and digestion.
- **The Physician's View**: Concerned with diagnosing and treating diseases and injuries, focusing on symptoms, causes, and treatment options.
- **The Athlete's View**: Emphasizes the body's performance, strength, and endurance, focusing on training, nutrition, and injury prevention.
- **The Artist's View**: Appreciates the aesthetic qualities of the human form, focusing on proportions, movement, and expression.

Please answer the following questions:

- For each viewpoint, identify specific types of models or representations that might be used to capture and communicate the relevant information. (e.g. anatomical diagrams, physiological flowcharts, medical imaging, training plans, artistic sketches)
- How might these different viewpoints be reconciled or integrated to gain a more comprehensive understanding of the human body? What challenges might arise in this process?
- Can you think of other viewpoints that might be relevant to understanding the human body? How might these additional viewpoints contribute to a more holistic perspective?

Consider a scenario where a patient is experiencing fatigue and shortness of breath. The physician might use medical imaging to identify a potential heart problem, while the physiologist might conduct tests to assess lung function and oxygen uptake. By integrating these different viewpoints and their related models, a more comprehensive diagnosis can be reached, leading to a more effective treatment plan.

3.10 Interface Design and Management

In MBSE, interfaces are important points where different system parts or various perspectives in a model meet and share information. The success of MBSE depends on careful design and management of these interfaces to ensure that complex systems work together smoothly and effectively (Friedenthal et al. 2014). This section looks into the details of interfaces in MBSE, including what they are, how they are modeled and managed throughout their lifecycle, and how they are verified.

3.10.1 Defining Interfaces

In the MBSE framework, an interface is a shared boundary that allows information, energy, or material to flow between two or more parts of the system. This includes internal interfaces that manage interactions within a system and external interfaces that connect the system to its environment or other external systems. Interfaces include the rules, protocols, and constraints that frame these interactions, ensuring that the information exchanged is meaningful, consistent, and compatible.

3.10.2 Modeling Techniques for Interfaces

MBSE uses various modeling techniques and notations to clearly represent interfaces in system models. These techniques provide a clear specification of interface properties, behaviors, and constraints, helping stakeholders to understand and analyze the complex interactions between system parts. Here are some commonly used approaches:

- **Interface Diagrams**: These diagrams offer a visual representation of interfaces, showcasing the connected components, the nature of the exchanged information or signals, the protocols, and any relevant constraints.
- **Port-and-connector Notations**: This approach models interfaces as ports on components linked by connectors that represent the flow of information or energy. It fosters a structured and modular representation of interfaces, facilitating their reuse and management.
- **Interface Definition Languages (IDLs)**: IDLs provide a formal and precise mechanism for defining interfaces, meticulously specifying the data types, operations, and protocols supported. They enable automated code generation and streamline interoperability between components developed using diverse languages or platforms.

3.10.3 Interface Management Throughout the Lifecycle

Interface management is a continuous effort that spans the entire system lifecycle, from the initial elicitation of requirements to the final stages of system integration and testing (Friedenthal et al. 2014). Within MBSE, this process is supported by a suite of tools and techniques that enable the definition, analysis, and evolution of interfaces as the system design evolves. Key activities in this process include:

- **Requirements Analysis**: The early identification and capture of interface requirements, ensuring their traceability to system-level objectives and constraints.
- **Interface Design**: The meticulous definition of the structure, behavior, and protocols governing interfaces, taking into account factors such as performance, reliability, and security.
- **Interface Analysis**: The systematic analysis of interfaces to proactively identify potential conflicts, inconsistencies, or performance bottlenecks.
- **Interface Verification and Validation (V&V)**: The rigorous assurance that interfaces adhere to their specified requirements and function seamlessly within the broader system context.
- **Interface Change Management**: The effective management of changes to interfaces throughout the system's lifecycle, ensuring consistent propagation across all affected models and components.

3.10.4 Interface V&V

V&V of interfaces are essential for guaranteeing their intended functionality and contribution to the overall system's correctness and reliability. This multifaceted process includes:

- **Static Analysis**: A meticulous examination of interface models and specifications to ensure consistency, completeness, and adherence to established design rules and standards.
- **Simulation and Testing**: The simulation and testing of interfaces under diverse scenarios and conditions, encompassing error handling and fault tolerance, to verify their behavior.
- **Formal Verification**: The application of formal methods to rigorously prove the correctness of interface designs and implementations, particularly crucial for safety-critical systems.

3.10.5 Challenges and Best Practices

The design and management of interfaces in complex systems can be fraught with challenges, as interfaces often represent integration points where conflicts can arise between different components or viewpoints. Some common challenges include:

- **Ambiguity and Incompleteness**: Interface specifications may suffer from ambiguity or incompleteness, potentially leading to misunderstandings and integration issues.
- **Inconsistency**: Interfaces may evolve independently, resulting in inconsistencies and compatibility problems.
- **Performance Bottlenecks**: Poorly designed interfaces can introduce performance bottlenecks that can adversely impact the overall system's performance.

To navigate these challenges, it is crucial to adopt best practices for interface design and management, such as:

- **Early and Continuous Involvement**: Engaging stakeholders from diverse disciplines early in the design process to ensure comprehensive capture and addressing of interface requirements.
- **Clear and Unambiguous Specifications**: Using formal notations and modeling techniques to define interfaces with clarity and precision, leaving no room for ambiguity.
- **Version Control and Configuration Management**: Implementing robust version control and configuration management practices to meticulously track changes to interfaces and maintain consistency across different models and components.

- **Automated Testing and Verification**: Leveraging automated tools and techniques to rigorously test and verify interfaces throughout the development lifecycle.
- **Collaboration and Communication**: Fostering open communication and collaboration between different teams and stakeholders involved in interface design and implementation.

By adhering to these best practices and harnessing the capabilities of MBSE, system engineers can effectively manage interfaces, ensuring the seamless integration and interoperability of complex systems.

3.10.6 Connecting the System With Its Environment

Interfaces are important for managing how internal parts of a system work together and for connecting a system to its environment. They enable the system to sense, engage with, and impact its environment (Rechtin 2017). This allows the system to achieve its goals in its working environment. Key aspects include:

- **Sensing the Environment**: A system's ability to perceive and respond to its environment hinges on its interfaces with sensors and other input devices. These interfaces capture data from the external world, such as temperature, pressure, light, or user commands, and translate them into a format that the system can process and understand. The design of these interfaces must consider factors such as sensor accuracy, noise filtering, and data synchronization to ensure that the system receives reliable and timely information about its environment.
- **Acting Upon the Environment**: Similarly, interfaces enable a system to act upon its environment through actuators, displays, or other output devices. These interfaces translate the system's internal commands and data into actions that affect the external world, such as controlling motors, displaying information, or communicating with other systems. Interface design in this context must address issues such as actuator precision, response time, and safety to ensure that the system's actions are accurate, timely, and do not pose any risks to its environment.
- **Adapting to Change**: The environment in which a system operates is rarely static. Changes in external conditions, user needs, or technological advancements can necessitate adaptations in the system's behavior or functionality. Well-designed interfaces can facilitate such adaptations by providing a flexible and modular framework for interaction with the environment. This allows

the system to evolve and remain relevant in the face of changing circumstances, without requiring major redesigns.

- **Ensuring Compatibility and Interoperability**: In many cases, a system needs to interact with other systems or components that may have been developed independently or by different organizations. Interfaces play a crucial role in ensuring compatibility and interoperability between these disparate entities. The use of open standards and well-defined interface specifications can facilitate seamless integration and data exchange, enabling the system to collaborate effectively with its external counterparts.
- **Interface Design for the Environment**: MBSE provides a powerful framework for designing and managing interfaces that connect a system to its environment. Through the use of models, simulations, and analysis techniques, engineers can explore different interface options, evaluate their impact on system behavior and environmental interactions, and make informed trade-offs between competing factors. MBSE also enables the early identification and mitigation of interface-related risks, ensuring that the system can operate safely and effectively within its intended environment.

3.10.7 The Rover Case

The logical architecture diagram of the Mars rover (Figure 2.1) shows a mixture of subsystems, each acting as a separate unit with specific roles. However, the real strength of the system is not only in these individual units but in their teamwork toward the rover's mission. This teamwork relies on clear interfaces that act as communication channels, indicating how data and commands flow between the subsystems. For example, the Navigation Subsystem may provide an interface for the Mobility Subsystem to use positioning data, while the Science Subsystem could have an interface to send collected information to the Communication Subsystem.

These interfaces, although not visible in the logical architecture diagram, are crucial for turning a bunch of separate subsystems into a cohesive, working system. They set the rules for information exchange, enabling each subsystem to function properly while supporting the rover's overall objectives. Careful design and handling of these interfaces are critical in systems engineering, impacting the system's performance, reliability, and flexibility. The rover's external interfaces are vital, allowing it to sense its environment, interact with it, and communicate with Earth. Clear external interfaces are necessary for the rover to move, carry out scientific exploration, and withstand the harsh conditions on Mars.

Exercise: Identifying and defining interfaces in the Rover's logical architecture

Consider the logical architecture diagram (Figure 2.1) of the Mars rover, which depicts various subsystems.

Part 1: Interface Identification

Examine the diagram and identify at least three crucial interfaces between different subsystems. For each interface:

1. Name the two subsystems it connects.
2. Describe the type of information or signals that would likely be exchanged through this interface.
3. Explain the importance of this interface for the rover's overall mission success.

Part 2: Interface Specification

Choose one of the interfaces you identified and elaborate on its potential specifications. Consider aspects such as:

1. Data formats and protocols used for communication
2. Timing and synchronization requirements
3. Error handling and fault tolerance mechanisms
4. Security and access control considerations

Part 3: Interface Evolution

Discuss how the requirements or design of an interface might evolve during the rover's development lifecycle. Consider factors such as:

1. Changes in subsystem functionalities or capabilities
2. Integration of new sensors or instruments
3. Adaptation to unforeseen environmental conditions on Mars
4. Lessons learned from testing and simulation

Part 4: MBSE and Interface Management

Explain how MBSE tools and techniques can support the design, analysis, and management of interfaces in the rover project. Consider aspects such as:

1. Modeling notations for representing interfaces (e.g. interface diagrams, port-and-connector notations)
2. Simulation and analysis tools for verifying interface behavior and performance

3 Traceability mechanisms for linking interface requirements to system-level objectives
4 Collaboration platforms for facilitating communication and coordination among subsystem teams

3.10.8 Summary

Interfaces are important for a system to sense, engage with, and impact its environment. They enable communication, data sharing, and adjustment, helping the system to achieve its goals in its working environment. In the field of MBSE, proper design and supervision of both internal and external interfaces are essential to system realization and the satisfaction of stakeholder requirements.

3.11 Modeling Standards and Tools

The success of MBSE mostly depends on clear modeling languages, common data formats, and strong tools for making, analyzing, and managing system models. This section looks at important standards and tools related to MBSE, emphasizing how they help with effective and efficient MBSE.

3.11.1 Modeling Languages

Several modeling languages are commonly used in MBSE, each with its strengths and limitations depending on the specific needs of the project. Here are some prominent examples:

- **UML**: Originally developed for software engineering, UML offers a broad range of modeling diagrams for representing various aspects of software systems (Rumbaugh 2005). While not as specifically tailored to systems engineering as SysML, UML can still be used effectively for modeling certain aspects of systems, such as software components and their interactions (Gomaa 2011).
  - Advantages: Widely adopted and understood, large community and resources, flexible and adaptable to different domains.
  - Limitations: Less specialized for systems engineering compared to SysML, potential for ambiguity and inconsistency in interpretation, may require additional profiles or extensions for specific system aspects.

- **SysML**: SysML provides a comprehensive set of graphical and textual modeling constructs for representing system requirements, structure, behavior, and parametrics (Friedenthal et al. 2014). Its standardized notation and semantics facilitate communication and collaboration among stakeholders, and enable the integration of different system views (Delligatti 2013).
  - Advantages: Rich set of diagrams for various system aspects, strong support for requirements traceability, standardized notation, and semantics.
  - Limitations: Steeper learning curve compared to UML, may be overkill for simpler systems, tool support can vary in maturity and features.
- **AADL**: A domain-specific language for modeling software-intensive systems, particularly in the context of real-time and embedded systems (Feiler and Gluch 2012). AADL provides constructs for representing system architecture, behavior, and performance characteristics, enabling analysis and verification of system properties (Hugues et al. 2008).
  - Advantages: Strong support for real-time and embedded systems modeling, focus on architecture and performance analysis, enables formal V&V.
  - Limitations: Less widely adopted compared to SysML or UML, steeper learning curve, may require specialized tools and expertise.
- **Modelica**: A language for modeling complex cyber–physical systems, including their behavior and dynamics (Fritzson 2015). Modelica enables the creation of executable models that can be used for the simulation and analysis of system performance (Tiller 2012).
  - Advantages: Powerful for modeling and simulating cyber–physical systems, supports multi-domain modeling, enables optimization, and design exploration.
  - Limitations: Less suitable for modeling nonphysical aspects of systems, such as requirements or software behavior, may require specialized tools and expertise.

The choice of modeling language depends on the specific needs of the project, the domain of application, and the expertise of the modeling team. It is essential to select a language that is expressive enough to capture the relevant aspects of the system while being understandable and usable by the relevant stakeholders (Estefan et al. 2007).

3.11.2 Data Exchange Formats

Interoperability between different modeling tools and platforms is crucial for effective MBSE. Standardized data exchange formats enable the sharing and reuse of models across different tools and facilitate collaboration among diverse teams (Estefan et al. 2007). Here are some commonly used data exchange formats:

- **XML Metadata Interchange (XMI)**: A standard format for exchanging UML and SysML models, XMI enables the transfer of models between different modeling tools (Object Management Group 2015).
- **Requirements Interchange Format (ReqIF)**: A standard format for exchanging requirements, ReqIF enables the transfer of requirements between different requirements management tools and facilitates the integration of requirements with system models (Object Management Group 2016).
- **Comma-separated Values (CSV)**: A simple and widely used format for exchanging data in tabular form. CSV can be used for exporting data from models for further analysis or for importing data into models from other sources (Shafranovich 2005).

3.11.3 Modeling Tools

Numerous software tools support MBSE, offering various features and capabilities for creating, analyzing, and managing system models (Estefan et al. 2007). Some popular MBSE tools include:

- **Cameo Systems Modeler**: A comprehensive MBSE tool that supports SysML and other modeling languages, offering features for model simulation, analysis, and document generation.
- **Enterprise Architect**: A widely used UML modeling tool that also supports SysML and other modeling languages, offering features for model simulation, code generation, and team collaboration.
- **PTC Integrity Modeler**: An MBSE tool that integrates with other PTC products for requirements management and lifecycle management, offering features for model simulation and analysis.

The selection of modeling tools depends on various factors, including budget, functionality, ease of use, and integration with other tools and processes within the organization. It is crucial to select tools that meet the specific needs of the project and support the chosen modeling language and methodology.

3.11.4 Modeling Standards Organizations

Several organizations are involved in the development and maintenance of modeling standards relevant to MBSE. Some of the key organizations include:

- **OMG**: Responsible for the development and maintenance of UML, SysML, and other modeling standards.
- **INCOSE**: Plays a leading role in promoting MBSE practices, offering resources and guidance on MBSE methodologies and tools.

- **ISO and IEEE**: Develops and maintains various standards related to systems and software engineering, including those relevant to MBSE.

These organizations play a crucial role in ensuring the interoperability and consistency of modeling languages, tools, and practices, contributing to the overall advancement and adoption of MBSE.

3.11.5 Emerging Trends and Technologies

The MBSE tool landscape is constantly evolving, with new trends and technologies emerging to address the growing complexities of system design and development. Some of the notable trends include:

- **Cloud-based Modeling Platforms**: These platforms offer the flexibility and scalability to access and collaborate on models from anywhere, anytime, promoting distributed teamwork and reducing infrastructure costs.
- **Collaborative Modeling Environments**: These environments enable real-time collaboration and concurrent editing of models, fostering communication and knowledge sharing among team members.
- **Integration of Artificial Intelligence (AI) and ML**: AI and ML techniques are being integrated into modeling tools to automate tasks, provide intelligent recommendations, and enhance decision-making.

3.12 Modeling Requirements

As mentioned in the previous chapter, systems engineering and requirements engineering are related and help each other. Models are important for improving different parts of requirements engineering. Using models, MBSE offers a clear and visual way to gather system requirements. This improves communication, analysis, and validation during the system lifecycle (Friedenthal et al. 2014; SEBoK Editorial Board 2024).

3.12.1 Capturing Stakeholder Needs

The SEBoK shows that involving stakeholders and understanding their needs and expectations is a key step in the systems engineering process (SEBoK Editorial Board 2024). MBSE facilitates this process by providing tools and methods to clearly capture stakeholders' needs. For example, use cases, context, and activity models help to elicit and record both functional and extra-functional requirements (Weilkiens 2011). These models act as a shared language for discussion and

collaboration between stakeholders and the development team, helping to improve understanding and agreement on system requirements. Here are two diagrams that are commonly used to capture stakeholder needs:

- **Use Case Diagrams**: Visually depict the interactions between actors (users or external systems) and the system under development, helping in the identification of key functionalities and scenarios that the system should support (Holt 2021).
- **Context Diagrams**: Illustrate the system boundaries and its interfaces with external entities, clarifying the scope and context of the system within its environment (SEBoK Editorial Board 2024; Holt 2021).

3.12.2 Specifying and Managing Requirements

MBSE helps define requirements using clear modeling languages and notations such as SysML (Friedenthal et al. 2014). Requirements can be shown as model parts within diagrams such as requirement diagrams, use case diagrams, and state machines. This provides a clear visual means of understanding and analyzing them.

The SEBoK emphasizes the need to manage and trace requirements throughout the system lifecycle (SEBoK Editorial Board 2024). Models in MBSE help to break down high-level requirements into detailed requirements, ensuring that they are complete and consistent. For example, requirements diagrams in SysML help to organize requirements in a hierarchy, showing their relationships and dependencies (Holt 2021). This makes it easier to manage complex requirements and keep track of them, supporting a stable requirements base.

MBSE also boosts requirements traceability by clearly linking requirements, system components, and other model elements. From these models, traceability matrices or diagrams can be created to show how requirements affect system design, implementation, and testing (Weilkiens 2011). This traceability is essential for checking the completeness and consistency of requirements, spotting any gaps or issues, and handling changes in requirements throughout the system lifecycle (SEBoK Editorial Board 2024).

3.12.3 Validating and Verifying Requirements

Models in MBSE enable requirements to be verified using analysis, simulation, and testing (Friedenthal et al. 2014). Executable models, such as state machines or activity diagrams enable system behavior to be simulated and verified against specific requirements. Model-based testing techniques, such as test cases generation from the models, can be used to create tests that cover the requirements

well, which aligns with the SEBoK's discussion about verification and validation processes (Holt 2021; SEBoK Editorial Board 2024).

3.12.4 Iterative Refinement and Evolution

In addition, MBSE helps to update and improve requirements during the system's lifecycle (Weilkiens 2011). As the design of the system evolves and new information comes to light, the requirements models can be changed to reflect the latest understanding of the system and its environment. Models allow stakeholders to work together to review, discuss, and provide feedback on requirements. Change impact analysis can be conducted on the models to see how changes in requirements may affect the system's design, helping to make better decisions and manage risks.

The SEBoK highlights the need to use tools and platforms to manage requirements (SEBoK Editorial Board 2024). Tools and platforms for MBSE provide features such as version control, change tracking, and monitoring the status of requirements. These tools can be used to keep one source of truth and ensure that the requirements align with the system models (Friedenthal et al. 2014). This connection facilitates requirements management and helps to keep the requirements and system design in sync as the development process progresses.

3.12.5 Summary

The relationship between MBSE and requirements engineering is important for creating complex systems successfully. Using models, MBSE improves the effectiveness and efficiency of requirements engineering, which in turn improves system quality, reduces development time, and boosts customer satisfaction. Combining models and requirements engineering, as suggested by the SEBoK, is key to managing complexity and delivering systems that fulfill the needed functionality, quality, and performance requirements in today's fast-changing technological environment (SEBoK Editorial Board 2024).

3.13 Conclusion

In this chapter, we examined the fundamental principles of modeling in the field of MBSE. We began by highlighting the important role of models in human history, going back to the earliest efforts to make sense of the world. Next, we examined the relationships between systems, languages, and models, showing how languages help to express models and the balance between expressiveness and manageability. The Rugby metamodel served as a useful tool for understanding the

properties of computational models and how they evolve through different levels of abstraction.

We also reviewed the various types of models used in MBSE, which include descriptive and prescriptive models, as well as static and dynamic models, alongside mathematical, physical, digital, and formal models. We emphasized the need to select the right models based on the specific needs and objectives of a project. The concept of viewpoints was discussed, illustrating how different perspectives can be captured in models for a clearer understanding of complex systems. The advantages of formalization were discussed, showing how formal languages and modeling methods can improve accuracy, consistency, and analysis in system design.

Moreover, this chapter has highlighted the relationship between modeling and requirements engineering, showing how models can connect stakeholder requirements and system design. It discussed the role of models in requirements gathering, analysis, specification, validation, and management, emphasizing their role in improving communication, teamwork, and traceability in the system development process.

As we move to the next chapter on "Modeling Paradigms," we will expand on these basic modeling ideas and examine some specific theoretical models that support MBSE. We will go over different modeling techniques, such as object-oriented modeling or hierarchical breakdown, and analyze how they apply to representing the structure, behavior, and characteristics of complex systems. The next chapter will enhance our understanding of the theoretical concepts that influence the creation and use of models in MBSE, helping us manage the challenges of system design and development effectively.

Questions

1 Which of the following best describes model transformation in MBSE?
   - **a** The process of converting models into natural language descriptions
   - **b** The automatic or semiautomatic conversion of one model into another, often at a different level of abstraction or for a different purpose
   - **c** The manual rewriting of models to improve their readability
   - **d** The elimination of models after system implementation

2 Which of the following is a key benefit of using models in MBSE?
   - **a** Increased ambiguity in system design
   - **b** Enhanced communication and collaboration
   - **c** Reduced precision in system specifications
   - **d** Increased complexity in documentation

(Continued)

(Continued)

3 What is the primary purpose of a formal language in MBSE?
- **a** To increase the expressiveness of natural language
- **b** To provide a flexible framework for system design
- **c** To ensure precision and reduce ambiguity
- **d** To enhance the aesthetic appeal of models

4 Which of the following best describes the trade-off between expressivity and tractability in modeling languages?
- **a** More expressive languages are always more tractable.
- **b** Less expressive languages are more difficult to use.
- **c** More expressive languages are less tractable.
- **d** Tractability is independent of expressivity.

5 In the context of MBSE, what is the role of a viewpoint?
- **a** To provide a single perspective on a system
- **b** To capture specific stakeholder concerns
- **c** To eliminate the need for multiple models
- **d** To simplify the system architecture

6 Which of the following is an example of a dynamic model?
- **a** UML class diagram
- **b** 3D CAD model
- **c** State machine diagram
- **d** Entity-relationship diagram

7 What is the main advantage of using SysML over UML in systems engineering?
- **a** SysML is simpler to learn than UML.
- **b** SysML is specifically tailored for systems engineering.
- **c** SysML has fewer diagrams than UML.
- **d** SysML is more widely used in software engineering.

8 How does MBSE enhance requirements traceability?
- **a** By eliminating the need for requirements documentation
- **b** By establishing explicit links between requirements and system elements
- **c** By using natural language exclusively
- **d** By reducing the number of system models

9 What is the significance of the Rugby metamodel in MBSE?
- **a** It provides a framework for understanding computational models.
- **b** It simplifies the process of model creation.
- **c** It focuses on the aesthetic design of models.
- **d** It eliminates the need for formal languages.

10 Which of the following is a challenge associated with transitioning from natural languages to formal languages in system design?
- **a** Increased ambiguity in system specifications
- **b** Reduced precision in system models
- **c** Steeper learning curve for formal languages
- **d** Decreased analyzability of system properties

Answers

1-b; 2-c; 3-c; 4-b; 5-c; 6-b; 7-b; 8-a; 9-c; 10-b

References

Baier, C., Katoen, J.P., and Larsen, K.G. *Principles of Model Checking*. The MIT Press. MIT Press, 2008. ISBN 9780262304030. https://books.google.fr/books?id=5dvxCwAAQBAJ.

Bruegge, B. and Dutoit, A.H. *Object-oriented Software Engineering: Using UML, Patterns, and Java*. Prentice Hall, 2010. ISBN 9780136061250. https://books.google.fr/books?id=VYdf2ONuRZIC.

Chomsky, N. *Syntactic Structures*. A Mouton classic. Mouton de Gruyter, 2002. ISBN 9783110172799. https://books.google.fr/books?id=SNeHkMXHcd8C.

Clarke, E.M., Grumberg, O., Kroening, D., et al. *Model Checking, 2*e. Cyber Physical Systems Series. MIT Press, 2018. ISBN 9780262038836. https://books.google.fr/books?id=OJV5DwAAQBAJ.

Clements, P., Bachmann, F., Bass, L., et al. *Documenting Software Architectures: Views and Beyond*. SEI Series in Software Engineering. Pearson Education, 2010. ISBN 9780132488594. https://books.google.fr/books?id=UTZbsrA4qAsC.

Delligatti, L. *SysML Distilled: A Brief Guide to the Systems Modeling Language*. Pearson Education, 2013. ISBN 9780133430332. https://books.google.fr/books?id=3bMJAgAAQBAJ.

Diestel, R. *Graphentheorie*. Springer Lehrbuch. Springer, 2017. ISBN 9783961340040. https://books.google.fr/books?id=3pFRDwAAQBAJ.

Dijkstra, E. The humble programmer. In *ACM Turing Award Lectures*, 1972:1.

Codd, E.F. A relational model of data for large shared data banks. *Communications of the ACM* 1970:13(6):377–387.

Estefan, J.A. Survey of model-based systems engineering (MBSE) methodologies. *INCOSE MBSE Focus Group* 2007:25(8):1–12.

Feiler, P.H. and Gluch, D.P. *Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language*. SEI Series in Software Engineering. Pearson Education, 2012. ISBN 9780133132908. https://books.google.fr/books?id=T_nOltBoX64C.

Friedenthal, S., Moore, A., and Steiner, R. *A Practical Guide to SysML: The Systems Modeling Language*. The MK/OMG Press. Elsevier Science, 2014. ISBN 9780128002025. https://books.google.fr/books?id=97XfoAEACAAJ.

Fritzson, P. *Principles of Object-Oriented Modeling and Simulation with Modelica 3.3: A Cyber-Physical Approach*. Wiley, 2015. ISBN 9781118859162. https://books.google.fr/books?id=wgIaBgAAQBAJ.

Gomaa, H. *Software Modeling and Design: UML, Use Cases, Patterns, and Software Architectures*. Cambridge University Press, 2011. ISBN 9781139494731. https://books.google.fr/books?id=TqZi-hAX17YC.

Halmos, P.R. *Naive Set Theory*. Dover Books on Mathematics. Dover Publications, 2017. ISBN 9780486814872. https://books.google.fr/books?id=lgdJDgAAQBAJ.

Holt, J. *Systems Engineering Demystified: A practitioner's handbook for developing complex systems using a model-based approach*. Packt Publishing, 2021. ISBN 9781838985448. https://books.google.fr/books?id=EL0YEAAAQBAJ.

Hugues, J., Zalila, B., Pautet, L., et al. From the prototype to the final embedded system using the ocarina AADL tool suite. *ACM Transactions on Embedded Computing Systems (TECS)* 2008:7(4):1–25.

INCOSE. *INCOSE Systems Engineering Handbook*. Wiley, 2023. ISBN 9781119814290. https://books.google.fr/books?id=HhjBEAAAQBAJ.

International Organization for Standardization, International Electrotechnical Commission, Institute of Electrical, Electronics Engineers, and IEEE-SA Standards Board. *2476 - ISO/IEC/IEEE International Standard—Systems and Software Engineering—Vocabulary*. IEEE Std. IEEE, 2010. ISBN 9780738162058. https://books.google.fr/books?id=d5c8zQEACAAJ.

ISO/IEC/IEEE. Systems and software engineering–architecture description. *ISO/IEC/IEEE* 42010: *2011 (E)(Revision of ISO/IEC 42010: 2007 and IEEE Std 1471-2000*), 2011:1–46.

Jantsch, A. *Modeling Embedded Systems and SoC's: Concurrency and Time in Models of Computation*. Electronics & Electrical. Elsevier Science, 2004. ISBN 9781558609259. https://books.google.fr/books?id=GMamoSg9A00C.

Kleppe, A.G., Warmer, J.B., and Bast, W. *MDA Explained: The Model Driven Architecture: Practice and Promise*. Object technology. Addison–Wesley, 2003. ISBN 9780321194428. https://books.google.fr/books?id=-uodwVPBG6YC.

Mendelsohn, E. *Introduction to Mathematical Logic*. The Wadsworth & Brooks/Cole Mathematics Series. Springer US, 2012. ISBN 9781461572886. https://books.google.fr/books?id=UkP0BwAAQBAJ.

Object Management Group. OMG unified modeling language (OMG UML), superstructure, August 2011. https://www.omg.org/spec/UML/2.5.1.

Object Management Group. Xml metadata interchange (XMI) specification. 2015. https://www.omg.org/spec/XMI/2.5.1/.

Object Management Group. Requirements interchange format (ReqIF), version 1.2. 2016. https://www.omg.org/spec/ReqIF/1.2/ (accessed 24 April 2023).

Quine, W.V.O. Two dogmas of empiricism. *Perspectives in the Philosophy of Language* 2000:189–210.

Rechtin, E. *Systems Architecting of Organizations: Why Eagles Can't Swim*. Systems Engineering. CRC Press, 2017. ISBN 9781351411943. https://books.google.fr/books?id=Ars6DwAAQBAJ.

Rumbaugh, J. *The Unified Modeling Language Reference Manual*. Addison–Wesley object technology series. Pearson Education, 2005. ISBN 9788178081069. https://books.google.fr/books?id=SGewzQEACAAJ.

SEBoK Editorial Board. *The Guide to the Systems Engineering Body of Knowledge (SEBoK)*. INCOSE, 2024.

Selic, B. The pragmatics of model-driven development. *IEEE software* 2003: 20(5):19–25.

Shafranovich, Y. Common format and mime type for comma-separated values (CSV) files. Technical report 2005.

Spivey. M. *The Z Notation: A Reference Manual*. Prentice Hall international Series in Computer Science. Prentice Hall, 1992. ISBN 9780139785290. https://books.google.fr/books?id=wa1QAAAAMAAJ.

Tiller. M. *Introduction to Physical Modeling with Modelica*. The Springer International Series in Engineering and Computer Science. Springer US, 2012. ISBN 9781461515616. https://books.google.fr/books?id=7vcGCAAAQBAJ.

Voelter, M. and Benz, S. *DSL Engineering: Designing, Implementing and Using Domain-specific Languages*. CreateSpace Independent Publishing Platform, 2013. ISBN 9781481218580. https://books.google.fr/books?id=J2i0lwEACAAJ.

Weilkiens, T. *Systems Engineering with SysML/UML: Modeling, Analysis, Design*. The MK/OMG Press. Elsevier Science, 2011. ISBN 9780080558318. https://books.google.fr/books?id=LKC9F5gQt3AC.

Wiegers, K.E. and Beatty, J. *Software Requirements*. Developer Best Practices. Pearson Education, 2013. ISBN 9780735679627. https://books.google.fr/books?id=nbpCAwAAQBAJ.

Wikipedia Contributors. Seven bridges of Königsberg, 2023. https://en.wikipedia.org/wiki/Seven_Bridges_of_K%C3%B6nigsberg.

4

Modeling Paradigms

4.1 Structural Models

Structural modeling approaches are important for model-based systems engineering (MBSE), as they allow for the representation of the basic configuration and connections in complex systems (Friedenthal et al. 2014). They are essential for building clear, complete, and organized models that help in system design, analysis, and communication between all parties involved.

As illustrated in Figure 2.1 (Chapter 2), structural models in MBSE show the main parts of a system, such as its components, connections, and layered structure. They offer a straightforward representation of what the system is made of, helping engineers to better manage complexity.

4.1.1 Purposes and Benefits

Structural models provide several key benefits in MBSE processes:

- **Complexity Management**: By breaking down complex systems into manageable components and relationships, structural models help engineers handle intricate system architectures.
- **Improved Communication**: These models serve as a common reference point for diverse stakeholders, enhancing understanding and reducing misinterpretations.
- **Consistency and Traceability**: Structural models ensure consistency across different system views and enable traceability between system elements and requirements.
- **Design Optimization**: By providing a clear view of system structure, these models facilitate the identification of design improvements and optimizations.

4.1.2 The Evolution of Structural Modeling

The field of structural modeling in MBSE has evolved to include different techniques, each with its own benefits and uses. The approach chosen often relies on what type of system is being modeled, the particular objectives of the modeling activity, and the preferences of the systems engineering team.

4.1.3 Challenges in Structural Modeling

However, it's important to acknowledge some challenges associated with structural modeling:

- **Model Complexity**: As systems become more intricate, structural models can become complex and difficult to manage (Madni et al. 2019).
- **Static Nature**: While excellent for representing system structure, these models may not capture dynamic system behavior adequately (Friedenthal et al. 2014).
- **Tool Limitations**: The effectiveness of structural modeling can be constrained by the capabilities of available modeling tools (Estefan 2007).
- **Integration Challenges**: Integrating structural models with other types of models (e.g. behavioral, performance) can be challenging (Weilkiens 2011).

4.1.4 Key Structural Modeling Approaches

In the following subsections, we will examine the important structural modeling techniques used in MBSE. We will cover what they are, where they can be used, and how they contribute to good system representation and analysis. Knowledge of these techniques is important for systems engineers to select and use the right ones for their projects and domains.

4.1.4.1 Object-oriented Models

Object-oriented models in MBSE use concepts from object-oriented programming in systems engineering. This provides an effective means of representing and analyzing complex systems (Booch et al. 2007). The object-oriented systems engineering method, proposed by the Object Management Group (OMG), combines object-oriented techniques with traditional systems engineering techniques to improve the efficiency of system-level development.

Key features of object-oriented models in MBSE include:

- **Abstraction**: Representing complex systems as simplified models by focusing on essential characteristics while hiding unnecessary details.
- **Encapsulation**: Grouping related data and functions into objects and promoting information hiding.

- **Inheritance**: Allowing objects to inherit properties and behaviors from parent classes, facilitating reuse, and creating hierarchical relationships between system elements.
- **Polymorphism**: Enabling objects to take on multiple forms and respond differently to the same message based on their specific type, enhancing flexibility in system design.

Object-oriented models in MBSE usually use languages such as Systems Modeling Language (SysML). These languages provide a standard way to represent system structures, behaviors, and requirements based on object-oriented concepts.

The advantages of using object-oriented models in MBSE include:

- Improved modularity and reusability of system components.
- Enhanced flexibility in system design and easier accommodation of changes.
- Better alignment with real-world concepts, facilitating stakeholder understanding.
- Seamless integration of systems engineering with object-oriented software engineering.

As MBSE develops, combining object-oriented models with other modeling methods is likely to become more important for managing the complexity of today's engineered systems (INCOSE 2023).

4.1.4.2 Hierarchical Decomposition

Hierarchical decomposition is an important technique in systems engineering. It involves dividing complex systems into simpler parts, such as subsystems and components. This approach is key for understanding, analyzing, and designing intricate systems in many fields. Hierarchical decomposition can be approached from two primary directions:

- **Top-down Decomposition**: This approach starts with the overall system and progressively breaks it down into subsystems and components (Wikipedia Contributors 2023). It emphasizes planning and a complete understanding of the system before delving into details. Top-down decomposition is particularly useful for:
  - Identifying system-level requirements and allocating them to subsystems
  - Providing a clear overview of the system architecture
  - Facilitating communication among stakeholders
- **Bottom-up Composition**: This method involves building up the system from individual components. It starts with specifying the details of base

elements and then linking them to form larger systems. Bottom-up composition is advantageous for:
  - Leveraging existing components or technologies
  - Enabling early testing and validation of individual modules
  - Promoting reusability and modularity in system design

A critical aspect of hierarchical decomposition is the maintenance of an appropriate level of granularity at each decomposition level, known as the *whole-to-part ratio* or *steepness index* (Chiriac et al. 2011). Finding the optimal balance is essential for effective system modeling and analysis. The hierarchical decomposition approach offers several benefits in systems engineering:

- It provides a systematic way to manage complexity in large-scale systems.
- It facilitates the allocation of requirements to specific subsystems and components.
- It supports modular design, allowing for easier system modification and maintenance.
- It enables parallel development of different system components, potentially reducing overall development time.

However, it's important to note that hierarchical decomposition also has potential challenges:

- Ensuring consistency across different levels of abstraction can be difficult.
- There's a risk of losing sight of system-wide properties when focusing on individual components.
- The decomposition process itself can be subjective and may vary based on the designer's perspective.
- A "broad and shallow" structure may not adequately represent important subsystems.
- An excessively narrow structure can be impractical for complex system representation.

To address these challenges and leverage the strengths of both top-down and bottom-up approaches, many system engineering methodologies use a hybrid strategy. This combined approach allows for:

- A comprehensive understanding of the system at multiple levels of abstraction
- Iterative refinement of the system design
- Balancing between system-level goals and component-level constraints
- Flexibility in adapting to changes in requirements or technologies

When using hierarchical decomposition in MBSE, languages such as SysML offer standard methods to represent system hierarchies using diagrams such as

block definition diagrams and internal block diagrams (IBDs) (Friedenthal et al. 2014). These diagrams make it easier to share system structure between the various stakeholders and enable different analyses during the life of the system. An example of this technique can be seen in Figure 2.1 (Chapter 2), which shows an IBD of the rover system.

4.1.4.3 Functional Decomposition

Functional decomposition is a key technique in systems engineering. It involves dividing a complex system or process into smaller, easier-to-handle parts based on their specific functions (van Eck et al. 2007). This technique is especially useful in the early stages of system design and analysis, as it enables complexity to be understood without restricting the design to particular physical solutions too early.

Key Steps: The process of functional decomposition typically involves the following steps:

1. **Identify Main Functions**: Determine the primary functions of the system based on stakeholder needs and goals.
2. **Decompose into Subfunctions**: Break down main functions into smaller and more specific subfunctions until reaching a level of granularity where functions are simple enough to be easily understood and managed.
3. **Analyze Relationships**: Examine the relationships and dependencies between different functions and subfunctions.
4. **Map to Physical Components**: Relate functions to the physical components that perform them, often using a functions/components matrix.

Representation Techniques: Functional decomposition can be represented using various techniques, including:

- **Hierarchical Structures**: Tree diagrams or nested diagrams showing different levels of abstraction.
- **Flowcharts or Process Maps**: Visual diagrams illustrating the interconnections between different functions.
- **Data Flow Diagrams**: Graphical representations of data flows within a system.
- **Functional Decomposition Diagrams**: Specialized diagrams showing the breakdown of functions in a top-down format.

Benefits and Applications: Functional decomposition offers several advantages in systems engineering:

- Simplifies representations of complex systems, making them easier to understand and analyze.
- Facilitates requirements gathering and analysis.

- Improves communication among stakeholders by providing clear visual representations.
- Supports modular design and promotes reusability of functions.
- Enables parallel development of different system components.

This technique finds applications in various fields, including software engineering, business process improvement, and robotics.

Limitations and Considerations: While functional decomposition is a powerful tool, it's important to be aware of its limitations:

- Risk of oversimplification, potentially overlooking complex interactions between components.
- Time-consuming process, especially for large and complex systems.
- May not capture all aspects of a system, such as extra-functional requirements or external influences.
- Success depends on the domain expertise of those performing the analysis.

In summary, functional decomposition is a useful technique in systems engineering. When applied correctly, it enables complex systems to be understood and created, facilitating communication and requirements gathering. However, it is important to use it in conjunction with other analysis techniques to gain an overall view of the system under study.

Example: Figure 4.1 shows the functional decomposition of an IFE system extracted from a model provided by the Capella tool, illustrating its main components and interactions. This decomposition highlights essential functionalities such as audio and video management, passenger interaction, and content control. The diagram shows how passengers interact with the system to access services, while the cabin crew can manage content selection. Additionally, integration with onboard systems for data exchange and ground operations for content updates is described.

This diagram provides a clear overview of the system's functional architecture, making it easier for stakeholders to understand and discuss. Still, as highlighted in the previous chapter, it is important to be cautious when interpreting the meanings of functional decomposition diagrams to prevent misunderstandings and bad decisions.

4.1.4.4 Assembly Decomposition

Assembly decomposition is a systematic approach to understand and analyze complex systems based on their physical configuration and how they are assembled (Chiriac et al. 2011). This approach is particularly used in product design, manufacturing, and systems engineering because it provides an understanding of how parts and subassemblies relate to each other.

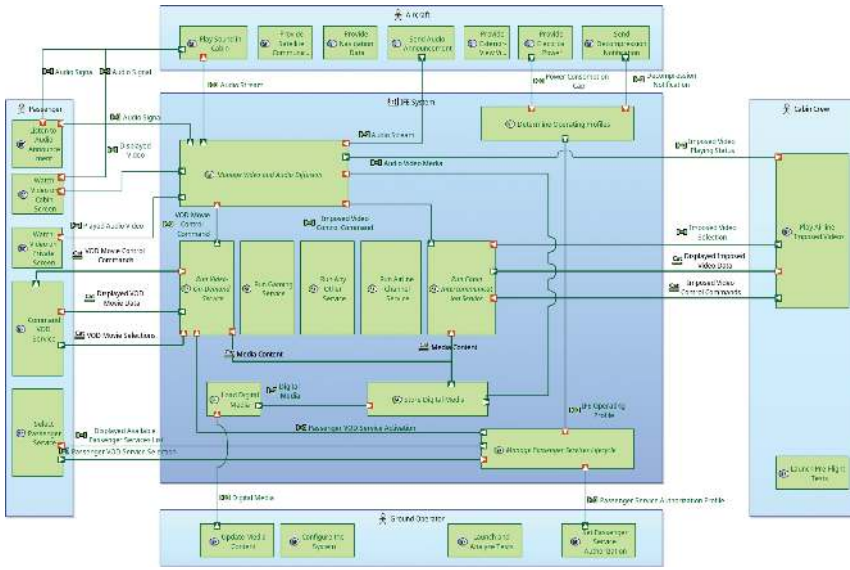


Figure 4.1 Functional decomposition of an in-flight entertainment (IFE) system.

Key Steps: The process of assembly decomposition typically involves the following steps:

1. **Visual Inspection**: Observe how components are physically attached. This step involves a detailed examination of the system's physical structure, identifying individual parts and their connections.
2. **Identify Subassemblies**: Determine which parts can be easily assembled as units. This step focuses on grouping components that form logical subassemblies based on their physical relationships.
3. **Remove Large Subassemblies**: Identify and remove large chunks or subassemblies that can be easily separated from the main system.
4. **Record in Design Structure Matrix (DSM)**: Document the decomposition using a DSM. The DSM is a powerful tool for representing and analyzing the relationships between components in complex systems.

Key Considerations: Key considerations in assembly decomposition include:

- **Granularity**: Maintain an appropriate level of detail, considering the parts-to-whole ratio at each decomposition level.
- **System Integrity**: Keep certain subsystems (e.g. HVAC, drives, controls) intact during decomposition stages.

- **Hierarchical Structure**: The decomposition process naturally creates a hierarchical structure, with each level representing a more detailed breakdown of the system.

Benefits and Applications: The assembly decomposition approach offers several benefits in systems engineering:

- It provides a clear understanding of the physical structure of complex systems.
- It facilitates the identification of modular components and potential areas for design improvement.
- It supports the optimization of assembly processes and manufacturing strategies.
- It aids in maintenance planning and spare parts management.

Limitations: However, it's important to note that assembly decomposition also has some limitations:

- It may not capture all functional relationships between components, as some interactions may not be visible through physical inspection alone.
- For highly integrated systems, it can be challenging to identify clear subassemblies.
- The process can be time-consuming for very complex systems with numerous components.

To solve these problems, assembly decomposition is commonly combined with other system analysis techniques, such as functional decomposition or network analysis. This combined approach gives a better understanding of the physical and functional parts of complex systems.

In real-world use, assembly decomposition has produced good results in different fields, such as car manufacturing, airplane engineering, and consumer electronics. For example, in car manufacturing, assembly decomposition helps improve vehicle design for simpler production and maintenance activities.

Example: Figure 4.2 presents the assembly decomposition of an IFE system extracted from a model provided by the Capella tool, illustrating its physical components and their interconnections. This representation provides a clear and organized view of the system's hardware architecture, showing the relationships between units such as the Private Video Display Unit and Cabin Management Unit.

Key benefits of this assembly decomposition include:

- **Clarity in System Architecture**: Visually breaks down the system into manageable components, improving comprehension.
- **Component Relationships**: Highlights connections and dependencies between processors, network adapters, and other units.

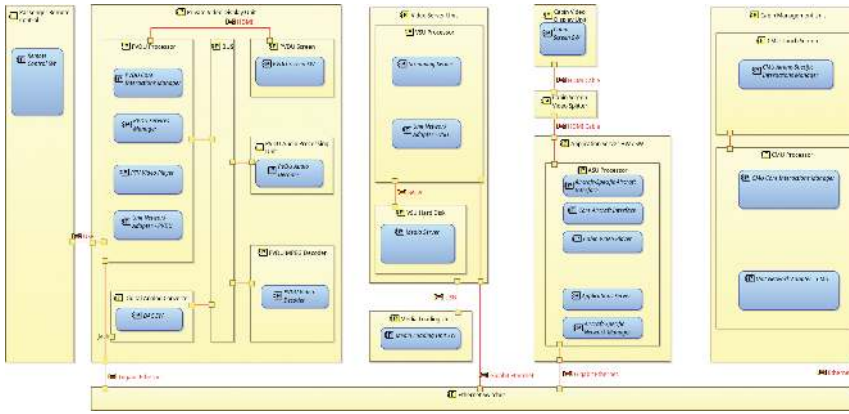


Figure 4.2 Assembly decomposition of an IFE system.

- **Facilitates Maintenance and Upgrades**: Enables identification of independently maintainable or upgradeable components.
- **Supports Manufacturing and Assembly**: Provides a blueprint for accurate system assembly.
- **Enhances Communication**: Serves as a shared reference for effective collaboration across teams.

4.1.4.5 Service-based Decomposition

Service-based decomposition is a way of breaking down a system by examining the services it offers. This aligns well with service-oriented architecture principles (Laskey and Laskey 2009). This technique is especially useful for software-intensive systems and complex configurations, as it focuses on separate services instead of large, single structures.

Key Characteristics: Service-based decomposition is characterized by the following elements:

- **Service Identification**: Defining core services that the system provides to users or other systems.
- **Interface Definition**: Specifying standardized interfaces for service access, often using protocols such as SOAP or REST.
- **Loose Coupling**: Ensuring services are independent and can be modified without affecting other system parts.
- **Reusability**: Designing services to be reusable across different applications or contexts.

Advantages: Service-based decomposition offers several benefits in systems engineering:

- **Modular Design**: Facilitates easier maintenance and updates of individual services.
- **Interoperability**: Promotes seamless interaction between different systems and platforms.
- **Scalability**: Supports flexible scaling of specific services based on demand.
- **Alignment with Modern Practices**: Complements cloud-native applications and microservices architectures.

Challenges: Despite its advantages, service-based decomposition also presents some challenges:

- **Granularity**: Determining the appropriate level of service granularity can be tough.
- **Dependency Management**: Ensuring consistency and managing dependencies between services requires careful planning.
- **Performance Overhead**: Network communication between services may introduce latency.
- **Complexity:** The distributed nature of services can increase system complexity.

Applications: Service-based decomposition finds applications in various domains:

- **Enterprise Software**: For creating flexible, scalable business applications.
- **Cloud Computing**: In designing distributed cloud services and microservices architectures.
- **Internet of Things (IoT)**: For managing diverse, interconnected devices and their services.
- **E-commerce Systems**: In building modular, scalable online platforms.

In summary, service-based decomposition provides a powerful way to design complex, distributed systems. Its focus on modularity, reusability, and loose coupling fits with current software development methods and the requirements for scalable, adaptable architectures. Nevertheless, to succeed, careful thought must be given to the size of services, dependency management, and overall system complexity.

Exercises

1. Identify the main structural elements (components, interfaces, relationships) of a bicycle.
2. Create a simple structural model (using any suitable representation) for a home security system, including components such as sensors, alarms, and control panels.

4.2 Behavioral Models

Dynamic modeling approaches are important in MBSE because they show how complex systems act over time (Estefan 2007). While structural models give a fixed picture of system design, behavioral models help us understand how a system changes, reacts to inputs, and works with its environment (Friedenthal et al. 2014). These models allow engineers to simulate and verify system behavior before creating physical prototypes, helping to reduce development costs and risks (Madni et al. 2019).

4.2.1 Types of Behavioral Models

In MBSE, there are generally three types of behavioral modeling: continuous, discrete, and hybrid behavior models (Madni et al. 2019). Continuous models usually use differential equations to capture systems that change smoothly over time (Friedenthal et al. 2014). Discrete models, including state machines and Petri Nets, are used for systems that change state in discrete steps or with certain events (Estefan 2007).

Many real-world systems exhibit both continuous and discrete behaviors; this has led to hybrid modeling techniques, which mix continuous and discrete behavioral representations to better capture and analyze such systems. Indeed, hybrid models are helpful in MBSE for systems that involve both physical processes (usually continuous) and control logic (usually discrete), including cyber–physical systems. This combination facilitates important systems engineering activities, such as requirements analysis, design creation, and system verification and validation (V&V).

There are many types of models that we can't cover here, and this chapter aims to inform the reader that there is a wide variety of models covering different domains and needs. We will take a closer look at some behavioral modeling techniques more closely in the next sections, focusing on their uses, advantages, and drawbacks within MBSE. We will review particular techniques and tools for modeling both continuous and discrete behaviors, along with mixed methods that link the two.

4.2.2 Continuous Behavior Models

Models of continuous behavior are key elements of MBSE for representing systems with changes that flow smoothly over time (Friedenthal et al. 2014). These models are essential for explaining physical processes, control systems, and situations where state variables change continuously rather than in discrete steps

(Cellier and Greifeneder 2013). The importance of continuous behavior models in MBSE stems from several key factors:

- **Accurate Representation**: They provide a more precise description of many real-world systems, especially those governed by physical laws (Hart 2015).
- **Early V&V**: Continuous models enable engineers to simulate and analyze system behavior before physical prototypes are built, reducing development costs and risks (Madni et al. 2019).
- **Integration with Control Systems**: They are fundamental for designing and analyzing feedback control systems, which are prevalent in many engineered systems (Hatcliff et al. 2012).
- **Multi-domain Modeling**: Continuous models facilitate the integration of different physical domains (e.g. mechanical, electrical, thermal) within a single system model (Karnopp et al. 2012).

In MBSE, continuous behavior models typically fall into two main categories:

- **Differential Equation-based Models**: These models use ordinary differential equations (ODEs) or partial differential equations (PDEs) to describe system dynamics (Cellier and Greifeneder 2013).
- **Bond Graphs**: A domain-independent graphical approach for modeling multi-energy domain systems (Karnopp et al. 2012).

The choice between these approaches is generally based on the type of system being modeled, the level of detail needed, and the team's preferences (Friedenthal et al. 2014).

In the next subsections, we will look at these continuous modeling methods more closely, covering their characteristics, uses, and drawbacks in the field of MBSE. We will also look at the difficulties in using these models and the techniques for solving them, such as numerical simulation techniques, and ways of validating the models.

4.2.2.1 Differential Equations

Differential equations are essential for many continuous behavior models in MBSE (Cellier and Greifeneder 2013). They provide a sound mathematical framework for explaining how systems change over time, making them crucial for modeling dynamic systems in various fields of engineering (Hart 2015).

There are two main types of differential equations used in MBSE:

- **ODEs**: These equations involve functions of a single independent variable (typically time) and their derivatives. ODEs are used to model systems where state variables change with respect to a single parameter.

- **PDEs**: These equations involve functions of multiple independent variables and their partial derivatives. PDEs are used for systems where state variables depend on multiple parameters, such as time and space.

The application of differential equations in MBSE spans various domains, including:

- **Mechanical Systems**: Modeling motion, vibration, and structural dynamics.
- **Electrical Systems**: Analyzing circuits and electromagnetic phenomena.
- **Thermal Systems**: Describing heat transfer and temperature distribution.
- **Fluid Dynamics**: Modeling fluid flow and aerodynamics.
- **Control Systems**: Designing and analyzing feedback control systems.
- **Behavioral Sciences**: Representing changes in psychological or social variables over time.

In the context of MBSE, differential equations offer several advantages:

- **Continuous Representation**: They provide a smooth, continuous description of system behavior, capturing subtle changes and interactions.
- **Predictive Power**: Once solved, differential equations can predict system behavior under various conditions.
- **Analytical Insights**: They often reveal fundamental properties and relationships within the system.
- **Integration with Other Models**: Differential equations can be combined with discrete models to create hybrid systems.

However, using differential equations in MBSE also presents challenges:

- **Complexity**: Many real-world systems lead to complex equations that are difficult to solve analytically.
- **Numerical Methods**: Often, numerical techniques are required to approximate solutions, introducing potential errors and computational overhead.
- **Parameter Identification**: Determining the correct parameters for the equations can be challenging, especially for complex systems.
- **Validation**: Ensuring that the model accurately represents the real system's behavior requires careful attention.

To solve these issues, MBSE users often resort to special software tools that combine differential equation solvers with other simulation and modeling features. These tools enable continuous models based on differential equations to be linked with different parts of system design and analysis.

In summary, differential equations are very important for MBSE because they provide a sound mathematical basis for modeling the behavior of continuous systems. Their ability to represent complex dynamics and interactions makes them

a key resource for engineers and researchers dealing with a variety of systems, whether mechanical, electrical, behavioral, or social.

4.2.2.2 Bond Graphs

Bond graphs provide a robust general way of modeling continuous systems (Karnopp et al. 2012). This graphical modeling language offers a consistent way of showing the flow and conservation of energy in different physical domains, making it especially useful in MBSE for complex systems that span multiple domains.

Key features of bond graphs include:

- **Graphical Representation**: Bond graphs provide a visual representation of energy flow between system components, enhancing intuitive understanding of system dynamics.
- **Multi-domain Modeling**: They allow for seamless integration of mechanical, electrical, hydraulic, and thermal systems within a single model, facilitating the analysis of complex interdisciplinary systems.
- **Object-oriented Approach**: Bond graphs support hierarchical decomposition and reusable component models, aligning well with object-oriented system design principles.
- **Causality Analysis**: They facilitate the automatic determination of computational causality, helping in model formulation and simulation.
- **Systematic Model Development**: The bond graph approach provides a structured method for developing system models, from conceptual design to detailed analysis.

Bond graphs are particularly useful for modeling complex, multi-domain systems such as:

- Aircraft systems (e.g. the Airbus A380 with its extensive electrical systems),
- Biomedical systems (e.g. cardiovascular models),
- Mechatronic systems,
- Robotics and autonomous systems,
- Renewable energy systems (e.g. wind turbines, solar panels).

In the context of MBSE, bond graphs offer several advantages:

- They provide a unified modeling framework across different engineering disciplines, facilitating interdisciplinary collaboration.
- The graphical nature of bond graphs helps in system visualization and communication among stakeholders.
- Their systematic approach to model development supports early-stage design decisions and system optimization.
- Bond graphs can be easily integrated with other modeling techniques, such as block diagrams or state machines, for comprehensive system representation.

However, it's important to note that bond graphs also have some limitations:

- They may have a steeper learning curve compared to some other modeling techniques, especially for those unfamiliar with energy-based modeling.
- For very large systems, bond graph models can become visually complex, potentially reducing their intuitive appeal.
- While excellent for energy-based systems, they may be less suitable for modeling information flow or purely logical systems.

Despite these difficulties, bond graphs still serve as a powerful tool in the MBSE toolkit, especially for systems with diverse energy domains or intricate physical interactions. Their ability to offer a combined, energy-focused view of different systems makes them crucial for simulating and evaluating the continuous behavior of complex engineered systems.

4.2.2.3 Numerical Simulation Techniques

Continuous systems are usually described with differential equations, and digital computers need, therefore, numerical methods for simulation and analysis (Cellier and Greifeneder 2013). These numerical approaches are key for addressing complex continuous behavior models that most of the time do not have analytical solutions. Key aspects of numerical simulation techniques include:

- **Time Discretization**: Continuous time is approximated using small time steps, allowing for the simulation of dynamic behavior over time.
- **Numerical Integration Methods**: Algorithms such as Euler's method, Runge–Kutta methods, or more advanced techniques are used to solve differential equations numerically.
- **Adaptive Time Steps**: Some simulators use variable time steps to balance accuracy and computational efficiency, adjusting step size based on the rate of change in the system.
- **Finite Element Method**: This technique divides the problem domain into smaller, simpler parts called finite elements. It's particularly useful for solving PDEs in complex geometries.
- **Finite Difference Method**: This method approximates derivatives using finite differences. It's often used for solving fluid mechanics and heat transfer problems with stationary boundaries.

Challenges in numerical simulation of continuous behavior models include:

- **Accuracy vs. Computational Efficiency**: There's often a trade-off between simulation accuracy and computational speed, especially for large-scale systems.

- **Stiff Systems**: Some continuous systems exhibit both fast and slow dynamics, requiring specialized numerical methods to handle them efficiently.
- **Numerical Stability**: Ensuring that small errors in the computation do not accumulate and lead to meaningless results.
- **Model Validation**: Verifying that the numerical simulation accurately represents real system behavior, especially for complex systems.

Recent advancements in numerical simulation techniques include:

- **Parallel Computing**: Using multicore processors or distributed systems to speed up simulations of large-scale models (Kim et al. 2018).
- **Machine Learning Integration**: Combining traditional numerical methods with machine learning techniques for more efficient and accurate simulations (Kochkov et al. 2021).
- **Multi-scale Modeling**: Developing techniques to simulate systems across different spatial and temporal scales simultaneously (Kim et al. 2018).

In the context of MBSE, numerical simulation methods are important for studying and confirming continuous behavior models. They enable engineers to predict system operation, improve designs, and verify requirements prior to creating physical prototypes, which helps to reduce development times and costs.

4.2.2.4 Example

A well-known example of using differential equations in an analog system is the mass-spring-damper configuration. This system has three linked parts: a mass, a spring, and a damper, and is frequently used to represent mechanical systems.

The motion of the mass is described by Newton's second law, leading to the following differential equation:

$$m\frac{d^2x}{dt^2} + c\frac{dx}{dt} + kx = F(t) \tag{4.1}$$

Where:

- x is the displacement of the mass.
- $\frac{dx}{dt}$ is the velocity.
- $\frac{d^2x}{dt^2}$ is the acceleration.
- $F(t)$ is an external force applied to the system.

This model enables engineers to study how mechanical systems act when forces are applied, to guess oscillations, and to create systems that exhibit the desired dynamic behavior. It is used in many fields, such as automotive suspension design and vibration analysis, highlighting its importance in engineering.

4.2.3 Discrete Behavior Models

While continuous models are important for systems that change smoothly, discrete models are useful for understanding event-driven behaviors, logical processes, and systems that change state at certain times (Estefan 2007).

The importance of discrete behavior models in MBSE stems from several factors:

- **Event-driven Systems**: Many modern systems, especially in software-intensive domains, exhibit behaviors that are best described by discrete state changes triggered by events (Madni et al. 2019).
- **Concurrency and Parallelism**: Discrete models can effectively represent concurrent and parallel processes, which are common in complex systems (Friedenthal et al. 2014).
- **Nonlinear Behaviors**: They can capture nonlinear and emergent behaviors that may be challenging to represent with continuous models (Estefan 2007).
- **System Analysis**: Discrete models facilitate the analysis of system performance, reliability, and other quality properties through techniques such as Markov chain analysis or Monte Carlo simulation (Madni et al. 2019).
- **V&V**: They provide a formal basis for verifying system behavior against requirements and validating design decisions (Friedenthal et al. 2014).

Discrete behavior models in MBSE can take various forms and variants, including finite-state machines (FSMs), Petri Nets, activity diagrams, discrete event models (DEMs), agent-based models. Each of these modeling methods has its own features and can be adapted to different types of systems or behaviors (Estefan 2007). The choice of technique often depends on the specific properties of the system under study, the modeling objectives, and the preferences of the engineering team.

In the following sections, we will take a closer look at these distinct modeling techniques, discussing their properties, uses, advantages, and drawbacks in the context of MBSE. We will also see how these models can be linked to continuous models to illustrate complex systems that exhibit both discrete and continuous behavior, leading to mixed modeling approaches (Madni et al. 2019).

4.2.3.1 Finite-state Machines

FSMs, or finite-state automata, are key models in computer science and systems engineering. They offer an effective way to show and analyze discrete behaviors in complex systems (Hopcroft et al. 2006). In the field of MBSE, FSMs provide a straightforward way to model system behavior, especially for event-driven and reactive systems (Friedenthal et al. 2014).

Key Characteristics: FSMs are characterized by the following elements:

- **States**: A finite set of conditions or situations in which a system can exist.
- **Transitions**: Rules or conditions that govern the change from one state to another.

- **Events**: Inputs that trigger state transitions.
- **Actions**: Behaviors associated with states or transitions.
- **Initial State**: The starting condition of the system.
- **Final State(s)**: Optional end conditions for the system.

The simplicity and rigor of FSMs make them particularly valuable in MBSE for:

- Modeling system modes and operational states.
- Representing control logic and decision-making processes.
- Analyzing system behavior and verifying design requirements.
- Facilitating communication between stakeholders through visual representations.

Applications in MBSE: FSMs find wide applications across various domains within MBSE:

- **Requirements Engineering**: Capturing and formalizing behavioral requirements.
- **Systems Architecture**: Representing operational modes of complex systems, such as spacecraft or autonomous vehicles.
- **Software Design**: Modeling user interfaces, protocol implementations, and game logic.
- **Hardware Design**: Describing the behavior of digital circuits and control systems.
- **V&V**: Enabling formal analysis and model checking of system properties.

Advantages and Limitations: FSMs offer several advantages in MBSE:

- Simplicity and intuitive representation.
- Formal mathematical foundation enabling rigorous analysis.
- Compatibility with various modeling and simulation tools.
- Scalability through hierarchical and modular approaches.

However, they also have limitations:

- Potential state explosion for complex systems.
- Difficulty in representing concurrent behaviors.
- Limited expressiveness for timing and continuous behaviors.

To address these issues, different extensions and types of FSMs have been proposed, which we will examine in the next subsections. These types improve the analysis capabilities of basic FSMs, making them more suitable for modeling varied and complex systems in MBSE.

Variants: FSMs have several important variants, each with specific characteristics and applications:

- **Deterministic Finite-state Machine**:
  - For each state and input symbol, there is exactly one defined transition to the next state.
  - Provides predictable and unambiguous behavior.
  - Useful for applications requiring precise control and certainty.
- **Nondeterministic Finite-state Machine**:
  - Allows multiple transitions from a state for a given input symbol.
  - Introduces branching and flexibility in the machine's behavior.
  - Advantageous for complex applications involving decision-making and search algorithms.
- **Mealy Machine**:
  - Outputs depend on both the current state and the input.
  - Often used in digital circuit design for systems where output changes with input.
- **Moore Machine**:
  - Outputs depend only on the current state, not on the input.
  - Simpler to design and analyze than Mealy machines.
- **Extended Finite-state Machine**:
  - Incorporates additional capabilities beyond basic FSMs.
  - Transitions are expressed by "if statements" with trigger conditions.
  - Includes data operations and enables more complex behavior modeling.
- **Two-way Deterministic Finite Automaton**:
  - Allows input to be read back and forth with no limit on how many times an input symbol can be read.
  - A generalization of the DFSM, offering more flexibility in input processing.
- **Pushdown Automaton**:
  - Extends FSMs with a stack memory.
  - Mainly used for parsing context-free languages.
- **Timed State Machine**:
  - Incorporates timing constraints into state transitions.
  - Used in real-time systems and protocol modeling.
- **Probabilistic State Machine**:
  - Transitions between states are associated with probabilities.
  - Applied to natural language processing and machine learning.

These variants offer different capabilities and are chosen according to the specific requirements of the system being modeled or the problem to be solved.

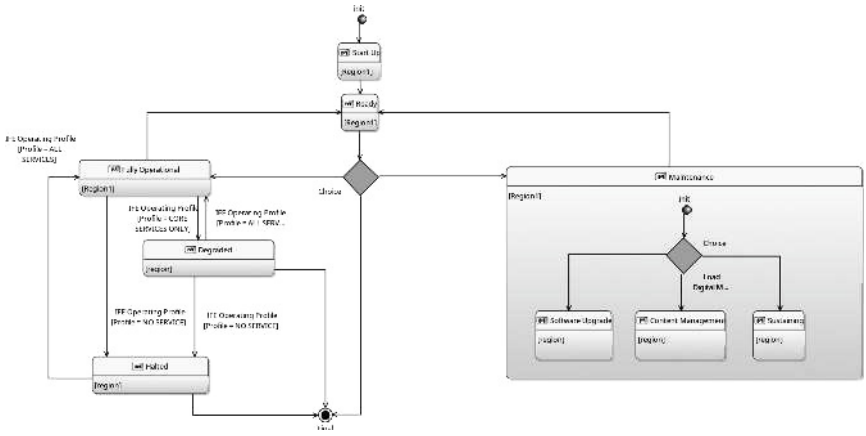


Figure 4.3 Finite-state MFSM of an IFE system.

The choice of FSM variant can significantly impact the design, implementation, and performance of the resulting system.

Example: Figure 4.3 shows the FSM of an IFE system extracted from a model provided by the Capella tool, visually mapping its operational states and transitions. This representation offers key benefits:

- **Clarity in System Behavior**: Depicts the various states (startup, operational, maintenance) and the transitions between them.
- **Simplified Communication**: Provides an intuitive representation of system behavior, helping understanding among diverse stakeholders.
- **Enhanced Analysis and Design**: Facilitates the analysis of system behavior and the design of robust control logic.
- **Support for V&V**: Can be used to verify system requirements and validate behavior.
- **Facilitation of Maintenance and Upgrades**: Helps in understanding state transitions for maintenance and future modifications.

FSMs allow us to model how systems act over time, helping to make designs clearer and more consistent. But, as mentioned before, it is important to pay close attention to meanings to avoid misunderstandings and bad choices. Specifically, FSMs in Unified Modeling Language (UML) have different semantic variation points that need to be clarified to avoid confusion.

4.2.3.2 Petri Net

Petri Nets, introduced by Carl Adam Petri in 1962 (Petri 1962), are a powerful graphical and mathematical modeling tool used to describe and analyze

concurrent, asynchronous, distributed, parallel, nondeterministic, and stochastic systems. They are particularly valuable in MBSE for modeling complex, dynamic systems.

Key Elements and Concepts: The basic structure of a Petri Net consists of four main elements:

- **Places** (represented by circles): Model conditions or states
- **Transitions** (represented by bars or rectangles): Model events or actions
- **Arcs** (represented by arrows): Connect places to transitions and vice versa
- **Tokens** (represented by dots inside places): Indicate the state of the system

The dynamic behavior of a Petri Net is modeled by the flow of tokens through the net when transitions are triggered. This mechanism can be used to represent the dynamics of a complex system, including:

- Concurrency and parallelism
- Synchronization and resource sharing
- Conditional branching and decision-making
- Sequencing and process flow

Advantages in MBSE: Petri Nets offer several key benefits in the context of MBSE:

- **Visual Clarity**: The graphical nature aids in system visualization and communication among stakeholders.
- **Formal Analysis**: The underlying mathematical structure enables rigorous analysis of system properties.
- **Versatility**: Petri Nets can model a wide range of systems, from simple sequential processes to complex distributed systems.
- **Scalability**: Hierarchical extensions allow for the modeling of large-scale systems.
- **Integration**: Petri Nets can be integrated with other modeling techniques to provide a comprehensive system representation.

Analysis Techniques: Petri Nets support various analysis techniques, including:

- **Reachability Analysis**: Determines if a particular system state (marking) can be reached (Murata 2002).
- **Boundedness Analysis**: Assesses limits on the number of tokens in places, crucial for resource management (Peterson 1981).
- **Liveness Analysis**: Checks for the absence of deadlocks in the system (Murata 2002).
- **Invariant Analysis**: Identifies structural properties of the net that remain constant during execution (Peterson 1981).

- **Performance Analysis**: Evaluates system performance metrics, particularly in timed or stochastic variants (Molloy 1982).

Variants and Extensions: To address specific modeling needs, several variants of Petri Nets have been developed:

- **Timed Petri Nets**: Incorporate time delays associated with transitions or places (Ramchandani 1974).
- **Stochastic Petri Nets**: Use probabilistic firing times for transitions, enabling performance and reliability analysis (Molloy 1982).
- **Colored Petri Nets (CPNs)**: Allow tokens to have data values ("colors"), extending their expressivity for more control (Jensen et al. 2007).
- **Hierarchical Petri Nets**: Support the nesting of subnets, facilitating the modeling of complex systems (Fehling 1993).

Challenges and Limitations: Despite their strengths, Petri Nets face some challenges in practical application:

- **State Space Explosion**: For large systems, the number of possible states can grow exponentially, making analysis computationally intensive (van der Aalst and van Hee 2004).
- **Learning Curve**: The formalism may require significant learning effort compared to some other modeling techniques (Girault and Valk 2013).
- **Tool Support**: While specialized tools exist, Petri Nets are not natively supported in many standard MBSE environments (Jensen et al. 2007).
- **Integration**: Combining Petri Net models with other MBSE artifacts can be challenging and may require custom solutions (van der Aalst and van Hee 2004).

In summary, Petri Nets provide a robust technique for modeling complex concurrent systems in MBSE. They combine graphical and formal analysis, which is useful for systems requiring concurrency, resource sharing, and synchronization. However, to use them well, one must pay close attention to the system's requirements and the tools available.

Example: Figure 4.4 illustrates a CPN representing a network protocol, extracted from Jensen et al. (2007). This visual and mathematically rigorous modeling tool offers several key benefits for complex systems, such as network protocols:

- **Visual Clarity**: The CPN provides a clear visual representation of the protocol's structure and behavior, using colors and shapes to distinguish system elements and data/control flow.
- **State and Event Modeling**: Places represent states (e.g. "Send"), while transitions model events (e.g. "Send Packet"), enabling visualization of system states and operations.

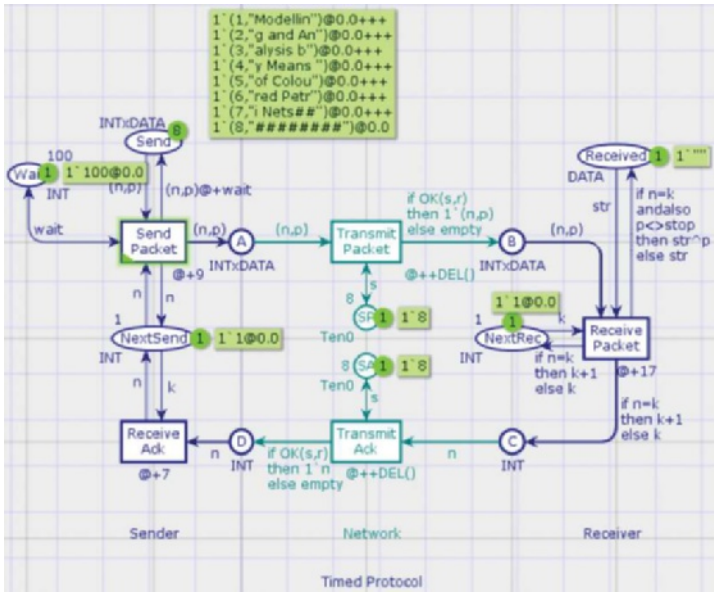


Figure 4.4 CPN representation of a network protocol.

- **Data Flow and Concurrency**: The CPN illustrates data movement (tokens) and concurrent activities, crucial for understanding network behavior.
- **Conditional Behavior and Formal Analysis**: Guard conditions on transitions allow for modeling decision-making, while the formal foundation of CPNs enables rigorous analysis of protocol properties.
- **Simulation and Hierarchical Modeling**: CPNs support simulation for dynamic analysis and hierarchical modeling for representing complex systems at different levels of abstraction.

This CPN representation combines data and control flow, giving a clear view of how the protocol works. It is useful for designing and analyzing network protocols or distributed systems, helping in visualization, simulation, and formal verification. This process enables problems to be detected and performance to be improved before actual implementation.

4.2.3.3 Activity Diagram

Activity diagrams are visual tools used to model the flow of control or objects in a system. As part of UML, they are important for showing the dynamic aspects of a system. Activity diagrams describe both sequential and simultaneous activities

of a process, illustrating the overall flow of control and data within the system or organization (Booch et al. 2005).

Key Concepts: Activity diagrams use a rich set of elements to describe system behavior:

- **Initial and Final Nodes**: Mark the start and end points of the process flow.
- **Action Nodes**: Represent individual steps or activities within the process.
- **Decision Nodes and Merge Nodes**: Illustrate decision points and the merging of alternative flows.
- **Fork and Join Nodes**: Depict the splitting and synchronization of parallel activities.
- **Swimlanes**: Organize actions based on actors or responsible entities.
- **Object Nodes and Object Flows**: Represent data or objects and their movement through the process.

Objectives: Activity diagrams serve a variety of purposes within MBSE:

- **Process Modeling**: Visualizing and documenting complex business processes or workflows (White 2004).
- **Software Design**: Modeling the logic of algorithms, use cases, or complex operations (Rumbaugh 2005).
- **Business Process Engineering**: Analyzing, improving, and redesigning business processes (Aguilar-Savén 2004).
- **Requirements Analysis**: Capturing and representing functional requirements of a system (Booch et al. 2005).
- **Parallel Processing Representation**: Modeling concurrent and multithreaded processes (Fowler 2018).
- **Decision-making Visualization**: Illustrating complex decision paths and their outcomes (White 2004).
- **System Integration**: Modeling interactions between different system components or subsystems (Rumbaugh 2005).

Variants: Activity diagrams have evolved to include several variants, each addressing specific modeling needs:

- **Basic Activity Diagram**: The standard UML activity diagram with actions, control flows, and basic control structures. Used for simple process modeling and basic algorithm design (Fowler 2018).
- **Swimlane Activity Diagram**: Incorporates swimlanes to organize activities by actor, department, or system component. Useful for visualizing responsibilities and interactions in business processes (White 2004).

- **Object Flow Activity Diagram**: Emphasizes the flow of objects or data through the process. Includes object nodes and object flows to show how data is processed and transformed (Booch et al. 2005).
- **Hierarchical Activity Diagram**: Allows nesting of activities within other activities. Useful for modeling complex systems with multiple levels of encapsulation (Rumbaugh 2005).
- **Timed Activity Diagram**: Incorporates timing constraints and durations for activities. Used in real-time systems and performance analysis (Douglass 2004).
- **Stochastic Activity Diagram**: Includes probabilistic branches and activities. Applied in simulation and risk analysis scenarios (Bause and Kritzinger 2013).
- **Interruptible Activity Diagram**: Includes interruptible regions to model activities that can be interrupted by external events. Useful for modeling exception handling and event-driven processes (Booch et al. 2005).

Example: Figure 4.5 depicts an activity diagram for a coffee payment system extracted from a Cameo System Modeler sample model. This visual representation effectively illustrates the sequence of actions and decision points involved in the payment process.

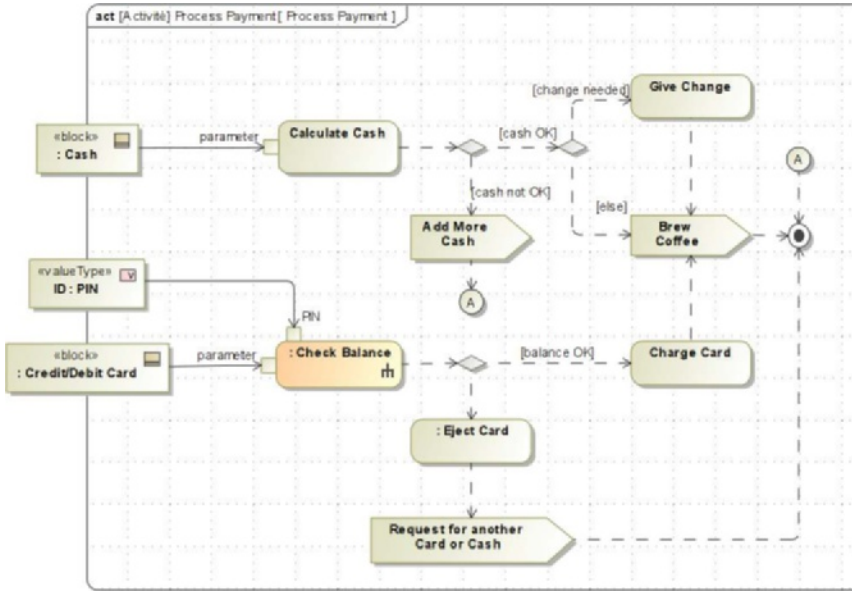


Figure 4.5 Activity diagram for a coffee payment system.

Key benefits of this activity diagram include:

- **Visual Clarity**: Provides a clear overview of the entire registration process, facilitating understanding among stakeholders.
- **Process Flow**: Clearly depicts the sequence of actions, decision points, conditional branches, and parallel activities.
- **Decision Points and Error Handling**: Illustrates decision points and alternative paths for exception handling.
- **Parallel Activities**: Demonstrates concurrent processes, helping in understanding process efficiency.
- **Modularity and Communication**: Breaks down the complex process into manageable steps, enhancing comprehension and communication.
- **Process Optimization and Requirements Clarification**: Facilitates the identification of bottlenecks and validation of system requirements.
- **Documentation**: Serves as valuable documentation for training, maintenance, and future enhancements.

In summary, this activity diagram offers a comprehensive and easily understandable representation of the coffee payment process, contributing to improved system design, implementation, and management.

4.2.3.4 Kahn Process Networks

Kahn Process Networks (KPNs) were created by Gilles Kahn in 1974 (Kahn 1974) and serve as a useful framework for modeling and examining concurrent systems, especially those involving asynchronous communication and dataflow. KPNs allow for a clear and formal representation of how independent processes interact via unidirectional unbounded first-in, first-out (FIFO) channels.

Key Characteristics: KPNs are distinguished by several key characteristics:

- **Processes**: The system is composed of independent processes that execute concurrently. Each process has its own local state and operates autonomously, performing computations and communicating with other processes through channels.
- **Channels**: Unidirectional, unbounded FIFO channels allow communication between processes. Processes send data tokens to channels, and other processes receive tokens from channels, ensuring asynchronous and non-blocking communication.
- **Dataflow**: The execution of processes is driven by the availability of data on their input channels. A process can only execute when it has sufficient input tokens on all its input channels.
- **Determinism**: KPNs exhibit deterministic behavior, meaning that for a given set of inputs, the output of the system is always the same, regardless of the order in which processes execute.

Limitations: While KPNs offer a powerful modeling framework, they also have some limitations:

- **Nondeterminism**: Although KPNs exhibit deterministic behavior at the system level, the order in which processes execute is nondeterministic, which can make debugging and analysis more challenging.
- **Limited Expressiveness**: KPNs are primarily suited for modeling dataflow systems with asynchronous communication. They may not be the most appropriate choice for systems with complex synchronization or timing requirements.
- **Implementation Complexity**: Implementing KPNs can be challenging, especially for large-scale systems, as it requires careful management of communication channels and synchronization mechanisms.

Advantages in MBSE: KPNs offer several advantages in the context of MBSE:

- **Concurrency Modeling**: They provide a natural and intuitive way to model concurrent systems, capturing the asynchronous and parallel nature of many real-world systems.
- **Modularity and Composability**: KPNs promote modular design, as processes can be developed and tested independently. The composability of KPNs allows for the construction of complex systems by connecting simpler processes.
- **Formal Semantics**: The formal semantics of KPNs enable rigorous analysis and verification of system properties, such as absence of deadlock, liveness, and boundedness.
- **Applicability**: KPNs have been successfully applied in various domains, including digital signal processing (DSP), embedded systems, and software design.

KPNs are useful for modeling and studying concurrent systems in MBSE. They can demonstrate asynchronous communication and dataflow, making them suitable for systems with independent processes interacting via message passing. Although KPNs have some drawbacks, their clear rules and modular structure bring significant benefits for system design, analysis, and verification.

4.2.3.5 Synchronous Data Flow

Synchronous Data Flow (SDF) is a specific computational model created by Edward A. Lee and David G. Messerschmitt in the 1980s (Lee and Messerschmitt 1987). It is especially useful for the design and analysis of DSP and multimedia tasks. SDF is a restriction of Kahn process networks where the number of tokens a process produces and consumes is known beforehand.

Key Characteristics: SDF models are characterized by the following features (Lee and Messerschmitt 1987; Bhattacharyya et al. 2012):

- **Graph Structure**: SDF is represented as a directed graph where nodes represent computations (actors), edges represent communication channels, and tokens represent data items.
- **Static Scheduling**: The fixed production and consumption rates allow for compile-time scheduling of actor firings.
- **Determinism**: SDF graphs are inherently deterministic, ensuring consistent outputs for given inputs.
- **Periodic Execution**: SDF graphs can be executed on a fixed, repeating schedule.

Key Concepts: Important concepts of SDF include (Lee and Messerschmitt 1987; Bhattacharyya et al. 2012):

- **Actor Firing**: The execution of an actor, consuming input tokens and producing output tokens.
- **Balance Equations**: Ensure token production equals consumption over a complete graph cycle.
- **Repetition Vector**: Specifies the number of firings for each actor to return the graph to its initial state.
- **Consistency**: An SDF graph is consistent if a nonzero repetition vector exists.
- **Deadlock-free Execution**: The graph can execute indefinitely without memory overflow.

Advantages: SDF offers several benefits for system modeling:

- Predictable behavior and performance
- Efficient static scheduling
- Simplified analysis and optimization
- Natural representation of many DSP algorithms
- Support for formal verification techniques

Limitations: Despite its strengths, SDF has some limitations (Lee and Messerschmitt 1987; Bhattacharyya et al. 2012):

- **Limited Expressiveness**: SDF cannot directly model data-dependent behavior or dynamic rate changes.
- **Asynchronous Processes**: SDF struggles to account for processes with varying token rates.
- **State Explosion**: Complex models can lead to large state spaces, making analysis challenging.

- **Inflexibility**: The deterministic nature of SDF may restrict modeling of systems with uncertain or variable data rates.

Variants: Several variants of SDF have been developed to address its limitations:

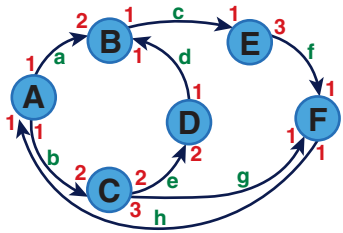
- **Cyclo-static Dataflow**: Allows rates to vary cyclically in a predefined pattern (Bilsen et al. 1996).
- **Boolean Dataflow**: Introduces data-dependent actors for increased expressiveness (Bebelis et al. 2013).
- **Parameterized Dataflow**: Allows dynamic parameter changes between iterations (Bhattacharya and Bhattacharyya 2001).
- **Scenario-aware Dataflow**: Supports switching between predefined scenarios (Stuijk et al. 2011).

Applications: SDF is widely used in various domains, including:

- DSP systems
- Multimedia applications
- Software-defined radio
- Image and video processing
- Parallel computing and hardware synthesis

In MBSE, SDF is a useful tool for modeling and analyzing systems with predictable and repetitive behaviors. It helps in static analysis and optimization, which are important for the design of embedded systems and signal processing applications. Nevertheless, engineers need to recognize its limitations and consider using different variants or models when working with more dynamic systems, or those that rely on sporadic data flows.

Example: Figure 4.6 shows an application with six functions (A–F) that exchange data at fixed production and consumption rates. Answers to questions concerning the schedulability of tasks, their order of execution, and the size of buffers required can be found by solving the Diophantine equation from the



$$\begin{bmatrix} 1 & -2 & 0 & 0 & 0 & 0 \\ 1 & 0 & -2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & -1 & 0 \\ 0 & -1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 2 & -2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & -1 \\ 0 & 0 & 3 & 0 & 0 & -1 \\ -1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} * \vec{X} = \vec{0}, \vec{X} = \begin{pmatrix} 2 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 3 \end{pmatrix}$$

Figure 4.6 Synchronous data flow example.

graph's incidence matrix. If there is a solution, this means the system can be scheduled, and it indicates the execution rate of each function. Solving the transpose of the incidence matrix will give the buffer sizes required between functions.

4.2.3.6 Discrete Events

DEMs or Discrete Event Simulations (DES) represent systems as a sequence of distinct events occurring at specific times. Each event marks a change in the state of the system, with no changes assumed to occur between events (Banks 2005). This approach is particularly useful for modeling complex systems where state changes are not continuous but occur at discrete points (Fishman 2013).

Key Characteristics: Key characteristics of DEMs include:

- **Event-driven**: The system state changes only when an event occurs.
- **Time advancement**: Simulation time jumps from one event to the next (next-event time progression).
- **State variables**: Represent the system state and are updated at each event.
- **Event list**: Maintains a queue of future events to be processed.
- **Statistical accumulators**: Collect data on system performance.

Objectives: DEMs serve several important purposes in system modeling and analysis (Banks 2005; Fishman 2013; Law 2024):

- **System Analysis**: To understand and optimize complex systems' behavior.
- **Performance Evaluation**: To assess system efficiency and identify bottlenecks.
- **Capacity Planning**: To determine resource requirements and system limitations.
- **What-if Analysis**: To test different scenarios and system configurations.
- **Risk Assessment**: To evaluate potential outcomes and their probabilities.
- **Process Improvement**: To identify areas for enhancing system performance.
- **Decision Support**: To help in strategic and operational decision-making.

Variants: Several variants of DEMs have been developed to address specific modeling needs:

- **Three-phase Approach**: Divides the simulation into time advance, B-events (bound to occur), and C-events (conditional) (Pidd 1998).
- **Process-interaction Approach**: Models the system as a set of interacting processes (Law 2024).
- **Activity-based Approach**: Focuses on modeling activities and their durations (Pidd 1998).

- **Event-scheduling Approach**: Emphasizes the scheduling and handling of events (Fishman 2013).
- **State-based Approach**: Focuses on defining system states and transitions between them (Banks 2005).
- **Combined Discrete-continuous Simulation**: Integrates discrete events with continuous state changes (Cellier and Greifeneder 2013).
- **Distributed Discrete Event Simulation**: Allows simulation to be executed across multiple processors or computers (Fujimoto 2001).
- **Agent-based Discrete Event Simulation**: Combines discrete event modeling with agent-based concepts (Macal and North 2005).

These types serve different purposes and are selected according to the needs of the system being modeled or the issues to be analyzed. They provide a flexible set of tools for representing and studying complex systems in many fields, such as manufacturing, logistics, healthcare, and computer networks.

4.2.4 Hybrid Models

Hybrid models offer a powerful technique in MBSE by combining continuous and discrete modeling methods to represent systems with both continuous and event-driven behaviors. These models are important for complex systems that cannot be fully represented with just continuous or discrete techniques, as they typically involve interactions between physical processes (often continuous) and control logic or decision-making (often discrete). The ability to incorporate both elements within a single framework makes hybrid models essential for the representation and analysis of a variety of real-world systems.

4.2.4.1 Key Characteristics

Hybrid models are distinguished by several key characteristics that enable them to effectively represent complex system behaviors:

- **Integration of Continuous and Discrete Dynamics**: Hybrid models seamlessly combine differential equations (for continuous behavior) with state machines or other discrete event formalisms, allowing for the representation of both smooth, time-varying dynamics and state changes triggered by events (Lygeros et al. 2008).
- **Mode Switching**: They capture systems that switch between different modes of operation, each characterized by distinct continuous dynamics. These mode switches are often triggered by discrete events or conditions, enabling the representation of systems with varying behaviors depending on their operational context (Connor and Linton 2007).

- **Time Scales**: They can represent phenomena occurring at different time scales, from fast continuous dynamics to slower discrete state changes. This capability is crucial for modeling systems where different components or subsystems operate at vastly different speeds or frequencies, such as in cyber–physical systems or embedded control systems (Lygeros et al. 2008).

4.2.4.2 Overcoming Challenges in Verification and Simulation

The complexity of hybrid models poses several important challenges in their verification, validation, and simulation:

- **Numerical Instability**: Interfacing continuous and discrete simulations can lead to numerical instability, particularly at mode-switching points. This instability can result in inaccurate or unreliable simulation results, requiring careful handling and specialized numerical methods.
- **Validation Complexity**: Validating hybrid models is particularly challenging due to the interaction between continuous and discrete components. Traditional validation techniques may not be sufficient, and new approaches are needed to ensure the accuracy and fidelity of hybrid models.
- **Lack of Standardized Conceptual Models**: There is no widely accepted standard for representing hybrid conceptual models, making it difficult to communicate and validate model designs across different domains and tools.
- **Computational Intensity**: Simulating hybrid models often requires significant computational resources, especially when dealing with systems that have multiple time scales or frequent mode switches.
- **Zeno Behavior**: Hybrid models can theoretically exhibit Zeno behavior, where an infinite number of discrete transitions occur in a finite time interval, making simulation practically impossible. Addressing Zeno behavior often requires careful model design and the use of appropriate simulation techniques.
- **Tool Limitations**: Many simulation tools are designed for either continuous or discrete simulation, but not both. This lack of integrated tools can make it difficult to implement and simulate complex hybrid models efficiently.
- **Formal Verification**: Proving properties of hybrid systems is often undecidable, requiring approximation techniques that may not provide guaranteed results. This poses challenges in ensuring the correctness and safety of hybrid systems, especially in safety-critical applications.
- **Maintaining Consistency**: Ensuring consistency between the continuous and discrete parts of the model, especially during mode transitions, is a significant challenge. Inconsistencies can lead to errors and unexpected behavior in the simulation.

4.2.4.3 Addressing the Challenges

Researchers and practitioners are actively developing various approaches to address these challenges:

- Advanced numerical methods for stable integration of continuous and discrete dynamics.
- Development of specialized validation techniques for hybrid models.
- Creation of standardized frameworks for representing hybrid conceptual models.
- Usage of high-performance computing and parallel processing techniques.
- Development of integrated simulation tools that support both continuous and discrete paradigms.
- Application of formal methods and approximation techniques for the verification of hybrid systems.

4.2.4.4 The Power of Hybrid Models

Despite these difficulties, hybrid models remain useful for representing complex systems in MBSE, especially in fields such as cyber–physical systems or control systems. Research in this field is still striving to make hybrid modeling and simulation methods more reliable, effective, and practical.

By connecting continuous and discrete types of modeling, hybrid models provide a broader and more real-life picture of the behavior of complex systems. They help engineers to understand the complex links between physical processes and control systems, resulting in better predictions, smarter design choices, and improved system performance. As MBSE keeps evolving, hybrid models are likely to play a more important role in the design and analysis of future engineered systems.

4.2.4.5 Example

Consider a room equipped with a thermostat-controlled heating system. The room temperature changes continuously over time, while the thermostat operates in a discrete manner, switching the heater on or off according to temperature thresholds.

Figure 4.7 illustrates a very simple hybrid system architecture of such a system. In this example, we can see two components interconnected by a common interface:

- **Sensor**: Models the continuous temperature dynamics
- **Heater Control**: Manages the discrete switching between heating states

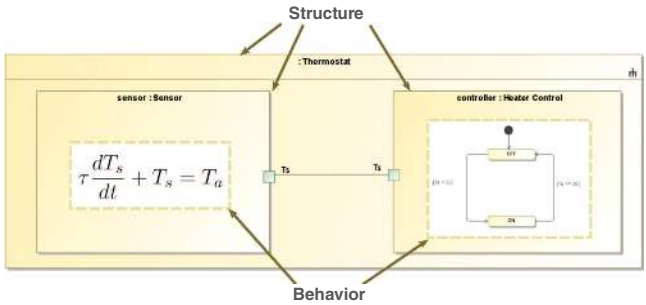


Figure 4.7 Hybrid model example.

This figure aptly illustrates the hybrid nature of the system:

- **Continuous Component**: The sensor (left side) represents the continuous aspect of the system. It's described by a differential equation that models the temperature change over time, where:
  - T_s**: Temperature sensed by the sensor (output)**
  - T_a**: Actual ambient temperature (input)**
  - τ **: Time constant of the sensor (a parameter characterizing the sensor's response time)**
- **Discrete Component**: The controller (right side) represents the discrete aspect, showing a state machine with ON and OFF states.

Despite its simplicity, this model can be used to illustrate various simulation problems we have to face regarding the simulation of such systems:

- **Different Time Scales**: The continuous system evolves in continuous time, while the discrete system changes at specific events. Synchronizing these different time scales in simulation can be challenging.
- **State Changes**: Abrupt changes in the system behavior occur when the thermostat switches states. These discontinuities can cause numerical issues in simulation.
- **Stiff Systems**: The temperature dynamics might be much slower than the switching speed of the thermostat, leading to a stiff system that's difficult to simulate efficiently.
- **Event Detection**: Accurately detecting when the temperature crosses the threshold values is crucial, but can be computationally expensive.
- **Numerical Methods**: Choosing appropriate numerical methods that can handle both the continuous dynamics and discrete events is challenging.
- **Accuracy vs. Efficiency**: Balancing simulation accuracy with computational efficiency, especially for long-term behavior analysis, can be difficult.

- **Tool Integration**: Simulating this system might require integrating different tools specialized for continuous and discrete simulations, which can introduce compatibility issues.

4.3 Agent-based Modeling

Agent-based modeling (ABM) is based on concepts from different fields, such as game theory, complex systems, emergence, computational sociology, multi-agent systems, and evolutionary programming. This approach provides a fresh vision of system dynamics that complements standard modeling methods (Macal and North 2005).

4.3.1 Key Characteristics

ABM is characterized by the following features that distinguish it from other modeling paradigms:

- **Autonomous Agents**: The system is composed of individual agents that encapsulate their own properties, behaviors, and decision-making capabilities. These agents act independently based on their internal states and interactions with their environment (Bonabeau 2002).
- **Interactions**: Agents engage in interactions with each other and their environment, exchanging information, resources, or influencing each other's behaviors. These interactions can be local or global, direct or indirect, and can lead to complex emergent phenomena (Macal and North 2005).
- **Emergence**: ABM captures the emergence of complex system-level behaviors from the interactions of relatively simple agents following basic rules. This bottom-up approach allows for the exploration of unexpected and nonlinear system dynamics (Gilbert 2019).
- **Heterogeneity**: ABM can represent the inherent heterogeneity and individual differences among agents, allowing for the modeling of diverse populations and their impact on system behavior (Bonabeau 2002).
- **Spatial and Temporal Dynamics**: ABM can incorporate spatial representations and temporal evolution, capturing how agents move, interact, and change over time within their environment (Gilbert 2019).

4.3.2 The Dual Nature of ABM: Structural and Behavioral

ABM serves as both a structural and behavioral model, bridging the gap between these two modeling paradigms. This dual nature is explained in the three following paragraphs.

4.3.2.1 Structural Representation

As a structural model, ABM defines the composition and organization of a system:

- **Agent Definition**: ABM specifies the types of agents present in the system, their attributes (e.g. age, location, resources), and their relationships with other agents. This forms the structural backbone of the model, defining the entities that interact and influence each other (Macal and North 2005).
- **Environment Representation**: It defines the structure of the environment in which agents operate, including spatial layouts, resource distributions, and any constraints or limitations that may affect agent behavior (Gilbert 2019).
- **Interaction Networks**: ABM can represent the structural connections between agents, forming social networks, communication channels, or other interaction topologies. These networks can influence the flow of information and resources within the system (Newman 2003).
- **Hierarchies and Groupings**: ABM can model hierarchical structures or groupings of agents, reflecting organizational or social structures. This allows for the representation of complex systems with multiple levels of organization and decision-making (Epstein 2012).

4.3.2.2 Behavioral Dynamics

At the same time, ABM captures the dynamic behavior of the system:

- **Agent Behavior Rules**: It defines how agents make decisions and act based on their current state, their interactions with other agents, and their perception of the environment. These rules can be simple or complex, deterministic or stochastic, and can incorporate learning and adaptation mechanisms (Bonabeau 2002).
- **Interaction Dynamics**: ABM simulates how agents interact with each other and their environment over time. These interactions can lead to changes in agent states, resource distributions, and the overall system configuration (Macal and North 2005).
- **Emergent Behaviors**: ABM captures system-level behaviors that emerge from the collective actions and interactions of individual agents. These emergent behaviors may not be predictable from the individual agent rules alone, highlighting the power of ABM in understanding complex systems (Gilbert 2019).
- **Adaptive Processes**: ABM can model how agents learn and adapt their behaviors based on experiences, feedback, or interactions with other agents. This allows for the representation of systems that evolve and change over time in response to their environment (Epstein 2012).

- **Time Evolution**: It represents how the system evolves over time, often incorporating both discrete events (e.g. decisions, transactions) and continuous processes (e.g. resource consumption, movement) (Borshchev 2015).

4.3.2.3 Integration of Structural and Behavioral Aspects

The power of ABM lies in its ability to integrate structural and behavioral elements, creating a holistic representation of complex systems:

- **Structure-behavior Feedback**: The structural composition of the system influences agent behaviors, while the collective behaviors of agents can, in turn, modify the system structure. This feedback loop allows for the modeling of systems that co-evolve and adapt over time (Gilbert 2019).
- **Multi-scale Modeling**: ABM can represent how micro-level interactions between agents lead to macro-level structural changes and vice versa. This multi-scale perspective is crucial for understanding complex systems where phenomena at different levels of organization are interconnected (Olsen and Siegelmann 2013).
- **Dynamic Network Evolution**: ABM can model how interaction networks between agents evolve over time based on their behaviors and decisions. This allows for the representation of systems where relationships and communication patterns change dynamically (Newman 2003).
- **Emergence of Structure**: ABM can demonstrate how structural patterns and organizations emerge from the interactions of agents following simple behavioral rules. This bridges the gap between structural and behavioral modeling, showing how structure can arise from dynamic processes (Epstein 2012).

4.3.3 Applications in MBSE

In the context of MBSE, ABM offers unique advantages:

- **Complex System Design**: ABM helps in understanding how system-level behaviors emerge from component-level interactions, facilitating the design of complex engineered systems where traditional modeling approaches may fall short (Bonabeau 2002).
- **Requirements Elicitation**: ABM can be used to explore and validate system requirements by simulating various scenarios and stakeholder interactions, ensuring that the system design meets the needs of all involved parties (Macal and North 2005).
- **Architecture Evaluation**: ABM allows for the evaluation of different system architectures by simulating their performance under various conditions, helping engineers to identify the most suitable design choices (Gilbert 2019).

- **Socio-technical Systems**: ABM is particularly useful for modeling systems that involve human factors and social dynamics alongside technical components, capturing the complex interplay between technology and human behavior (Epstein 2012).
- **V&V**: ABM can support the V&V process by allowing engineers to test system behavior under a wide range of conditions, ensuring that the system meets its requirements and performs as expected (Macal and North 2005).

By serving as both a structural and behavioral model, ABM provides a comprehensive approach to the representation of complex systems in MBSE. It enables engineers to capture the intricate relationships between system structure and behavior, leading to more accurate and insightful models of complex engineered systems.

4.3.4 Example

Figure 4.8 shows a traffic simulation using agents, taken from the NetLogo model library. This example shows how ABM can be used to study complex systems. The simulation shows the flow of traffic, where each car is represented by an agent. Users can adjust parameters such as the number of cars and acceleration, while monitoring real-time emergent properties such as car speeds and driver patience.

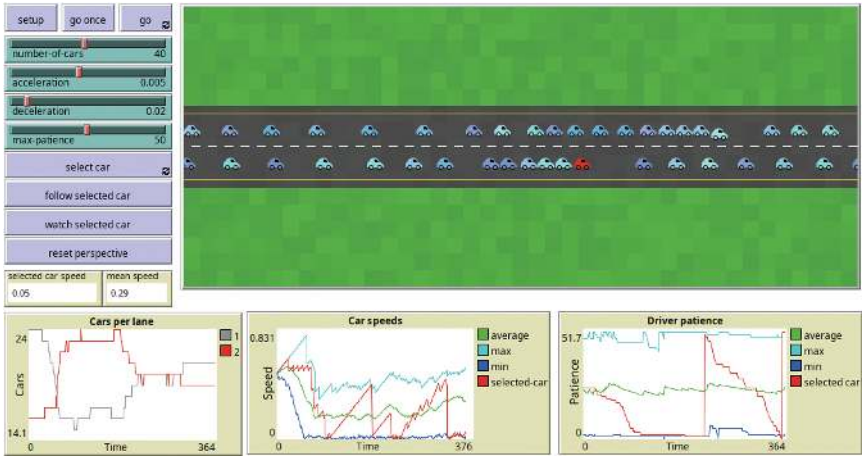


Figure 4.8 Agent-based model simulation of car traffic.

Key benefits of this ABM simulation include:

- **Dynamic Visualization**: Real-time visual representation of traffic flow and emergent patterns.
- **Individual Agent Behavior**: Models diverse driver behaviors and vehicle characteristics.
- **Parameter Adjustment and Scenario Testing**: Enables experimentation with various scenarios and traffic management strategies.
- **Multi-lane Dynamics**: Studies lane-changing behaviors and their impact on traffic flow.
- **Data Collection and Analysis**: Provides real-time data for quantitative analysis.
- **Emergent Phenomena**: Reveals traffic jams and other complex behaviors arising from simple interactions.
- **Interdisciplinary Insights**: Incorporates insights from various fields for a holistic understanding.
- **Scalability and Education**: Can be scaled to larger networks and used for effective communication with stakeholders.

This ABM simulation is a powerful tool for understanding, analyzing, and predicting traffic behavior, informing urban planning, traffic management, and policy decisions.

Exercise: Choosing a behavioral model

Consider the following scenarios and select the most suitable behavioral modeling approach for each case. Justify your choice by explaining how the chosen model aligns with the system's characteristics and requirements.

1. A vending machine that dispenses snacks and drinks based on user input and payment. The machine should handle different states (e.g. idle, accepting payment, dispensing items) and transitions between them.
2. A traffic light control system at a busy intersection. The system should coordinate the timing of red, yellow, and green lights for multiple directions based on predefined time intervals and traffic flow patterns.
3. A manufacturing plant with multiple machines, each performing specific tasks on a product. The machines operate concurrently and pass the product from one station to another until the final product is assembled.

(Continued)

(Continued)

4 A smart thermostat that adjusts the temperature of a room based on user preferences, time of day, and occupancy. The thermostat should learn from user behavior and adapt its control strategy over time.
5 A social media platform where users can create profiles, connect with friends, post updates, and interact with each other's content. The platform should model the dynamic interactions and emergent behaviors of the user community.

For each scenario, consider the following behavioral modeling approaches discussed in the chapter:

- FSMs
- Petri Nets
- Activity diagrams
- DEMs
- Agent-based models
- Continuous behavior models (e.g. differential equations)
- Hybrid models

Reflect on the strengths and limitations of each approach and select the one that best captures the essential behavioral aspects of the system in question. Provide a brief explanation of your reasoning.

4.4 Beyond Structure and Behavior

In addition to structural and behavioral models, MBSE encompasses a crucial dimension: models that capture and analyze the extra-functional properties of systems (Sentilles et al. 2009). These properties, which relate to safety, performance, reliability, security, and maintainability, are critical to ensuring that a system not only meets its main objectives, but also operates well and safely in its intended environment. Extra-functional models help to capture, assess, and improve these properties during the system's lifecycle, contributing to the overall design and creation of reliable and trustworthy systems.

The role of extra-functional modeling is very important. In our complex and connected world, systems must not only carry out their designated tasks but also do so dependably, safely, and securely. Ignoring these extra-functional properties can result in serious issues, from financial losses to human casualties. Thus, it is essential for systems engineers to pay close attention to these features and integrate them into the MBSE process right from the start.

4.4.1 Types of Extra-functional Models

The field of extra-functional modeling is diverse, with various models that address different aspects of system behavior:

- **Safety Models**: These models focus on identifying, analyzing, and mitigating potential hazards and risks associated with a system. Prominent examples include Fault Tree Analysis (FTA), Failure Mode and Effects Analysis (FMEA), and Hazard and Operability Analysis (HAZOP) (Xing and Amari 2008; Stamatis 2003; Kletz 2018). Another important approach in this domain is model-based safety analysis (MBSA), which integrates safety analysis into the MBSE process by leveraging system models to identify and assess safety risks (Joshi et al. 2006). MBSA enables a more proactive and iterative approach to safety analysis, allowing engineers to identify and mitigate potential hazards early in the development process.
- **Performance Models**: These models evaluate and predict system performance characteristics, such as throughput, response time, and resource usage. Queuing models, Petri Nets, and discrete event simulation are commonly used for performance analysis (Balsamo et al. 2003; Murata 2002; Banks 2005). Performance modeling is crucial for ensuring that a system can handle the expected workload and deliver the desired level of quality of service. It helps in identifying potential bottlenecks, optimizing resource allocation, and making informed decisions about system design and configuration.
- **Reliability Models**: These models assess the ability of a system to perform its intended function under specified conditions for a given period. Reliability Block Diagrams, Markov models, and Accelerated Life Testing are key techniques in reliability modeling (Rausand and Hoyland 2003; Trivedi 2016; Nelson 2009). Reliability modeling is essential for predicting and improving the dependability of a system, minimizing downtime, and ensuring that the system meets the required availability and maintainability goals.
- **Security Models**: These models focus on identifying and mitigating potential security threats and vulnerabilities in a system. Threat modeling, attack trees, and methodologies like Secure Tropos are essential in ensuring system security (Myagmar et al. 2005; Schneier 1999; Mouratidis and Giorgini 2007). With the increasing prevalence of cyber threats, security modeling has become an indispensable part of MBSE. It helps in identifying potential attack vectors, assessing the impact of security breaches, and designing effective countermeasures to protect the system and its users.
- **Maintainability Models**: These models assess the ease with which a system can be maintained, repaired, or modified over its lifecycle. Maintenance Task Analysis and Reliability-Centered Maintenance are prominent approaches in this domain (Dhillon 2006; Moubray 2001). Maintainability modeling is

crucial for ensuring that a system can be efficiently and cost-effectively maintained throughout its operational life. It helps in optimizing maintenance strategies, reducing downtime, and minimizing the total cost of ownership.

These extra-functional models provide systems engineers with a comprehensive set of tools for analyzing and optimizing various aspects of system behavior. By leveraging these models, engineers can gain valuable insights into the system's safety, performance, reliability, security, and maintainability, enabling them to make informed design decisions and trade-offs.

4.4.2 Integration within MBSE

The integration of extra-functional models into the MBSE process is essential for achieving systems that meet their requirements and operate reliably and safely (Madni et al. 2022). This integration involves:

- **Requirement Definition**: Clearly defining extra-functional requirements and constraints within the system model. This involves capturing the desired levels of safety, performance, reliability, security, and maintainability and expressing them in a precise and measurable manner. By incorporating these requirements into the system model from the outset, engineers can ensure that they are given due consideration throughout the development process.
- **Model Linking**: Establishing explicit links between extra-functional models and relevant system elements, such as components, interfaces, and functions. This enables traceability and consistency between the different aspects of the system model. For example, linking a fault tree model to the corresponding system components allows engineers to easily identify the impact of component failures on system safety.
- **Design Evaluation and Optimization**: Using extra-functional models to assess and compare design alternatives, enabling informed decisions that optimize system performance, safety, and other critical properties. By integrating extra-functional models into the design process, engineers can evaluate the impact of different design choices on the system's extra-functional properties and make trade-offs based on the project's priorities and constraints.
- **Lifecycle Management**: Continuously monitoring and updating extra-functional models throughout the system's lifecycle to ensure their relevance and accuracy as the system evolves. This involves regularly reviewing and refining the models based on the latest system data, operational feedback, and changing requirements. By maintaining up-to-date extra-functional models, engineers can proactively identify and address potential issues, ensuring that the system continues to meet its extra-functional objectives throughout its lifecycle.

Additionally, including extra-functional models in MBSE encourages a broad and collaborative approach to system development. It brings together experts from different fields, such as safety engineers, performance analysts, and security specialists, to work together and share their knowledge within the system model. This contributes to build a shared understanding of extra-functional requirements and promotes better communication and decision-making within the project team.

In summary, extra-functional modeling is an important part of MBSE that extends beyond traditional focuses on structure and behavior. By using extra-functional models within their MBSE processes, systems engineers can create systems that are not only functionally correct but also safe, reliable, secure, and easy to maintain. This comprehensive approach to development is crucial for addressing the complex issues of modern systems and ensuring effective deployment and operation in the real world.

4.4.3 Example

MBSA is a robust method that integrates safety analysis into the wider MBSE process. It helps in evaluating system safety in a detailed manner, using the system models created during design. Let's look at an example of MBSA with an FTA for a car headlight system.

Figure 4.9, taken from Lee et al. (2021), shows an FTA for the failure mode "Switch to low beam failed" in a dual-function DC motor system for car headlights.

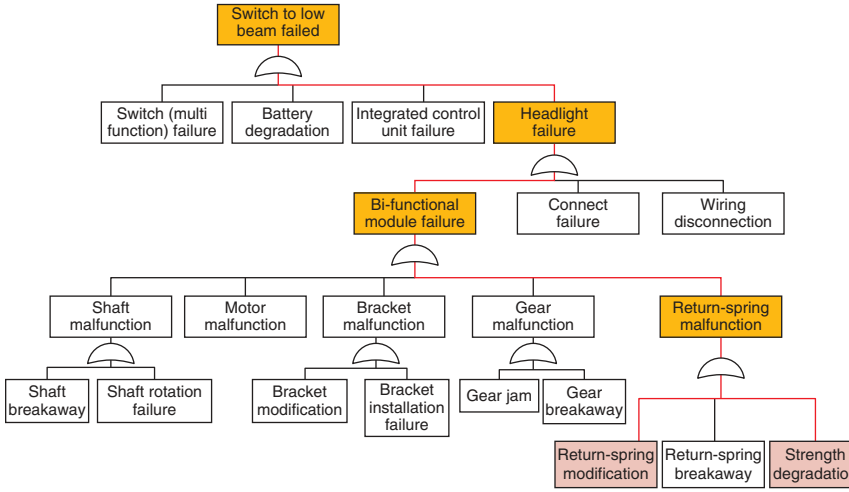


Figure 4.9 FTA of an automobile headlight system.

This fault tree shows how MBSA can be used to study possible failure modes and their causes in a complex system. Key observations from this analysis include:

- **Top-level Event**: The analysis focuses on the specific failure mode of "Switch to low beam failed," which is a critical safety concern for automobile operation.
- **Hierarchical Structure**: The fault tree breaks down the top-level event into more specific failure modes, allowing for a systematic analysis of potential causes.
- **Multiple Failure Paths**: The diagram shows that the top-level failure can result from various component failures or malfunctions, including switch failure, battery issues, control unit problems, or headlight failure.
- **Component-level Analysis**: The fault tree extends to detailed component-level failures, such as bi-functional module failures and their potential causes (e.g. shaft malfunction, motor malfunction).
- **Logical Relationships**: The use of OR gates in the diagram indicates that any of the lower-level events can lead to the higher-level failure, highlighting the system's vulnerability to multiple failure modes.
- **Critical Components**: The analysis identifies critical components, such as the return spring, whose malfunction can directly contribute to the system failure.

This MBSA approach offers several benefits in the context of system safety:

- **Systematic Identification of Failure Modes**: It provides a structured method to identify and analyze potential failure modes at various levels of the system.
- **Prioritization of Risks**: By visualizing the relationships between different failure modes, engineers can prioritize which components or subsystems require additional safety measures or redundancies.
- **Integration with System Models**: The fault tree can be directly linked to the system architecture models, ensuring consistency between safety analysis and system design.
- **Traceability**: MBSA facilitates traceability between system requirements, design decisions, and safety analysis, supporting a more integrated approach to system design and development.
- **Iterative Analysis**: As the system design evolves, the fault tree can be easily updated, allowing for continuous safety assessment throughout the development lifecycle.
- **Communication Tool**: The visual nature of the fault tree makes it an effective tool for communicating safety concerns and mitigation strategies to various stakeholders.

By using MBSA techniques such as FTA in their MBSE process, system engineers can find, investigate, and reduce potential safety issues in complex systems. This combined technique makes safety a central element of system design, resulting in more robust and reliable systems.

4.5 Modeling the System Environment

In MBSE, it is important to capture the system's environment. The system environment includes all external factors and conditions that can affect the system's design, development, and operation (Estefan 2007; Friedenthal et al. 2014). Understanding and accurately modeling the system environment is paramount for several reasons (Kossiakoff et al. 2020):

- **Context for Requirements**: It provides critical context for system requirements and constraints, ensuring the designed system is fit for its purpose and operates effectively in its intended configuration.
- **Risk Mitigation**: By anticipating potential risks and challenges arising from environmental interactions, engineers can design more robust and reliable systems.
- **System Adaptability**: The environment is dynamic and evolves over time due to technological advancements, changing user needs, and shifting regulations. A flexible and adaptive modeling approach is crucial to accommodate these changes throughout the system lifecycle.

4.5.1 Modeling Techniques

MBSE practitioners use various techniques to model the system environment:

- **Physical Environment Modeling**: Represents tangible external conditions, such as temperature, humidity, and mechanical stresses.
- **Operational Environment Modeling**: Captures the contexts and conditions of system use, including user behaviors, operational scenarios, and demand patterns.
- **Regulatory Environment Modeling**: Represents legal and standards-based factors that constrain or guide system development and operation.
- **Technological Environment Modeling**: Models the broader technology ecosystem, encompassing interoperability standards, technology lifecycles, and cybersecurity.

Common tools and techniques for environmental modeling include:

- **Parametric Models**: Represent quantifiable factors and their relationships to system performance.

- **Behavioral Models**: Capture dynamic interactions between the system and its environment.
- **Scenario-based Modeling**: Explores system behavior under diverse conditions.
- **Stochastic Modeling**: Accounts for uncertainties and variations in environmental factors.

A major problem in environmental modeling is finding a good balance between how accurate the model is and how easy it is to use (Friedenthal et al. 2014). Detailed models can provide useful insights, but they can also hinder understanding and analysis. Engineers need, therefore, to think carefully about the appropriate level of abstraction when seeking to demonstrate any property of the system under study.

4.5.2 Environmental Considerations and System Complexity

Including environmental factors in system modeling significantly increases complexity due to:

- **Breadth and Diversity of Factors**: Accounting for numerous physical, operational, and regulatory variables expands the model's scope and increases the number of relationships to analyze.
- **Dynamic and Stochastic Behavior**: Many environmental factors change unpredictably over time, making them more difficult to model than deterministic system components.
- **Holistic Modeling Approach**: Engineers must consider the system's internal operations and its interactions with the environment, including potential emergent behaviors. This often necessitates sophisticated modeling techniques, such as ABM.
- **Stakeholder Integration**: Environmental modeling may involve additional stakeholders (e.g. environmental experts, regulators, and community representatives) who bring new perspectives and requirements to the system model.

Despite its challenges, modeling the environment leads to:

- Increased robustness and adaptability of the system
- Improved risk management and mitigation
- More accurate performance predictions
- Better alignment with regulations and stakeholder needs

4.5.3 The Rover Case

The importance of environmental modeling is vividly illustrated by the Martian rover example (Figure 2.1). The rover's success hinges on its ability to navigate and operate in the harsh and unpredictable Martian environment, including among others:

- **Temperature Extremes**: Martian temperatures fluctuate drastically between day and night, requiring the rover's thermal management system to be robust and adaptable. Modeling these temperature extremes is crucial for ensuring that critical components operate within their specified temperature ranges and the rover can survive the Martian night.
- **Dust Storms**: Mars is known for its frequent and intense dust storms that can reduce visibility and potentially damage the rover's solar panels, sensors, and other sensitive equipment. Modeling these dust storms is essential for designing dust mitigation strategies and ensuring the rover's continued operation during such events.
- **Terrain Navigation**: The Martian terrain is varied and challenging, with rocks, craters, and slopes that the rover must navigate. Modeling the terrain is essential for developing robust path planning algorithms and ensuring the rover's safe and efficient movement.
- **Limited Bandwidth**: Communication with Earth is constrained by limited bandwidth and signal delays, requiring efficient data transmission protocols and autonomous decision-making capabilities. Modeling these communication constraints is crucial for designing a system that can effectively collect and transmit data despite these limitations.

These examples highlight the interconnected nature of environmental factors and their profound influence on system design. MBSE, by providing a framework for modeling these factors and their interactions, empowers engineers to develop systems that are not only functionally sound but also well-adapted to their intended operational environments.

4.5.4 Summary

Modeling environmental factors is an important step in creating robust, adaptable, and context-aware systems. Problems arise from managing increased complexity and finding a balance between model details and modeling objectives. Continuous improvements in MBSE approaches, tools, and practices will be key to tackling these issues. By adopting a holistic approach that includes both the system and its environment, engineers can develop systems that are not only operationally effective but also fit for their intended environments.

Exercise: Environment-aware modeling for an ATM system

You are the systems engineer responsible for designing a new generation of ATM systems. Your organization wants these ATMs to be more robust, secure, and adaptable to various deployment locations and user needs. You decide to leverage MBSE and environment-aware modeling to achieve these goals.

Part 1: Identifying Environmental Factors

Consider the different environmental factors that could influence the design, deployment, and operation of an ATM system. For each of the four environment types (physical, operational, regulatory, and technological), list at least three specific factors that are relevant to the ATM example.

Example:

Physical Environment: Extreme temperatures, humidity, or vandalism attempts.

Part 2: Impact on System Behavior

For each environmental factor you identified in Part 1, describe its potential impact on the ATM system's behavior, performance, reliability, or security.

Example:

Extreme Temperatures: High temperatures could lead to overheating of internal components, potentially causing malfunctions or system failures. Low temperatures could affect the display screen or make the keypad difficult to use.

Part 3: Modeling Techniques

Select three environmental factors from your list in Part 1 and suggest specific modeling techniques that could be used to represent these factors and their interactions with the ATM system within an MBSE framework.

Example:

Vandalism Attempts: DES could be used to model the frequency and impact of vandalism attempts, considering factors such as location, time of day, and types of attacks. The model could then be used to evaluate the effectiveness of different security measures (e.g. reinforced casing, alarm systems, surveillance cameras).

Part 4: Architectural Decisions

Based on your analysis in Parts 1–3, suggest at least three specific architectural decisions or design features that could be implemented in the new ATM system to mitigate the identified environmental risks and enhance its robustness, security, and adaptability.

Example:

Implement a robust thermal management system that can regulate internal temperatures effectively across a wide range of thermal conditions.

Part 5: Reflection
Write a brief reflection on how considering the system environment in MBSE changes your perspective on the ATM system design. Consider the following points:

- What new insights did you gain by thinking about environmental factors?
- How did the process of environmental modeling influence your design choices?
- What challenges did you encounter in incorporating environmental considerations, and how did you address them?

4.6 Conclusion

The different types of models discussed in this chapter form the basis of MBSE. We examined structural, behavioral, and extra-functional models to understand how they help us to grasp complex systems. From simple structural models to dynamic behavioral simulations and the insights from extra-functional analysis, we have seen how models reveal the details of system design and development.

In this chapter, we have looked at various ways of formalizing key aspects of a system: its structure, behavior, and extra-functionality. These techniques are not independent; they need to work together to address different issues. They generally follow an agile approach that helps us explore the design space. The choice of one or more models to help with system design and analysis is based on several factors examined in the second chapter. The difficulty lies in the way these models are defined and combined, which raises questions about composability and consistency.

The growth of MBSE has led to a wide range of modeling techniques, each with its own benefits and drawbacks. It is essential to select the right models based on the characteristics of the system and the objectives of the analysis. The complexity of modern systems often requires using several modeling techniques, leading to both theoretical and practical challenges that call for creative solutions.

To conclude this chapter, it is important to note that MBSE is an evolving and growing field, driven by new technologies and the increasing complexity of engineered systems. The future of MBSE is bright, with new modeling techniques and tools that will enhance our ability to represent and analyze complex systems.

The insights and lessons learned from our examination of theoretical models help systems engineers make informed decisions when selecting and using the right modeling techniques for their projects. By harnessing the power of models, we can manage the complexities of modern systems, improve communication and teamwork, and ultimately, achieve successful system design and development.

However, creating models is just one part of the MBSE process. To make sure these models truly reflect the intended system and satisfy stakeholder needs, we need to rigorously test them. This leads us to the next essential stage in the MBSE process: V&V.

In the next chapter, we will look at the concepts and techniques of model V&V. We will see how V&V activities can confirm the accuracy, completeness, and consistency of our models, and how they connect abstract representations to real-world systems. By understanding and using V&V techniques, we can improve the reliability and effectiveness of our models, resulting in better and more successful system implementations. As we move from model development to model verification, we continue our journey toward mastering all aspects of MBSE.

Questions

1. What is the primary purpose of behavioral models in MBSE?
   - **a** To capture the static architecture of a system
   - **b** To represent the dynamic aspects of a system over time
   - **c** To model only continuous system behaviors
   - **d** To eliminate the need for structural models
2. Which of the following is NOT a key characteristic of FSMs?
   - **a** States
   - **b** Transitions
   - **c** Continuous time evolution
   - **d** Initial state
3. What is the main advantage of using Petri Nets for modeling concurrent systems?
   - **a** They provide a graphical representation and a formal mathematical basis.
   - **b** They are limited to modeling only sequential processes.
   - **c** They cannot represent resource sharing or synchronization.
   - **d** They are not suitable for modeling large-scale systems.
4. Which of the following is a key feature of SDF models?
   - **a** Fixed token production and consumption rates
   - **b** Dynamic changes in token rates during execution
   - **c** Lack of deterministic behavior
   - **d** Inability to support static scheduling
5. What is the primary purpose of using discrete event models (DEM) in system modeling and analysis?
   - **a** To model systems with continuously changing state variables
   - **b** To represent systems with only deterministic behavior

c To understand and optimize the behavior of complex systems with discrete state changes
d To model systems without considering the concept of events

6 Which of the following is a key characteristic of agent-based models?
a Lack of interactions between agents
b Emergence of complex system-level behaviors from agent interactions
c Homogeneity among agents
d Absence of spatial and temporal dynamics

7 What is the main challenge in verifying and validating hybrid models?
a Simplicity of the models
b Lack of interactions between continuous and discrete components
c Abundance of standardized conceptual models
d Complexity arising from the interaction between continuous and discrete components

8 Which of the following is NOT an advantage of using object-oriented models in MBSE?
a Improved modularity and reusability
b Enhanced flexibility in system design
c Better alignment with real-world concepts
d Guaranteed simplicity for all types of systems

9 What is the primary benefit of using hierarchical decomposition in systems engineering?
a It provides a systematic way to manage complexity in large-scale systems.
b It eliminates the need for considering system-level properties.
c It guarantees the absence of subjectivity in the decomposition process.
d It removes the need for iterative refinement of the system design.

10 Which of the following statements best describes the role of theoretical models in MBSE?
a They provide a single, fixed approach to represent all types of systems.
b They eliminate the need for practical implementation and real-world testing.
c They offer a diverse array of techniques to capture and analyze complex system behaviors.
d They are independent of the specific goals and requirements of the system being modeled.

Answers

1-b; 2-c; 3-a; 4-a; 5-c; 6-b; 7-d; 8-d; 9-a; 10-c

References

Aguilar-Savén, R.S. Business process modelling: review and framework. *International Journal of production economics* 2004:90(2):129–149.

Balsamo, S., Personè, V.D.N., and Inverardi, P. A review on queueing network models with finite capacity queues for software architectures performance prediction. *Performance Evaluation* 2003:51(2–4):269–288.

Banks, J. *Discrete-event System Simulation*. Prentice Hall international series in industrial and systems engineering. Pearson Prentice Hall, 2005. ISBN 9780131446793. https://books.google.fr/books?id=CWZRAAAAMAAJ.

Bause, F. and Kritzinger, P.S. Stochastic Petri Nets: an introduction to the theory. 2013https://www.researchgate.net/publication/258705139_Stochastic_Petri_Nets_-An_Introduction_to_the_Theory. ISBN: 3-528-15535-3

Bebelis, V., Fradet, P., Girault, A., et al. BPDF: a statically analyzable dataflow model with integer and boolean parameters. In *2013 Proceedings of the International Conference on Embedded Software (EMSOFT)*, 2013:1–10. IEEE.

Bhattacharya, B. and Bhattacharyya, S.S. Parameterized dataflow modeling for DSP systems. *IEEE Transactions on Signal Processing* 2001:49(10):2408–2421.

Bhattacharyya, S.S., Murthy, P.K., and Lee, E.A. *Software Synthesis from Dataflow Graphs*. The Springer International Series in Engineering and Computer Science. Springer US, 2012. ISBN 9781461313892. https://books.google.fr/books?id=bCTvBwAAQBAJ.

Bilsen, G., Engels, M., Lauwereins, R., et al. Cycle-static dataflow. *IEEE Transactions on Signal Processing* 1996:44(2):397–408.

Bonabeau, E. Agent-based modeling: methods and techniques for simulating human systems. *Proceedings of the National Academy of Sciences* 2002:99(Suppl_3):7280–7287.

Booch, G., Maksimchuk, R., Engle, M., et al. *Object-Oriented Analysis and Design with Applications*. Addison–Wesley Object Technology Series. Pearson Education, 2007. ISBN 9780132797443. https://books.google.fr/books?id=8Rg5wxGj818C.

Booch, G., Rumbaugh, J., and Jacobson, I. *The Unified Modeling Language User Guide*. Addison–Wesley object technology series. Addison–Wesley, 2005. ISBN 9780321267979. https://books.google.fr/books?id=BqFQAAAAMAAJ.

Borshchev, A. The big book of simulation modeling: multimethod modeling with anylogic 6. 2015.https://www.amazon.fr/Big-Book-Simulation-Modeling-Multimethod/dp/0989573176/ref=tmm_hrd_swatch_0. ISBN: 0989573176

Cellier, F.E. and Greifeneder, J. *Continuous System Modeling*. Springer New York, 2013. ISBN 9781475739220. https://books.google.fr/books?id=aajqBwAAQBAJ.

Chiriac, N., Hölttä-Otto, K., Lysy, D., et al. Three approaches to complex system decomposition. In *DSM 2011: Proceedings of the 13th International DSM Conference*, 2011.

Connor, G. and Linton, O. Semiparametric estimation of a characteristic-based factor model of common stock returns. *Journal of Empirical Finance* 2007:14(5):694–717.

Dhillon, B.S. *Maintainability, Maintenance, and Reliability for Engineers*. CRC Press, 2006. ISBN 9781420006780. https://books.google.fr/books?id=v2HLBQAAQBAJ.

Douglass, B.P. *Real Time UML: Advances in the UML for Real-Time Systems*. Addison–Wesley object technology series. Addison–Wesley, 2004. ISBN 9780321160768. https://books.google.fr/books?id=LewzKVv3A20C.

Epstein, J.M. *Generative Social Science: Studies in Agent-Based Computational Modeling*. Princeton Studies in Complexity. Princeton University Press, 2012. ISBN 9781400842872. https://books.google.fr/books?id=6jPiuMbKKJ4C.

Estefan, J.A. Survey of model-based systems engineering (MBSE) methodologies. *INCOSE MBSE Focus Group* 2007:25(8):1–12.

Fehling, R. A concept of hierarchical Petri Nets with building blocks. In *Advances in Petri Nets 1993 12*, 1993:148–168. Springer.

Fishman, G.S. *Discrete-Event Simulation: Modeling, Programming, and Analysis*. Springer Series in Operations Research and Financial Engineering. Springer New York, 2013. ISBN 9781475735529. https://books.google.fr/books?id=y5fuBwAAQBAJ.

Fowler, M. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. Addison–Wesley Object Technology Series. Pearson Education, 2018. ISBN 9780134865126. https://books.google.fr/books?id=VTdtDwAAQBAJ.

Friedenthal, S., Moore, A., and Steiner, R. *A Practical Guide to SysML: The Systems Modeling Language*. The MK/OMG Press. Elsevier Science, 2014. ISBN 9780128002025. https://books.google.fr/books?id=97XfoAEACAAJ.

Fujimoto, R.M. Parallel and distributed simulation systems. In *Proceeding of the 2001 Winter Simulation Conference (Cat. No. 01CH37304)*, 2001:1:147–157. IEEE.

Gilbert, N. *Agent-Based Models*. Quantitative Applications in the Social Sciences. SAGE Publications, 2019. ISBN 9781506355603. https://books.google.fr/books?id=Az85yQEACAAJ.

Girault, C. and Valk, R. *Petri Nets for Systems Engineering: A Guide to Modeling, Verification, and Applications*. Springer Berlin Heidelberg, 2013. ISBN 9783662053249. https://books.google.fr/books?id=PNmoCAAAQBAJ.

Hart, L.E. Introduction to model-based system engineering (MBSE) and SysML. In *Delaware Valley INCOSE Chapter Meeting*, 2015:30. Ramblewood Country Club Mount Laurel.

Hatcliff, J., Leavens, G.T., Leino, K.R.M., et al. Behavioral interface specification languages. *ACM Computing Surveys (CSUR)* 2012:44(3):1–58.

Hopcroft, J.E., Motwani, R. Ullman, J.D. *Introduction to Automata Theory, Languages, and Computation, 3rd edition*, 2006. ISBN: 9780321455369. Pearson/Addison Wesley, United Kingdom. https://www.google.fr/books/edition/Introduction_to_Automata_Theory_Language/tEafQgAACAAJ?hl=fr.

INCOSE. *INCOSE Systems Engineering Handbook*. Wiley, 2023. ISBN 9781119814290. https://books.google.fr/books?id=HhjBEAAAQBAJ.

Jensen, K., Kristensen, L.M., and Wells, L. Coloured Petri Nets and CPN tools for modelling and validation of concurrent systems. *International Journal on Software Tools for Technology Transfer* 2007:9:213–254.

Joshi, A., Heimdahl, M.P., Miller, S.P., et al. Model-based safety analysis. Technical Report, 2006.

Kahn, G. The semantics of a simple language for parallel programming. In *IFIP Congress*, 1974:74:471–475.

Karnopp, D.C., Margolis, D.L., and Rosenberg, R.C. *System Dynamics: Modeling, Simulation, and Control of Mechatronic Systems*. EngineeringPro collection. Wiley, 2012. ISBN 9781118160077. https://books.google.fr/books?id=rSVHAAAAQBAJ.

Kim, O.D., Rocha, M., and Maia, P. A review of dynamic modeling approaches and their application in computational strain optimization for metabolic engineering. *Frontiers in Microbiology*, 2018:9:1690.

Kletz, T.A. *Hazop & Hazan: Identifying and Assessing Process Industry Hazards*, 4e. CRC Press, 2018. ISBN 9781351441353. https://books.google.fr/books?id=0ksPEAAAQBAJ.

Kochkov, D., Smith, J.A., Alieva, A., et al. Machine learning–accelerated computational fluid dynamics. *Proceedings of the National Academy of Sciences* 2021:118(21):e2101784118.

Kossiakoff, A., Biemer, S.M., Seymour, S.J., et al. *Systems Engineering Principles and Practice*. Wiley Series in Systems Engineering and Management. Wiley, 2020. ISBN 9781119516668. https://books.google.fr/books?id=gGzoDwAAQBAJ.

Laskey, K.B. and Laskey, K. Service oriented architecture. *Wiley Interdisciplinary Reviews: Computational Statistics* 2009:1(1):101–105.

Law, A.M. *Simulation Modeling and Analysis*, 6e. McGraw-Hill Series in industrial Engineering and Management Science. McGraw-Hill Education, 2024. ISBN 9781264268245. https://books.google.fr/books?id=cXu6zwEACAAJ.

Lee, C., Lim, M., Kim, C., et al. Reliability analysis of accelerated destructive degradation testing data for bi-functional DC motor systems. *Applied Sciences* 2021:11(6):2537.

Lee, E.A. and Messerschmitt, D.G. Synchronous data flow. *Proceedings of the IEEE* 1987:75(9):1235–1245.

Lygeros, J., Tomlin, C., Sastry, S., et al. Hybrid systems: modeling, analysis and control. *Electronic Research Laboratory*, University of California, *Berkeley, CA, Tech. Rep. UCB/ERL M* 2008:99:6.

Macal, C.M. and North, M.J. Tutorial on agent-based modeling and simulation. In *Proceedings of the Winter Simulation Conference*, 2005:14–pp. IEEE.

Madni, A.M., Boehm, B., Erwin, D., et al. *Recent Trends and Advances in Model Based Systems Engineering*. Springer International Publishing, 2022. ISBN 9783030820831. https://books.google.fr/books?id=lN1mEAAAQBAJ.

Madni, A.M., Madni, C.C., and Lucero, S.D. Leveraging digital twin technology in model-based systems engineering. *Systems* 2019:7(1):7.

Molloy. Performance analysis using stochastic petri nets. *IEEE Transactions on computers* 1982:100(9):913–917.

Moubray, J. *Reliability-centered Maintenance*. G—Reference, Information and Interdisciplinary Subjects Series. Industrial Press, 2001. ISBN 9780831131463. https://books.google.fr/books?id=bNCVF0B7vpIC.

Mouratidis, H. and Giorgini, P. Secure tropos: a security-oriented extension of the tropos methodology. *International Journal of Software Engineering and Knowledge Engineering* 2007:17(2):285–309.

Murata, T. Petri Nets: properties, analysis and applications. *Proceedings of the IEEE* 2002:77(4):541–580.

Myagmar, S., Lee, A.L., and Yurcik, W. Threat modeling as a basis for security requirements. 2005. Citeseer.

Nelson, W.B. *Accelerated Testing: Statistical Models, Test Plans, and Data Analysis*. Wiley Series in Probability and Statistics. Wiley, 2009. ISBN 9780470317471. https://books.google.fr/books?id=Dk85hUWrhp8C.

Newman, M.E.J. The structure and function of complex networks. *SIAM review* 2003:45(2):167–256.

Olsen, M.M. and Siegelmann, H.T. Multiscale agent-based model of tumor angiogenesis. *Procedia Computer Science* 2013:18:1016–1025.

Peterson, J.L. *Petri Net Theory and the Modeling of Systems*. Foundations of Philosophy Series. Prentice Hall, 1981. ISBN 9780136619833. https://books.google.fr/books?id=xNtQAAAAMAAJ.

Petri, C.A. Kommunikation mit automaten. 1962. https://edoc.sub.uni-hamburg.de/informatik/volltexte/2011/160.

Pidd, M. *Computer Simulation in Management Science*. Wiley, 1998. ISBN 9780471979319. https://books.google.fr/books?id=WmBRAAAAMAAJ.

Ramchandani, C. Analysis of asynchronous concurrent systems by timed Petri Nets. 1974. https://dspace.mit.edu/handle/1721.1/149425.

Rausand, M. and Hoyland, A. *System Reliability Theory: Models, Statistical Methods, and Applications*. Wiley Series in Probability and Statistics—Applied Probability and Statistics Section. Wiley, 2003. ISBN 9780471471332. https://books.google.fr/books?id=OtvWDwAAQBAJ.

Rumbaugh, J. *The Unified Modeling Language Reference Manual*. Addison–Wesley object technology series. Pearson Education, 2005. ISBN 9788178081069. https://books.google.fr/books?id=SGewzQEACAAJ.

Schneier, B. Attack trees. *Dr. Dobb's journal* 1999:24(12):21–29.

Sentilles, S., Štěpán, P., Carlson, J., et al. Integration of extra-functional properties in component models. In *International Symposium on Component-Based Software Engineering*, 2009:173–190. Springer.

Stamatis, D.H. *Failure Mode and Effect Analysis: FMEA from Theory to Execution*. ASQ Quality Press, 2003. ISBN 9780873895989. https://books.google.fr/books?id=TTxI8jbTkVwC.

Stuijk, S., Geilen, M., Theelen, B., et al. Scenario-aware dataflow: modeling, analysis and implementation of dynamic applications. In *2011 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation*, 2011:404–411. IEEE.

Trivedi, K.S. *Probability and Statistics with Reliability, Queuing, and Computer Science Applications*. Wiley, 2016. ISBN 9781119314202. https://books.google.fr/books?id=_yOXDAAAQBAJ.

van der Aalst, W. and van Hee, K.M. *Workflow Management: Models, Methods, and Systems*. Cooperative information systems. MIT Press, 2004. ISBN 9780262720465. https://books.google.fr/books?id=O1xW1_Za-I0C.

van Eck, D., McAdams, D.A., and Vermaas, P.E. Functional decomposition in engineering: a survey. In *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, 2007:48043:227–236.

Weilkiens, T. *Systems Engineering with SysML/UML: Modeling, Analysis, Design*. The MK/OMG Press. Elsevier Science, 2011. ISBN 9780080558318. https://books.google.fr/books?id=LKC9F5gQt3AC.

White, S.A. Introduction to BPMN. *IBM Cooperation* 2004:2. http://yoann.nogues.free.fr/IMG/pdf/07-04_WP_Intro_to_BPMN_-_White-2.pdf.

Wikipedia Contributors. Hierarchical decomposition. https://en.wikipedia.org/wiki/Hierarchical_decomposition,

Xing, L. and Amari, S.V. Fault tree analysis. In *Handbook of Performability Engineering*.595–620. Springer. 2008.

5

Model Verification and Validation

5.1 Verification

The verb "to verify" comes from the Low Latin "verificare," meaning "to present as true" (verus: true, facere: to do). Its meaning then evolved into "to examine the truthfulness of a thing." Today, system-level verification consists of determining the correctness and veracity of what is expressed in the system by both sides, taking into account differences in viewpoints and expectations.

In model-based systems engineering (MBSE), verification is important for checking the quality and accuracy of system models. Verification determines whether a model correctly represents the intended design and complies with established rules on syntax, semantics, and consistency (Engel 2010).

Verification confirms that the model is built correctly according to its specifications and that there are no internal errors or inconsistencies. It answers the question: "*Are we building the product correctly?*"

- It takes place throughout the development process.
- It verifies that the deliverables of a phase meet the conditions established at the start of that phase.
- It uses techniques such as reviews, inspections, audits, walk-throughs, and testing (unit tests, integration tests, etc.).

By verifying models early in the development lifecycle, potential issues and defects can be identified and fixed before they propagate to later stages, where the cost of correction is significantly higher (Stecklein et al. 2004). Model verification techniques include:

- **Syntax Checking**: Ensuring that the model adheres to the grammar and structure of the modeling language (Selic 2003).

- **Consistency Checking**: Verifying that the model elements are consistent with each other and with the overall system specification (Van Der Straeten et al. 2008).
- **Model Checking**: Exhaustively verifying the behavior of the model against specific properties expressed in temporal logic or other formal specifications (Baier et al. 2008; Clarke et al. 2018).

5.1.1 Key Stages of Verification

Verification includes several main steps that take place during the development process to check the model's quality and correctness at different times.

5.1.1.1 Audit Planning

The first stage of verification is audit planning, which involves:

- Defining the objectives and scope of the verification audits
- Identifying the deliverables and artifacts to be verified, such as requirements documents, design models, source code, and test cases
- Establishing the audit criteria and metrics to assess the quality and completeness of the artifacts
- Allocating the necessary resources and assigning responsibilities for conducting the audits
- Planning the schedule of verification activities aligned with the project timeline

Careful planning of verification audits is essential to ensure that all relevant aspects of the system model are thoroughly verified and that the verification process is efficient and effective (Hirshorn et al. 2017; Reid 2013).

5.1.1.2 Reviews and Inspections

The next stage involves systematic reviews and inspections of the model and its related documents:

- Organizing peer reviews and walkthroughs of requirements, design, and engineering artifacts to identify defects, omissions, and inconsistencies
- Performing detailed inspections of the model structure, behavior, and constraints to verify compliance with modeling standards and best practices
- Verifying that the model elements are consistent with each other and with the overall system specification
- Identifying and documenting problems, deviations, and improvement opportunities
- Tracking corrective actions through to completion and verifying their effectiveness

Reviews and inspections are powerful techniques for detecting defects early in the development process when it is less costly to fix them (Wiegers 2002; Gilb et al. 1993). They also promote knowledge sharing and collaboration within the development team.

5.1.1.3 Static Testing

Static testing techniques are applied to analyze the model without executing it. This includes:

- Performing static analysis of the model to check for potential defects, security vulnerabilities, and compliance with modeling standards and guidelines
- Using automated static analysis tools to detect issues such as data flow anomalies, control flow errors, and dead behaviors
- Checking the coverage of unit tests and identifying areas of the model that require additional testing
- Measuring model quality metrics such as complexity, maintainability, and readability
- Identifying high-risk areas of the model that require thorough testing and reviews

Static testing helps improve the quality and reliability of the system by identifying defects and improvement opportunities early in the development process.

5.1.1.4 Dynamic Testing

Dynamic testing involves executing the system model to verify its behavior and performance under different conditions. This includes:

- Running unit tests to verify the functionality of individual model components and subsystems
- Performing integration testing to verify the interactions and interfaces between different model components and subsystems
- Conducting system testing to validate the end-to-end behavior and performance of the complete system model
- Automating tests wherever possible and establishing regression test suites to verify the system's behavior after changes
- Analyzing test results and metrics, such as requirements coverage, defect density, and performance benchmarks

Dynamic testing is essential to validate the system model's behavior and ensure that it meets the specified functional and nonfunctional requirements.

5.1.1.5 Anomaly Management

Throughout the verification process, any anomalies, defects, or deviations identified must be properly managed and resolved. This involves:

- Capturing and documenting all anomalies found during reviews, inspections, and testing
- Categorizing and prioritizing anomalies based on their severity and impact
- Assigning anomalies to the appropriate team members for investigation and resolution
- Tracking the status of anomalies and ensuring that they are resolved in a timely manner
- Verifying the effectiveness of corrective actions through re-testing and reviews
- Analyzing anomaly trends and metrics to identify areas for process improvement

Effective anomaly management ensures that all defects and issues are properly addressed and that the system model is of high quality and fit for purpose.

5.1.1.6 Reporting and Continuous Improvement

Finally, verification activities and outcomes should be properly documented and reported to stakeholders. This includes:

- Preparing summary reports of verification activities, findings, and metrics
- Communicating verification status and results to project stakeholders and management
- Identifying lessons learned and areas for improvement in the verification process
- Defining and implementing process improvement initiatives based on verification feedback
- Continuously monitoring and optimizing verification activities throughout the project lifecycle

Regular reporting and continuous improvement help to ensure the effectiveness and efficiency of the verification process and contribute to the overall success of the system development effort.

5.1.2 Key Considerations of Verification

5.1.2.1 Controlling Complexity and Size

One of the main issues in the verification of complex systems is dealing with the large size and complexity of the system under study. Today's mission-critical systems generally have a complex design that includes various parts, such as

hardware, software, and interfaces. Careful verification of such systems can lead to a large number of reviews and tests. To manage this complexity, verification needs to be well-organized and focused on the most important parts of the system, using a risk-based approach to make the best use of verification resources.

5.1.2.2 Verification of Emerging Properties

In a complex critical system, certain important properties like safety or performance derive from the way individual parts collaborate. To verify these properties, a holistic approach is needed. Techniques such as architecture reviews, formal proofs, and system integration tests are essential to check the system's overall behavior and ensure it meets its requirements. This means grasping the system's structure and how its components relate to one another.

5.1.2.3 Compliance with Norms and Standards

Critical systems often comply with specific norms and standards, such as DO-178C for avionics, ISO 26262 for vehicles, or IEC 62304 for medical equipment. These standards contain strict rules regarding verification activities, the techniques to be used, and the evidence to be provided. Proving compliance with these standards makes the verification process more complex because it demands extra costly activities, paperwork, and monitoring. Organizations need to fully understand the applicable standards and put in place robust procedures to ensure that their verification efforts fulfill these requirements.

5.1.2.4 Verifying Robustness and Resilience

Besides checking the system's normal operation, it is important to assess the system's ability to cope with unexpected or harmful situations. This includes identifying possible points of failure, testing fault detection and recovery systems, and testing under fault injection and extreme conditions. Checking resilience requires a specific safety analysis approach, using techniques like Failure Mode and Effects Analysis (FMEA), Fault Tree Analysis (FTA), and Hazard and Operability Analysis (HAZOP). These analyses help identify potential risks and failure situations, facilitating the creation of test cases and verification plans to ensure system resilience.

5.1.2.5 Verification Independence and Objectivity

For the most crucial systems, standards usually call for independent verification, done by a team not involved in the development or even by an external organization. This enables more objective verification, free from the influence of development biases, but requires good coordination and information sharing between teams. Keeping track of versions and configurations is essential to ensure verification is based on the right scope and that any problems found are properly linked

to the relevant development documents. Independent verification complicates the process but is necessary for achieving the highest assurance levels.

5.1.2.6 Verification Tools and Automation

Given the large number of verification tasks required for a mission-critical system, tools and automation play a key role in productivity and quality. However, verification tools for critical systems also need to be validated and qualified according to the relevant standards. Setting up and maintaining qualified tool sets, as well as writing automated test scripts, requires considerable investment. Organizations must carefully weigh the costs and advantages of verification automation, taking into account considerations such as system complexity, expected duration, and potential for reuse in other projects.

5.2 Validation

The word "validate" comes from the Latin "validus," meaning "strong, effective." In the Middle Ages, it evolved to mean "having the conditions required to produce its effect." Today, validation is the process of assessing whether a model accurately represents the real-world system and meets the stakeholders' needs and requirements (Engel 2010). It answers the question: "Are we building the right model?" (Boehm 1984).

Validation ensures the model accurately reflects the system it represents and that it achieves its intended goal. The goal of validation is to confirm that the system addresses actual user needs and meets its intended purpose.

- It generally takes place at the end of the design and development processes.
- It uses techniques such as user acceptance testing (UAT), operational testing, and alpha/beta testing.
- It is performed by customers or end-users, or an independent third party.

In the context of MBSE, validation is important to ensure that the created system matches the customer's expectations and requirements. By checking models against the actual system and stakeholder needs, any discrepancies or overlooked requirements can be spotted quickly, reducing the risks of delivering a system that does not meet its intended use (Wymore 2018).

Model validation techniques include:

- **Face Validation:** Reviewing the model with subject-matter experts to assess its plausibility and correctness (Sargent 2010).
- **Empirical Validation:** Comparing the model's behavior and outputs with data collected from the real-world system or similar systems (Robinson 1997).
- **Acceptance Testing:** Demonstrating that the model meets the stakeholders' requirements and acceptance criteria (Wymore 2018).

5.2.1 Key Stages of Validation

Validation involves several key steps carried out during the development phase to check system's behavior, performance, and usability in the environment in which it will be used.

5.2.1.1 Validation Planning

The first stage of validation is planning, which involves:

- Defining the objectives and scope of the validation activities
- Identifying the user requirements and acceptance criteria that the system must satisfy
- Establishing the validation metrics and thresholds to assess the system's performance and usability
- Allocating the necessary resources and assigning responsibilities for conducting the validation activities
- Scheduling the validation activities in line with the project timeline, including user involvement

Careful planning of validation activities is essential to ensure that the system is thoroughly validated against its intended use and that the validation process is efficient and effective (Reid 2013).

5.2.1.2 Development of Validation Tests

The next stage involves developing the validation tests and scenarios that will be used to assess the system's behavior and performance:

- Creating realistic and representative test scenarios based on the system's intended use cases and operational scenarios
- Defining the test data sets and test environments that reflect the real-world conditions in which the system will operate
- Developing automated test scripts and test procedures to guide the validation activities and ensure their reproducibility
- Setting up the validation test environment, including any necessary hardware, software, and tools

Well-designed validation tests are crucial to identify potential issues and ensure that the system meets its intended use under realistic conditions (Engel 2010).

5.2.1.3 User Acceptance Testing

UAT is a critical stage of validation where end-users evaluate the system to ensure it meets their needs and expectations:

- Executing the validation test scenarios with representative users to assess the system's functionality, usability, and performance

- Collecting user feedback, opinions, and evaluations on the system's behavior and user experience
- Identifying any usability issues, missing functionalities, or areas for improvement based on user feedback
- Validating that the system meets the real needs and requirements of its intended users

UAT is essential to ensure that the system is accepted and adopted by its end-users and delivers value in real-life operational configuration (Morgan et al. 2019).

5.2.1.4 Operational and Performance Testing

Operational testing involves validating the system's behavior and performance in its intended operational environment under realistic conditions:

- Deploying and testing the system in a representative operational environment that reflects real-world conditions
- Conducting performance tests, load tests, and stress tests to validate the system's responsiveness, reliability, and resource usage under various operational workloads
- Validating the system's compliance with nonfunctional requirements such as safety, security, scalability, and maintainability in operational settings
- Testing the system's operational procedures, maintenance activities, and support processes to ensure their effectiveness and efficiency

Operational testing is crucial for identifying and resolving any issues that may arise when the system is deployed in its real environment and subjected to realistic operational requirements (Engel 2010).

5.2.1.5 Acceptance and Deployment

The final stage of validation is to obtain formal acceptance of the system by stakeholders and users and deploy it for operational use:

- Conducting acceptance reviews and walkthroughs with stakeholders to demonstrate that the system meets its intended requirements and acceptance criteria
- Analyzing and reporting on the results of all validation activities, including test results, user feedback, and operational performance metrics
- Identifying any remaining issues or non-conformities that need to be resolved before final system acceptance
- Obtaining formal sign-off and acceptance from stakeholders and users that the system is validated and ready for operational deployment
- Deploying the validated system in its intended operational environment and transitioning it to maintenance and support

Successful completion of the acceptance and deployment stage marks the transition of the system from development to operational use and the start of its operational lifecycle.

5.2.1.6 Comprehensive Coverage of Critical Scenarios

Critical systems must be validated for a wide variety of situations, including rare or emergency cases. Covering all crucial scenarios could require a lot of testing, which may take a long time and consume many resources (Engel 2010). Smart, risk-based validation methods should be developed to improve this coverage and emphasize the most crucial scenarios. Techniques such as combinatorial testing, model-based test case generation, and exploratory testing can help cover more scenarios while keeping the validation process manageable (Utting et al. 2012).

5.2.1.7 Reproduction of Actual Operating Conditions

To validate a critical system, it needs to be tested in conditions that are as similar as possible to its real-world environment. However, creating realistic operational conditions in a controlled environment can be difficult and expensive (Sargent 2010). This may involve creating complex test configurations, using detailed simulators, or performing live tests in the field. The validation environment must truly represent the system's intended use, including features such as workload, concurrency, data inputs, and environmental factors. Approaches like hardware-in-the-loop (HIL) testing, virtual commissioning, or digital twins can assist in creating realistic validation configurations that closely reflect actual operating conditions (Schluse et al. 2018).

5.2.1.8 Validation of Critical Extra-functional Properties

Besides confirming system operation, mission-critical systems must comply with strict rules regarding extra-functional properties such as safety, security, reliability, performance, and resilience. Checking these properties often requires special methods and knowledge, going beyond functional testing (Glinz 2007). For instance:

- Validating safety may involve hazard analysis, fault injection testing, or formal safety proofs
- Validating security may require penetration testing, vulnerability scanning, or security audits
- Validating reliability may involve stress testing, endurance testing, or accelerated life testing
- Validating performance may require load testing, benchmarking, or performance profiling

There are specific tools and techniques for checking each of these extra-functional properties, and they need to be selected and used with care (Chung et al. 2012).

5.2.1.9 Traceability and Documentation Management

Critical systems must comply with strict standards and regulations, which require detailed recording of the validation process. Keeping track of this compliance and creating necessary documents such as validation plans, test specifications, test reports, and safety records can be a considerable task (Cleland-Huang et al. 2012). It calls for careful management of configurations, control of changes, and tracking of versions to keep validation evidence steady and up-to-date throughout the system's lifecycle.

Often, specialized tools for managing requirements and testing are necessary to create and maintain the links between requirements, test cases, and validation results (Winkler and von Pilgrim 2009).

5.2.2 Key Considerations of Validation

5.2.2.1 Validation Costs and Timescales

Due to the extensive nature of validation for critical systems, the costs and timelines can be significant, sometimes exceeding the actual development work itself. It is important to plan and manage validation carefully to control costs and delays (Engel 2010). A streamlined validation approach should be established, focusing on the most important risks and using automation and reuse when possible. Techniques such as continuous testing, incremental validation, and risk-focused testing can help in reducing validation costs and timescales without sacrificing system quality and reliability (Marijan et al. 2013).

5.2.2.2 Skills and Responsibilities

Validating critical systems requires special skills and knowledge in areas such as domain knowledge, safety engineering, security testing, or reliability analysis. Organizations may need to train their validation teams or call on external experts to acquire the required skills (Morgan et al. 2019). Clear roles and responsibilities for validation tasks are necessary, often involving independent teams or external reviewers to ensure an objective review of system suitability. Good communication and teamwork between development and validation teams are essential for a smooth and efficient validation process.

5.3 The Evolution of V&V

The concepts of verification and validation (V&V) are ancient, stemming from our innate desire to ensure our creations work as they should. However, the formalization of V&V as clear processes in systems engineering is fairly new, emerging from the increasing complexity of engineered systems and the serious risks that can arise if they fail.

5.3.1 Early Foundations

The beginnings of V&V can be seen in early engineering work. For example, the builders of the pyramids used advanced surveying and measuring methods to check that their construction was correct. Likewise, builders of medieval cathedrals relied on scale models and mathematics to confirm their designs before starting large-scale construction. These early practices, although not officially called V&V, illustrate the fundamental ideas of ensuring that a system meets its specifications and works as intended.

5.3.2 The Industrial Revolution and Quality Control

The Industrial Revolution brought about major changes in the size and complexity of engineered systems. With the onset of mass production, the need for better quality control methods increased. In the early twentieth century, innovators like Walter A. Shewhart developed statistical process control techniques, which paved the way for modern quality assurance methods. These changes, mainly focused on manufacturing, introduced important ideas that would later affect V&V in systems engineering.

5.3.3 The Space Race and Formal V&V

The V&V was formalized during the Space Race in the 1960s. The Apollo program was very complex, and its mission was very important. Strict methods were needed to ensure that the system was reliable and safe. NASA created detailed V&V processes that clearly separated verification (checking whether the system was built correctly) from validation (checking whether the correct system was built).

During this time, structured testing methods, formal reviews, and simulation for system validation became commonplace. The success of the Apollo missions demonstrated the importance of systematic V&V processes, leading to wider use in other critical industries such as healthcare and defense.

A bit of history

In 1962, NASA sent Mariner 1 into space, aiming to fly by Venus to collect data. But shortly after it launched, the spacecraft lost its trajectory and had to be destroyed by the safety officer. This mission failed completely, leading to major financial losses and a considerable delay in NASA's efforts in space.

The failure happened due to a simple mistake in the spacecraft's guidance software. A hyphen was missing in a piece of code, which made the guidance system misinterpret the data. This minor mistake caused serious problems, showing how important it is to have thorough V&V processes.



Mariner 1. *Source*: Eric Long/Smithsonian / NASA / Public Domain.

The Mariner 1 issue is considered as one of the most expensive hyphen mistakes ever, costing around US$18.5 million, or US$165 million today. It served as a painful lesson for the aerospace sector, underlining the need for careful software testing and quality control.

After this failure, NASA and other space organizations considerably improved their V&V methods. They implemented stricter code reviews, added automated testing tools, and created formal methods to check the accuracy of important software parts. These increased V&V efforts have helped avoid similar problems in subsequent space missions.

The Mariner 1 story clearly shows the essential role these V&V processes play in ensuring the success and safety of complex systems. It is a reminder that even small mistakes can lead to serious problems, and emphasizes the need for a careful and organized approach to V&V.

5.3.4 The Software Crisis

In the 1970s and 1980s, as computer systems were increasingly used, the software industry was confronted to its own problems regarding reliability and correctness. The "software crisis" revealed the need for stricter development and testing methods. This prompted the use of V&V concepts from systems engineering in software engineering, leading to the development of practices such as unit testing, integration testing, and formal methods for software verification.

5.3.5 The Rise of Model-based Approaches

The late twentieth and early twenty-first centuries saw a shift toward model-based approaches in systems engineering, including MBSE. This shift greatly influenced V&V practices. Models became crucial parts of the development process, allowing for earlier and more thorough verification. Simulations and virtual testing made it possible to validate system behavior before building real prototypes, saving time and money.

The emergence of MBSE languages and tools, such as Systems Modeling Language (SysML), provided new techniques to represent and analyze system requirements, architecture, and behavior. This has improved traceability between requirements and system elements, strengthening verification efforts. Additionally, executable models allowed for dynamic validation of system behavior, enabling engineers to "run" their designs and see how they performed in different situations.

5.3.6 V&V Today

As we began the twenty-first century, the growing complexity of systems, specifically cyber–physical systems and systems-of-systems, has created new challenges for V&V. Traditional methods, although useful, have become insufficient for in-depth V&V of these complex, interconnected systems. This situation has led to the development of advanced V&V techniques, including:

- **Formal Methods**: Mathematical approaches to system specification and verification, allowing for rigorous proofs of system properties.
- **Model Checking**: Automated techniques for verifying that a system model satisfies a set of crucial properties.
- **HIL Testing**: Combining physical hardware with simulated components to validate system behavior in realistic conditions.
- **Continuous Integration and Continuous Validation (CI/CV)**: Integrating V&V activities throughout the development lifecycle, enabled by automation and DevOps practices.

5.3.7 The Current Landscape and Future Directions

Today, V&V stands as a critical discipline within systems engineering, recognized as essential for developing reliable, safe, and effective systems. The integration of V&V with MBSE practices continues to evolve, with ongoing research into areas such as:

- **Artificial Intelligence (AI)-assisted V&V**: Leveraging machine learning (ML) and AI to enhance test case generation, anomaly detection, and requirements analysis.
- **Digital Twins**: Using high-fidelity digital representations of physical systems for CV throughout the system lifecycle.
- **Formal Verification of AI/ML Systems**: Developing techniques to verify and validate the behavior of systems incorporating AI and ML components.

Future of V&V involves both challenges and opportunities. Modern systems are more autonomous and adaptable, and can include AI/ML components. These changes create new areas to explore in V&V. Meanwhile, improved computing power, modeling methods, and analytical tools are making V&V processes more comprehensive and efficient.

The V&V history shows how much the engineering community cares about making sure our systems are reliable, safe, and effective. As technology advances and faces more complex problems, V&V principles and practices keep evolving. They will remain key in shaping the future of systems engineering.

The benefits of numerical simulation for crash tests

Numerical crash test simulation is a computer method that models how vehicles behave in crashes using software and finite element analysis. These virtual tests help engineers to mimic real crash situations and assess vehicle safety and performance.

Numerical simulations greatly lessen the need for real crash tests, which saves time and money. Engineers can quickly make design changes by adjusting CAD files and running virtual tests rather than spending time and money on physical tests.

Recent research has shown a close link between numerical simulations and physical tests, with accuracy rates of up to 97% in some cases (Prabhaharan et al. 2022). Advanced techniques like deep learning are being used to:

- Optimize crashbox designs
- Predict structural performance
- Generate lightweight design solutions with high accuracy

By using numerical simulations, automotive engineers can improve vehicle safety design, reduce costs, and speed up development in crash protection technologies.

5.4 V&V Concerns

When performing checks and confirmations for complex systems, there are a number of important concerns that need to be considered to make sure the system is good, reliable, and fit for its intended use. These concerns cover many parts of the system, including its operation, safety, security, and ease of use.

5.4.1 Operation and Compliance

One of the main concerns of V&V is to confirm that the system correctly performs all its intended functions in accordance with its specifications and requirements. This means making sure the system is complete, consistent, and meets the necessary standards and regulations (Engel 2010). Testing the functions of the system is very important to see how it performs and whether it complies with the rules.

In addition to testing, other V&V tasks, such as reviews, inspections, and audits, are important for checking that the system complies with the relevant standards and guidelines. These tasks help detect problems, errors, or areas for improvement in the system design and execution.

5.4.2 Performance and Efficiency

Another important issue in V&V is to make sure that the system meets its performance objectives. This means checking that the system meets its requirements in terms of response time, throughput, resource use, and scaling under different operational situations and loads.

Performance testing methods such as load testing, stress testing, and benchmarking, are used to check how the system acts and performs in various scenarios. These tests help detect performance problems, resource limitation, or inefficiencies in the system's design or configuration.

Moreover, methods such as profiling and performance monitoring are used to measure and study the system performance properties, including, for example, CPU, memory, and I/O tasks usage. This information can be used to improve the system performance, identify areas of growth, and ensure they remain within acceptable limits.

5.4.3 Reliability and Availability

Reliability and availability are important issues for many systems, especially those that are mission-critical or need to be operational at all times. V&V activities must ensure that the system is reliable, meaning it can perform its task without failure and is ready to be used when needed.

Techniques for reliability testing, such as stress testing, endurance testing, and fault injection, are used to assess how the system can recover from failures in different situations. These tests can help detect reliability problems such as memory leaks, resource usage, or error propagation.

Availability testing verifies if the system can be used when needed. This includes testing the system's backup, failover, and disaster recovery systems to ensure it can keep providing service despite failures or issues.

Reliability and availability requirements are often expressed in terms of metrics like Mean Time Between Failures, Mean Time To Repair, and availability percentage. V&V activities focus on measuring and confirming these metrics to ensure the system meets reliability and availability objectives.

5.4.4 Security

Security is very important in today's threat-filled digital world. V&V activities must focus on checking the system's security, its ability to prevent unauthorized access, and its capacity to cope with possible attacks or breaches.

Security testing methods such as penetration testing, vulnerability analysis, and security audits are used to find security weaknesses or problems in the system's design, build, or configurations. These tests mimic real attack situations and try to take advantage of known or suspected security issues.

Besides testing, security V&V activities also include examining the system's security structure, cryptographic systems, access control rules, and ability to record and monitor security events. The objective is to make sure the system complies with security best practices and standards, such as NIST or ISO 27001, and offers adequate protection for its data and resources.

5.4.5 Safety

For systems where safety is of crucial importance, such as those in aerospace, automotive, or medical fields, V&V tasks must focus on checking that the system can operate safely and does not cause serious damage to people or the environment.

Besides traditional safety methods (FTA, FMEA, HAZOP, etc.), formal methods, including model checking and theorem proving, are also part of the safety-critical

V&V to mathematically confirm that the system complies with its safety rules and constraints. These methods ensure the system will remain safe in all kinds of situations.

5.4.6 Usability and User Experience

Usability and user experience matter in V&V. They influence how well users accept, adopt, and use the system. Therefore, V&V tasks should make sure the system is user-friendly, clear, and offers a good experience.

Usability testing is an important way of checking the system's user interface, navigation, and general usability. It typically involves users performing common tasks with the system while observing what they do, getting their feedback, and examining performance data such as task duration and error frequency.

Usability V&V techniques include:

- Heuristic evaluation to assess the system's compliance with established usability principles and guidelines
- Cognitive walkthrough to evaluate the system's learnability and ease of use for new users
- User surveys and interviews to gather subjective feedback on the system's usability and user satisfaction

Besides usability, V&V also looks at the overall user experience, including elements such as design, emotional connection, and accessibility. The aim is to make sure the system works properly and offers an engaging experience to users.

5.4.7 Maintainability and Scalability

Maintainability and scalability are important issues in V&V, especially for systems that will evolve and expand over time. V&V tasks must ensure that the system is designed and built to allow easy maintenance, changes, and upgrades in the future. Maintainability V&V techniques include:

- Model reviews and inspections to assess the quality, readability, and adherence to standards
- Modularity and coupling analysis to evaluate the system's architecture and design for maintainability
- Regression testing to ensure that modifications do not introduce new defects or break existing functionality

Scalability V&V, on the other hand, focuses on validating the system's ability to handle increased workload, data volume, or users without degrading performance or reliability. This involves techniques such as:

- Load and performance testing to assess the system's behavior under various scalability scenarios
- Capacity planning and modeling to predict the system's resource requirements and scalability limits
- Scalability testing to verify the effectiveness of the system's scaling mechanisms, such as load balancing or horizontal scaling

5.4.8 Summary

V&V activities should adopt a holistic approach that takes into account all key issues impacting the quality, reliability, and suitability of the system. The importance of each issue may vary according to the specific domains of the system, its criticality, and the needs of stakeholders.

To make sure all issues are properly managed, V&V teams usually create a traceability matrix. This matrix connects each issue to particular V&V activities, techniques, and documents. This process keeps a clear and trackable record of how each issue has been checked throughout the system development lifecycle.

Effective V&V requires a blend of technical skills, domain knowledge, and a cautious approach. It requires strong teamwork between V&V engineers, system developers, and stakeholders to confirm that the system meets its objectives and is useful for users.

By addressing key issues such as operation, compliance, performance, reliability, security, safety, usability, maintainability, and scalability, V&V activities are crucial to ensuring the quality, dependability, and success of complex systems.

5.5 Theoretical Bases

When examining the truth and value of a system from the perspective of all the parties involved, philosophy shows us that all arguments need a solid framework. Without it, they can easily fall into linguistic traps, such as sophisms. Science has developed significantly since early attempts to clarify argumentation, such as syllogism. Today, we have a wide range of theories to support our discourses. This includes the different types of logic that form the basis of the various paradigms discussed in the previous chapter.

5.5.1 First-order Logic

First-order logic is the basis for formal system specification. It allows us to express properties of data structures and operations using predicates, quantifiers, and rules for inference. First-order logic is particularly useful for reasoning about the correctness of a discourse. It is the basis of formal specification languages such as Z or B (Spivey 1992; Abrial 1996).

First-order logic is important for V&V tasks because it allows for the formal specification and verification of system properties. By using first-order logic to express requirements and design constraints, it is possible to mathematically show that an implementation meets its specification. This is especially important for safety-critical systems.

For example, think about an air traffic control system. One might specify in first-order logic that "for every aircraft, if two aircraft are in the same airspace, then they must keep a minimum distance from each other." This property can then be formally checked against the system design and implementation to confirm there is a risk of collision.

5.5.2 Temporal Logic

Temporal logics such as linear temporal logic (LTL) and computation tree logic can be used to express the way a system evolves over time. They define properties that are expected from a system, such as liveliness (something will happen at some point), reliability (something bad will not happen), and fairness (something will happen continuously). Temporal logics are essential for model-checking methods (Baier et al. 2008; Clarke et al. 2018).

Temporal logics are very important for V&V, especially when checking the behavior of reactive and concurrent systems. By formulating requirements about how the system behaves over time and comparing them with a model of the system dynamics (such as a state machine or a Petri Net), we can confirm that the system will always exhibit the expected behavior.

For example, for a traffic light controller, we can state in temporal logic that "the lights should not be green in both directions at once" (a safety property) or "every time a pedestrian requests it, the pedestrian light will eventually turn green" (a liveness property). Model-checking tools can then automatically check whether these properties are true for all possible executions of the model of the controller.

5.5.3 Automata Theory

Automata theory is concerned with models focusing on states and transitions, such as finite automata, stack automata, or Turing machines. Automata are used to model the behavior of sequential systems and analyze their properties, such as

language recognition or state accessibility. This theory is key in checking behavioral models (Hopcroft et al. 2006).

In V&V, automata theory is the basis for demonstrating and studying system operation. By using an automaton to represent a system's behavior, one can reason about its properties, such as reachability (can we reach a certain state?), safety (can we avoid undesired states?), or liveness (will we reach a desired state at some point?).

For example, the different states of the entities in a communication protocol and the messages they exchange can be represented as communicating automata. Verification methods stemming from automata theory, such as model checking or bisimulation checking, can then be used to check if the protocol meets its required properties, such as the absence of deadlocks or correct data transfer.

5.5.4 Graph Theory

Graph theory studies data structures composed of nodes and edges and the algorithms for manipulating them. Many checking issues can be reduced to graph issues, such as path-finding, depth or breadth traversals, cycle detection, or the computation of strongly related components. Graph theory can also be used to analyze the way data flows or control systems (Bondy and Murty 1976).

Graph theory is important in V&V in different ways. Many system designs, such as dependency graphs, call graphs, or control flow graphs, are basically graphs. Graph methods can help study these designs and confirm properties of the system structure or behavior.

For example, in a software configuration, we can extract the graph of calls between functions or methods. Graph search methods can then help find unreachable code (dead code) or cycles in the call graph (possible infinite loops). Graph-based measures, such as cyclomatic complexity, can also be calculated to evaluate code complexity and testability.

5.5.5 Computability and Complexity Theory

Computability theory examines what machines can compute, dividing problems into those that can be solved (decidable) and those that cannot (undecidable). It shows impossible tasks, such as the halting problem, which indicates that not all programs can be checked for termination. Complexity theory, on the other hand, examines how much time and memory are required to solve a problem and classifies problems into different complexity classes (Arora and Barak 2009).

Both computability and complexity theory are important for V&V in various ways. They show the limits of what can be verified: some properties cannot be

confirmed for all programs since they are undecidable. Additionally, these theories help in selecting the right verification methods based on the complexity of the problem.

For example, when checking a scheduling algorithm, we may want to show that the scheduler always ends and doesn't get stuck. However, because of the halting problem, we know that showing termination in general cases is undecidable. Therefore, we must use other methods, such as proving termination for a limited type of schedules or applying delays to reduce the risk of non-termination. Complexity theory also indicates that some scheduling issues, like finding the best schedule, are NP-hard. Thus, it is better in this case to avoid seeking an exact answer and instead use methods like heuristics or approximations.

5.5.6 Summary

These basic concepts are used in tools such as theorem provers, model-checkers, static analyzers, and test generators for V&V. They provide a solid mathematical basis for specifying, modeling, analyzing, and verifying systems automatically and reliably. It is important to use them properly, along with testing and reviewing, in order to make sure critical systems are correct and safe.

Of course, the theories presented here are far from exhaustive. We could have mentioned theories such as modal logic, epistemic logic, fuzzy logic, lambda calculus, pi calculus, and more. There is a lot of material available on these topics, with many more theories that can help with V&V. As the field of V&V grows, new theories and methods will likely emerge, giving us more ways to guarantee system correctness and reliability.

The diversity in these theoretical methods shows how complex and varied modern systems are. Each theory has its own benefits and drawbacks, helping engineers to select the best tools for their specific V&V problems. As we continue to push the boundaries of system complexity, the integration and synergy of these various theoretical foundations will become increasingly important in developing comprehensive and robust V&V strategies.

5.6 Motivations for V&V Models

Models enable the system to be described clearly and precisely using a common visual language. This reduces the risk of inconsistencies or different interpretations of requirements, which often lead to errors. Models provide a common ground for V&V, avoiding the ambiguous nature of natural language.

For instance, take a complex aerospace system with many requirements. Using a formal modeling language like SysML to describe these requirements in an organized visual way ensures, at least in theory, that everyone involved understands

the system specification clearly and consistently. This clear representation provides a reliable basis for later V&V activities, such as model reviews and test case generation.

5.6.1 Early Verification Through Model Analysis

Models enable early checking to be carried out during the design phase, before the system is even built. Static model analysis can automatically check aspects such as consistency and completeness. This enables the finding and fixing of design flaws early on, reducing verification costs and time later on.

For example, when creating a real-time embedded system, a formal model of its behavior, such as a timed automaton, can be analyzed with model-checking tools to ensure the system is free from deadlocks and that timing constraints are met. By finding and fixing any problems at the model stage, developers can prevent expensive rework and delays that would occur if these issues were found during later phases or testing.

5.6.2 Automatic Test Case Generation

Behavioral models such as state diagrams can be used to automatically produce test cases that cover all scenarios. This enables the systematic creation of a complete test suite that remains up-to-date with the model, unlike manual definitions, which can be tedious and incomplete. Model-based testing (MBT) supports thorough and efficient verification.

For example, think about a complex communication protocol modeled using a state machine diagram. By using MBT methods, such as coverage-based testing, a full set of test cases can be automatically generated from the model, ensuring all relevant states and transitions are tested. This method not only saves time and effort over manual test creation but also helps keep tests consistent and complete as the model evolves.

5.6.3 Traceability and Requirements Coverage

Models capture and trace system requirements in a structured way, allocating them to the model elements that implement them. This makes sure that all requirements are taken into account in design and testing. Traceability diagrams and coverage data can be automatically generated from models, helping with V&V tracking and reporting (Königs et al. 2012).

For example, in a large system development project, requirements can be expressed in a model and connected to the corresponding design parts, such as

use cases, components, or classes. During the V&V phase, traceability reports can be created to show that each requirement has been properly integrated into the design and tested. This can confirm that no requirements are missed and provides proof of requirements coverage for compliance and auditing purposes.

5.6.4 Validation by Simulation and Prototyping

Executable models, such as parametric SysML models or timed Petri Nets, enable system behavior to be simulated from the early design phase to the more detailed ones. This enables early checks of operational scenarios and identification of dynamic problems, even before integration occurs. Model-based prototyping is used to confirm user interfaces and usability. Simulation and prototyping help to reduce the risk of late discovery of validation issues (Broy 2010).

For instance, in the automotive industry, executable models of a car's control systems, such as engine management or advanced driver assistance systems, can be simulated in a virtual environment to check their behavior under different conditions. This enables engineers to find and fix performance and safety issues or undesired interactions early in development without the need for physical prototypes. Model-based simulation and prototyping reduce validation costs and times while boosting the overall quality and reliability of the final product.

5.7 Techniques of V&V in MBSE

The following subsections discuss the different techniques that can be used in MBSE for verifying and validating systems.

5.7.1 Model Checking

Model checking is an automated formal verification technique that exhaustively verifies the behavior of a model against specific properties expressed in temporal logic or other formal specifications (Clarke et al. 2018; Baier et al. 2008). It examines all possible states of the system to see if the model meets these properties, offering confidence about important system properties.

Model checking relates closely to V&V tasks in MBSE, as it provides a structured and automated way to confirm key system functions and constraints. By defining desired properties in a formal way, such as LTL, and using a model checker to analyze the model completely, engineers can confirm the system design aligns with these properties.

For example, consider the design of a safety-critical medical device, such as an infusion pump. Using model checking, engineers could specify safety properties, such as:

- The pump shall never deliver a dose exceeding the prescribed limit.
- If an occlusion is detected, the pump shall sound an alarm within three seconds.

The model of the infusion pump, including its control logic and interactions, can be evaluated against these properties using tools such as UPPAAL (Behrmann et al. 2004), SPIN (Holzmann 1997), or NuSMV (Cimatti et al. 2002). These tools will examine all possible states and transitions of the model to check that the safety properties are satisfied in every situation. If a property fails, the model checker will provide a counterexample, helping engineers to find and fix design issues.

This automated and in-depth analysis provides a level of verification that would be hard to reach through manual testing or simulation. It uncovers hidden defects or unusual cases that may be missed otherwise, which is very important for safety-critical systems.

Other formal verification techniques include:

- **Theorem Proving**: Uses a formal system of logical deduction to prove properties about a system, relying on axioms and inference rules to construct proofs. Tools such as Isabelle (Nipkow et al. 2003), Coq (Bertot et al. 2013), and PVS (Owre et al. 1992) are commonly used.
- **Constraint Solving:** Formulates the system and its desired properties as a set of constraints, and uses constraint solvers such as satisfiability modulo theories solvers to find solutions or prove that no solution exists. Tools like Z3 (De Moura and Bjørner 2008) are often used for this purpose.

Formal verification methods are very useful for highlighting important system properties tied to safety, security, and reliability. This includes checking for deadlock-free behaviors, reachability, liveness, and fairness. Industries such as aerospace, automotive, and medical devices rely on formal methods to verify the correctness of safety-critical parts (Woodcock et al. 2009).

Despite some drawbacks, such as the risk of state space explosion in complex models and the need for specialized knowledge in formal specification languages, model checking's automation, rigor, and ability to work with complex systems make it a key tool in the V&V toolbox, particularly for safety-critical and mission-critical uses.

5.7.2 Simulation

Simulation is a technique for dynamic analysis that runs a model over time to check how it behaves under various conditions (Law 2024; Banks 2005). It serves as a powerful tool for V&V in MBSE, enabling engineers to examine system dynamics, test different scenarios, and assess performance and reliability.

Simulation is particularly relevant to V&V activities because it enables:

- **Early Detection of Design Flaws**: By simulating the system model early in the development process, engineers can identify and fix design issues before they propagate to later stages, where the cost of fixing them is much higher.
- **Testing of Critical Scenarios**: Simulation allows engineers to test the system's behavior under a wide range of scenarios, including rare or extreme conditions that may be difficult, dangerous, or expensive to reproduce in real life. This is crucial for ensuring the system's safety, reliability, and robustness.
- **Evaluation of Emergent Behaviors**: Complex systems often exhibit emergent behaviors that arise from the interactions among their components. Simulation helps uncover these behaviors and assess their impact on the system's performance and functionality.

Simulation techniques used in V&V include:

- **Scenario Analysis** involves defining specific scenarios – combinations of inputs, events, and conditions – that the system is expected to handle. The model is then simulated under these scenarios to verify it behaves as expected. Scenarios can cover nominal operations, edge cases, fault conditions, and more. Tools like Simulink (MathWorks 2021) and Modelica (Fritzson 2015) support scenario modeling and simulation. For example, in the development of an aircraft control system, engineers would define scenarios such as takeoff, landing, turbulence, and engine failure. By simulating the model under these scenarios, they can verify that the control system responds appropriately and maintains safe operation.
- **Sensitivity Analysis** studies how changes in model inputs and parameters affect the outputs and performance. It helps identify which factors have the greatest influence on system behavior and can guide design decisions and optimization efforts. Techniques like one-factor-at-a-time, design of experiments, and variance-based methods are used (Saltelli et al. 2008). For example, in the design of a wind turbine, sensitivity analysis can be used to determine how factors such as wind speed, blade angle, and generator efficiency impact the turbine's power output and structural loads. This information can then be used to optimize the turbine design for maximum efficiency and reliability.

- **Monte Carlo Methods** involve running a large number of simulations with randomly sampled inputs based on probability distributions. This approach is useful for analyzing systems with uncertain or stochastic elements and can provide statistical insights into system performance, reliability, and risk. Tools like @Risk (Palisade Corporation 2021) and Oracle Crystal Ball (Oracle Corporation 2021) support Monte Carlo simulation. For example, in the assessment of a power grid's reliability, Monte Carlo simulation can be used to model the stochastic behavior of power generation, transmission, and demand. By running thousands of simulations with different combinations of generator outages, line failures, and load profiles, engineers can estimate the probability and impact of blackouts and identify weak points in the grid that require reinforcement.

Strengths of simulation for V&V include:

- Allows for the analysis of system behavior and performance under various conditions
- Supports the exploration of what-if scenarios and sensitivity analysis
- Can handle complex, stochastic, and time-dependent systems

However, simulation also has some limitations:

- Provides only a partial coverage of the system's behavior, as it relies on selected scenarios and inputs
- The accuracy of the results depends on the fidelity of the underlying model and its assumptions
- Can be computationally expensive, especially for large-scale and detailed models

For instance, before using autonomous vehicles on real roads, engineers make use of simulation to check and confirm their performance in different traffic, weather, and road situations. Simulation helps test the car's ability to understand its environment, make decisions, and control itself in a secure and managed space, enabling issues that could affect safety or reliability to be found and fixed. By simulating numerous driving scenarios, engineers can reinforce confidence in the system's capabilities and robustness and confirm it complies with the strict safety rules of autonomous driving. Chapter 8 illustrates the importance of simulation for autonomous vehicles V&V.

5.7.3 Testing

MBT is a useful method in MBSE that consists of creating test cases from system models, providing a clear and organized way to verify and validate the system.

MBT uses models to produce test cases, test data, and expected outcomes, making it easier to automate test running and evaluation.

Testing plays a crucial role in MBSE V&V activities by:

- **Providing Evidence of System Behavior and Functionality**: Testing allows engineers to observe and validate the system's actual behavior against the expected behavior specified in the models. It helps uncover discrepancies, errors, and omissions in the system design or implementation.
- **Enabling Early Defect Detection**: By generating test cases from models and executing them early in the development process, MBT helps identify defects and inconsistencies before they propagate to later stages, where the cost of fixing them is significantly higher.
- **Improving Test Coverage and Efficiency**: MBT techniques, such as requirements coverage, structural coverage, and scenario-based testing, help ensure that all important aspects of the system are thoroughly tested. Automated test case generation and execution reduce manual effort and enable more efficient testing.

Test case generation in MBT involves various techniques:

- **Requirements Coverage**: Deriving tests to cover all requirements linked to the model elements.
- **Structural Coverage**: Generating tests to cover all elements of the model, such as states, transitions, and decisions.
- **Data Coverage**: Creating tests that cover all important data ranges, boundaries, and partitions.
- **Scenario-based**: Defining key scenarios and generating tests to cover them.
- **Random Generation**: Using randomization to generate diverse tests within defined constraints.

Model-based techniques can derive data from value ranges, constraints, and dependencies specified in the model. Data can also be generated by simulating the model and capturing its outputs.

MBT execution and analysis often involve running tests on the model itself to verify its behavior before running them on the implemented system. This model-level testing enables defects to be detected early. MBT tools typically provide automated test execution and results analysis, with features such as:

- Automated comparison of expected and actual results
- Coverage analysis and reporting
- Regression testing support
- Integration with issue-tracking and requirements management tools

Strengths of testing for V&V include:

- Provides direct evidence of the system's behavior and functionality
- Can be applied at different levels of granularity, from unit testing to system-level testing
- Allows for the validation of both functional and nonfunctional requirements
- Significantly improves test coverage, efficiency, and maintainability compared to manual testing approaches

However, testing also has some limitations:

- Provides limited coverage, as it is impractical to test all possible inputs and scenarios
- Can be time-consuming and costly, especially for large and complex systems

An illustrative example of the importance of testing in V&V is the development of an aircraft flight control system. By modeling the behavior of the flight control system, engineers can automatically create test cases to examine different scenarios, such as regular flight, extreme maneuvers, and system failures. These test cases are then run on a simulator to check that the flight control system behaves correctly and safely in different situations. Any differences between predicted behaviors and what really happens are examined and used to improve the system design. This careful testing process helps make sure that the flight control system meets its safety and functional requirements before it is implemented, reducing the risk of costly and dangerous failures.

Tools such as Conformiq Creator, Smartesting CertifyIt, Simulink Design Verifier, IBM Rational Rhapsody, and Reactis facilitate many parts of MBT, such as test case generation, test data creation, and test execution analysis.

5.7.4 Combining Techniques

In practice, model checking, simulation, and testing are often used in combination to leverage their complementary strengths and mitigate their individual limitations. For example:

- Model checking can be used to verify critical properties of a system model.
- Simulation can be used to guide the selection of test cases for physical testing, focusing on the most critical or error-prone scenarios.
- Testing can be used to validate the results obtained from model checking and simulation, ensuring that the actual system behaves as expected.

The choice of which technique(s) to use depends on factors such as the complexity of the system, the availability of resources, the desired level of assurance,

and the stage of the development process. A well-balanced approach that combines model checking, simulation, and testing can provide a comprehensive and effective V&V strategy for complex systems.

Exercise: Selecting V&V techniques for an autonomous car system

You are the lead systems engineer for a project developing an autonomous car. Your team has completed the initial design phase, and you are now planning the V&V activities. The car includes various subsystems such as perception (sensors and cameras), decision-making (AI-based control system), actuation (steering, acceleration, braking), and human-machine interface.

For each of the following aspects of the autonomous car system, suggest appropriate V&V techniques and justify your choices; consider combining techniques where appropriate:

- **Perception System Accuracy**: The ability of the car to correctly identify and classify objects in its environment (e.g. other vehicles, pedestrians, road signs).
- **Decision-making Logic**: The AI-based control system's ability to make safe and appropriate driving decisions based on perceived information and predefined rules.
- **Actuator Response Time**: The speed and accuracy with which the car's physical systems (steering, acceleration, braking) respond to commands from the decision-making system.
- **Safety Features**: The effectiveness of emergency braking, collision avoidance, and other safety-critical features.
- **Human-machine Interface**: The usability and clarity of the interface that allows human passengers to interact with and monitor the autonomous system.
- **Overall System Integration**: The seamless interaction between all subsystems to ensure safe and efficient operation of the autonomous car as a whole.

For each aspect, consider the following V&V techniques:

- Formal methods
- Simulation and modeling
- Hardware-in-the-loop testing
- Software-in-the-loop testing
- Field testing
- Static analysis
- Dynamic analysis

(Continued)

(Continued)

- Fault injection testing
- UAT

Your responses should:

1. Identify the most appropriate V&V technique(s) for each aspect.
2. Explain why you chose those technique(s).
3. Discuss any limitations of the chosen techniques and how they might be mitigated.
4. Suggest how multiple techniques could be combined for more comprehensive V&V.

5.8 Standards and Good Practices

Systems engineering has developed a rich ecosystem of standards and best practices for guiding the development of complex systems. These standards create a shared language, framework, and processes to support consistency, quality, and compatibility across projects and organizations. The Systems Engineering Body of Knowledge (SEBoK) considers these standards as key to a systematic and organized approach to systems engineering (SEBoK Editorial Board 2024).

5.8.1 Key Standards in Systems Engineering

Several standards and guidelines are particularly relevant to systems engineering and MBSE:

- **ISO/IEC/Institute of Electrical and Electronics Engineers (IEEE) 15288: Systems and Software Engineering-System Lifecycle Processes** (ISO 2015): This standard provides a comprehensive framework for describing the lifecycle of systems, including V&V activities. It defines a set of processes that span the entire lifecycle and can be tailored to specific project needs.
- **NASA Systems Engineering Handbook** (Hirshorn et al. 2017): This handbook offers guidance on V&V practices in the context of NASA's systems engineering processes. It provides detailed descriptions of best practices, tools, and techniques used in space systems engineering.
- **International Council on Systems Engineering (INCOSE) Systems Engineering Handbook** (INCOSE 2023): This handbook includes guidance

on V&V as part of the systems engineering process, with a focus on model-based approaches. It serves as a comprehensive guide for systems engineering practitioners across various industries.
- **IEEE 1012: IEEE Standard for System, Software, and Hardware V&V** (Institute of Electrical, Electronics Engineers, and IEEE-SA Standards Board 2017): This standard provides a process framework for V&V, including requirements, activities, and documentation. It offers a structured approach to ensuring that systems meet their intended purpose and stakeholder needs.

Adherence to these standards makes sure that V&V practices are steady, strict, and in line with industry standards. They provide a shared basis for systems engineering work, facilitating communication and teamwork between various organizations and fields.

5.8.2 Documentation and Traceability

Proper documentation and traceability are essential for an effective model V&V. The SEBoK emphasizes the importance of maintaining clear and comprehensive records of the V&V process (SEBoK Editorial Board 2024). Key practices include:

- **Documenting V&V Plans, Procedures, and Results**: This includes specifying V&V objectives, methods, acceptance criteria, and reporting formats (Hirshorn et al. 2017). Clear documentation ensures that the V&V process is transparent, repeatable, and auditable.
- **Maintaining Traceability Between Requirements, Model Elements, and V&V Artifacts**: This helps ensure coverage and facilitates impact analysis when requirements or models change (Gotel and Finkelstein 1994). Traceability is crucial for managing complexity and ensuring that all system requirements are adequately verified and validated.
- **Using Tools and Methods that Support Automatic Generation of V&V Documentation and Traceability Matrices from Models**: Tools such as SysML diagrams and tables can be leveraged to automate the creation and maintenance of documentation and traceability information (Friedenthal et al. 2014). This automation helps reduce errors and improves the efficiency of the V&V process.

Good documentation and tracking methods contribute to the transparency, reproducibility, and maintenance of the V&V process. They provide a straightforward record of choices made, tests done, and results obtained, which is very valuable for project management, regulatory compliance, and updating systems in the future.

5.8.3 Automation and Tool Integration

Automating V&V activities and integrating V&V tools into the MBSE workflow can significantly improve efficiency and consistency. The SEBoK recommends leveraging automation and tool integration to enhance the V&V process (SEBoK Editorial Board 2024). Recommended practices include:

- **Using Model Checking Tools to Automatically Verify Model Properties and Consistency**: Tools like UPPAAL for timed automata models (Behrmann et al. 2004) or Alloy for structural constraints (Jackson 2019) can be used to automatically verify certain properties of the system model, reducing the potential for human error, and increasing the thoroughness of the verification process.
- **Integrating Simulation Tools with the Modeling Environment to Enable Automated Model Execution and Analysis**: For example, integrating Simulink with MATLAB (MathWorks 2021) allows for seamless simulation and analysis of system models, providing valuable insights into system behavior and performance.
- **Leveraging CI/CD Pipelines to Automatically Trigger V&V Activities When Models are Updated**: This ensures continuous model quality and provides rapid feedback on changes (Duvall et al. 2007). By integrating V&V activities into the development workflow, issues can be identified and addressed more quickly, reducing the cost and effort of later stages.

Automation and tool integration help to scale V&V efforts, reduce manual errors, and provide rapid feedback on model quality. They enable more frequent and comprehensive V&V activities, improving the overall quality and reliability of the system design.

5.8.4 Iterative and Continuous V&V

Rather than considering V&V as a one-time, end-of-phase activity, it should be carried out iteratively and continuously throughout the system lifecycle, as advocated by the SEBoK Editorial Board (2024). This approach:

- **Enables Early Detection and Correction of Defects**: By identifying issues early in the development process, the cost and effort of fixes can be significantly reduced.
- **Provides Frequent Feedback on Model Quality and Progress Toward V&V Objectives**: This allows for timely adjustments to the system design and V&V strategy.

- **Allows V&V Activities to Adapt to Evolving Requirements, Designs, and Risks**: As the system design evolves and new information becomes available, the V&V approach can be adjusted to ensure continued effectiveness.

Using V&V in agile and iterative development processes, such as Scrum or Spiral, can help make continuous V&V practices a regular part of the workflow (Larman and Basili 2003). This method makes V&V a key part of the development cycle instead of just a stage at the end of the project.

By creating clear standards, maintaining good documentation and traceability, using automation and tool integration, and adopting iterative and continuous V&V practices, organizations can establish a solid base for effective model V&V in their MBSE processes. This ensures that models are of high quality, meet requirements, and provide a trustworthy foundation for system development and decision-making.

5.9 Closing Design Space Exploration

In this section, we look at V&V and its importance in making sure a system meets its goals and functions properly. A key element is closing the design space exploration, which means reducing the number of design alternatives to find the best ones.

The theoretical models mentioned in Chapter 4 use different expressive languages, as shown in Chapter 3. The choice of these languages is based on specific aims such as documentation, formal verification, testing, or simulation. Each language is based on a logic that fits these objectives.

Figure 5.1, which builds upon Figure 3.5, shows the complex relationship between models, languages, and V&V purposes. This figure highlights the fact that closing the design space requires a robust method that clearly states objectives at

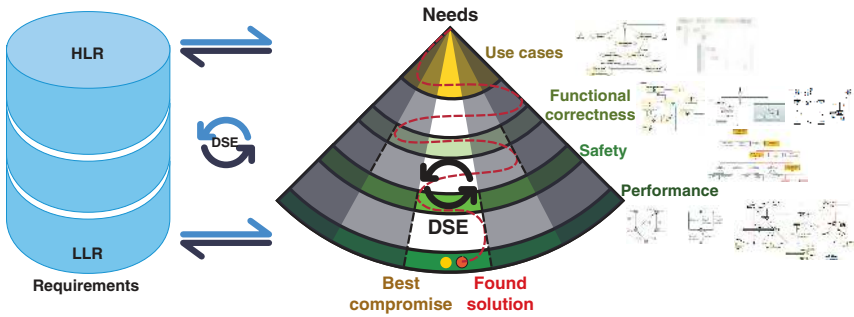


Figure 5.1 Design space exploration closure through V&V.

each phase of the development, which may include functional correctness, operational safety, performance improvement, meeting deadlines, or cost-effectiveness.

The selection of the right modeling language is essential and must reflect the levels of V&V representation required, whether for formal techniques, simulation, or testing. Yet, this choice also brings a big challenge; it's important to maintain clear connections between different languages given their assumptions in both composition and refinement situations.

In refinement, the key issue is to show that adding more details to a representation does not change the consistency of previous representations. For instance, when transitioning from a basic Petri Net to a hierarchical Petri Net, it is important to maintain the properties that have already been established.

Composition has its own difficulties. The choice of models depends on the type of different subsystems or components that form the overall system. For example, if a system includes both continuous and discrete components, we have to pick suitable models for each subsystem (such as a continuous model and a discrete model) and also grasp the behaviors that may arise from their interactions, as explained in Chapter 4.

This approach for limiting design space exploration is in line with the V&V techniques presented earlier. It highlights the need to select suitable modeling paradigms and languages to support effective V&V activities. By paying close attention to the interactions between models, languages, and V&V goals, we can better navigate the complex system design landscape and ensure our final solution meets all necessary requirements and constraints. The chosen method must be adaptable enough to allow smooth transitions between various levels of abstraction, enabling thorough exploration of the design space.

While it may seem appealing to use a single framework, such as Unified Modeling Language or SysML, hoping it will solve model integration issues, this idea is flawed in several ways. First, the use of these languages often limits design space exploration because of their great expressiveness. Second, the unification they provide is mainly syntactic, not semantic. Indeed, many semantic variation points exist within these unifying frameworks, and currently, there is no theoretical framework that can combine all modeling paradigms effectively. As a result, the search for a genuinely unified modeling approach continues to be a challenge in systems engineering.

5.10 Reducing Costs and Delays

Early and continuous V&V in the system development lifecycle is very important. As shown in Figure 5.2, taken from the NASA Systems Engineering Handbook (Hirshorn et al. 2017), the expense of resolving issues increases significantly as

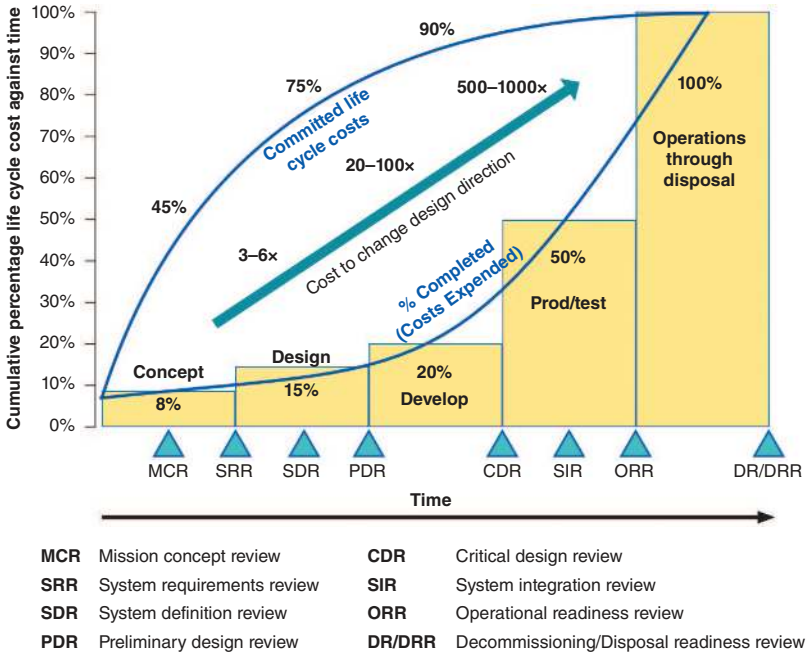


Figure 5.2 Costs of engineering. *Source*: https://www.researchgate.net/figure/Product-life-cycle-costs-at-each-design-stage-7_fig1_383144695.

the project moves through the various phases of its lifecycle. This increase in costs highlights the essential requirement for sound V&V practices, especially those using model-based methods.

5.10.1 The Cost Escalation Curve

The figure presents a compelling visualization of how costs associated with resolving problems or implementing changes escalate as a project moves through its lifecycle:

1. **Early Stages**: During the concept and design phases, the cost to implement changes or fix issues is relatively low, typically ranging from 1× to 6× the initial cost.
2. **Mid-stages**: As the project moves into development and production/test phases, the cost multiplier increases significantly, ranging from 20× to 100×.
3. **Late Stages**: In the operations phase, the cost to address issues can skyrocket to 500–1000× the original cost.

This exponential increase in costs highlights the importance of identifying and addressing issues as early as possible in the system lifecycle.

5.10.2 Leveraging MBSE for Early V&V

MBSE provides powerful tools and techniques for conducting early V&V activities, thereby helping to mitigate the cost-escalation risk:

1. **Requirements Validation**: MBSE allows for the creation of formal, executable models of system requirements. These models can be analyzed and simulated early in the lifecycle to validate their completeness, consistency, and feasibility.
2. **Design Verification**: System models created in MBSE tools can be used to verify design decisions against requirements long before physical prototypes are built. This enables early detection of design flaws or inconsistencies.
3. **Behavioral Analysis**: Executable models of system behavior can be created and analyzed to verify system responses under various conditions, identifying potential issues in system logic or performance.
4. **Interface Verification**: MBSE facilitates the creation of detailed interface models, allowing for early verification of component interactions and data flows.
5. **Trade-off Analysis**: Models can be used to explore different design alternatives and their impacts on system performance, cost, and schedule, enabling informed decision-making early in the project.

5.10.3 Continuous V&V Throughout the Lifecycle

While early V&V is crucial, the MBSE approach also supports continuous V&V throughout the system lifecycle:

1. **Model Evolution**: As the system design matures, models can be refined and expanded, allowing for continuous verification against evolving requirements.
2. **Regression Testing**: Changes to the system can be quickly verified against the entire model, ensuring that modifications don't introduce unintended consequences.
3. **Virtual Integration**: Before physical integration, components can be virtually integrated and tested in the model environment, identifying potential integration issues early.
4. **Test Case Generation**: Models can be used to automatically generate comprehensive test cases, ensuring thorough coverage of system behaviors and scenarios.

Using MBSE for early and continuous V&V helps organizations to reduce the risks of costly changes and subsequent delays. This approach is in line with the "shift-left" idea in systems engineering, which means that testing and validation tasks take place earlier in development.

The potential for cost savings is large. By finding and fixing problems when costs are still low – between 1× and 6× – rather than when they reach 500–1000×, organizations can gain significant advantages in cost savings and quicker market releases.

In summary, looking at the NASA cost escalation curve strongly justifies the benefits of MBSE V&V practices. By investing in effective modeling and early V&V efforts, organizations can greatly reduce total costs and delays and improve the quality and reliability of their systems.

5.11 Conclusion

In this chapter, we have examined the main concepts of V&V in MBSE. We discussed the essence of V&V, its main stages, and the problems it faces. We also covered different V&V methods, such as model checking, simulation, and testing, as well as the standards and best practices that guide these processes.

The role of V&V in making sure that systems are high-quality, reliable, and effective is very important. As highlighted, V&V tasks are essential for finding and fixing problems early in the development process, thereby reducing costs and improving overall system performance. Combining V&V with MBSE practices reinforces these advantages, offering a more comprehensive and organized way of developing systems.

Before moving on to the next chapter on "System Architecting Principles," it is essential to understand how V&V is related to system architecture. A system's architecture greatly influences how easily it can be verified and validated. Good architecture facilitates and speeds up V&V tasks, while poor architecture complicates and slows them down.

By learning about both V&V and system architecting principles, systems engineers and architects can collaborate to develop robust, reliable, and high-quality systems that meet stakeholder requirements and can cope with real-world challenges. This integrated approach, combining effective V&V methods with sound architectural techniques, is crucial for successful systems engineering in today's complex technological environment.

Questions

1. What is the primary purpose of verification in the context of MBSE?
   - **a** To ensure the system meets stakeholder needs
   - **b** To check if the system is built according to specifications
   - **c** To validate the system's performance in its operational environment
   - **d** To identify new system requirements
2. Which of the following best describes Validation in MBSE?
   - **a** Ensuring the system is built correctly
   - **b** Checking if the system meets its intended purpose
   - **c** Verifying the accuracy of system models
   - **d** Testing individual system components
3. What is a key difference between Verification and Validation?
   - **a** Verification is performed after validation.
   - **b** Validation focuses on "building the right system" while verification focuses on "building the system right."
   - **c** Verification is only concerned with software components.
   - **d** Validation is only performed at the end of the system lifecycle.
4. Which V&V technique involves analyzing the system's behavior without executing it?
   - **a** Dynamic testing
   - **b** Static analysis
   - **c** Simulation
   - **d** Prototype testing
5. What is the purpose of formal methods in V&V?
   - **a** To replace all other V&V techniques
   - **b** To provide mathematical proof of system correctness
   - **c** To simulate system behavior
   - **d** To generate test cases automatically
6. Which of the following is a benefit of MBSE on V&V activities?
   - **a** Elimination of all system errors
   - **b** Reduced need for testing
   - **c** Improved traceability between requirements and V&V activities
   - **d** Automatic generation of all test cases
7. What is a key challenge in V&V for complex systems?
   - **a** The need for mathematical expertise
   - **b** The difficulty in verifying emergent behaviors
   - **c** The high cost of simulation tools
   - **d** The lack of standardized V&V processes

8 Which V&V approach involves incrementally building and testing the system?
   a Big bang testing
   b Waterfall model
   c Incremental V&V
   d Formal verification
9 What role do standards play in V&V activities?
   a They replace the need for V&V activities.
   b They provide guidelines and best practices for V&V.
   c They automate V&V processes.
   d They define system requirements.
10 How does MBSE contribute to continuous V&V throughout the system lifecycle?
   a By eliminating the need for V&V after system deployment
   b By enabling early V&V activities through model analysis
   c By replacing physical testing with virtual testing
   d By automating all V&V activities

Answers

1-b; 2-b; 3-b; 4-b; 5-b; 6-c; 7-b; 8-c; 9-b; 10-b

References

Abrial, J.R. *The B-book: Assigning Programs to Meanings*. Cambridge University Press, 1996. https://books.google.fr/books?id=aVKgAQAACAAJ.

Arora, S. and Barak, B. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009. ISBN 9780521424264. https://books.google.fr/books?id=8Wjqvsoo48MC.

Baier, C., Katoen, J.P., and Larsen, K.G. *Principles of Model Checking*. The MIT Press. MIT Press, 2008. ISBN 9780262304030. https://books.google.fr/books?id=5dvxCwAAQBAJ.

Banks, J. *Discrete-event System Simulation*. Prentice Hall International Series in Industrial and Systems Engineering. Pearson Prentice Hall, 2005. ISBN 9780131446793. https://books.google.fr/books?id=CWZRAAAAMAAJ.

Behrmann, G., David, A., and Larsen, K.G. A tutorial on Uppaal. *Formal Methods for the Design of Real-Time Systems* 2004:200–236.

Bertot, Y., Huet, G., Castéran, P., et al. *Interactive Theorem Proving and Program Development: Coq'Art: The Calculus of Inductive Constructions*. Texts in Theoretical

Computer Science. An EATCS Series. Springer Berlin Heidelberg, 2013. ISBN 9783662079645. https://books.google.fr/books?id=FeklBQAAQBAJ.

Boehm, B.W. Verifying and validating software requirements and design specifications. *IEEE Software* 1984:1(1):75.

Bondy, J.A. and Murty, U.S.R. *Graph Theory with Applications*. American Elsevier Publishing Company, 1976. ISBN 9780444194510. https://books.google.fr/books?id=4bwrAAAAYAAJ.

Broy, M. A logical basis for component-oriented software and systems engineering. *The Computer Journal* 2010:53(10):1758–1782.

Chung, L., Nixon, B.A., Yu, E., et al. *Non-Functional Requirements in Software Engineering*. International Series in Software Engineering. Springer US, 2012. ISBN 9781461552697. https://books.google.fr/books?id=MNrcBwAAQBAJ.

Cimatti, A., Clarke, E., Giunchiglia, E., et al. NuSMV 2: an opensource tool for symbolic model checking. In *Computer Aided Verification: 14th International Conference, CAV 2002* Copenhagen, Denmark, July 27–31, *2002*, *Proceedings 14*, 2002:359–364. Springer.

Clarke, E.M., Grumberg, O., Kroening, D., et al. *Model Checking*, 2e. Cyber Physical Systems Series. MIT Press, 2018. ISBN 9780262038836. https://books.google.fr/books?id=OJV5DwAAQBAJ.

Cleland-Huang, J., Gotel, O., and Zisman, A. *Software and Systems Traceability* 2012:2. Springer.

De Moura, L. and Bjørner, N. Z3: an efficient SMT solver. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 2008:337–340. Springer.

Duvall, P.M., Matyas, S., and Glover, A. *Continuous Integration: Improving Software Quality and Reducing Risk*. Addison–Wesley Signature Series. Pearson Education, 2007. ISBN 9780321630148. https://books.google.fr/books?id=PV9qfEdv9L0C.

Engel, A. *Verification, Validation, and Testing of Engineered Systems*. Wiley Series in Systems Engineering and Management. Wiley, 2010. ISBN 9780470527511. https://books.google.fr/books?id=H6N2CgAAQBAJ.

Friedenthal, S., Moore, A., and Steiner, R. *A Practical Guide to SysML: The Systems Modeling Language*. The MK/OMG Press. Elsevier Science, 2014. ISBN 9780128002025. https://books.google.fr/books?id=97XfoAEACAAJ.

Fritzson, P. *Principles of Object-Oriented Modeling and Simulation with Modelica 3.3: A Cyber-Physical Approach*. Wiley, 2015. ISBN 9781118859162. https://books.google.fr/books?id=wgIaBgAAQBAJ.

Gilb, T., Graham, D., and Finzi, S. *Software Inspection*. Software engineering. Addison–Wesley, 1993. ISBN 9780201631814. https://books.google.fr/books?id=nM5QAAAAMAAJ.

Glinz, M. On non-functional requirements. In *15th IEEE International Requirements Engineering Conference (RE 2007)* 2007:21–26. IEEE.

Gotel, O.C. and Finkelstein, C.W. An analysis of the requirements traceability problem. In *Proceedings of IEEE International Conference on Requirements Engineering* 1994:94–101. IEEE.

Hirshorn, S.R., Voss, L.D., and Bromley, L.K. *NASA Systems Engineering Handbook*. Technical report 2017.

Holzmann, G.J. The model checker spin. *IEEE Transactions on Software Engineering* 1997:23(5):279–295.

Hopcroft, J.E., Motwani, R., Ullman, J.D. *Introduction to Automata Theory, Languages, and Computation.* Introduction to Automata Theory, Languages, and Computation.2006. Pearson/Addison Wesley. ISBN: 9780321455369.

INCOSE. *INCOSE Systems Engineering Handbook*. Wiley, 2023. ISBN 9781119814290. https://books.google.fr/books?id=HhjBEAAAQBAJ.

Institute of Electrical, Electronics Engineers, and IEEE-SA Standards Board. *IEEE Standard for System, Software and Hardware Verification and Validation*. IEEE, 2017. https://books.google.fr/books?id=VYN_zQEACAAJ.

ISO. ISO/IEC/IEEE 15288: 2015. Systems and software engineering—content of systems and software life cycle process information products (Documentation), International Organization for Standardization/International Electrotechnical Commission: Geneva, Switzerland, 2015.

Jackson, D. Alloy: a language and tool for relational models. *IEEE Transactions on Software Engineering* 2019:45(12):1295–1307.

Königs, S.F., Beier, G., and Figge, A. Traceability in systems engineering—review of industrial practices, state-of-the-art technologies and new research solutions *Advanced Engineering Informatics* 2012:26(4):924–940.

Larman, C. and Basili, V.R. Iterative and incremental developments. A brief history. *Computer* 2003:36(6):47–56.

Law, A.M. *Simulation Modeling and Analysis*, 6e. McGraw-Hill series in industrial engineering and management science. McGraw-Hill Education, 2024. ISBN 9781264268245. https://books.google.fr/books?id=cXu6zwEACAAJ.

Marijan, D., Gotlieb, A., and Sen, S. Test case prioritization for continuous regression testing: an industrial case study. In *2013 IEEE International Conference on Software Maintenance* 2013:540–543. IEEE.

MathWorks. Simulink—simulation and model-based design.2021. https://www.mathworks.com/products/simulink.html.

Morgan, P., Hambling, B., Samaroo, A., et al. *Software Testing: An ISTQB-BCS Certified Tester Foundation Guide, 4th edition.* BCS, The Chartered Institute for IT. 2019. ISBN: 9781780174921.

Nipkow, T., Paulson, L.C., and Wenzel, M. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2003. ISBN 9783540459491. https://books.google.fr/books?id=xwdqCQAAQBAJ.

Oracle Corporation. Oracle crystal ball. 2021. https://www.oracle.com/middleware/technologies/crystalball.html.

Owre, S., Rushby, J.M., and Shankar, N. PVS: A prototype verification system. In *International Conference on Automated Deduction* 1992748–752. Springer.

Palisade Corporation. @Risk: risk analysis and simulation add-in for Microsoft Excel. 2021. https://lumivero.com/products/at-risk/.

Prabhaharan, S.A., Balaji, G., Annamalai, K. et al. Numerical simulation of crashworthiness parameters for design optimization of an automotive crash-box. *International Journal for Simulation and Multidisciplinary Design Optimization* 2022.

Reid, S. Software and systems engineering software testing part 1: concepts and definitions. Technical Report, ISO/IEC/IEEE 29119-1, Technical Report, 2013.

Robinson, S. Simulation model verification and validation: increasing the users' confidence. In *Proceedings of the 29th Conference on Winter Simulation* 1997:53–59.

Saltelli, A., Ratto, M., Andres, T., et al. *Global Sensitivity Analysis: The Primer*. Wiley, 2008. ISBN 9780470725177. https://books.google.fr/books?id=wAssmt2vumgC.

Sargent. G. Verification and validation of simulation models. In *Proceedings of the 2010 Winter Simulation Conference* 2010:166–183. IEEE.

Schluse, M., Priggemeyer, M., Atorf, L., et al. Experimentable digital twins—streamlining simulation-based systems engineering for industry 4.0. *IEEE Transactions on Industrial Informatics* 2018:14(4):1722–1731.

SEBoK Editorial Board. The Guide to the Systems Engineering Body of Knowledge (SEBoK). INCOSE, 2024.

Selic, B. The pragmatics of model-driven development. *IEEE Software* 2003:20(5):19–25.

Spivey, J.M. *The Z Notation: A Reference Manual*. Prentice Hall International Series in Computer Science. Prentice Hall, 1992. ISBN 9780139785290. https://books.google.fr/books?id=wa1QAAAAMAAJ.

Stecklein, J.M., Dabney, J., Dick, B., et al. Error cost escalation through the project life cycle. In *14th Annual International Symposium* 2004. Number JSC-CN-8435.

Utting, M., Pretschner, A., and Legeard, B. A taxonomy of model-based testing approaches. *Software Testing, Verification and Reliability* 2012:22(5):297–312.

Van Der Straeten, R., Mens, T., and Van Baelen, S. Challenges in model-driven software engineering. In *International Conference on Model Driven Engineering Languages and Systems* 2008:35–47. Springer.

Wiegers, K.E. *Peer Reviews in Software: A Practical Guide*. Addison–Wesley information technology series. Addison–Wesley, 2002. ISBN 9780201734850. https://books.google.fr/books?id=d7BQAAAAMAAJ.

Woodcock, J., Larsen, P.G., Bicarregui, J., et al. Formal methods: practice and experience. *ACM Computing Surveys (CSUR)* 2009:41(4):1–36.

Wymore, A.W. *Model-Based Systems Engineering*. Systems Engineering. CRC Press, 2018. ISBN 9781351431088. https://books.google.fr/books?id=00taDwAAQBAJ.

6

System Architecting Principles

6.1 Fundamental Principles

System architecture is important for the design and development of complex systems. It provides a blueprint for system structure, behavior, and properties. This framework helps in making decisions throughout the system's lifecycle, ensuring that the final outcome is fit for its purpose and provides value to those involved (Maier 1998). This section examines the main concepts underlying good system architecture design, highlighting key elements such as separation of concerns, modularity, abstraction, and scalability.

6.1.1 Managing Complexity

Basically, system architecture is about managing complexity. Modern systems are complex, containing many connected parts with dynamic interactions, which can be quite challenging for engineers and designers. System architecture helps address this complexity by breaking the system into smaller, more manageable parts and defining how they relate and interact with each other (Baldwin and Clark 2000).

6.1.2 Separation of Concerns

The separation of concerns is a key principle in effective system architecture design. It suggests breaking the system into different parts, each focusing on a specific issue or area, such as functionality, performance, or security. This facilitates the development, testing, and maintenance of individual parts, which reduces the risk of unintended problems and allows for easier adjustments to changing needs.

6.1.3 Modularity and Encapsulation

Modularity is another key principle that encourages organizing a system into separate, loosely connected modules with clear interfaces (Parnas 1972). This structure improves flexibility, reusability, and ease of maintenance, since changes in one module do not greatly affect others.

Encapsulation, related to modularity, boosts scalability by hiding a module's inner workings, reducing dependencies, and facilitating changes (Parnas 1972).

The Phonebloks project

Phonebloks was a modular smartphone idea launched by Dave Hakkens in 2013. It aimed to tackle electronic waste and encourage eco-friendly consumer electronics. The main concept was to build a smartphone with parts that could be exchanged or upgraded.



Phone modularization. *Source*: Dave Hakkens / Wikimedia Commons / CC BY-SA 3.0.

This project beautifully illustrated the benefits of modularization and encapsulation. The smartphone was conceptualized as a grid of standardized "bloks" – discrete, independent modules representing different functional components such as:

- Processing unit
- Battery
- Camera

- Memory
- Communication modules

The advantages of this modularization are numerous:

- Easy component replacement
- Reduced electronic waste
- Lower repair costs
- Customizable user experience
- Extended device lifespan

Each module of this phone was connected through a standardized backplane, allowing:

- Hot-swappable components
- Independent upgradeability
- Simplified maintenance
- Reduced manufacturing complexity

The project did not reach complete market use, but it significantly impacted industry views on product design, showcasing how modularity can improve sustainability and user experience in consumer electronics.

Moreover, the Phonebloks modular configuration changed the way we perform verification and validation (V&V) of phone designs. It allowed for the testing of single components, facilitating performance and troubleshooting. Thanks to the use of standardized interfaces, validation processes have become more focused and simpler, reducing testing complexity. This approach supports quick technological updates and thorough system evaluations. By modifying the usual V&V methods, the modular design makes fault-finding easier, allows for small improvements, and enables more flexible quality control methods.

In the end, Phonebloks represented a forward-thinking approach to product design, questioning the usual one-piece electronic device structures.

6.1.4 Abstraction and Scalability

Abstraction enables architects to focus on the important properties and behaviors of the system, without getting bogged down by unnecessary details (Shaw and Garlan 1996). By setting clear abstractions and interfaces, architects can better manage complexity, support interoperability, and allow the system to evolve and change over time.

Scalability is an important factor in system architecture, especially for systems that must evolve and meet higher demands (Bondi 2000). System architects need

to design scalable systems, which means adding new features or enhancing existing ones without having to make major changes to the whole system. Loose connections and clear interfaces help with scalability because they allow parts of the system to be added, removed, or switched without major effects on other parts (Fielding 2000).

6.1.5 The Interplay of Principles

The concepts of separation of concerns, modularity, abstraction, and scalability are related and support each other. By consistently applying these ideas across the system architecture, architects can build systems that are clearer, easier to maintain, and better able to meet changing requirements (Bass et al. 2012). However, it's crucial to understand that these principles can sometimes conflict and need careful balancing based on the system's specific objectives and constraints (Maier 1998).

6.1.6 Beyond Technical Considerations

Good system architecture goes beyond technical details; it also requires teamwork with stakeholders and a good grasp of the issues at hand. Architects should work closely with users, clients, and others to gather and prioritize requirements, making sure the system fits their expectations. This demands good communication and interpersonal skills, along with the ability to explain complex technical ideas in ways that nontechnical stakeholders can understand (Maier 1998).

6.1.7 Summary

The fundamental principles of system architecting – separation of concerns, modularity, abstraction, and scalability – provide a solid foundation for the development of complex systems. By applying these principles and collaborating with stakeholders, architects can design systems that are more reliable, flexible, and suited to users' needs. As systems become increasingly complex, a clear understanding and application of these principles will be essential for future system development success.

6.2 Types of System Architectures

System architecture is like a blueprint for how a system's parts fit together, how they work with each other, and the principles that shape their design and future (Bass et al. 2012). The Systems Engineering Body of Knowledge (SEBoK) shows the different types of architectural styles that have emerged to meet the various needs and problems of today's systems, each having its own pros and cons in terms

of scalability, maintainability, performance, and other characteristics (SEBok Editorial Board 2024). This section delves into some common system architectures and what they mean for systems engineering.

6.2.1 Monolithic Architectures

Monolithic architectures are the usual way to design systems, where the whole system is made as a complete unit with components that are tightly connected (Newman 2021). All functionality is included in a single executable or deployable package, making development, testing, and deployment easier, especially for small systems with clear and stable requirements (SEBoK Editorial Board 2024).

However, as systems grow in size and complexity, monolithic architectures encounter challenges:

- **Scalability Limitations**: Scaling individual components independently becomes increasingly difficult, hindering adaptability to growing demands.
- **Maintainability Challenges**: Tight coupling between components makes modifications and updates complex, as changes can ripple through the entire system.
- **Technology Lock-in**: Reliance on a single technology stack can impede the adoption of new technologies or frameworks without substantial rework.

The SEBoK underscores the importance of considering these limitations when selecting an architecture, especially for complex systems with evolving requirements (SEBoK Editorial Board 2024).

6.2.2 Layered Architectures

Layered architectures provide a modular approach, organizing the system into distinct layers, each with its own functions and levels of detail (Buschmann et al. 2007). This approach corresponds to the SEBoK emphasis on system decomposition and separation of concerns, splitting complex systems into smaller parts with clear interfaces (SEBoK Editorial Board 2024).

A typical layered architecture comprises:

- **Presentation Layer**: Handles user interface and interaction.
- **Application Layer**: Contains business logic and coordinates system functionality.
- **Domain Layer**: Represents core business concepts and rules.
- **Infrastructure Layer**: Provides low-level technical capabilities, such as data persistence and communication.

Layered architectures facilitate the separation of concerns, allowing developers to focus on each layer for development and maintenance and allowing changes to be made without affecting the whole system (Fowler 2012).

But, as mentioned in the SEBoK, these architectures may lead to performance issues due to the need for cross-layer communication, and can become rigid when handling concerns that span multiple layers (Evans 2004; SEBoK Editorial Board 2024).

6.2.3 Service-oriented Architectures

Service-oriented Architecture (SOA) organizes the system as a set of loosely connected, independent services (Erl 2016). Each service focuses on a specific business function and interacts with others using clear interfaces and protocols. The SEBoK considers SOA as a means of achieving greater flexibility and adaptability in system development, supporting the concepts of service orientation and loose coupling (SEBoK Editorial Board 2024).

Key principles of SOA include:

- **Service Autonomy**: Services are self-contained units with clear boundaries and contracts, enabling independent development, deployment, and evolution.
- **Loose Coupling**: Services interact through standardized interfaces and protocols, minimizing dependencies and facilitating independent changes.
- **Service Composition**: Services can be combined and orchestrated to create complex business processes and adapt to changing requirements.

SOA contributes to creating flexible, reusable, and agile systems, making them easier to adapt to changing needs and technologies (Papazoglou 2003). But the SEBoK highlights issues such as the increased complexity of service management, governance, and security (Josuttis 2007). It's then important to design services with the right level of detail to reap the benefits of SOA while reducing risks.

6.2.4 Microservices Architectures

Microservices architecture builds on SOA by providing more detailed service orientation (Newman 2021). It breaks the system down into small, independently deployable services, each focusing on one business capability. The SEBoK discusses microservices for greater scalability and agility, maintaining SOA principles while highlighting the need for modular design and independent deployment.

Key characteristics of microservices architectures include:

- **Small, Focused Service**s: Each service addresses a specific business capability, enabling independent development, deployment, and scaling.
- **Lightweight Communication**: Services interact through lightweight protocols (e.g. HTTP/REST, messaging), minimizing dependencies and facilitating independent evolution.
- **Technology Diversity**: Each service can use its own data storage and technology stack, offering flexibility and adaptability.
- **Business-oriented Organization**: Services are organized around business capabilities, promoting alignment with business goals.

Microservices offer advantages such as better modular design, scalability, and the use of various technologies, allowing for separate development, deployment, and scaling of services, promoting flexibility, and reducing the impact of failures (Lewis and Fowler 2014). But the SEBoK also highlights problems such as the added difficulty of finding services, communication, and data consistency, showing that careful planning and management are needed for good system performance (Nadareishvili et al. 2016).

6.2.5 Event-driven Architectures

Event-driven Architecture (EDA) focuses on the creation, retrieval, and use of events to facilitate communication and interaction among loosely connected parts (Michelson 2006). The SEBoK looks at EDA to improve the responsiveness and scalability of systems that require asynchronous and distributed processing.

Key concepts in EDA include:

- **Events**: Significant occurrences or state changes triggering actions or responses.
- **Event Producers**: Components generating events based on internal state or external stimuli.
- **Event Consumers**: Components reacting to events and performing actions based on event data.
- **Event Channels**: Mechanisms for transmitting events between producers and consumers (e.g. message queues, publish-subscribe systems).

EDA allows for loose connections, helping systems grow, scale, and respond well through asynchronous event processing. Nevertheless, the SEBoK highlights issues such as ensuring events are processed in the right order, dealing with failures, and maintaining data consistency, underscoring the need for careful design and configuration (Hohpe and Woolf 2012).

6.2.6 Choosing the Right Architecture

Choosing the right system architecture is important since this choice has a considerable effect on the success of system development. Every architectural style has its own trade-offs, so a detailed look at the system's needs, constraints, and objectives is necessary. The SEBoK offers useful advice for making this choice, stressing the need to think about different aspects such as scalability, maintainability, performance, development speed, and the requirement for flexibility.

Choosing an architecture isn't a simple one-size-fits-all solution; it requires a detailed understanding of how different architectural features interact and how they affect the system's overall design and function. To facilitate this understanding, let's examine some of the key dimensions that architects must consider when evaluating different architectural approaches:

- **Scalability**: The ability of the architecture to accommodate growth in system size, complexity, or workload. A scalable architecture can handle increasing demands without requiring significant redesign or re-engineering.
- **Modularity**: The degree to which the system can be divided into independent, interchangeable components. A modular architecture promotes flexibility, reusability, and maintainability, as changes in one component have minimal impact on others.
- **Performance**: The efficiency and responsiveness of the system under various conditions. A high-performance architecture delivers fast response times, high throughput, and efficient resource usage.
- **Maintainability**: The ease with which the system can be modified, updated, or repaired. A maintainable architecture facilitates changes and enhancements without introducing unintended side effects or compromising system stability.
- **Flexibility**: The ability to adapt to changing requirements or environments. A flexible architecture can accommodate new features, technologies, or operational contexts without requiring major rework.
- **Complexity**: The overall intricacy of the system design and implementation. While some level of complexity is inevitable in modern systems, architects strive to manage complexity through techniques such as abstraction, modularity, and encapsulation.
- **Interoperability**: The ease of integration with other systems or components. An interoperable architecture enables seamless communication and data exchange with external systems, promoting collaboration and extensibility.

To guide the selection of an appropriate architecture, consider the following decision framework:

- Assess System Requirements:
  - Identify key functional and nonfunctional requirements.

  - Determine expected system scale and growth projections.
  - Consider performance and latency constraints.
- Evaluate Organizational Factors:
  - Assess team size, skills, and experience.
  - Consider development and deployment processes.
  - Evaluate budget and resource constraints.
- Analyze Technical Constraints:
  - Identify integration requirements with existing systems.
  - Consider technology stack and platform limitations.
  - Evaluate security and compliance requirements.
- Prioritize Architectural Qualities:
  - Rank the importance of scalability, modularity, performance, etc.
  - Consider long-term maintainability and evolution needs.
  - Evaluate the need for flexibility and adaptability.
- Consider Trade-offs:
  - Use the comparative table to assess strengths and weaknesses of different architectures.
  - Evaluate how each architecture aligns with prioritized qualities.
  - Consider hybrid approaches that combine elements of multiple architectures.
- Prototype and Validate:
  - Develop proof-of-concept implementations for top candidates.
  - Conduct performance testing and scalability analysis.
  - Gather feedback from stakeholders and development teams.
- Make an Informed Decision:
  - Select the architecture that best balances requirements and constraints.
  - Document the rationale for the chosen architecture.
  - Plan for potential challenges and mitigation strategies.

By examining these points and using the comparisons made, architects can pick the right architectural approach for their system work. It is important to note that picking an architecture is not a one-off job. It needs regular checks and modifications as the system develops and improves.

Spotify's transformation: The microservices advantage

The music streaming service Spotify embarked on a major change between 2013 and 2014, moving its backend from a monolithic architecture to one based on microservices. This migration was prompted by the limitations of their old architecture and the need to cope with rapid growth and new ideas.

(Continued)

(Continued)

Before this change, Spotify's backend was a large Java application set up on around 100 servers. This monolithic structure presented several significant challenges:

- **Risky and Time-consuming Deployments**: The sheer size of the monolith made deployments long and fraught with risk, hindering the company's ability to release new features and improvements quickly.
- **Extended Build Times**: Compiling and building the entire application was a time-consuming process, slowing down development cycles and impeding agility.
- **Scalability Issues**: As Spotify's user base grew exponentially, the monolithic architecture struggled to scale efficiently to meet increasing demands, potentially impacting user experience and service availability.
- **Tight Coupling**: The interconnected nature of the monolith made it difficult and slow to implement changes or add new features, as modifications in one part of the system could have ripple effects throughout the entire application.

To address these challenges, Spotify made the strategic decision to transition to a microservices-based architecture. This involved gradually decomposing the monolith into smaller, autonomous services, each designed to be:

- **Independently Developed**: Each service could be developed, tested, and maintained by a dedicated team, promoting agility and ownership.
- **Separately Deployed**: Services could be deployed independently, enabling faster and more frequent releases without impacting the entire system.
- **Individually Scalable**: Each service could be scaled horizontally to handle increased load, ensuring optimal resource usage and performance.
- **Managed by a Dedicated Team**: Each service was owned and managed by a small, cross-functional team, fostering accountability and expertise.

The migration to microservices yielded significant benefits for Spotify:

- **Accelerated Deployment Frequency**: Spotify went from deploying once per week to multiple times daily, dramatically increasing their ability to ship new features and improvements, enabling them to respond quickly to user feedback and market trends.
- **Enhanced Scalability and Resilience**: The microservices architecture allowed Spotify to scale specific components as needed, improving overall system resilience and ensuring a seamless user experience even during peak usage periods.

- **Reduced Build Times**: Smaller, focused services resulted in much faster build times, enabling more rapid development cycles and accelerating innovation.
- **Increased Team Autonomy**: Teams gained end-to-end control over their services, fostering a sense of ownership, encouraging innovation, and promoting accountability.

To manage its growing microservices ecosystem, Spotify has created Helios, a tool for continuous deployment. This tool makes it possible to manage and coordinate numerous microservices across the entire system, facilitating deployment and operation.

Spotify's shift to microservices demonstrates the potential benefits of this architectural style for large-scale applications. Although moving from a monolithic architecture is a tough and time-consuming task requiring careful planning, the results can be significant. Spotify's journey illustrates how microservices can lead to quicker innovation, better scalability, and enhanced operational efficiency amid fast growth and changing market needs. It is a compelling example for companies seeking to upgrade their systems and take advantage of the agility and flexibility that microservices can bring.

6.3 Open Architectures

Open architectures have become a powerful approach for creating complex systems that are adaptable and can evolve over time. The main idea behind open architectures is the use of open standards, modular configurations, and clear interfaces that support the integration and collaboration of components from different suppliers. This differs from closed, proprietary architectures that limit flexibility and can result in vendor dependency and outdated technology. The SEBoK examines the benefits of open architectures in managing complexity and encouraging teamwork in systems engineering.

6.3.1 Key Principles

The essence of open architectures can be distilled into three fundamental principles:

- **Open Standards**: The use of open, publicly available standards for interfaces, protocols, and data formats ensures that components from different vendors can seamlessly interact and exchange information. This promotes interoperability and avoids the limitations of proprietary solutions.

- **Modular Design**: The system is decomposed into loosely coupled, self-contained modules with well-defined interfaces. This modularity enables independent development, deployment, and evolution of components, fostering flexibility and adaptability.
- **Collaboration and Community**: Open architectures make it easier for different developers, vendors, and users to work together and share knowledge. This helps the emergence of new ideas, speeds up development, and supports a lively community around the architecture.

6.3.2 Advantages of Open Architectures

The adoption of open architectures offers several significant advantages for organizations and stakeholders:

- **Increased Flexibility and Adaptability**: Open architectures enable systems to evolve and adapt to changing requirements, technologies, and operational contexts. By leveraging modular design and well-defined interfaces, components can be added, removed, or replaced with minimal impact on the overall system.
- **Improved Interoperability and Integration**: The use of open standards and interfaces facilitates the integration of components from different vendors, enabling the creation of best-of-breed solutions and promoting a more collaborative and diverse ecosystem. This interoperability also allows for the integration of legacy systems and the incorporation of new technologies as they emerge.
- **Reduced Vendor Lock-in and Technology Obsolescence**: Open architectures mitigate the risks associated with relying on a single vendor or proprietary technology. By promoting competition and enabling the substitution of components, open architectures reduce the potential for vendor lock-in and help organizations avoid the costs and disruptions associated with technology obsolescence.

6.3.3 Challenges and Considerations

While open architectures offer significant benefits, their implementation also presents challenges:

- **Standardization and Compatibility Issues**: The success of open architectures relies on the existence and adoption of robust, widely accepted standards. The lack of mature standards or the presence of competing standards can hinder interoperability and limit the benefits of openness.

- **Security and Intellectual Property Concerns**: Open architectures may raise concerns regarding the protection of sensitive information and intellectual property. Organizations must carefully manage the sharing of data and knowledge and establish appropriate security measures and governance structures.
- **Governance and Ecosystem Management**: Developing and operating open architectures often involves multiple stakeholders, requiring effective governance mechanisms and ecosystem management practices to ensure alignment of interests, resolve conflicts, and maintain long-term sustainability.

6.3.4 Examples

Several examples demonstrate the successful application of open architectures in various domains:

- **The Open Systems Interconnection Model**: This standardized architecture for communication protocols has enabled the development of the Internet and facilitated interoperability between diverse networking technologies (Zimmermann 1980).
- **The Automotive Open System Architecture Standard**: Promotes the reuse and interchangeability of software components in the automotive industry, reducing development costs and enabling rapid integration of new features (Fürst et al. 2009).
- **The Modular Open Systems Approach**: Adopted by the US Department of Defense to facilitate the development of flexible, upgradable, and cost-effective military systems by leveraging open standards and modular design principles (Department of Defense 2015).

These examples illustrate the potential of open architectures to create systems that are more adaptable, resilient, and responsive to evolving needs and opportunities.

6.3.5 The A320 NEO Case

The Airbus A320 series, especially the A320 NEO (New Engine Option), illustrates the success of an open architecture approach in aircraft design. The open architecture of the A320 has been key to its flexibility, adaptability, and long lifespan, allowing Airbus to meet changing market needs and new technologies effectively.

The A320's open architecture has yielded several key benefits:

- **Flexibility and Adaptability**: The modular design and well-defined interfaces of the A320 have allowed Airbus to seamlessly upgrade the aircraft with new engines (CFM LEAP-1A or Pratt & Whitney PW1100G) and aerodynamic

enhancements (like sharklets) to create the A320neo. This adaptability has enabled Airbus to swiftly respond to market demands for more fuel-efficient aircraft, ensuring its continued competitiveness in its narrow-body market segment.

- **Commonality and Integration**: The open architecture approach has fostered high commonality between different A320 family members (A318, A319, A320, A321), extending to systems, components, and flight deck design. This commonality streamlines maintenance, crew training, and operations for airlines, while the open systems architecture facilitates the integration of new technologies and upgrades over time, allowing Airbus to continually enhance the aircraft's performance and capabilities.
- **Future-proofing and Cost-effectiveness**: The A320's open architecture design has enabled Airbus to incorporate new technologies and meet evolving regulatory requirements without requiring a complete redesign. This future-proofing, coupled with the ease of customization and integration of preferred systems and equipment, translates into cost-effectiveness for airlines, further solidifying the A320's appeal.
- **Longevity and Sustainability**: The adaptability afforded by the open architecture has contributed to the A320 family's remarkable longevity, with the basic design remaining competitive for over three decades through various upgrades and improvements. Moreover, the open architecture has facilitated the integration of fuel-saving technologies and more efficient engines, leading to improved environmental performance over time.

In contrast, the Boeing 737 MAX experienced critical issues, mainly due to its closed architecture. The need for larger engines led to major design changes, resulting in the implementation of the MCAS system, which raised safety concerns. This highlights the risks of closed systems and shows how openness can help improve adaptability and reduce risks.

The A320's open architecture is a good example of how open systems can foster innovation, flexibility, and continued success. By using open standards, modular design, and teamwork, Airbus has developed an aircraft model that succeeds in a competitive market. The A320's success can motivate system designers in various fields.

6.3.6 Summary

Open architectures mark a shift in system design, focusing on flexibility, interoperability, and teamwork. By adopting open standards and a modular approach, organizations can build systems that better handle today's complex technological landscape. Although open architectures have some challenges, their advantages in adaptability and sustainability make them appealing for many modern systems.

6.4 Architectural Patterns and Tactics

Architectural patterns and tactics are important for creating systems that meet certain quality requirements. Patterns provide tested solutions for common design issues, helping architects choose wisely and improve system aspects such as performance, scalability, reliability, and security.

Tactics offer detailed control, letting architects assess how single design choices affect quality attributes and manage trade-offs between different issues. The combination of patterns and tactics supports good system design, leading to architectures that are both simple and robust.

6.4.1 The Power of Patterns

Architectural patterns provide reusable solutions for typical design problems found in specific situations (Buschmann et al. 2007). They capture the core of a solution, offering a broad structure and a set of components that interact to achieve desired quality characteristics. The choice and the use of patterns must be based on a solid grasp of the system's specific needs and constraints, making sure that the selected pattern fits with the overall architectural vision and objectives.

Some prominent architectural patterns that have found widespread adoption include (Buschmann et al. 2007):

- **Model-View-Controller (MVC)**: This pattern promotes modularity, flexibility, and maintainability by separating the representation of information (Model) from user interaction (View) and the logic that controls the interaction (Controller). MVC is particularly prevalent in software-intensive systems, where it facilitates the development of user interfaces that are both responsive and adaptable.
- **Pipes and Filters**: This pattern decomposes a system into a series of independent, reusable components (filters) connected by pipes that transmit data. The modular and flexible nature of this pattern makes it well-suited for data processing applications, where data transformations and manipulations can be performed in a sequential and streamlined manner.
- **Blackboard**: The blackboard pattern facilitates collaborative problem-solving by enabling multiple specialized subsystems (knowledge sources) to share and modify a common data structure (blackboard). This pattern is particularly useful in knowledge-based systems and artificial intelligence applications, where diverse sources of information need to be integrated and analyzed to find a solution.
- **Broker**: The broker pattern defines a distributed communication infrastructure that allows decoupled components to interact transparently. It promotes

interoperability and scalability in distributed systems by providing a central point of coordination and communication between components.

These patterns, among others, offer a valuable starting point for designing systems with desirable quality attributes. However, it is essential to remember that patterns are not silver bullets – their effectiveness hinges on their judicious application within the appropriate context.

6.4.2 The Precision of Tactics

Architectural tactics are design choices that influence compliance with certain quality characteristics (Bass et al. 2012). They are more detailed than patterns and focus on specific parts of the system's design. Tactics enable architects to think about how their design decisions impact quality attributes and to make thoughtful trade-offs among different concerns.

Some common architectural tactics used to achieve various quality attributes include:

- **Performance**: Caching, load balancing, and concurrency are tactics that can be used to enhance system performance by optimizing resource usage, distributing workload, and enabling parallel execution of tasks.
- **Scalability**: Horizontal scaling, sharding, and caching are tactics that can be used to improve system scalability by adding more resources, partitioning data, or storing frequently accessed data in memory for faster retrieval.
- **Reliability**: Redundancy, failover, and monitoring are tactics that contribute to system reliability by providing backup components, automatic recovery mechanisms, and proactive detection of potential failures.
- **Security**: Authentication, authorization, and encryption are tactics that enhance system security by controlling access to resources, verifying credentials, and protecting sensitive data from unauthorized access.

Choosing and using architectural tactics should follow a detailed examination of the system's quality requirements and the trade-offs between the tactics. Architects need to consider how each tactic affects the overall system design and the possible interactions or conflicts between different tactics.

6.4.3 Model-based Systems Engineering and Architectural Decisions

Model-based systems engineering (MBSE) provides a powerful framework for incorporating architectural patterns and tactics into the system design process. Languages like Systems Modeling Language provide tools for showing architectural elements and the links between them, enabling different design alternatives to be visualized and studied. Simulation and analysis tools can assess the effects

of various patterns and tactics on system behavior and performance, supporting better decision-making and trade-off analysis.

When selecting and applying architectural patterns and tactics in MBSE, it is crucial to consider the following factors:

- **Specific Quality Attribute Requirements**: These requirements, derived from business goals and stakeholder needs, serve as the guiding principles for architectural decision-making.
- **Trade-offs and Conflicts**: Architects must carefully navigate the trade-offs and conflicts between different quality attributes and their corresponding patterns and tactics.
- **System Context**: The domain, technology stack, and organizational constraints all play a role in shaping the architectural choices.
- **Impact on System Complexity, Maintainability, and Evolution**: It is essential to consider the long-term implications of architectural decisions on the system's complexity, maintainability, and ability to evolve over time.

By examining these factors and using MBSE tools, architects can create systems that meet quality requirements while dealing with complexity and ensuring they can be maintained. The combination of architectural patterns, tactics, and MBSE helps architects design systems that are functional, flexible, reliable, and in tune with the changing demands of stakeholders and technology.

6.5 Evaluating and Updating System Architectures

Evaluating and updating system architectures are key parts of the systems engineering process. As systems become more complex and face changing needs and situations, it's crucial to review the effectiveness of the system architecture and adjust it if necessary. This keeps the system on track to achieve its goals and provide value to stakeholders (Bass et al. 2012).

6.5.1 The Evaluation Process

The evaluation of system architectures includes a detailed examination of how they meet system requirements, support required quality characteristics, and enable future changes and growth (Clements et al. 2002). This evaluation occurs at different points in the system lifecycle, and uses both qualitative and quantitative methods (Kazman et al. 1996).

6.5.2 The Architecture Trade-off Analysis Method

One common method for assessing system architectures is the Architecture Trade-off Analysis Method (ATAM) (Kazman et al. 2000). ATAM is a scenario-based approach that gathers stakeholders to find and prioritize important quality

features like performance, security, modifiability, and usability. The method then looks at the architectural choices and design options that affect these features, identifying risks and trade-offs and suggesting improvements.

6.5.3 Modularity and Flexibility Assessment

Checking modularity and flexibility is important in architecture. Modular designs split the system into parts that can function independently of each other, offering benefits such as better reusability, maintenance, and adaptability (Baldwin and Clark 2000). To check modularity, we examine how components collaborate, checking how well they interconnect, as well as potential areas that might require changes or improvements (Sarkar et al. 2008).

Flexibility refers to the system's ability to evolve in line with new requirements, technologies, and conditions without the need for many changes (Fricke and Schulz 2005). Measuring flexibility consists of checking how well the architecture can handle anticipated changes, such as new features or technologies, and how resilient it is to unexpected changes and issues (Ross et al. 2008).

6.5.4 The Dynamics of Architectural Evolution

The evolution of system architectures is essential for systems to remain relevant, effective, and responsive to changing needs and contexts (Lehman 1996). This evolution can be driven by various factors:

- Changes in stakeholder requirements or priorities
- Advancements in technology or the emergence of new standards
- Shifts in the operational environment or context of use
- Identification of performance bottlenecks or quality issues
- Need for improved scalability, reliability, or security

6.5.5 A Systematic Approach

Evolving a system architecture requires a systematic and disciplined approach to manage complexity and risk (Nord et al. 2012). This typically involves:

1. Analyzing the current architecture to pinpoint areas for improvement or adaptation.
2. Defining desired architectural changes and their rationale.
3. Assessing the impact of proposed changes on functionality, quality attributes, and stakeholders.

4. Developing an implementation plan, including migrations, transitions, or coexistence strategies.
5. Executing the plan and monitoring system performance to ensure the changes' intended effect.
6. Documenting the evolved architecture and updating relevant artifacts (models, specifications, etc.).

6.5.6 Balancing Stability and Change

One of the main challenges of architecture evolution is dealing with the conflict between stability and change (Lehman 1996). Architectures need to provide a steady foundation while being able to adapt when changes are needed. Finding this right combination requires thoughtful thinking about the system's goals, constraints, and environment, as well as using suitable patterns, tactics, and methods (Bass et al. 2012).

6.5.7 Stakeholder Collaboration

Another important part is working with and organizing many stakeholders (Nord et al. 2012). Changes in architecture can affect many areas, so clear communication, teamwork, and agreement among all parties involved, such as developers, operators, maintainers, and users, are necessary (Clements et al. 2002).

6.5.8 Summary

Evaluating and updating system architectures is an essential part of systems engineering. They enable organizations to examine how the architecture is working, find areas for improvement, and adjust it to keep the system relevant and valuable. A structured, careful, and team-oriented approach to these tasks is key for the long-term success of complex systems.

Exercise: Architecting an e-commerce system

You are the lead architect for a growing e-commerce company. The company currently has a monolithic architecture, but is experiencing scalability issues as it expands its product offerings and customer base. The management team wants to modernize the system to handle increased traffic, improve maintainability, and allow for rapid feature development. They are also considering opening up parts of their platform to third-party developers.

(Continued)

(Continued)

Part 1: Architectural Analysis

1. List three key advantages and three key drawbacks of the current monolithic architecture for an e-commerce system.
2. Identify five core services that could be candidates for microservices in an e-commerce system. Briefly explain why each would benefit from being a separate service.
3. Describe two scenarios where a hybrid approach (combining elements of monolithic and microservices architectures) might be beneficial for an e-commerce system.

Part 2: Open vs. Closed Architecture Considerations

1. Identify three potential risks associated with adopting an open architecture for an e-commerce system and suggest mitigation strategies for each.
2. Describe a compromise between open and closed architectures that could be suitable for an e-commerce platform. Explain the reasoning behind this compromise.

Part 3: Architectural Patterns and Tactics

1. Suggest three architectural patterns that could be applied to improve the scalability and maintainability of the e-commerce system. Briefly explain how each pattern would address specific challenges.
2. Identify two availability tactics and two performance tactics that could be used in the e-commerce system architecture. Explain how these tactics would be implemented in a microservices-based approach.
3. Propose a strategy for gradually transitioning from the current monolithic architecture to a more modular, scalable architecture. Include at least three key steps in this transition process.

Part 4: Design Challenge

Based on your analysis and considerations from the previous parts, propose a high-level architecture for the e-commerce system that addresses the following requirements:

- Scalability to handle peak loads during sales events
- Flexibility to add new features and integrate with third-party services
- Improved maintainability and ease of updates
- Balance between open and closed components
- Consideration for future international expansion

Your proposal should include:

1. A diagram illustrating the main components of the architecture
2. A brief explanation of key architectural decisions and their rationale
3. Identification of at least two areas where compromises were made between different architectural approaches, explaining the trade-offs involved

Part 5: Reflection

Write a short reflection (200–300 words) on the challenges of designing an e-commerce system architecture that balances various competing concerns. Consider the following points:

1. The tension between architectural purity and practical compromises
2. The role of business requirements in shaping architectural decisions
3. How evolving technologies and market trends might influence architectural choices in the future

6.6 Linking MBSE to Architectural Concerns

MBSE provides a powerful framework to deal with important architectural issues mentioned earlier. Using MBSE principles and tools, organizations can implement these architectural principles and monitor them during the system's lifecycle. The Model-driven Architecture (MDA) Y model, illustrated in Figure 6.1, offers a useful way to understand how MBSE helps address architectural issues. The MDA Y model distinguishes between four main types of models:

- **Computation Independent Model (CIM)**: This model focuses on the system context and requirements from a business perspective, without considering the system's structure or processing details.
- **Platform Independent Model (PIM)**: This model represents the system's structure and behavior, independent of any specific technology or platform.
- **Platform Description Model (PDM)**: This model provides information about the target platform (technology) that is used during the transformation process from PIM to Platform Specific Model (PSM).
- **PSM**: This model combines the specifications in the PIM with the details of a particular implementation technology or platform described by the PDM.

MBSE uses various model types to gradually develop the system architecture, moving from broad business needs to specific platform-based implementations. This approach helps maintain a clear focus on different aspects and lets us tackle architectural issues at the right level of detail.

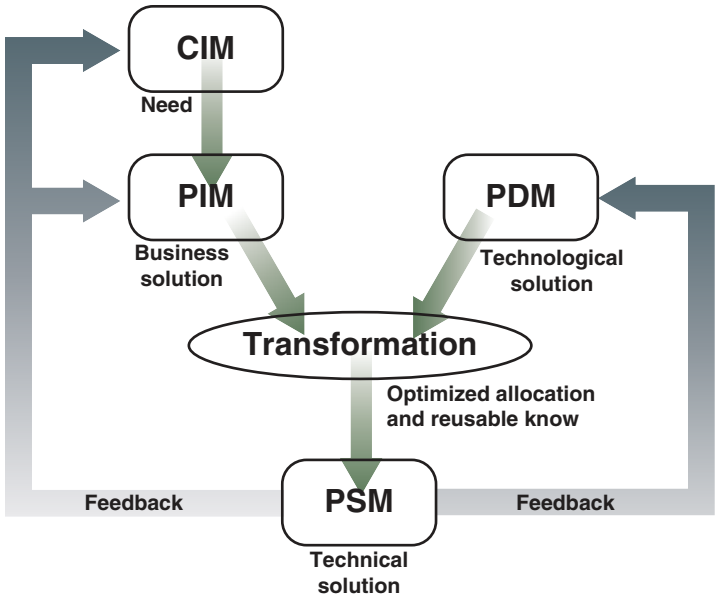


Figure 6.1 The MDA Y model.

6.6.1 Modularity and Interoperability

Regarding modularity, MBSE allows for breaking the system down into loosely connected, separately deployable modules at the PIM level. This modular structure can then be linked to certain implementation technologies and platforms at the PSM level, ensuring that modularity remains intact throughout the development process.

In the same way, MBSE promotes interoperability by enabling the establishment of clear interfaces and communication protocols at the PIM level. These general specifications can then be turned into platform-specific implementations, ensuring that system components can work together effectively across various technologies and domains.

6.6.2 Complexity Management and Adaptability

By creating models at different levels of detail, from CIM to PSM, MBSE lets architects focus on important parts of the system during each development stage. This layered modeling approach keeps the system architecture clear and manageable, even as it becomes larger and more complex. Moreover, MBSE promotes adaptability by allowing the creation of flexible architectures. By outlining system architecture at the PIM level, without tying it to specific technologies, MBSE helps the

system to develop and adjust over time. As new technologies emerge or requirements evolve, these platform-independent models can be updated to fit new specific implementations, making sure the system remains adaptable to changing demands.

6.6.3 Traceability and Consistency

Finally, MBSE ensures traceability and consistency between different architectural views and abstraction levels. By keeping a clear link between the CIM, PIM, and PSM models, MBSE guarantees that system requirements, architecture, and implementation stay consistent and traceable throughout the development process. This traceability is crucial for managing changes and ensuring that the system architecture aligns with business objectives and needs.

6.6.4 Illustrating MDA Benefits

The MDA approach, as illustrated in Figure 6.2, offers a powerful framework for designing and evolving complex systems. This approach provides several key benefits that address the challenges of modern system architecting. One of the most significant advantages of the MDA approach is the separation and independence of different architectural viewpoints:

- **Functional Architecture**: The top-left quadrant of the image shows the functional architecture, representing the system's logical structure and behavior. This view focuses on what the system does, regardless of how it's implemented.
- **Platform**: The top-right quadrant represents the platform, which includes the middleware, operating system, and hardware components. This separation allows architects to consider the execution environment independently of the system's functionality.
- **Realization**: The bottom quadrant shows the realization or allocated physical architecture, which maps the functional components onto specific hardware and software elements.

This separation of concerns allows architects to focus on each aspect independently, handling complexity and improving flexibility. The MDA approach facilitates the coevolution of functional and physical architectures through:

- **Bidirectional Mapping**: The arrows between the functional architecture and realization indicate a bidirectional relationship. This allows changes in one domain to be reflected in the other, ensuring consistency as the system evolves.

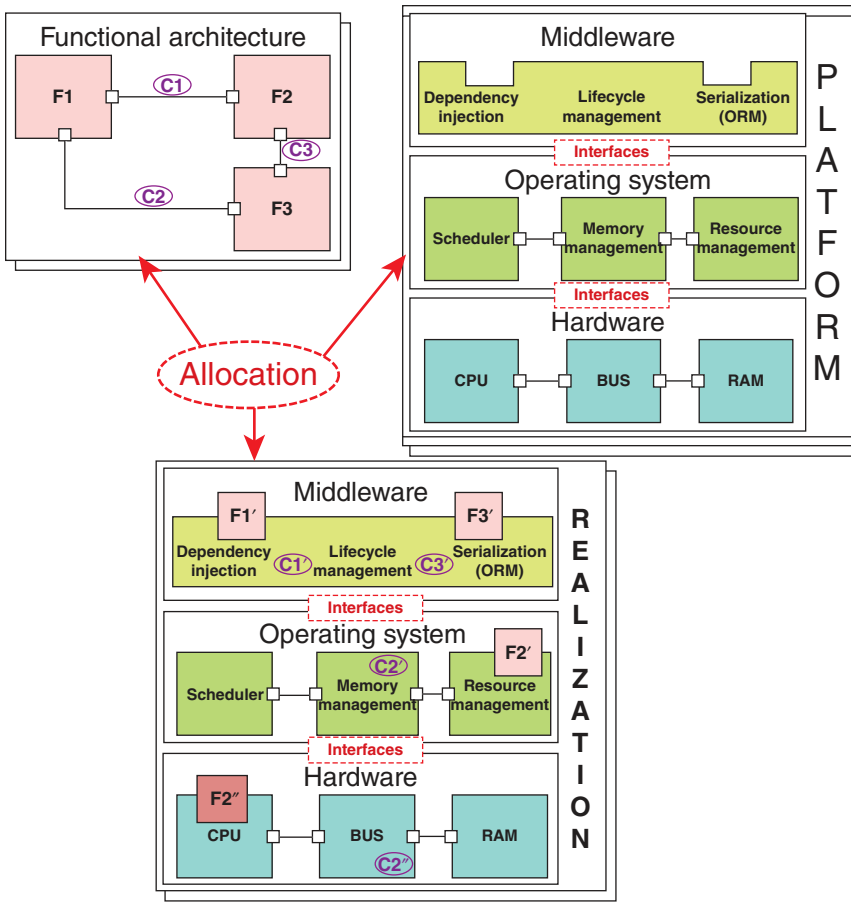


Figure 6.2 The MDA Y model in the context of hardware/software co-design.

- **Iterative Refinement**: Architects can iteratively refine both the functional and physical architectures, making adjustments in one view and propagating those changes to the other. This supports an agile development process where the system design can adapt to changing requirements or technological advancements.

The MDA approach emphasizes the importance of well-defined interfaces, which provide several benefits:

- **Interface-based Composition**: Components in both the functional and physical architectures are composed based on their interfaces (represented by

the small squares connecting components). This interface-centric approach promotes modularity and reduces coupling between components.

- **Component Replaceability**: By focusing on interfaces rather than implementations, the MDA approach enables easy replacement of components. As long as a new component adheres to the same interface and constraints (performance, reliability, etc.), it can be substituted without affecting the rest of the system. This is particularly valuable for:
  - Technology upgrades: Replacing outdated components with more advanced ones.
  - Vendor flexibility: Switching between different suppliers' components.
  - Performance optimization: Swapping in more efficient implementations.
- **Middleware Abstraction**: The platform layer, including middleware, provides an additional level of abstraction. This allows components to be replaced or upgraded at the platform level without necessarily impacting the functional or physical architectures.

The "Allocation" arrows in the figure highlight another key benefit of the MDA approach:

- **Traceability**: The explicit mapping between functional components and their physical realizations provides clear traceability. This is crucial for understanding the impact of changes, managing requirements, and ensuring comprehensive test coverage.
- **Flexible Allocation**: The allocation process allows for different deployment scenarios. A single functional component might be allocated to multiple physical components, or vice versa, providing flexibility in how the system is realized.

The MDA provides a powerful approach for system design. It splits up different issues, supports the growth of various parts, focuses on design based on interfaces, and enables clear assignments and tracking. This makes it easier and more flexible to create and change complex systems in today's fast-moving technology world. This approach fits well with open architecture concepts, and enables the creation of modular, scalable, and easy-to-maintain systems.

In summary, MBSE offers a comprehensive framework to tackle essential design issues such as modularity, interoperability, complexity management, adaptability, and traceability. By using the principles and tools from MBSE, as shown by the MDA Y model, organizations can successfully apply and oversee these design ideas throughout the system's lifecycle, making sure the final system is reliable, adaptable, and meets business needs.

Exercise: Implementing MDA for e-commerce system transition

Using the e-commerce system discussed in the previous exercise, we will develop PIM, PDM, and PSM models to facilitate the architectural transition.

Part 1: PIM

Using the core services identified in the previous exercise, create a high-level PIM for the e-commerce system:

1. Identify and list the key business entities (e.g. user, product, order, etc.).
2. Create a Unified Modeling Language class diagram representing these entities and their relationships.
3. Define the main operations for each entity without specifying implementation details.
4. Develop a sequence diagram for a key business process (e.g. placing an order).

Part 2: PDM

Based on the microservices architecture decision from the previous exercise:

1. Define the target technology stack for the microservices implementation.
2. Specify the communication protocols between services (e.g. REST, gRPC).
3. Outline the data storage solutions for different types of data.
4. Describe the deployment environment (e.g. containerization, orchestration).

Part 3: PSM

For each core service identified in the PIM:

1. Create a detailed class diagram that includes technology-specific elements.
2. Specify the application programming interface design (e.g. REST endpoints or GraphQL schema).
3. Define the data model for the service's storage solution.
4. Outline any service-specific libraries or frameworks to be used.

Part 4: Model Transformations

1. Describe the process of transforming the PIM to the PSM using the PDM.
2. Identify any automation tools that could assist in this transformation.
3. Discuss how you would handle custom business logic during the transformation.

Part 5: Incremental Implementation Strategy

1. Propose a strategy for incrementally implementing the microservices architecture.
2. Explain how you would maintain system functionality during the transition.
3. Describe how you would validate each microservice against the original PIM.

Part 6: Reflection

Write a brief reflection (200–300 words) on:

1. The benefits of using MDA for architectural transition.
2. Challenges you anticipate in applying MDA to a real-world e-commerce system.
3. How MDA can help in managing the complexity of the transition process.

6.7 Conclusion

In this chapter, we began examining the basic concepts behind systems architecting. We discussed the main points related to systems and their architectures, stressing the need for a complete, systems-focused approach to handle the complex nature of current systems. Guidelines such as the separation of concerns, modularity, abstraction, and scalability were highlighted as key points for good system design.

We also looked at different architectural styles, from monolithic to microservices, each with its own benefits and challenges in achieving system objectives. The concept of open architectures, which focuses on flexibility, interoperability, and collaboration, was introduced as a valuable approach for adapting to fast-changing technologies. Lastly, we examined how architectural patterns and tactics can help architects make smart choices and create systems that fulfill specific quality requirements.

The process of systems architecting is continuous, involving constant assessment and improvement of architectures to maintain their relevance and effectiveness as needs and contexts change. Using MBSE enhances this journey, offering a comprehensive framework for applying and managing architectural ideas throughout the system's lifecycle.

As we move to the next chapter on "Methodological Bases," we will expand on these basic principles and look at the structured methods and practices that direct systems architecting. We will review various frameworks, processes, and best practices that help architects turn business needs into actual systems, ensuring

effective design, development, and use of complex systems. This exploration of methodological bases will provide us with the necessary tools and techniques to successfully navigate the complex field of systems architecting and create systems that satisfy a variety of stakeholder and societal needs.

Questions

1. What is the primary purpose of system architecture?
   - **a** To develop isolated components
   - **b** To provide a blueprint for the structure, behavior, and properties of a system
   - **c** To focus solely on software development
   - **d** To eliminate the need for stakeholder involvement
2. Which of the following is NOT a fundamental principle of system architecture?
   - **a** Separation of concerns
   - **b** Modularity
   - **c** Continuous integration
   - **d** Scalability
3. What is the main advantage of modular architecture?
   - **a** Enhanced flexibility, reusability, and maintainability
   - **b** Increased system complexity
   - **c** Reduced performance
   - **d** Limited scalability
4. Which architectural style is characterized by organizing the system into distinct layers with specific responsibilities?
   - **a** Monolithic architecture
   - **b** Layered architecture
   - **c** Event-driven architecture
   - **d** Microservices architecture
5. What is a key characteristic of SOA?
   - **a** Tightly coupled components
   - **b** Single, cohesive unit of deployment
   - **c** Services encapsulating specific business capabilities
   - **d** Limited scalability
6. Which of the following is a principle of open architectures?
   - **a** Use of proprietary standards
   - **b** Tight coupling between components
   - **c** Modular design with well-defined interfaces
   - **d** Limited interoperability

7 What is the primary purpose of architectural patterns?
- **a** To provide reusable solutions to common design problems
- **b** To increase system complexity
- **c** To limit system scalability
- **d** To reduce system performance

8 Which of the following is NOT a typical architectural tactic?
- **a** Caching for performance improvement
- **b** Load balancing for scalability
- **c** Encryption for security
- **d** Continuous deployment for reliability

9 What is the primary goal of evaluating system architectures?
- **a** To increase system complexity
- **b** To assess the architecture's ability to satisfy requirements and support quality attributes
- **c** To reduce development costs
- **d** To eliminate the need for future modifications

10 How does MBSE support system architecting?
- **a** By focusing solely on software development
- **b** By eliminating the need for architectural patterns
- **c** By providing a framework to implement and manage architectural principles throughout the system lifecycle
- **d** By replacing the need for human architects

Answers

1-b; 2-c; 3-a; 4-b; 5-c; 6-c; 7-a; 8-d; 9-b; 10-c

References

Baldwin, C.Y. and Clark, K.B. *Design Rules: The Power of Modularity*. Design Rules. MIT Press, 2000. ISBN 9780262024662. https://books.google.fr/books?id=oaBOuo4mId8C.

Bass, L., Clements, P., and Kazman, R. *Software Architecture in Practice: Software Architect Practice_c3*. SEI Series in Software Engineering. Pearson Education, 2012. ISBN 9780132942782. https://books.google.fr/books?id=-II73rBDXCYC.

Bondi, A.B. Characteristics of scalability and their impact on performance. In *Proceedings of the 2nd International Workshop on Software and Performance*, 2000:195–203.

Buschmann, F., Henney, K., and Schmidt, D.C. *Pattern-Oriented Software Architecture, On Patterns and Pattern Languages*. Pattern-oriented Software Architecture. Wiley, 2007. ISBN 9780470512579. https://books.google.fr/books?id=4MVhfLZlUF8C.

Clements, P., Kazman, R., and Klein, M. *Evaluating Software Architectures: Methods and Case Studies*. SEI Series in Software Engineering. Addison–Wesley, 2002. ISBN 9780201704822. https://books.google.fr/books?id=VatQAAAAMAAJ.

Department of Defense. Modular open systems approach (MOSA) in Defense Acquisition Programs, 2015. https://www.cto.mil/sea/mosa/.

Erl, T. *Service-Oriented Architecture: Analysis and Design for Services and Microservices*. The Pearson Service Technology Series from Thomas Erl. Pearson Education, 2016. ISBN 9780133858693. https://books.google.fr/books?id=vMm8DQAAQBAJ.

Evans, E. *Domain-driven Design: Tackling Complexity in the Heart of Software*. Addison–Wesley, 2004. ISBN 9780321125217. https://books.google.fr/books?id=xColAAPGubgC.

Fielding, R.T. *Architectural Styles and the Design of Network-based Software Architectures*. University of California, Irvine, 2000. https://books.google.fr/books?id=1xN9NwAACAAJ.

Fowler, M. *Patterns of Enterprise Application Architecture*. Addison–Wesley Signature Series (Fowler). Pearson Education, 2012. ISBN 9780133065213. https://books.google.fr/books?id=vqTfNFDzzdIC.

Fricke, E. and Schulz, A.P. Design for changeability (DFC): principles to enable changes in systems throughout their entire lifecycle. *Systems Engineering* 2005:8(4).

Fürst, S., Mössinger, J., Bunzel, S., et al. AUTOSAR—a worldwide standard is on the road. In *14th International VDI Congress Electronic Systems for Vehicles, Baden-Baden*, 2009:62. Citeseer.

Hohpe, G. and Woolf, B. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison–Wesley Signature Series (Fowler). Pearson Education, 2012. ISBN 9780133065107. https://books.google.fr/books?id=qqB7nrrna_sC.

Josuttis, N.M. *SOA in Practice: The Art of Distributed System Design*. O'Reilly Media, 2007. ISBN 9780596551551. https://books.google.fr/books?id=jUn0mXGXUIcC.

Kazman, R., Abowd, G., Bass, l., et al. Scenario-based analysis of software architecture. *IEEE Software* 1996:13(6):47–55.

Kazman, R., Klein, M., and Clements, P. *ATAM: method for architecture evaluation*. AD-a382 629. Carnegie-Mellon Univ Pittsburgh PA Software Engineering Institute, 2000. https://books.google.fr/books?id=qMGD0AEACAAJ.

Lehman, M.M. Laws of software evolution revisited. In *European Workshop on Software Process Technology*, 1996:108–124. Springer.

Lewis, J. and Fowler, M. Microservices: a definition of this new architectural term, 2014. https://martinfowler.com/articles/microservices.html.

Maier, M.W. Architecting principles for systems-of-systems. *Systems Engineering: The Journal of the International Council on Systems Engineering* 1998:1(4):267–284.

Michelson, B.M. Event-driven architecture overview. *Patricia Seybold Group* 2006:2(12):10–1571.

Nadareishvili, I., Mitra, R., McLarty, M., et al. *Microservice Architecture: Aligning Principles, Practices, and Culture*. O'Reilly Media, 2016. ISBN 9781491956342. https://books.google.fr/books?id=Ev2wDAAAQBAJ.

Newman, S. *Building Microservices: designing fine-grained system*, 2021. O'Reilly Media, O'Reilly ISBN 1492034029.

Nord, R.L., Ozkaya, I., Kruchten, P., et al. In search of a metric for managing architectural technical debt. In *2012 Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture*, 2012:91–100. IEEE.

Papazoglou, M.P. Service-oriented computing: concepts, characteristics and directions. In *Proceedings of the Fourth International Conference on Web Information Systems Engineering*, 2003. *WISE*, 2003:3–12. IEEE.

Parnas, D.L. On the criteria to be used in decomposing systems into modules. *Communications of the ACM* 1972:15(12):1053–1058.

Ross, A.M., Rhodes, D.H., and Hastings, D.E. Defining changeability: reconciling flexibility, adaptability, scalability, modifiability, and robustness for maintaining system lifecycle value. *Systems Engineering* 2008:11(3):246–262.

Sarkar, S., Kak, A.C., and Rama, G.M. Metrics for measuring the quality of modularization of large-scale object-oriented software. *IEEE Transactions on Software Engineering* 2008:34(5):700–720.

SEBoK Editorial Board. *The Guide to the Systems Engineering Body of Knowledge (SEBoK)*. INCOSE, 2024.

Shaw, M. and Garlan, D. *Software Architecture: Perspectives on an Emerging Discipline*. An Alan R. Apt book. Prentice Hall, 1996. ISBN 9780131829572. https://books.google.fr/books?id=5KlQAAAAMAAJ.

Zimmermann, H. OSI reference model-the iso model of architecture for open systems interconnection. *IEEE Transactions on Communications* 1980:28(4):425–432.

7

Methodological Bases

7.1 Preliminary Considerations

Choosing and adapting an appropriate methodology is important for achieving the success of any systems engineering project. There are many methodologies in the literature, so it is necessary to understand their goals and context before selecting any of them (Estefan 2007; Weilkiens 2011). Factors such as the project's size, team skills, and tools available are essential in this decision. Moreover, clearly defining process steps, their inputs and outputs, who is in charge, and the tools used is key to having an efficient development structure (Sage and Rouse 2014).

7.1.1 Navigating the Design Space

A key choice in systems engineering is the approach: top-down, bottom-up, or hybrid (Kossiakoff et al. 2020; Weilkiens 2011). The top-down approach focuses on planning and understanding goals first. It starts with high-level objectives and breaks them down into smaller, detailed elements. This strategy works best for projects with clear goals and stable requirements.

On the other hand, the bottom-up approach uses current components and technologies to build the system from the outset, combining them together. It supports a gradual and iterative process, which can be beneficial in fast-changing technological environments.

7.1.2 The Meet-in-the-middle Approach

Bear in mind that in the real world, most projects do not start from scratch. Most systems engineering tasks build on what is already known, using available technologies and components. The meet-in-the-middle method takes this into account, and aims to combine the benefits of top-down and bottom-up approaches.

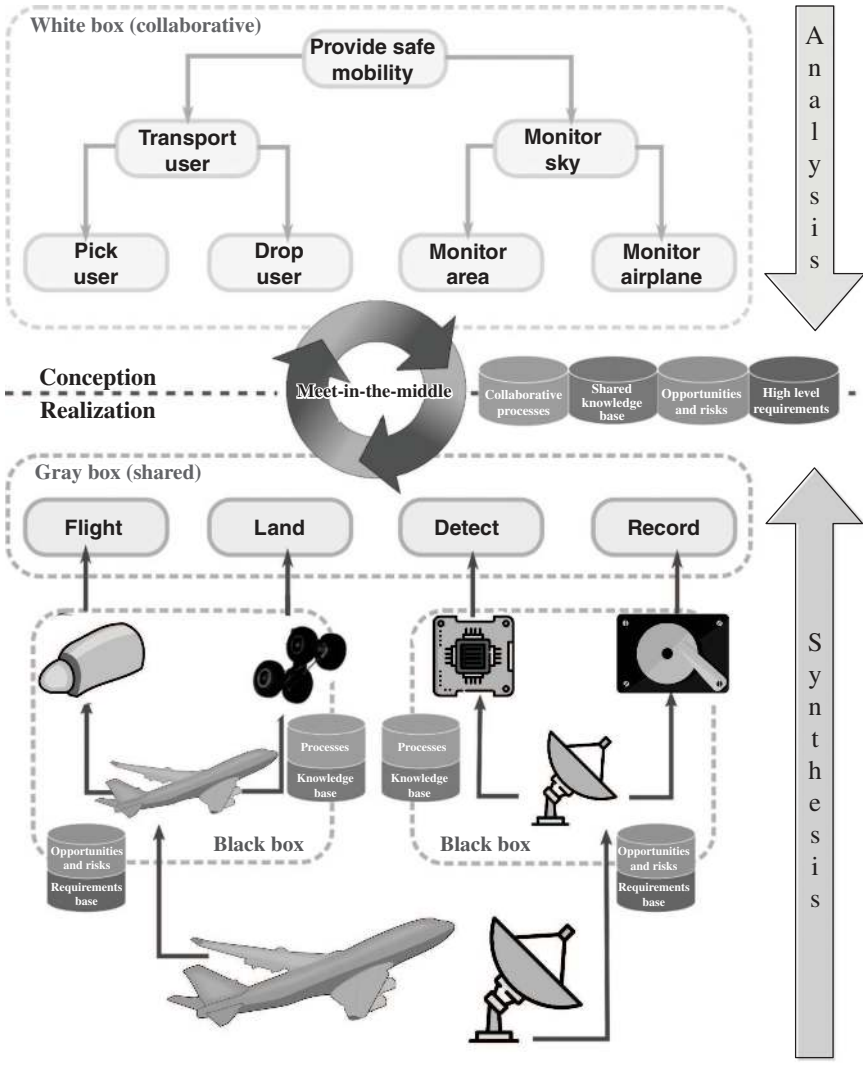


Figure 7.1 Meet-in-the-middle approach.

As illustrated by Figure 7.1, the "meet-in-the-middle" approach blends top-down and bottom-up methods to balance their strengths and weaknesses (Kossiakoff et al. 2020). This approach examines requirements from a top-down perspective to shape a conceptual solution, while simultaneously assessing the existing system from a bottom-up perspective. By combining insights from both sides, the meet-in-the-middle approach helps create models that reflect both the

desired system operations and the constraints imposed by existing components and technologies. The comparison of desired system models (prescriptive) and actual implementation models (descriptive) helps engineers to assess system viability and development costs quickly when dealing with changing constraints.

The benefits of this approach are numerous:

- Feasibility Assessment
  - Identifies gaps between desired and current states
  - Highlights technical challenges
  - Reveals implementation barriers
  - Pinpoints areas requiring innovation
- Cost Evaluation
  - Enables precise resource estimation
  - Helps identify cost-intensive modifications
  - Supports make-or-buy decisions
  - Facilitates budget planning
- Risk Management
  - Early detection of potential issues
  - Better understanding of technical debt
  - Improved project planning
  - More accurate timeline projections

This model-based comparison enables clear decisions to be made when developing complex systems, ensuring that they are technically feasible and economically reasonable. Indeed, a key part of the meet-in-the-middle approach is gap analysis. This process compares the proposed solution with the current system to find differences and areas where improvements can be made. This analysis helps to figure out if the project is feasible, estimate engineering expenditure, and inform choices about reusing parts, reworking, or creating new elements (Kossiakoff et al. 2020).

Figure 7.2 illustrates the gap analysis process, highlighting its role in bridging the gap between the existing and desired state of the system. Gap analysis provides a clear structure for exploring the design space, assessing risks, and making informed choices about the system's structure and application (Kossiakoff et al. 2020).

By using a blend of prescriptive and descriptive models, the meet-in-the-middle method allows for the reuse of existing components at various abstraction levels, which boosts efficiency and makes good use of established knowledge and skills. It also helps produce proof and evidence for certification agencies, guaranteeing that the system complies with regulatory and safety standards (Friedenthal et al. 2014; Weilkiens 2011).

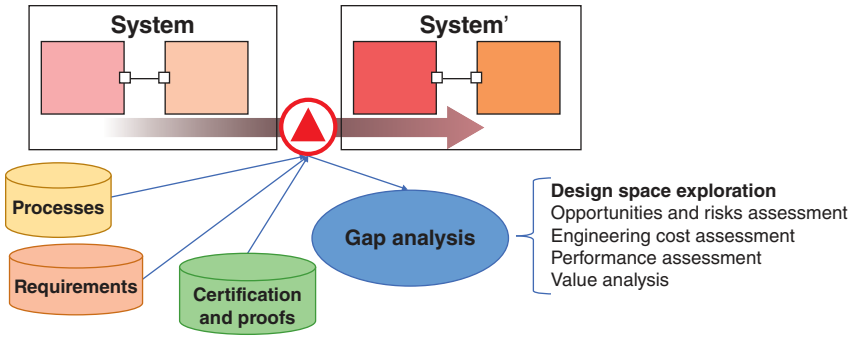


Figure 7.2 Gap analysis.

7.1.3 Exploring Architectural Alternatives

A key part of any robust systems engineering process is making and examining various architectural choices. The main objective is not to settle for a single design, but to present a range of valid alternatives, each carefully assessed. This allows the stakeholders to make informed decisions that fit their specific needs, constraints, and preferences (Maier 2009).

Examining different architectural choices promotes innovation, encouraging a broader search for potential solutions and possibly revealing unique designs that might not be seen otherwise. It adds flexibility to the decision-making process, helping stakeholders handle complex trade-offs between different criteria and choose the architecture that best meets their objectives. This approach also helps manage uncertainty by providing a variety of alternatives that can deal with various future situations. In addition, it creates a teamwork environment where the engineering team and the client work together in decision-making, ensuring that the final design is aligned with the customer's vision and objectives.

7.1.4 Decision-making and Architecture Optimization

To meet the challenges of assessing and comparing different architectural alternatives, systems engineers rely on a set of useful tools, including methods from multi-criteria decision making (MCDM) and architecture optimization. MCDM techniques, such as the analytic hierarchy process (Saaty 1990) and technique for order of preference by similarity to ideal solution (Tzeng and Huang 2011), offer organized approaches for evaluating various, sometimes conflicting criteria and ranking alternatives in a structured way.

At the same time, architecture optimization methods, such as genetic algorithms (Holland 1992) and simulated annealing (Kirkpatrick et al. 1983), assist engineers

in effectively searching through large design spaces, creating and improving architectural alternatives to find Pareto-optimal solutions that present the best trade-offs between different objectives.

The combination of MCDM and optimization techniques provides systems engineers with the skills needed to navigate the complex field of architectural choices, enabling them to make informed decisions that balance the needs of stakeholders, technical possibilities, and project constraints.

7.1.5 Summary

Selecting and adjusting the right methodology, as well as examining and assessing different architectural alternatives, are important steps along the way. The meet-in-the-middle approach, supported by gap analysis and the smart use of MCDM and optimization methods, provides a solid and flexible framework for meeting the challenges of system development.

By adopting these early ideas, organizations can promote teamwork and informed decisions, making sure that the chosen architecture fits well with stakeholder needs, technical practicality, and project constraints. This creates a foundation for the next stages of model-based systems engineering (MBSE), where models and careful methods can be used to turn architectural ideas into a successful product.

Exercise: Choosing a methodological approach

You are the lead systems engineer for a new project to develop an autonomous delivery drone system for a large e-commerce company. The system will include drones, a central control system, charging stations, and integration with the existing order management system. The company has some experience with drone technology but has never implemented a fully autonomous delivery system before.

Evaluate whether a top-down, bottom-up, or meet-in-the-middle approach would be most appropriate for this project. Consider the following factors:

- Existing technologies and components
- Project complexity and scale
- Team expertise and experience
- Time and budget constraints
- Regulatory requirements
- Stakeholder expectations

(Continued)

(Continued)

Please answer the following questions:

1. What are the main advantages of using a top-down approach for this project?
2. What benefits could a bottom-up approach offer in this scenario?
3. How might a meet-in-the-middle approach balance the strengths and weaknesses of top-down and bottom-up methods for this project?
4. Which specific aspects of the project would benefit most from a top-down analysis?
5. What existing components or technologies could be leveraged in a bottom-up approach?
6. How would the choice of methodology impact the project's ability to meet regulatory requirements for autonomous drones?
7. Consider the stakeholder expectations. How might the choice of methodology affect the project's ability to deliver a solution that meets their needs?
8. Given the company's limited experience with fully autonomous systems, how might this influence the choice of a methodology?

Propose a high-level plan for implementing your chosen methodological approach, outlining key steps and considerations.

Discuss potential risks associated with your chosen approach and suggest mitigation strategies.

7.2 Methodological Foundations

The heart of all MBSE methodologies is their ability to adapt to different application domains. Many factors impact the selection of a specific methodology, and modifications are often needed to meet project requirements, but all MBSE methodologies share certain characteristics (Estefan 2007; Weilkiens 2011; Madni and Sievers 2018). The Systems Engineering Body of Knowledge (SEBoK) offers important insights into these shared characteristics, highlighting the need for a methodical and organized approach to using MBSE. The model-centric approach, which is a key aspect of MBSE, allows for a more thorough, uniform, and automated systems engineering process, marking a major change from older document-centric approaches.

Additionally, MBSE methodologies focus on abstraction and separation of concerns. By abstracting details about how things are implemented and focusing on

the main features and functions of the system, MBSE provides a clearer view of the problem space and helps to find different solutions (Weilkiens 2011). This abstraction promotes clear communication, in-depth analysis, and effective decision-making throughout the system development process.

7.2.1 The "4+1" View Model: A Cornerstone of MBSE

A basic structure for understanding MBSE methodologies is the "4+1" view model, suggested by Philippe Kruchten (2002). This model offers a comprehensive sketch of the design of a system from different angles, with each angle focused on certain concerns. The four key views of the "4+1" model are:

- The **Logical View**, which captures the functional requirements and the decomposition of the system into objects or classes.
- The **Process View**, which describes the system's dynamic aspects and the interactions between processes.
- The **Development View**, which focuses on the organization of the actual system modules and subsystems.
- The **Physical View**, which maps the system onto the implementation solution, and reflects the distributed aspects of the system.

Besides these four views, the "4+1" model includes a fifth central view: the **Scenarios**. Scenarios, or use cases, are important for understanding, framing and confirming the architecture. They connect the other views, enabling the architecture to be verified and validated against requirements and constraints. Figure 7.3 illustrates the relationships between the different views and the central role of scenarios in the "4+1" model.

The "4+1" view model is widely used in MBSE because it provides a clear way of representing and thinking about the different parts of a system. Using this framework helps systems engineers to create a complete and coherent picture of the system, facilitating communication, analysis, and decision-making.

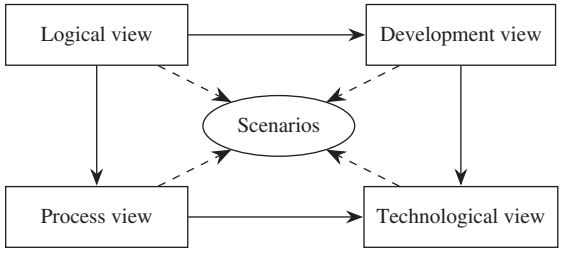


Figure 7.3 The "4+1" view model.

7.2.2 The Primacy of Scenarios in MBSE

The "4+1" view model reminds us of the central role of scenarios in MBSE. They represent real-world use cases that show how the system will operate, providing a better understanding of its behavior and requirements. By analyzing various scenarios, including nominal, alternative, and exception cases, engineers can:

- Gain insights into the system's behavior under various conditions
- Identify potential issues and risks early in the development process
- Ensure that the system meets its requirements and fulfills its intended purpose
- Facilitate effective communication and collaboration among stakeholders

In essence, scenarios act as a bridge between the abstract world of models and the concrete reality of system operation, guiding decision-making at every stage of the development lifecycle.

7.2.3 Tailoring the Methodology to Project Needs

The "4+1" view model provides a solid foundation to tailor MBSE methodologies to fit the project's specific features and constraints (Estefan 2007). This change means choosing and modifying the right methods, tools, and processes to match the project's objectives, scope, and resources. Some key factors to consider when tailoring an MBSE methodology include:

- The domain and complexity of the system being developed
- The size and distribution of the project team
- The regulatory and compliance requirements
- The available tools and infrastructure
- The project timeline and budget constraints

Tailoring the methodology is a key part of making sure that MBSE is used effectively in the project context. This requires a good understanding of the project's needs and constraints, as well as of the available MBSE methods and tools.

7.2.4 Continuous Improvement and Evolution

MBSE methods are not set in stone; they evolve and improve over time, in line with lessons learned from projects and new tools and technologies. Continuous improvement is one of the MBSE's core idea as organizations seek to improve and optimize their processes to achieve better results.

This continuous improvement process includes regular checks and reflections on how well the MBSE methodology works. It identifies areas for improvement

and applies specific changes. This may involve updating modeling rules, improving the toolset or adapting the process to fit new project or organization needs.

Successful continuous improvement in MBSE requires a culture of learning and teamwork, where experiences and knowledge are exchanged among teams and projects. It also requires investment in training and support to make sure all involved parties have the skills and knowledge to use the shared methodology effectively.

7.2.5 Summary

The basic ideas of MBSE provide the framework and rules for making good use of model-based methodologies in systems engineering. By using well-known frameworks such as the "4+1" view model, adapting methods to fit project requirements, and following standards and best practices, organizations can establish a grounded and effective MBSE practice. Scenarios play a key role in improving systems understanding, design, and validation.

However, the success of MBSE relies not only on a sound methodology, but also on a commitment to continuous improvement, and the ability to evolve with the growing needs of projects and stakeholders. By adopting these principles and methods, organizations can make the most of MBSE to provide high-quality complex systems that meet the requirements of their stakeholders and society.

Exercise: ATM system scenario analysis

An ATM system allows bank customers to perform financial transactions without the need for a human teller. The system typically includes functions such as cash withdrawal, balance inquiry, fund transfer, and deposit.

Part 1: Nominal Scenario

Write a detailed nominal scenario for a cash withdrawal transaction.

Example structure:

1. User inserts card
2. System validates card
3. User enters PIN
4. System verifies PIN
5. User selects "Withdraw Cash"

Part 2: Alternative Scenarios

Develop two alternative scenarios for the cash withdrawal process. Alternative scenarios represent different but valid paths through the use case.

(Continued)

(Continued)

Example alternative scenarios could include:

1. User chooses to withdraw an amount not available in preset options
2. User decides to change the account for withdrawal after initially selecting one

Part 3: Exception Scenarios

Create two exception scenarios for the ATM system. Exception scenarios describe what happens when something goes wrong.

Possible exception scenarios:

1. Insufficient funds in the account
2. Card reader malfunction
3. ATM runs out of cash

For each exception scenario:

1. Describe the initial conditions
2. Outline the sequence of events
3. Explain how the system should handle the exception
4. Describe the end state of the system

Part 4: Scenario Analysis

For each scenario developed in parts 1–3:

1. Identify the key actors involved
2. List the preconditions necessary for the scenario
3. Describe the post-conditions after the scenario completes
4. Highlight any security considerations relevant to the scenario

Part 5: Reflection

1. Explain the importance of considering alternative and exception scenarios in system design.
2. Discuss how these scenarios might influence the ATM system's architecture, user interface design and V&V activities.
3. Propose a method for prioritizing which scenarios to implement first in the development process.

7.3 Formalizing Processes

As stated by Osterweil, "Software processes are software too" (Osterweil 2011), the processes used to build software solutions can also be seen as a software specification. These processes are indeed made up of actions, data, exchanges, or decisions. As such, they can in fact be modeled as software. This idea applies well to system processes, as they are a mixture of control and information flows, with the main difference being that human actors usually carry out the execution. This point of view highlights the important benefits that come from formalizing system processes, such as improved verification, automation, and documentation.

The SEBoK stresses that a clear and structured process is crucial for successful systems engineering, which is in line with process engineering principles (Osterweil 1997). Formalizing processes means gathering and showing the main activities, artifacts, and roles in a clear way that helps ensure consistency, traceability, and reproducibility across projects, as stated by the Capability Maturity Model Integration (CMMI).

The level of detail in process models can vary according to the needs and constraints of the particular domain and organization. Nevertheless, the main goal remains the same: to create a solid basis for effective process management, continuous improvement, and successful outcomes in systems engineering. Organizations that adopt process formalization can fully harness the benefits of MBSE and improve the development of complex systems more efficiently, with higher quality, and greater stakeholder satisfaction.

7.3.1 Key Elements of Process Formalization

For complex systems, relevant details to capture in process formalization include (Acuna et al. 2001; Dumas 2016):

- **Process Steps**: Defining the sequence of activities and tasks involved in the process.
- **Responsibilities**: Assigning clear roles and responsibilities for each activity and artifact.
- **Activity Inputs/Outputs**: Specifying the inputs required for each activity and the outputs generated.
- **Activity Activation/Termination conditions**: Defining the conditions that trigger the start and the end of each activity.
- **The State of an Artifact Before, During, and After the Execution of an Activity**: Tracking the evolution of artifacts throughout the process.
- **Activity Scheduling**: Defining the order and dependencies between activities, as well as opportunities for parallel execution.

7.3.2 Benefits of Process Formalization

Formalizing processes through appropriate notations and tools offers numerous benefits, as highlighted in the SEBoK:

- **Increased Efficiency**: Automating repetitive and error-prone manual tasks through process formalization significantly reduces the effort and time required for development activities (Sendall and Kozaczynski 2003).
- **Improved Consistency**: Formalized processes ensure consistent and synchronized activities across different abstraction levels and domains, reducing the risk of inconsistencies and errors, contributing to the overall system coherence as discussed in the SEBoK (Mens and Van Gorp 2006).
- **Enhanced Traceability**: Process formalization establishes explicit traceability links between activities, artifacts, and stakeholders, enabling better understanding and management of relationships and dependencies, aligning with the SEBoK's emphasis on traceability as a key principle of systems engineering (Galvao and Goknil 2007).
- **Reusability and Maintainability**: Well-defined and modular formalized processes can be reused across different projects and evolved over time, promoting the development of reusable assets and facilitating the maintenance of the process ecosystem, contributing to organizational learning and efficiency (SEBoK Editorial Board 2024).
- **Verification of Process Consistency and Correctness**: Formalized processes enable systematic verification of consistency and correctness with regard to desired objectives, ensuring alignment with project goals and requirements (Chrissis et al. 2011; SEBoK Editorial Board 2024).
- **Automation of Routine Tasks**: Expressing processes in a machine-readable language facilitates automation of routine tasks, reducing manual effort, improving efficiency, and minimizing the risk of human errors (Acuna et al. 2001).
- **Better Control of Complexity and Scaling**: Process formalization helps manage the complexity and scale of development processes by breaking them down into well-defined activities, artifacts, and responsibilities. Organizations can handle larger and more complex projects effectively, aligning with the SEBoK's focus on managing complexity as a core challenge in systems engineering (Osterweil 1997).

7.3.3 Methodology vs. Process

The SEBoK emphasizes the importance of having a robust and suitable process that fits the principles and best practices of the methodology, helping to improve process maturity and consistency (Friedenthal et al. 2014; SEBoK Editorial Board

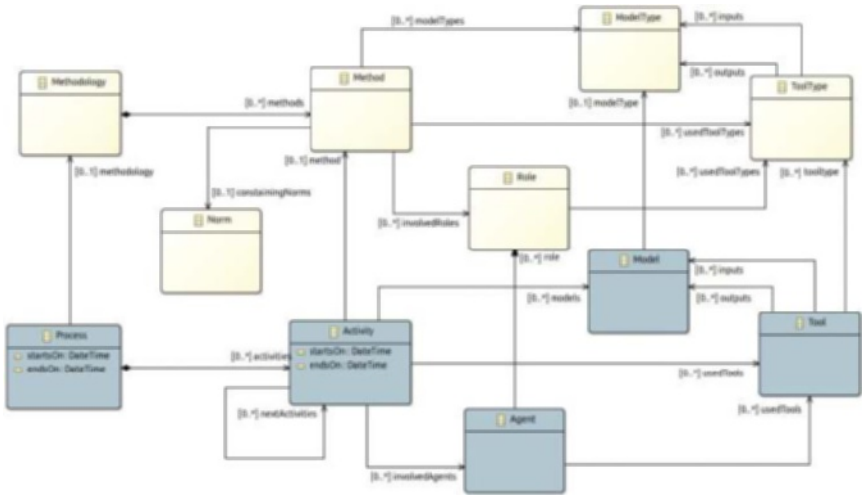


Figure 7.4 Process definition.

2024) as required by the CMMI. It makes a clear separation between *process* and *methodology*, noting that these are different but related concepts (Estefan 2007).

Figure 7.4 shows how methodology and process relate, highlighting their roles in good systems engineering. Although closely related, methodology and process in project management and system development have unique characteristics. A methodology is a collection of concepts, practices, and guidelines that suggest a general way to solve issues or achieve objectives. It is a general guide that describes the main strategy and approach for managing projects.

In contrast, a process represents the direct application of a methodology for a specific project or context. It involves a detailed application of the methodology's concepts, with clear start and finish dates, specific tasks, and designated resources. Whereas a methodology focuses on the theory, a process translates that theory into real actions, schedules, and results, taking into account the context of the process execution. Essentially, a methodology describes the "why" and "what" of project management, whereas a process answers the "how" and "when."

7.3.4 Automation in MBSE

In the area of MBSE, the SEBoK points out that some tasks can be automated to improve efficiency and consistency (Weilkiens 2011). By using model transformations, code generation, and automated analysis methods, MBSE can simplify tasks and reduce the need for manual work.

However, the SEBoK also stresses that automation needs to be balanced with human skills and decision-making to achieve good results (Madni and Sievers 2018). Successful automation and process formalization require recognizing which tasks can be automated while retaining human supervision where necessary. The SEBoK offers advice on how to strike this balance, taking into account the complexity of the task, the tools available, and the skills of the team.

7.3.5 Summary

Process formalization is key to effective MBSE, enabling organizations to create clear, consistent, and repeatable processes that support the entire system development lifecycle. By documenting and representing important tasks, materials and roles in an organized way, and applying automation where suitable, organizations can gain efficiency, consistency, and traceability in their MBSE practices. Striking the right balance between automation and human skills is essential for ensuring good results and getting the most out of the model-based approach.

7.4 Modeling Processes

The heart of MBSE is its emphasis on models, which are the key elements used to represent, check, and discuss systems over their lifespan. However, the usefulness of modeling goes beyond the system alone. In MBSE, we can also use modeling methods on the processes used in system development.

Process modeling helps us to better understand how processes are set up, how they act, and how they relate to each other, enabling us to find ways to improve them, streamline workflows, and boost overall productivity. Using MBSE for process modeling highlights the self-aware quality of the MBSE approach, where the concepts and methods used for system modeling are also used for the development processes.

7.4.1 Integrating Process Models with MBSE

To fully realize the benefits of process modeling in MBSE, it is essential to integrate process models with the overall MBSE approach. This integration can be achieved through several strategies:

- **Linking Process Models to System Models**: Establishing explicit links between process models and system models enables traceability and impact analysis (Broy 2010).
- **Using Process Models to Guide MBSE Activities**: Process models can serve as a roadmap for executing MBSE activities, providing guidance on the

sequence of tasks, the required inputs and outputs, and the roles and responsibilities involved (Estefan 2007).
- **Leveraging Process Metrics for MBSE Improvement**: Process models provide a foundation for collecting and analyzing process metrics, such as cycle time, defect rates, and resource usage. These metrics can be used to assess the effectiveness of the MBSE process and identify areas for improvement, enabling data-driven decision-making and continuous process optimization (Van Solingen et al. 2002; Alves et al. 2010).

7.4.2 Challenges and Best Practices

While process modeling offers significant benefits for MBSE, it also presents some challenges that organizations should be aware of:

- **Complexity and Maintenance**: As processes evolve and become more complex, maintaining and updating process models can become challenging (Rosemann and De Bruin 2005).
- **Balancing Standardization and Flexibility**: While process standardization is important for consistency and efficiency, it is also necessary to allow for flexibility and adaptation to project-specific needs (Tatikonda and Rosenthal 2000).
- **Ensuring Stakeholder Buy-in**: Implementing process modeling requires the engagement and support of various stakeholders (Kaindl 2000).

To address these challenges and ensure successful process modeling in MBSE, organizations should consider the following best practices:

- **Start Small and Iterate**: Begin with modeling key processes or subprocesses and gradually expand the scope over time. Use an iterative approach to refine and improve process models based on feedback and lessons learned.
- **Engage Stakeholders Early and Often**: Involve relevant stakeholders, including engineers, managers, and process owners, in the process modeling effort from the beginning. Seek their input and feedback to ensure that process models accurately reflect reality and meet their needs.
- **Use Appropriate Modeling Notations and Tools**: Select modeling notations and tools that are well-suited for the specific needs of the organization and the MBSE process. Consider factors such as ease of use, scalability, and integration with existing tools and platforms.
- **Provide Training and Support**: Offer training and support to help stakeholders understand and apply process modeling concepts and tools effectively. Foster a culture of continuous learning and improvement, encouraging the sharing of best practices and lessons learned.

- **Regularly Review and Update Process Models**: Establish a regular cadence for reviewing and updating process models to ensure they remain accurate and relevant. Use process performance data and feedback from stakeholders to identify areas for improvement and make necessary adjustments.

7.4.3 Summary

Process modeling is important to MBSE, as it helps organizations better understand their systems engineering and improve it over time. By representing key activities, roles, and products in the MBSE process, organizations can create a framework for systems engineering that encourages clarity, teamwork, and productivity.

Process models are useful for communication, analysis, automation, and improvement, leading to better execution of MBSE and quality systems. But process modeling needs careful attention to challenges and best practices application.

As MBSE evolves, combining process modeling with model-based approaches will be more crucial. By considering processes as important elements that can be modeled, examined, and improved, organizations can improve efficiency, flexibility, and quality in their systems engineering work.

7.5 Process Orchestration

Process orchestration, mentioned in the SEBoK, is like a conductor for the many activities of the MBSE. It involves managing how the various processes, subprocesses, and activities relate and interact in a complex systems engineering project (Peltz 2003). Process orchestration is mainly concerned with planning, carrying out, and tracking the workflow among various organizations, departments, and stakeholders to ensure smooth and efficient operations.

7.5.1 The Benefits of Process Orchestration

The SEBoK highlights some important benefits of process orchestration:

- **Streamlined and Optimized Workflow**: By automating and coordinating processes, process orchestration reduces manual interventions, eliminates redundancies, and minimizes delays, leading to improved efficiency and productivity (Leymann and Roller 2000). The result is a streamlined workflow where tasks flow seamlessly from one stage to the next, optimizing resource usage and accelerating project timelines.

- **Consistency and Standardization**: Process orchestration ensures consistent execution of processes according to predefined standards, rules, and best practices, promoting consistency across different teams and departments. This leads to improved quality and predictability of outcomes, as everyone involved in the project adheres to the same guidelines and procedures (van der Aalst and van Hee 2004).
- **Visibility and Transparency**: Process orchestration provides end-to-end visibility into the flow of work, enabling real-time monitoring and identification of bottlenecks and areas for improvement. This transparency fosters better decision-making, and empowers teams to proactively address issues and optimize their workflows for enhanced efficiency (Kloppmann et al. 2004; SEBoK Editorial Board 2024).
- **Agility and Adaptability**: In today's dynamic business environment, the ability to adapt quickly to changing needs and requirements is paramount. Process orchestration enables organizations to modify their processes in response to evolving contexts, ensuring responsiveness, and competitiveness (Smith 2003).
- **Collaboration and Integration**: Process orchestration facilitates collaboration and integration among different teams and systems, promoting seamless information flow, and breaking down silos. By fostering a more cohesive and collaborative environment, process orchestration enhances communication, knowledge sharing, and overall team productivity (Leymann and Roller 2000).
- **Compliance and Governance**: In regulated industries and complex projects, adherence to internal policies, industry standards, and regulatory requirements is critical. Process orchestration, by embedding compliance rules and controls within the workflow, helps organizations maintain compliance, and mitigate risks (Sadiq et al. 2007).
- **Scalability and Reusability**: As organizations grow and expand, their processes need to scale efficiently. Process orchestration enables the seamless scaling of processes, promoting the reuse of process components and accelerating time-to-market for new products, and services (van der Aalst and van Hee 2004).

7.5.2 Tools and Techniques

The SEBoK provides advice on the tools and methods required for successful process orchestration. This includes process modeling languages and workflow management systems. These tools help represent, automate, and keep track of complex workflows, which helps organizations improve their processes, and meet their objectives.

Some popular tools and techniques for process orchestration in MBSE include:

- **Business Process Modeling and Notation (BPMN)**: A graphical representation for specifying business processes in a business process model. BPMN provides a standard notation for process modeling, enabling clear communication, and understanding of processes across different stakeholders (Object Management Group 2011a).
- **Unified Modeling Language (UML) Activity Diagrams**: A standardized general-purpose modeling language in the field of software engineering. UML Activity Diagrams are commonly used to model the flow of control and data between activities in a process (Object Management Group 2011b).
- **Integration Frameworks**: Platforms that enable the integration and orchestration of different tools and systems used in the MBSE process. Examples include the Open Services for Lifecycle Collaboration (OSLC) (2022) and the Linked Data Platform (LDP) (W3C 2015).
- **Model-based Process Management Tools**: Specialized tools that support the definition, execution, and monitoring of MBSE processes using model-based approaches. These tools often provide features such as process modeling, simulation, and automation capabilities. Examples include Cameo Process Modeler and IBM Rational Method Composer.

Thanks to these tools and methods, organizations can design, implement and monitor their MBSE processes, and benefit from all the advantages of a model-based approach.

7.5.3 Summary

Process orchestration is very important for effective MBSE application. By planning and managing how processes connect and depend on each other, organizations can work better, reduce risks, and succeed in complex systems engineering tasks. It acts like a conductor, making sure that different MBSE activities operate smoothly together. This improves workflows, facilitates teamwork, and ultimately delivers high-quality systems that meet the needs of stakeholders and society.

7.6 Model Transformations

Model transformations are important in MBSE, as they contribute to automating tasks and enhancing process efficiency and maturity. The SEBoK describes model transformations as the automatic creation of a target model from a source model, using clear specifications called transformation rules (Kleppe et al. 2003). These rules explain how to convert elements from the source to the target language,

enabling the creation of new models or artifacts that retain the key information and meaning from the original model. The SEBoK notes that model transformations draw on consistent expertise and help ensure consistency throughout the development process (Wymore 2018; INCOSE 2023).

Model transformations are a complex way of changing models over time, similar to how chemicals react. Initial ideas, like raw materials, are carefully changed to create improved model outcomes. Just as chemical reactions follow specific rules and use energy, model transformations shift representations between different areas and levels of detail while keeping key meanings intact.

7.6.1 The Mechanics of Transformation

Transformation work involves detailed methods that change model structures without losing their core ideas, enabling us to move from abstract ideas to real applications. They can happen within the same modeling language (endogenous) or between different languages and areas (exogenous) (Mens and Van Gorp 2006), creating a flexible path for how systems are represented.

People play an important role in this process, not only by observing, but also by interpreting and guiding the transformation. They establish rules for changing models, check intermediate versions, address possible meaning issues, and make sure the core goals and needs are clearly reflected in each version of the model. This approach considers models as active, changeable objects instead of fixed ones. Every transformation is a careful translation that refines and develops the original model toward a final form, much like how a concept gradually develops into a real product through careful refinement and planning.

Figure 7.5 shows what model transformation looks like, where a transformation tool uses transformation rules on a source model to produce a target model in a different language or format.

7.6.2 Benefits of Model Transformations

The SEBoK emphasizes several key benefits of model transformations in the context of MBSE (Sendall and Kozaczynski 2003; Mens and Van Gorp 2006; Galvao and Goknil 2007; Hemel et al. 2010; Moha et al. 2009):

- **Increased Efficiency**: Automating repetitive and error-prone manual tasks significantly reduces the effort and time required for model-based development, contributing to improved productivity and faster development cycles.
- **Improved Consistency**: Model transformations ensure consistent and synchronized models across different abstraction levels and domains, reducing the risk of inconsistencies and errors and promoting a unified understanding of the system.

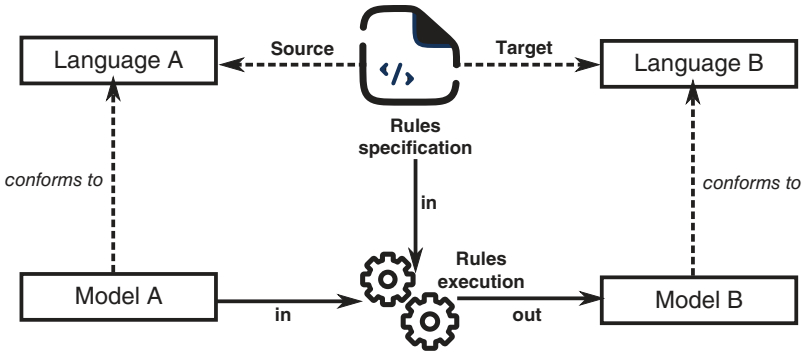


Figure 7.5 Transforming models.

- **Enhanced Traceability**: Model transformations establish explicit traceability links between source and target models, enabling better management of relationships and dependencies between different artifacts, aligning with the SEBoK's emphasis on traceability as a key principle of systems engineering.
- **Reusability and Maintainability**: Well-defined and modular model transformations can be reused across different projects and evolved over time, promoting the development of reusable assets and facilitating the maintenance of the modeling ecosystem, contributing to organizational learning and efficiency.

7.6.3 Applications of Model Transformations

Model transformations are a key enabler for MBSE, as they facilitate the automated generation of various artifacts from system models, including:

- **Documentation**: Generating documents such as requirements specifications, design documents, and user manuals directly from the system model ensures consistency and reduces the effort required for manual documentation.
- **Analysis Models**: Creating specialized models for analysis purposes, such as performance models or reliability models, allows for a deeper understanding of system behavior and facilitates informed decision-making.
- **Configuration Files/Source Code Generation**: Generating configuration files/code directly from the system model can automate the implementation process and ensure consistency between the design and implementation, reducing the risk of errors and inconsistencies.

Moreover, model transformations enable the reuse of domain knowledge captured in the form of reusable transformation rules. This is very useful in the MBSE context, where system models include the work of many engineering domains. By retaining and using specific knowledge through transformations, companies can make their work faster, more consistent, and of higher quality across various projects.

7.6.4 Limits of Model Transformations

Model transformations are powerful tools, but they have important limits that affect their effectiveness everywhere. The main issue is that we can't always guarantee we can reverse transformations. A change from model A to model B doesn't always mean we can change back from B to A exactly.

The key problem is keeping information and meaning. Transformations can clarify what we already know and how things work, but they can't always create new ideas or concepts that weren't in the original model or its context. Transformations often simplify information, which may lead to the loss of important context that isn't clearly defined in the transformation rules.

Additionally, transformations depend on how expressive and detailed the source and target modeling languages are. Sometimes, important details can be lost while translating, especially when switching between different models or ideas with varying levels of detail.

Experts therefore play an essential role in this process. They are not only there to check generated models but also to interpret, complete, and evaluate the generated/changed models. Transformations can help organize and clarify existing methods, but they can't replace human creativity, intuition, or the ability to come up with entirely new ideas.

7.6.5 Tools and Languages

The SEBoK gives advice on different model transformation languages and tools, such as Query/View/Transformation (QVT) and Atlas Transformation Language (ATL). These languages and tools help carry out efficient model transformations in MBSE. The listed languages and tools offer a common way to define and execute model transformations, making it easier for different MBSE tools and systems to work together. Some popular model transformation languages and tools include:

- **QVT**: A standard defined by the Object Management Group (OMG) for model transformation in the context of Model Driven Architecture (MDA) (Object Management Group 2016).

- **ATL**: A model transformation language and toolkit developed by the AtlanMod team, providing a hybrid approach that combines declarative and imperative constructs (Jouault et al. 2008).
- **Epsilon Transformation Language**: A rule-based model-to-model transformation language that is part of the Epsilon family of languages and tools for model management (Kolovos et al. 2008).
- **VIsual Automated model TRAnsformations**: A model transformation framework that provides a rule-based language for defining transformations, and a reactive transformation engine for executing them efficiently (Varró et al. 2016).

These languages and tools have different functions and abilities, serving various usages and preferences. Companies need to choose the right transformation tools carefully based on their needs, their current tools, and the knowledge and skills of their teams.

7.6.6 Challenges and Best Practices

While model transformations offer significant benefits, their effective application in MBSE also presents some challenges:

- **Complexity**: Defining and managing complex transformation rules can be challenging, especially for large and heterogeneous models. It is essential to adopt modular and hierarchical approaches to transformation design, promoting readability, maintainability, and reusability (Mens and Van Gorp 2006).
- **Performance**: Executing transformations on large models can be computationally expensive. Optimization techniques, such as incremental and parallel transformation execution, should be considered to improve performance (Hemel et al. 2010).
- **Debugging and Testing**: Debugging and testing model transformations can be complex, as they involve dealing with both the source and target models. Specialized debugging and testing techniques, such as model-based debugging and metamorphic testing, can help address these challenges (Moha et al. 2009).

To overcome these challenges and ensure the success of model transformations in MBSE, organizations should adopt best practices such as:

- **Defining Clear Transformation Requirements**: Clearly specify the purpose, inputs, outputs, and constraints of each transformation, ensuring that they align with the overall MBSE goals and processes (Sendall and Kozaczynski 2003).

- **Adopting Modular and Reusable Transformation Design**: Decompose complex transformations into smaller, reusable modules, promoting maintainability and facilitating the sharing and reuse of transformation knowledge across projects (Mens and Van Gorp 2006).
- **Leveraging Traceability Mechanisms**: Establish and maintain traceability links between source and target models, enabling impact analysis, change propagation, and consistency management throughout the transformation process (Galvao and Goknil 2007).
- **Continuously Monitoring and Optimizing Performance**: Regularly assess the performance of transformations, identifying bottlenecks and applying optimization techniques as needed to ensure efficient execution, even for large and complex models (Hemel et al. 2010).
- **Investing in Transformation Testing and Validation**: Develop comprehensive testing and validation strategies for transformations, leveraging techniques such as model-based testing, metamorphic testing, and property-based testing to ensure the correctness and reliability of the generated artifacts (Moha et al. 2009).

7.6.7 Summary

Model transformations are important for MBSE. They automate tasks, ensure consistency, and improve traceability in development. By using model transformations, companies can speed up their MBSE processes, reduce manual work, and improve the quality and efficiency of their systems engineering work.

The ability to create different artifacts from system models and to reuse domain knowledge through transformation rules reinforces the benefits of MBSE. This makes it a solid choice for dealing with the difficulties of modern system development. Still, using model transformations effectively needs careful attention to related challenges, and the use of best practices to make sure they work well.

As MBSE keeps evolving, model transformations are likely to play a more important role. Advances in transformation languages, tools, and techniques will ensure that organizations are better equipped to make full use of model-based methods, allowing them to produce high-quality, complex systems more effectively.

7.7 MBSE and Project Management

Successful management of complex system development projects depends not only on technical skills, but also on good project management. MBSE, which focuses on model-based creation, and project management, which is about planning and control, are two areas that, when well combined, can greatly improve project results. The meeting point of these fields creates strong cooperation, fostering a more effective, open, and successful way to develop systems.

7.7.1 Key Areas of Intersection

The interplay between MBSE and project management manifests in several key areas (Friedenthal et al. 2014; Holt 2021; Madni and Sievers 2018):

- **Requirements Management**: MBSE provides a structured approach to capturing and managing system requirements through models, aligning closely with project management's focus on scope definition and management. By leveraging MBSE models, project managers can effectively track requirement changes and their impact on project scope, schedule, and resources.
- **Risk Management**: MBSE models can be used to identify and analyze technical risks in system design and implementation. This information can be seamlessly integrated into the project risk management process, allowing for more comprehensive risk assessment and proactive mitigation planning.
- **Schedule and Resource Management**: The detailed system models in MBSE can inform more accurate estimates of task durations and resource needs. This information is crucial for project managers in developing realistic project schedules and resource allocation plans.
- **Stakeholder Communication**: MBSE models provide a visual and comprehensive representation of the system, serving as a powerful tool for stakeholder communication. Project managers can leverage these models to effectively communicate project status, issues, and decisions to both technical and nontechnical stakeholders.
- **Change Management**: Changes in system design or requirements, as captured in MBSE models, can be directly linked to project change management processes. This integration allows for a better assessment of the impact of changes on project parameters, enabling project managers to make informed decisions, and adjust plans accordingly.
- **Quality Management**: MBSE's focus on model-based verification and validation (V&V) aligns seamlessly with project quality management processes. The use of models for system testing and validation can provide more robust quality assurance throughout the project lifecycle.

7.7.2 Challenges and Opportunities

While the integration of MBSE and project management offers significant benefits, it also presents challenges (Estefan 2007, Friedenthal et al. 2014, Holt 2021, Madni and Sievers 2018):

- **Tool Integration**: Ensuring seamless integration between MBSE tools and project management software can be technically challenging, but is crucial for effective collaboration and data exchange between the two disciplines.

- **Skill Set Development**: Project managers need to develop a basic understanding of MBSE principles and practices to effectively leverage the insights from system or process models. Similarly, systems engineers need to appreciate project management constraints and methodologies to ensure that their work aligns with project goals and timelines.
- **Cultural Shift**: Integrating MBSE and project management often requires a cultural shift in organizations, moving toward a more collaborative and model-centric approach to project execution. This shift may involve changes in processes, roles, and responsibilities, requiring careful planning and change management efforts.

However, these challenges also present opportunities for innovation and improvement in both disciplines. For example, the development of integrated MBSE and project management platforms could revolutionize the way complex projects are managed, providing a unified environment for seamless collaboration and data sharing.

7.7.3 Future Directions

As both MBSE and project management continue to evolve, we can expect to see even closer integration between these disciplines. Emerging trends include:

- **Artificial Intelligence (AI)-driven Integration**: AI could be used to automatically update project schedules and resource allocations based on changes in MBSE models, enabling more proactive and responsive project management.
- **Virtual and Augmented Reality**: These technologies could be used to create immersive environments where project managers and stakeholders can interact with MBSE models, facilitating better understanding and decision-making.
- **Agile MBSE**: The integration of agile project management methodologies with MBSE practices could lead to more flexible and responsive system development processes, enabling faster adaptation to requirements and market conditions changes.

7.7.4 Summary

Merging MBSE and project management is a good area for improving the operation and efficiency of complex system development projects. By using the strengths of both domains, organizations can improve alignment between technical work and project management, leading to better results.

As these fields develop and connect, we can look forward to new methods and tools that will boost this partnership, creating a new phase of model-driven project management. The key is to create a team culture, invest in skill development, and seize opportunities for new ideas wherever MBSE meets project management.

7.8 Evaluating the Effectiveness of MBSE

Using MBSE is a significant investment for organizations, requiring both resources and a change in culture. To ensure this investment remains effective and justified, it is important to create a solid framework for measuring the effectiveness of MBSE methods (Friedenthal et al. 2014; Madni and Sievers 2018). Demonstrating the value of MBSE to those involved is key for maintaining continuous support and resources, as well as encouraging the use and improvement of model-based practices.

7.8.1 Key Performance Indicators

Assessing the effectiveness of MBSE must include a comprehensive set of Key Performance Indicators (KPIs) that cover different aspects of the systems engineering process and its effects on the organization (Holt 2021). These KPIs can be broadly categorized into four main areas:

- **Process Efficiency**: Metrics that measure improvements in the engineering process itself, such as reduced development time, improved productivity, and streamlined workflows.
- **Product Quality**: Indicators that assess the quality of the system being developed, such as reduced defect rates, improved reliability, and enhanced performance.
- **Project Performance**: Measures that evaluate the overall success of the project, such as adherence to schedule and budget, stakeholder satisfaction, and achievement of business objectives.
- **Organizational Impact**: Metrics that gauge the broader impact of MBSE on the organization, such as improved collaboration, knowledge sharing, and innovation.

7.8.2 Specific Metrics for Evaluation

The following list provides a set of specific metrics that can be used to evaluate MBSE effectiveness across the four key areas mentioned earlier (Friedenthal et al. 2014; Madni and Sievers 2018):

- **Requirements Traceability**:
  - Percentage of requirements traced to design elements
  - Time spent on requirements management and traceability
- **Model Consistency and Completeness**:
  - Number of model inconsistencies detected
  - Percentage of system elements represented in the model
- **Design Review Efficiency**:
  - Time spent in design reviews
  - Number of design issues identified early vs. late in the process
- **Reuse and Standardization**:
  - Percentage of reused model elements across projects
  - Adherence to modeling standards and best practices
- **Simulation and Analysis**:
  - Number of design alternatives evaluated through simulation
  - Time saved through model-based analysis vs. traditional methods
- **Documentation Generation**:
  - Time saved in generating project documentation
  - Consistency between model and documentation
- **Defect Detection and Resolution**:
  - Number of defects detected in early stages vs. late stages
  - Time to resolve defects
- **Stakeholder Communication**:
  - Stakeholder satisfaction with project communication
  - Time spent in stakeholder meetings and presentations
- **Project Timeline and Budget**:
  - Adherence to project schedule
  - Cost savings or overruns compared to traditional methods
- **Knowledge Management**:
  - Time spent searching for information
  - Reduction in duplicate work across teams

7.8.3 Evaluation Approaches

To effectively measure these metrics and evaluate MBSE impact, organizations should consider the following approaches (Friedenthal et al. 2014; Holt 2021):

- **Baseline Comparison**: Establish a baseline of metrics before MBSE implementation to enable before-and-after comparisons.
- **Pilot Projects**: Conduct pilot projects using MBSE methodologies and compare outcomes with similar projects using traditional methods.

- **Surveys and Interviews**: Gather qualitative feedback from team members, stakeholders, and customers on the perceived benefits and challenges of MBSE.
- **Tool Analytics**: Leverage built-in analytics from MBSE tools to track usage patterns, model quality, and process efficiency.
- **Continuous Monitoring**: Implement continuous monitoring of key metrics to track trends and improvements over time.
- **Post-project Reviews**: Conduct thorough reviews after project completion to assess the overall impact of MBSE on project outcomes.

7.8.4 Challenges in Measuring MBSE Effectiveness

While measuring MBSE effectiveness is crucial, it comes with challenges that organizations should be aware of (Friedenthal et al. 2014; Madni and Sievers 2018):

- **Long-term Nature of Benefits**: Some benefits of MBSE may only become apparent over multiple projects or extended timeframes.
- **Isolating MBSE Impact**: It can be difficult to attribute improvements solely to MBSE, as other factors may also influence project outcomes.
- **Qualitative Benefits**: Some benefits, such as improved communication or better decision-making, may be challenging to quantify.
- **Learning Curve**: Initial MBSE adoption may temporarily decrease efficiency as teams learn new tools and processes.

To deal with these issues, organizations should adopt a broad approach to evaluation that blends numerical data with personal contributions, examines both immediate and future effects, and considers the learning process tied to MBSE use. Consistent checking and updating of the evaluation system will keep it useful and efficient as the organization's MBSE practices develop.

7.8.5 Summary

Evaluating how well MBSE methods work is important for continuous improvements and justifying the spending on model-based strategies. By creating clear measurements, using both numerical and personal evaluation methods, and tackling the challenges of measurement, organizations can gather useful information on how MBSE affects their projects and overall engineering work.

The measurements and evaluation techniques in this section provide organizations with a starting point to measure their MBSE efficiency. Nevertheless, each organization should adjust its evaluation system to meet its own situation,

objectives, and development level. Regular checks and updates of these measurements will ensure that the evaluation method itself evolves as MBSE practices improve, giving the organization continuous benefits.

In the end, by adopting a mindset focused on measurement and continuous improvement, organizations can reinforce the benefits of MBSE and prepare for success in a more complex and competitive systems engineering environment.

7.9 MBSE Adoption: Challenges and Limitations

The use of MBSE marks a big change in the way systems engineering is conducted, bringing many advantages, such as greater efficiency, quality, and better teamwork. Still, this change has its own difficulties. Organizations just starting MBSE often face different problems that need careful thought and active plans to deal with. This section looks at some main problems and limitations associated with adopting MBSE, as well as possible ways of solving them.

7.9.1 Cultural Resistance and Skills Gap

A major challenge when adopting MBSE is the cultural resistance and the skills gap that can be present in organizations. Many engineers and managers are used to traditional document-based methods and might be reluctant to accept the MBSE's model-based approach. Moreover, the organization may lack sufficient MBSE knowledge and experience, which can slow down the correct use of model-based methods (Friedenthal et al. 2014; Madni and Sievers 2018).

To address these challenges, organizations can:

- **Provide Comprehensive Training Programs**: Equip engineers and managers with the necessary knowledge and skills to understand and apply MBSE principles and practices. Training should cover both the theoretical foundations and practical applications of MBSE, using real-world examples and hands-on exercises (Holt 2021).
- **Foster Leadership Support**: Secure buy-in from senior management to champion the adoption of MBSE and create a supportive environment for change. Leadership should communicate the strategic importance of MBSE and allocate the necessary resources for its successful implementation (Friedenthal et al. 2014).
- **Organize Teams Strategically**: Include both experienced modelers and traditional engineers in project teams to facilitate knowledge transfer and bridge the skills gap. Encourage collaboration and mentoring between team members to foster a culture of continuous learning and improvement (Madni and Sievers 2018).

- **Communicate the Benefits**: Clearly articulate the advantages of MBSE to all stakeholders, emphasizing its potential for improved efficiency, quality, and collaboration. Use case studies and success stories to demonstrate the tangible benefits of MBSE adoption (Holt 2021).

7.9.2 Integration with Existing Processes and Tools

Combining MBSE practices and tools with current systems engineering processes can be difficult. Legacy processes and data may not easily align with MBSE frameworks, requiring thoughtful planning and adjustment. Moreover, organizations may have devoted substantial resources to their current tools and infrastructure, making the transition to MBSE tools quite challenging (Estefan 2007).

To overcome these integration challenges, organizations can:

- **Conduct a Thorough Assessment**: Evaluate the current state of processes, tools, and data to identify potential integration points and challenges. This assessment should involve stakeholders from various departments to ensure a comprehensive understanding of the existing ecosystem (Madni and Sievers 2018).
- **Develop a Phased Implementation Plan**: Gradually introduce MBSE practices alongside existing processes, allowing for a smooth transition and minimizing disruption. Start with pilot projects to demonstrate the value of MBSE and refine the integration approach based on lessons learned (Friedenthal et al. 2014).
- **Leverage Integration Frameworks and Standards**: Use industry standards and frameworks, such as the OSLC and the LDP, to facilitate integration between MBSE tools and existing projects (Open Services for Lifecycle Collaboration 2022; W3C 2015). These standards provide a common language and protocol for data exchange and tool interoperability.
- **Invest in Tool Evaluation and Selection**: Carefully evaluate and select MBSE tools that offer robust integration capabilities with existing systems and data sources. Consider factors such as compatibility, scalability, and vendor support.
- **Address Data Migration and Transformation**: Develop strategies for migrating legacy data into MBSE models and ensuring data consistency across different tools and platforms. This may involve data cleaning, transformation, and validation processes to ensure the accuracy and integrity of the migrated data (Estefan 2007).

7.9.3 Model Management and Complexity

MBSE adds further complexity to managing many models and their connections throughout the system lifecycle. As models grow in size and complexity, keeping consistency, traceability, and version control becomes more difficult (Friedenthal et al. 2014; Madni and Sievers 2018). Moreover, working on models with teams in different locations and guaranteeing the integrity and security of models are major challenges.

To address these model management and complexity challenges, organizations can:

1. **Establish a Model Governance Framework**: Define clear policies, procedures, and responsibilities for model creation, modification, and approval. This framework should include guidelines for model naming conventions, metadata management, access control, and version control.
2. **Implement Model Repositories and Version Control**: Use model repositories and version control systems to manage the lifecycle of models effectively. These tools enable teams to collaborate on models, track changes, and maintain a single source of truth (Holt 2021).
3. **Develop Model Organization and Decomposition Strategies**: Establish guidelines for organizing and decomposing models into manageable and reusable units. This may involve creating model libraries, defining model interfaces, and establishing hierarchical relationships between models (Friedenthal et al. 2014).
4. **Leverage Model-based Collaboration Tools**: Implement collaboration tools that enable distributed teams to work together on models in real time, such as web-based modeling environments, and model review platforms. These tools facilitate communication, coordination, and consensus-building between team members (Madni and Sievers 2018).
5. **Ensure Model Security and Access Control**: Implement robust security measures to protect model data from unauthorized access, modification, and loss. This may involve encryption, backup and recovery mechanisms, and role-based access control to ensure that only authorized personnel can access and modify models.

7.9.4 V&V Challenges

As mentioned in Chapter 5, the V&V of complex system models present specific difficulties in MBSE. Typical V&V methods, which depend heavily on physical testing and document reviews, may not be sufficient to guarantee the accuracy and completeness of models (Friedenthal et al. 2014). Moreover, the higher level of abstraction in MBSE models can create challenges in tracking requirements and

confirming that the system works as intended. To overcome these V&V challenges, organizations can:

- **Develop Model-based V&V Strategies**: Establish a comprehensive V&V strategy that leverages the power of models to facilitate early and continuous V&V. This may involve model-based testing, formal verification techniques, and simulation-based validation approaches (Madni and Sievers 2018).
- **Integrate V&V Activities into the MBSE Process**: Embed V&V activities throughout the MBSE process, from requirements modeling to system integration and testing. This enables early detection and resolution of issues, reducing the cost and time associated with late-stage defects resolution.
- **Leverage Automated Verification Tools**: Use automated verification tools that can analyze models for consistency, completeness, and correctness. These tools can help identify potential issues and ensure that models adhere to established standards and best practices (Friedenthal et al. 2014).
- **Establish Traceability Between Requirements, Models, and Tests**: Maintain clear traceability links between requirements, models, and test cases to ensure that the system meets its intended functionality and performance. This traceability enables impact analysis and facilitates the identification of affected models and tests when requirements change (Holt 2021).
- **Foster Collaboration Between Systems Engineers and V&V Teams**: Encourage close collaboration between systems engineers and V&V teams throughout the MBSE process. This collaboration ensures that V&V considerations are incorporated early in the design process, and that models are developed with testability and verifiability in mind (Madni and Sievers 2018).

7.9.5 Scaling MBSE Adoption

Scaling MBSE use in large organizations and complex systems raises more difficulties. It is important to have a clear strategy and governance framework to maintain consistency, interoperability, and teamwork between various teams, projects, and domains (Friedenthal et al. 2014). Moreover, managing the complexity and number of models in large systems can put pressure on current tools and infrastructure. To address the challenges of scaling MBSE adoption, organizations can:

- **Develop an Enterprise-wide MBSE Strategy**: Establish an enterprise-wide MBSE strategy that defines the vision, goals, and roadmap for adoption. This strategy should align with the organization's overall business objectives and consider the specific needs and constraints of different business units and domains (Madni and Sievers 2018).
- **Establish a Center of Excellence (CoE)**: Create a dedicated MBSE CoE that provides guidance, support, and best practices for MBSE adoption across

the organization. The CoE should be responsible for developing standards, templates, and training materials, as well as facilitating knowledge sharing and collaboration among teams.

- **Implement a Federated Model Repository**: Deploy a federated model repository that enables teams to manage and share models across the enterprise. This repository should support version control, access control, and model integration capabilities to ensure consistency and traceability across the system lifecycle (Friedenthal et al. 2014).
- **Leverage Cloud-based MBSE Platforms**: Consider adopting cloud-based MBSE platforms that can scale to support large-scale systems and distributed teams. These platforms offer scalable computing resources, collaborative modeling environments, and integration with other enterprise systems (Holt 2021).
- **Foster a Culture of Continuous Improvement**: Encourage a culture of continuous improvement and learning across the organization. Regularly assess the effectiveness of MBSE practices, gather feedback from teams, and implement improvements based on lessons learned (Madni and Sievers 2018).

7.9.6 Summary

Adopting MBSE is a major change that needs careful planning and continuous improvement. Although the difficulties and limitations mentioned here may look hard, they can be overcome through smart thinking, strong organizational commitment and the use of best practices from other successful MBSE projects.

Organizations should consider MBSE adoption as a step-by-step method, starting with small pilot projects, and then gradually extending it to the whole organization. By providing in-depth training, gaining support from leaders, and setting up a solid governance structure, organizations can create a climate conducive to adopting MBSE, and take full advantage of it.

Exercise: Migrating to an MBSE process

You are the lead systems engineer for a complex satellite communication system project. The project is currently using traditional document-based systems engineering methods. Your organization has decided to transition to MBSE to improve efficiency, traceability, and collaboration. You are tasked with planning and executing this transition.

(Continued)

(Continued)

Part 1: Process Formalization

1. Referring to the process model in Figure 7.4, identify the key components you need to define for your MBSE process (e.g. Activities, Methods, Tools, Roles).
2. Outline a strategy to formalize the existing engineering processes into an MBSE framework. Consider:
   - How will you map current activities to MBSE activities?
   - What new activities might need to be introduced?
   - How will you ensure that the new process aligns with existing methodologies and norms?

Part 2: Tool Selection and Integration

1. Based on the *ToolType* and *Tool* elements in the process model:
   - Propose a set of MBSE tools that would be suitable for your satellite communication system project.
   - Explain how these tools align with the *ModelTypes* required for your project.
2. Develop a plan for tool integration, considering:
   - Data exchange between tools
   - Version control and configuration management
   - User training and adoption strategies

Part 3: Role Definition and Training

1. Using the role element from the process model:
   - Define the key roles required in your MBSE process.
   - Describe how these roles differ from the current roles in your traditional process.
2. Create a training plan to help team members transition to their new MBSE roles, addressing:
   - Required skills and competencies
   - Training methods and resources
   - Timeline for role transition

Part 4: Model Development Strategy

1. Outline a strategy for developing and managing the models required in your MBSE process:
   - Identify the key model types needed for the satellite communication system.

- Describe how these models will interact and evolve throughout the project lifecycle.

2 Propose a method for ensuring model consistency and traceability across different aspects of the system.

Part 5: Measurement and Evaluation

1 Develop a set of metrics to measure the effectiveness of your MBSE process implementation, considering:
   - Process efficiency
   - Model quality and consistency
   - Stakeholder satisfaction
   - Project outcomes (e.g. reduced errors, improved time-to-market)
2 Create a plan for collecting and analyzing these metrics throughout the transition process.

Part 6: Challenge Mitigation

Identify three major challenges you anticipate in the transition to MBSE, based on the concepts presented in this chapter. For each challenge:

1 Describe the potential impact on the project.
2 Propose a mitigation strategy.
3 Explain how you would measure the effectiveness of your mitigation strategy.

Part 7: Reflection

Write a brief reflection (300–400 words) on:

1 The key differences between traditional and MBSE processes in the context of your satellite communication system project.
2 The potential long-term benefits and risks of transitioning to MBSE.
3 How the structured approach provided by the process model (as shown in the image) can help in managing the complexity of the transition.

7.10 A Software Development Process Example

Figure 7.6 gives a basic but clear representation of a software engineering process that includes important MBSE concepts. This section will go through the process, noting the mix of manual and automated tasks, the opportunities for automation with model transformations, and the iterative nature of the development cycle.

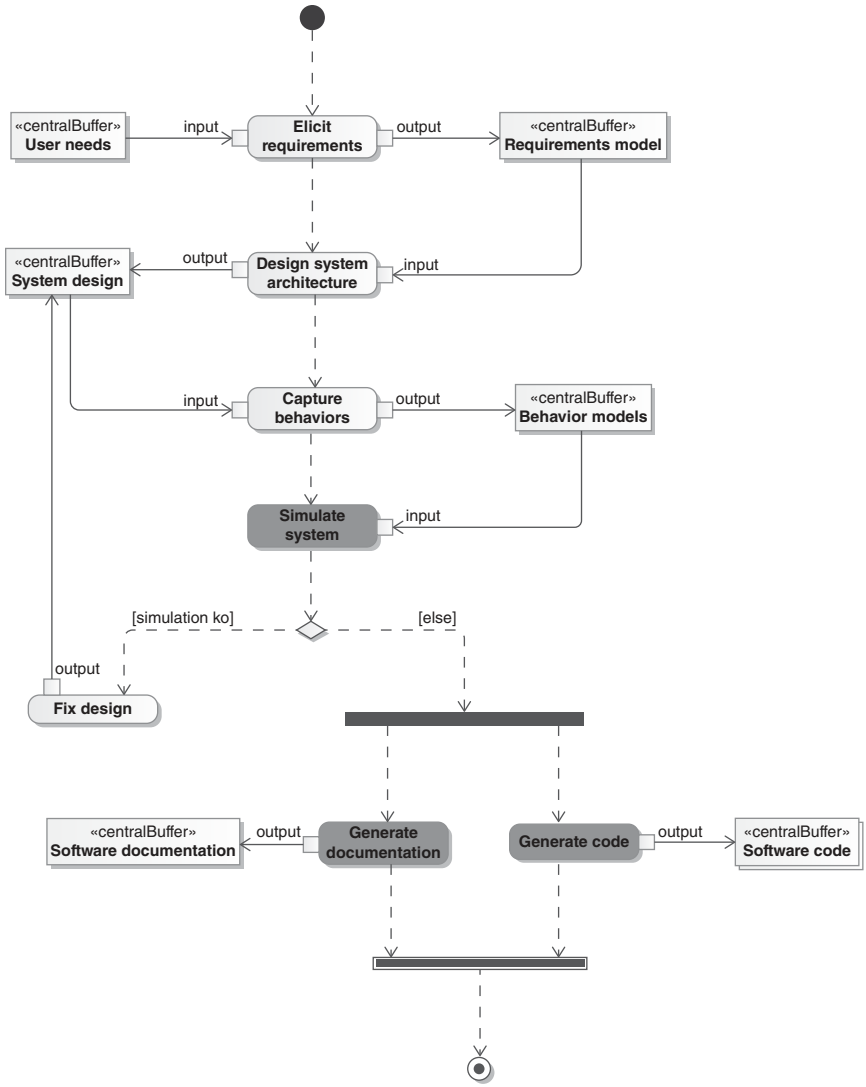


Figure 7.6 Simplified software engineering process with MBSE.

Here is a short description of each task depicted in this activity diagram; tasks in red are those supported by automation (model transformation):

- **Elicit Requirements**: The initial step involves gathering and documenting user needs. It requires human interaction, communication skills, and the ability to interpret stakeholder needs into formal requirements. The output is

typically a requirements document, often supplemented by Systems Modeling Language (SysML) requirements diagrams for visual representation and traceability.

- **Design System Architecture**: While high-level architectural decisions require human expertise, MBSE tools can assist in creating and analyzing architectural models. The output includes SysML block definition diagrams, internal block diagrams, and parametric diagrams, capturing the system's structure, interfaces, and constraints.
- **Capture Behaviors**: Defining system behaviors involves human creativity but can be supported by modeling tools for creating state machines, activity diagrams, and sequence diagrams. The output is a collection of behavior models describing how the system should function under various conditions.
- **Simulate System**: This step leverages the behavior models to run simulations, allowing for early validation of system behavior without physical prototypes. The potential for automation here is high, with model transformations enabling the generation of executable simulations from behavior models.
- **Fix Design**: Based on simulation results, engineers may need to revisit and adjust the system design. This step highlights the iterative nature of the process, allowing for continuous refinement and improvement based on feedback and analysis.
- **Generate Documentation and Code**: Automated generation of software documentation and code skeletons from models, reducing manual effort and ensuring consistency. Model-to-text transformation tools can be leveraged to generate various artifacts, including requirements specifications, design documents, user manuals, and source code.

Here are the main key observations we can make about this model:

- **Automation Potential**: The process leverages automation in several key areas, particularly in tasks involving model analysis, transformation, and artifact generation. This aligns with the MBSE principle of using models as the primary means of information exchange and the source for generating various project artifacts.
- **Manual Tasks**: Activities requiring human judgment, creativity, and stakeholder interaction remain manual. These include requirements elicitation, high-level architectural design decisions, and design refinement based on simulation results.
- **Iterative Nature**: The process incorporates feedback loops, allowing for refinement of the design based on simulation results. This iterative approach is crucial in MBSE, enabling continuous validation and improvement of the system design.

- **Model-centric Approach**: The central role of models (requirements, system design, behavior) in driving subsequent activities exemplifies the core MBSE principle of using models as the primary artifact throughout the development lifecycle.
- **Integration of V&V**: While not explicitly shown in the diagram, the simulation step represents an early validation activity. In a more comprehensive process, additional V&V steps would be integrated throughout, including model checking, formal verification, and traditional testing of the generated code.
- **Traceability**: The flow of information between steps (e.g. from requirements to design to behavior) implies traceability between different aspects of the system, another key principle of MBSE.

This simple model shows how MBSE concepts can be used to facilitate the software engineering process, using automation when it can, while still valuing human skill in important decision-making tasks. It also highlights the possibility of early checking through simulation and the repetitive steps of system development, which are important benefits of MBSE.

Moreover, the model presented in this figure could be run by an orchestration tool, and serve as a reference for the different stakeholders involved in this process, especially using notification systems, automatic model transformations and analyses.

Exercise: Modeling a software design process

Create a process model for a simplified software design process based on Figure 7.6, using a modeling language of your choice (e.g. BPMN, UML Activity Diagram, or SysML Activity Diagram), and perform the following tasks:

1. Identify and list all the major activities in the process, including those not explicitly shown in the image (e.g. testing, deployment).
2. Determine the sequence and dependencies between activities.
3. Identify decision points and potential loops in the process (e.g. the loop back to "Fix Design" after simulation).
4. Define inputs and outputs for each activity.
5. Assign roles or responsibilities for each activity (e.g. Systems Engineer, Software Developer, Project Manager).
6. Identify any parallel activities that can be performed simultaneously.
7. Include at least two additional activities not shown in the original image that you think would improve the process.

8 Annotate your model with brief explanations of key decision points and any assumptions you've made.
9 Identify at least three areas in your process model where automation could be applied to improve efficiency.

7.11 Conclusion

In this section, we have looked at the basic concepts and real-world uses of MBSE, highlighting how it can change the way we handle complex issues and work together. Using formal models as key elements, MBSE supports a more organized and methodological way of approaching systems engineering, helping teams to manage the complex connections and dependencies found in modern systems.

The discussions highlighted several key advantages of MBSE:

- **Enhanced Communication and Collaboration**: By providing a common language and visual representation of the system, MBSE fosters a shared understanding among stakeholders, bridging the gap between technical and nontechnical participants.
- **Iterative Design and Validation**: The use of models allows for early validation and iterative refinement of design decisions, reducing the risk of costly errors and rework in later stages.
- **Improved System Quality and Traceability**: MBSE enhances the consistency, traceability, and verifiability of system designs, contributing to higher overall quality and alignment with stakeholder needs.
- **Adaptability to Change**: The iterative nature of MBSE supports rapid exploration of design alternatives and adaptation to evolving requirements and technological advancements.

Nevertheless, this section also highlighted the difficulties that remain, such as managing the complexity of linked models and making sure data is accurate and accessible. These issues emphasize the need for further research and development of strategies and tools to improve MBSE's effectiveness.

In the next chapter, we will examine a specific case study that illustrates how MBSE can be applied in a real-life situation. The EVA project, an innovative initiative to create an autonomous mobility service, demonstrates how the combination of AI with MBSE can solve classic scenario management problems, and foster innovation in complex systems engineering. This case study not only shows the real benefits of MBSE, but also points out areas for future investigation and improvement, paving the way for ongoing progress in this field.

Questions

1. What is the first step in the process model shown in Figure 7.6?
   - **a** Design System Architecture
   - **b** Elicit Requirements
   - **c** Capture Behaviors
   - **d** Simulate System
2. Which of the following is a key benefit of process orchestration in MBSE?
   - **a** Increased ambiguity in system design
   - **b** Streamlined and optimized workflow
   - **c** Reduced precision in system specifications
   - **d** Increased complexity in documentation
3. What is the primary purpose of model transformations in MBSE?
   - **a** To increase the expressiveness of natural language
   - **b** To provide a flexible framework for system design
   - **c** To automate the conversion between different modeling formalisms
   - **d** To enhance the aesthetic appeal of models
4. Which approach combines elements of both top-down and bottom-up strategies in MBSE?
   - **a** Waterfall approach
   - **b** Agile approach
   - **c** Meet-in-the-middle approach
   - **d** Spiral approach
5. In the context of MBSE, what is the role of scenarios in the "4+1" view model?
   - **a** To provide a single perspective on a system
   - **b** To drive the understanding, design, and validation of the system
   - **c** To eliminate the need for multiple models
   - **d** To simplify the system architecture
6. Which of the following is an automated task in the process model shown in the image?
   - **a** Elicit Requirements
   - **b** Design System Architecture
   - **c** Capture Behaviors
   - **d** Generate Documentation

7 What is the main advantage of using process modeling in MBSE?
   a It simplifies the development process.
   b It provides a visual representation of the MBSE process.
   c It eliminates the need for project management.
   d It reduces the number of system models.
8 How does MBSE enhance traceability in the development process?
   a By eliminating the need for documentation
   b By establishing explicit links between different artifacts
   c By using natural language exclusively
   d By reducing the number of system models
9 What is the significance of the "Fix Design" step in the process model?
   a It provides a framework for understanding computational models.
   b It simplifies the process of model creation.
   c It allows for iterative refinement based on simulation results.
   d It eliminates the need for formal languages.
10 Which of the following is a challenge associated with MBSE adoption?
   a Increased ambiguity in system specifications
   b Reduced precision in system models
   c Cultural resistance and skills gap
   d Decreased analyzability of system properties

Answers

1-b; 2-b; 3-c; 4-c; 5-b; 6-d; 7-b; 8-b; 9-c; 10-c

References

Acuna, S.T., De Antonio, A., Ferre, X., et al. The software process: modeling, evaluation and improvement. In *Handbook of Software Engineering and Knowledge Engineering: Volume I: Fundamentals* 2001:193–237. World Scientific.

Alves, V., Niu, N., Alves, C., et al. Requirements engineering for software product lines: a systematic literature review. *Information and Software Technology* 2010:52(8):806–820.

Broy, M. A logical basis for component-oriented software and systems engineering. *The Computer Journal* 2010:53(10):1758–1782.

Chrissis, M.B., Konrad, M., and Shrum, S. *CMMI for Development: Guidelines for Process Integration and Product Improvement*. SEI Series in Software Engineering. Pearson Education, 2011. ISBN 9780132700443. https://books.google.fr/books?id=dNyv0h91BJIC.

Dumas, J.D. *Computer Architecture: Fundamentals and Principles of Computer Design*, 2e. CRC Press, 2016. ISBN 9781498772747. https://books.google.fr/books?id=TZ6VDQAAQBAJ.

Estefan, J.A. Survey of model-based systems engineering (MBSE) methodologies. *INCOSE MBSE Focus Group* 2007:25(8):1–12.

Friedenthal, S., Moore, A., and Steiner, R. *A Practical Guide to SysML: The Systems Modeling Language*. The MK/OMG Press. Elsevier Science, 2014. ISBN 9780128002025. https://books.google.fr/books?id=97XfoAEACAAJ.

Galvao, I. and Goknil, A. Survey of traceability approaches in model-driven engineering. In *11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007)* 2007:313–313. IEEE.

Hemel, Z., Kats, L.C., Groenewegen, D.M., et al. Code generation by model transformation: a case study in transformation modularity. *Software & Systems Modeling* 2010:9:375–402.

Holland, J.H. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. Bradford book. MIT Press, 1992. ISBN 9780585038445. https://books.google.fr/books?id=cyV7nQEACAAJ.

Holt, J. *Systems Engineering Demystified: A Practitioner's Handbook for Developing Complex Systems Using a Model-Based Approach*. Packt Publishing, 2021. ISBN 9781838985448. https://books.google.fr/books?id=EL0YEAAAQBAJ.

INCOSE. *INCOSE Systems Engineering Handbook*. Wiley, 2023. ISBN 9781119814290. https://books.google.fr/books?id=HhjBEAAAQBAJ.

Jouault, F., Allilaire, F., Bézivin, J., et al. ATL: a model transformation tool. *Science of Computer Programming* 2008:72 (1–2):31–39.

Kaindl, H. A design process based on a model combining scenarios with goals and functions. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans* 2000:30(5):537–551.

Kirkpatrick, S., Gelatt Jr, C.D., and Vecchi, M.P. Optimization by simulated annealing. *Science* 1983:220(4598):671–680.

Kleppe, A.G., Warmer, J.B., and Bast, W. *MDA Explained: The Model Driven Architecture: Practice and Promise*. Object technology. Addison-Wesley, 2003. ISBN 9780321194428. https://books.google.fr/books?id=-uodwVPBG6YC.

Kloppmann, M., Konig, D., Leymann, F., et al. Business process choreography in webSphere: combining the power of BPEL and J2EE. *IBM Systems Journal* 2004:43(2):270–296.

Kolovos, D.S., Paige, R.F., and Polack, F.A. The epsilon transformation language. In *Theory and Practice of Model Transformations: First International Conference, ICMT 2008, Zürich, Switzerland, July 1–2, 2008 Proceedings 1* 2008:46–60. Springer.

Kossiakoff, A., Biemer, S.M., Seymour, S.J., et al. *Systems Engineering Principles and Practice*. Wiley Series in Systems Engineering and Management. Wiley, 2020. ISBN 9781119516668. https://books.google.fr/books?id=gGzoDwAAQBAJ.

Kruchten, P.B. The 4+1 view model of architecture. *IEEE Software* 2002:12(6):42–50.

Leymann, F. and Roller, D. *Production Workflow: Concepts and Techniques*. Prentice Hall PTR, 2000. ISBN 9780130217530. https://books.google.fr/books?id=Xc8eAQAAIAAJ.

Madni, A.M. and Sievers, M. Model-based systems engineering: motivation, current status, and research opportunities. *Systems Engineering* 2018:21(3):172–190.

Maier, M.W. *The Art of Systems Architecting*. Systems Engineering. CRC Press, 2009. ISBN 9781420079142. https://books.google.fr/books?id=AWLMBQAAQBAJ.

Mens, T. and Van Gorp, P. A taxonomy of model transformation. *Electronic Notes in Theoretical Computer Science* 2006:152:125–142.

Moha, N., Mahé, V., Barais, O., et al. Generic model refactorings. In *International Conference on Model Driven Engineering Languages and Systems* 2009:628–643. Springer.

Object Management Group. Business process model and notation (BPMN) version 2.0, 2011a. https://www.omg.org/spec/BPMN/2.0.

Object Management Group. OMG Unified Modeling Language (OMG UML), superstructure, August 2011b. https://www.omg.org/spec/UML/2.5.1.

Object Management Group. Meta object facility (MOF) 2.0 query/view/transformation specification, 2016. https://www.omg.org/spec/QVT/1.3.

Open Services for Lifecycle Collaboration. Open Services for Lifecycle Collaboration (OSLC), 2022. https://open-services.net/.

Osterweil, L.J. Software processes are software too, revisited: an invited talk on the most influential paper of ICSE 9. In *Proceedings of the 19th International Conference on Software Engineering* 1997:540–548.

Osterweil, L. Software processes are software too. In *Engineering of Software: The Continuing Contributions of Leon J. Osterweil* 2011:323–344. Springer.

Peltz, C. Web services orchestration and choreography. *Computer* 2003:36(10):46–52.

Rosemann, M. and De Bruin, T. Towards a business process management maturity model. In *ECIS 2005 Proceedings of the Thirteenth European Conference on Information Systems* 2005:1–12. Verlag and the London School of Economics, 2005.

Saaty, T.L. How to make a decision: the analytic hierarchy process. *European Journal of Operational Research* 1990:48(1):9–26.

Sadiq, S., Governatori, G., and Namiri, K. Modeling control objectives for business process compliance. In *Business Process Management: 5th International Conference, BPM 2007*, Brisbane, Australia, September 24–28, 2007. *Proceedings 5* 2007: 149–164. Springer.

Sage, A.P. and Rouse, W.B. *Handbook of Systems Engineering and Management*. Wiley Series in Systems Engineering and Management. Wiley, 2014. ISBN 9780470083536. https://books.google.fr/books?id=eFRwQuzPnEcC.

SEBoK Editorial Board. *The Guide to the Systems Engineering Body of Knowledge (SEBoK)*. INCOSE, 2024.

Sendall, S. and Kozaczynski, W. Model transformation: the heart and soul of model-driven software development. *IEEE Software* 2003:20(5):42–45.

Smith, H. Business process management—the third wave: business process modelling language (BPML) and its pi-calculus foundations. *Information and Software Technology* 2003:45(15):1065–1069.

Tatikonda, M.V. and Rosenthal, S.R. Successful execution of product development projects: balancing firmness and flexibility in the innovation process. *Journal of Operations Management* 2000:18(4):401–425.

Tzeng, G.H. and Huang, J.J. *Multiple Attribute Decision Making: Methods and Applications*. CRC Press, 2011. ISBN 9781439861585. https://books.google.fr/books?id=Uu_RBQAAQBAJ.

van der Aalst, W. and van Hee, K.M. *Workflow Management: Models, Methods, and Systems*. Cooperative information systems. MIT Press, 2004. ISBN 9780262720465. https://books.google.fr/books?id=O1xW1_Za-I0C.

Van Solingen, R., Basili, V., Caldiera, G., et al. Goal question metric (GQM) approach. *Encyclopedia of Software Engineering*, 2002.

Varró, D., Bergmann, G., Hegedüs, Á., et al. Road to a reactive and incremental model transformation platform: three generations of the VIATRA framework. *Software & Systems Modeling* 2016:15:609–629.

Weilkiens, T. *Systems Engineering with SysML/UML: Modeling, Analysis, Design*. The MK/OMG Press. Elsevier Science, 2011. ISBN 9780080558318. https://books.google.fr/books?id=LKC9F5gQt3AC.

World Wide Web Consortium (W3C). Linked Data Platform 1.0, 2015. https://www.w3.org/TR/ldp/.

Wymore, A.W. *Model-Based Systems Engineering*. Systems Engineering. CRC Press, 2018. ISBN 9781351431088. https://books.google.fr/books?id=00taDwAAQBAJ.

8

Case Study

8.1 Overview

In the field of complex systems engineering, striking a balance between new ideas and safety is a tough task, especially with rapidly evolving technology such as autonomous cars. Model-based systems engineering (MBSE) provides a useful framework for managing these difficulties, enabling more effective, collaborative, and thorough methods for system design and development. Through its emphasis on models as central artifacts, MBSE promotes:

- **Enhanced Communication and Collaboration**: Providing a common language and visual representation of the system, fostering shared understanding among diverse stakeholders (as discussed in Chapter 3).
- **Faster Development Cycles**: Enabling early validation and iterative refinement of design decisions through simulation and analysis (as discussed in Chapter 4).
- **Reduced Development Costs**: Minimizing the risk of costly errors and rework by identifying potential issues in the early stages (as discussed in Chapter 2).
- **Improved System Quality**: Enhancing the consistency, traceability, and verifiability of the system design, contributing to higher overall quality (as discussed in Chapter 5).

However, the inherent complexities of designing autonomous mobility solutions raise several challenges:

- **Capturing Diverse Perspectives**: Ensuring a shared understanding among stakeholders with diverse knowledge, expertise, and priorities.

- **Anticipating Emergent Behaviors**: Modeling the complex interactions between autonomous vehicles, infrastructure, users, and other systems, and anticipating potential unforeseen outcomes.
- **Adapting to Rapid Change**: Ensuring the system's design remains robust and relevant as technology advances and operational contexts evolve.

To solve these problems, the "Éco-mobilité par Véhicules Autonomes (EVA)" project – an important attempt to create a new self-driving vehicle mobility service, presented at AFIS, the French chapter of INCOSE, in 2018 – used AI to improve MBSE by focusing on automating the creation of relevant covering scenarios. This chapter will explain how EVA's combined methodology helped stakeholders make better decisions, reduce risks, and meet project objectives, showing the powerful impact of AI and MBSE on the future of autonomous driving mobility.

8.2 Project Overview and Motivation

The EVA project was launched to tackle serious issues such as traffic jams, pollution, and accidents by creating a smart mobility service. This large initiative gathered important players from the French mobility industry, forming a collaborative effort. As shown in Figure 8.1, each partner contributed their current systems and skills while keeping their proprietary information private.

The main objective was to create, use, manage, and maintain an SoS that would provide safer, more efficient, and greener mobility services. In this teamwork

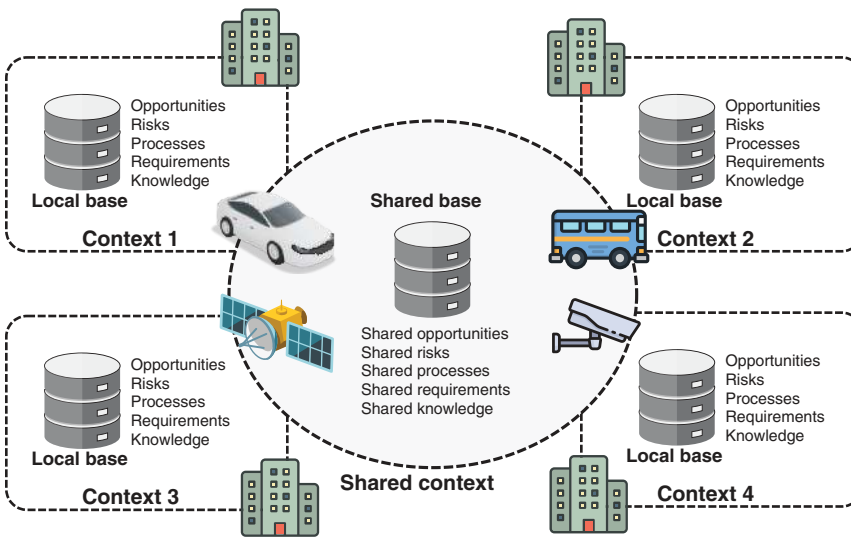


Figure 8.1 Extended enterprise implementing system of systems (SoS).

configuration, partners contributed their systems "as-is" and listed their needs for future improvements. Each partner worked within their specific situation, taking into account legal, environmental, political, and economic factors, which influenced the feasibility of the project and the importance of each partner's role.

In this fast-changing context, where rules and risks change quickly, having a shared real-time vision was essential for sound decision-making. The project aimed to develop a robust and flexible system that could address these fast-changing challenges.

8.2.1 Motivating MBSE

From the outset, the EVA project demonstrated the need for a structured systems engineering methodology, aligning closely with the core principles and benefits of MBSE as described in Chapter 2:

- **Complexity**: The project encompassed a challenging SoS involving autonomous vehicles, complex infrastructure, communication networks, and diverse stakeholders. Managing these interdependencies required a structured approach to model, analyze, and integrate individual contributions into a coherent system representation (as discussed in Chapter 3).
- **Collaboration and Communication**: The distributed nature of the extended enterprise required clear and consistent communication across organizational boundaries. MBSE's emphasis on models as primary communication artifacts helped overcome language barriers and fostered a shared understanding of the evolving system and its requirements (as discussed in Chapter 3).
- **Adapting to Change**: The rapid pace of technological advancement in the autonomous vehicle sector, coupled with evolving user needs and regulatory contexts, required a flexible design approach. MBSE's iterative nature, supported by modeling tools and simulations, facilitated rapid exploration of design alternatives and system refinement (as discussed in Chapter 4).
- **Risk Reduction and Value Optimization**: Safety was a paramount concern. The rigorous analysis and validation techniques supported by MBSE, combined with cost and performance optimization, aimed to deliver a system that met high safety standards and user expectations (as discussed in Chapter 5).

8.2.2 Key Capabilities Targeted

The EVA project prioritized five key capabilities to improve user experience and societal benefits:

- **Enhanced Personal Mobility**: Offering seamless, efficient, and convenient transportation options that integrate with existing infrastructure and reduce travel time.

- **System Integrity**: Ensuring safe and reliable operation of autonomous vehicles, safeguarding user well-being, and maintaining system integrity under unforeseen circumstances.
- **Improved Environmental Impact**: Promoting sustainability by incorporating electric vehicles, optimizing routes for efficiency, and minimizing environmental impacts.
- **Service Continuity**: Providing consistent and reliable service, mitigating disruptions from technical issues or operational constraints.
- **Congestion Reduction**: Designing the service to reduce urban traffic congestion, optimizing traffic flow, and improving transportation efficiency.

These abilities were examined using measures such as average waiting times, accident reduction, fuel consumption, and road used.

8.3 Blending "Meet-in-the-middle" and AI

The EVA project used the "meet-in-the-middle" approach to strike a good balance between the advantages of both top-down and bottom-up approaches in system design. This approach, as discussed in Chapter 7, fits well with the complex nature of the extended enterprise and the need to use existing resources and knowledge:

1. **Top-down Analysis**: The project began with a top-down analysis of the five key capabilities targeted by the mobility service. These high-level capabilities were systematically decomposed into increasingly granular sub-capabilities and requirements, providing a clear structure for defining the system's intended behavior and performance.
2. **Bottom-up Assessment**: Concurrently, a bottom-up assessment of existing systems within each participating organization was conducted. This analysis documented the current state of technology, identified limitations and constraints, and explored opportunities for integrating existing components and infrastructure into the envisioned mobility service.
3. **Gap Analysis**: Bridging the gap between the idealized vision of the system (top-down) and the constraints of the real world (bottom-up) was achieved through gap analysis, as illustrated in Figure 7.2. This analysis identified areas where existing systems needed upgrades or re-engineering and where new components or functionalities were necessary.

The information gathered during these three phases was crucial in guiding design decisions and ensuring that the resulting system could realistically meet stakeholder requirements and project feasibility. While the "meet-in-the-middle" methodology provides a solid foundation, the EVA project pushed the boundaries

of conventional MBSE by integrating Artificial Intelligence (AI) to address these key limitations:

- **Overcoming Expert-driven Scenario Limitations**: Traditional scenario creation in MBSE often relies on the limited knowledge and experiences of stakeholders, leading to scenarios that may not fully represent real-world complexities or may omit critical factors influencing system design and performance.
- **Harnessing AI for Scenario Generation**: The EVA project pioneered the use of AI in scenario generation. AI techniques enabled the analysis of massive datasets, the identification of patterns and relationships within data, and the generation of large quantities of realistic, diverse, and data-informed scenarios for testing the envisioned mobility system.
- **A Paradigm Shift to Data-driven Decisions**: By incorporating AI into scenario management, the EVA project transitioned from an "opinion-based" systems engineering approach to a truly "data-driven" one, empowering stakeholders to make more informed design decisions.

The choice of combining a classic systems engineering methodology with advanced AI laid the foundations for the EVA project's new ideas in MBSE.

Our starting point was the ARChitecture Analysis & Design Integrated Approach (ARCADIA) methodology, supported by the Capella modeling tool developed by Thales. The ARCADIA methodology is summarized in the following section.

8.4 The Arcadia Methodology

ARCADIA is a model-based methodology for systems engineering. It is supported by the Capella modeling tool, which is open-source. This methodology provides a way to design and analyze complex systems in a structured manner, encourages teamwork among stakeholders, and helps to bridge the gap between user requirements and technical specifications. ARCADIA's primary objectives are to:

- Enhance understanding of user needs
- Facilitate the design and analysis of complex systems through specialized notations and automated refinements
- Create a shared vision among project stakeholders
- Formalize interactions with subcontractors

ARCADIA follows a systematic approach divided into five distinct phases, each addressing specific aspects of system design, described in the following subsections.

8.4.1 Operational Analysis (Who? Why?)

This initial phase focuses on stakeholder expectations and the operational context of the system:

- Identify key players, their activities, and interactions
- Analyze operational scenarios to gain deeper insights into stakeholder needs
- Define the operational environment and constraints

The operational analysis helps establish a solid foundation for subsequent phases by ensuring a thorough understanding of business needs and context.

8.4.2 System Analysis (What?)

Building upon the operational analysis, this phase translates stakeholder needs into system capabilities:

- Derive operational capabilities into missions and system capabilities
- Identify functions that enable the realization of these capabilities
- Define function interfaces and constraints
- Identify critical functional chains

System analysis bridges the gap between user needs and system functionality, providing a clear picture of what the system must accomplish.

8.4.3 Logical Architecture (How?)

The logical architecture phase focuses on structuring the system's functions into logical components:

- Organize functions into a coherent logical architecture
- Consider capitalization and reuse objectives
- Define logical interfaces between components
- Allocate functions to logical components

This phase aims to create a technology-independent representation of the system's structure (platform-independent model), facilitating design flexibility and potential reuse.

8.4.4 Physical Architecture (With What?)

In this phase, the logical architecture is transformed into a concrete technical solution:

- Make technical choices for physical components and interfaces

- Provide high-level characterization to justify design decisions
- Demonstrate functional and extra-functional coverage of system functions and capabilities
- Address performance, reliability, and other quality attributes

The physical architecture represents the actual implementation of the system, considering technological constraints and opportunities.

8.4.5 End-product Breakdown Structure

The final phase focuses on the transition from system design to implementation:

- Define contracts with suppliers
- Manage the system-subsystem transition
- Specify requirements for each subsystem or component
- Prepare for integration, verification, and validation activities

End-product breakdown structure (EPBS) ensures a smooth transition from design to realization, providing a clear structure for system decomposition and supplier interactions.

8.5 AI-powered Scenario Management

EVA's AI-powered scenario management process consists of two main steps: building a domain-specific knowledge base and leveraging it for automated scenario generation and selection.

8.5.1 Extracting Domain Knowledge

A key part of EVA's approach was the creation of a comprehensive knowledge base that included essential parts of the mobility domain. Rather than just using expert viewpoints, which can be limited or biased, EVA used a method based on Natural Language Processing (NLP) techniques to review a large number of mobility-related documents. This method is illustrated in Figure 8.2.

To improve the quality and accuracy of scenarios used for important decisions, we suggested using knowledge taken from the literature to form an ontology. This project helped to classify main aspects of any mobility scenario, including users, vehicles, road types, roadside units, and traffic conditions. The ontology we created details different weather situations, types of roads, and expected behaviors of drivers and pedestrians. We used the Natural Language Toolkit framework to extract this knowledge and added insights from experts.

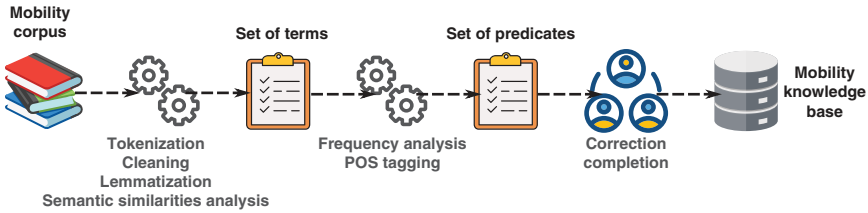


Figure 8.2 Ontology extraction using NLP.

8.5.2 AI-driven Scenario Generation

Typically, creating scenarios in MBSE is a time-consuming and manual task. EVA has taken a different route, using AI to automatically and systematically generate a large number of varied and realistic scenarios, covering a wider range of alternatives.

The ontology we extracted is exhaustive and can support countless scenarios. Since our study focused on a specific area in Paris, we selected the relevant part for more detailed analysis, incorporating expert advice. We used data such as weather, accident reports, and road layout to select 45 features for our study. Each feature was assigned a probability of occurrence and a criticality level.

Table 8.1 presents some selected features along with their probability of occurrence and criticality in relation to the object of our study.

In order to generate coherent scenarios, we proposed to define constraints between features. These constraints are illustrated by the following examples:

$$\neg(cross_road \wedge round_about) \tag{8.1}$$

$$fog \Rightarrow low_grip \tag{8.2}$$

For instance, Equation (8.1) is used to avoid scenarios with both cross-roads and roundabouts, while Equation (8.2) indicates that the presence of fog implies low grip within the same scenario.

With 45 selected features and 22 specified constraints, the total number of possible scenarios is extremely high (more than one trillion), and we can't afford to simulate all of them. To achieve an optimal solution within an acceptable timeframe, we used a heuristic approach to efficiently select the most representative scenarios for analysis, as explained in the next section.

8.5.3 The Role of Multi-criteria Decision-making

Given the vast number of possible scenarios, we investigated a new approach to classify them using metrics such as criticality and probability of occurrence.

Table 8.1 Scenario features.

Feature	Criticality	Probability
Cross road	B	A
Inclination break	B	B
Impaired driver	B	B
Fog	A	D
Roundabout	B	B
Low contrast	A	D
VRU obstacle	A	B
Teleoperation hacking	A	D
Electromagnetic disturbance	A	E
Safety distance violation	C	A
Communication loss	A	C
Right overtaking	A	C
AV failure	B	D
Vehicle hacking	A	D
Glare	B	B

For each generated scenario, we computed two scores according to Equations (8.3) and (8.4):

$$P_g = \prod_{i=1}^{n} P_i \tag{8.3}$$

$$C_g = \sum_{i=1}^{n} 2^{C_i} \tag{8.4}$$

These equations provide the global probability of occurrence and criticality for each scenario. In particular, Equation (8.4) uses an exponential law to emphasize higher levels of criticality.

Once calculated, these scores allowed us to represent scenarios in a graph linking criticality to probability of occurrence and select those present in the Pareto front, as shown in Figure 8.3.

The Pareto front efficiently reduced the total number of scenarios, highlighting the most representative ones for investigation (just over a thousand). The following scenario, selected from the Pareto front, illustrates the relevance of the approach

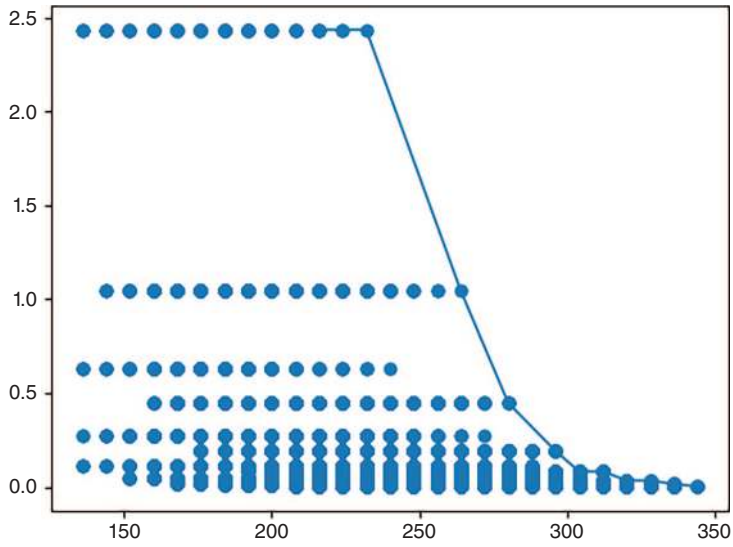


Figure 8.3 Pareto front of the generated scenarios.

for deducing specific system functions to guarantee the safety of transportation services.

> The autonomous vehicle drives on a sloppy slippery road with low visibility because of the fog. Electromagnetic disturbances cause a loss of communication with the control-command center. The road taken is often crossed by wild animals.

This scenario can lead to rapid interruption of transport service or endanger the lives of users. Therefore, we need functions to ensure proper coordination between vehicles and roadside units. We also need a function to enable the vehicle to park safely and inform the control operator when communication is fully restored.

This approach indeed affects the specification of behaviors and interactions required for safer transport services. As a result, examining various scenarios has helped us find the key functions the system shall implement, allowing for more detailed analyses based on more representative models.

This AI-based scenario management showed how using data techniques can improve MBSE, leading to better decision-making and more efficient systems.

8.6 Formal Checking and Simulations

Guided by the scenarios prioritized through EVA's AI-driven process, we transitioned from abstract concepts to concrete tasks, building formal models to represent, analyze, and simulate the behavior of the envisioned mobility system.

8.6.1 Formal Models and Their Advantages

Formal models are central to MBSE, providing a structured framework for analyzing, verifying, and communicating system behavior. The EVA project used several formal models to capture both user and vehicle behaviors, as well as system-level dynamics:

- **User and Vehicle Behaviors**:
  - We used UPPAAL to model and verify the modes and states of autonomous vehicles and users interacting with the service. UPPAAL's real-time capabilities allowed us to ensure that the system adhered to safety constraints and operational requirements.
  - Initially exploring workflows in Capella, we opted for Business Process Modeling and Notation (BPMN) to express scenario dynamics, leveraging its flexibility for modeling control flow and decision points. The captured models underwent formal verification using model checkers, providing mathematical assurance that workflows were free from any logical error.
- **System-level Dynamics**:
  - To understand the complex interactions within the SoS, we used Colored Petri Nets (CPNs) at different levels of abstraction. These models represented interactions among users, vehicles, and infrastructure at different levels of granularity.
  - CPNs were analyzed for properties such as deadlock, liveness, and boundedness, ensuring that the system met its objectives, even under adverse conditions.

Figure 8.4 shows the states of an autonomous vehicle modeled in UPPAAL. Joint execution of state machines for all entities in the operational context enabled us to better define the problem with stakeholders. As with Petri Nets, we formally validated all state machines and fixed any flaws in the specification.

Figure 8.5 illustrates the modeling of an end-to-end mobility workflow, including cyber-attack attempts, using colored, timed, and stochastic Petri Nets. The simulation of this Petri Net enabled us to agree on the objectives and functional scope of the mobility application. Beyond simulation, formal verification of this model, based on temporal logic (Computation Tree Logic), enabled us to detect and fix design flaws.

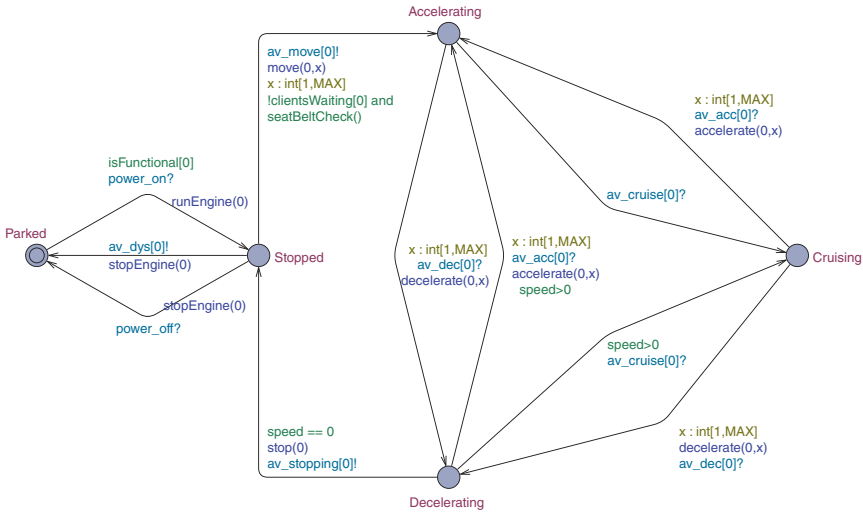


Figure 8.4 Autonomous state machines model.

8.6.2 System Simulation with a Multi-agent System

While formal methods provided rigorous verification, analyzing the scenario's emergent behavior and dynamic characteristics within a realistic context required comprehensive system simulations. This was achieved using a multi-agent system (MAS) approach, which is particularly well-suited to capture the complex dynamics of semi-supervised actors interacting with autonomous components. The selected scenario highlighted the need to simulate not only individual vehicle behaviors but also:

- Interaction between vehicles and the supervision center.
- Influence of less predictable agents, such as pedestrians and nonautonomous vehicles.

We combined Simulation of Urban MObility (SUMO) for macroscopic, continuous traffic simulation and Webots, a robotics simulator allowing detailed modeling of robot behavior and sensor interaction with a physics engine.

An open-street map-based model was developed to simulate generated scenarios in SUMO, including vehicles, pedestrians, infrastructure elements, and specifying behaviors based on consolidated UPPAAL and BPMN models. Communication protocols, failure modes, and vulnerabilities identified using CPN analysis were incorporated into vehicle agent behaviors to represent realistic scenarios of security compromises, data losses, and degraded performance.

Dynamic simulations provided us with valuable information:

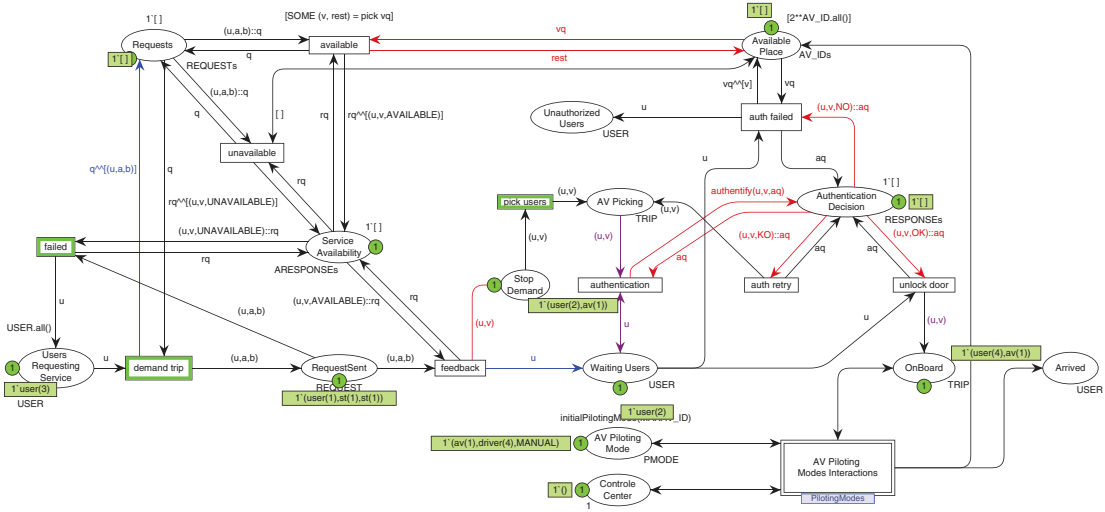


Figure 8.5 Mobility service Petri Net model.

- Numerous simulation runs with varying configurations of autonomous vehicles, users, traffic conditions, and potential disruptive events provided insights beyond what other models could offer.
- Quantifiable Measures: The dynamic simulation allowed us to quantitatively evaluate the impact of design decisions and operational strategies on:
  - Traffic flow patterns, including the ability to manage and optimize congestion.
  - Safety measures: Assess accident risk based on different scenarios.
  - System resource allocation, including power and communication bandwidth.

EVA's integrated use of modeling techniques effectively bridged the gap between abstract concepts and practical realities. By combining formal verification and the detailed analysis capabilities offered by SUMO/Webots simulations, we validated system design choices, uncovered emergent behaviors, identified potential errors at an early stage, and supported a rigorous, data-driven approach to system performance and safety.

8.7 Decision-making and Optimization

The EVA project leveraged simulations to gain insights into the dynamics of its mobility service. However, optimizing the system for cost-effectiveness and user experience required addressing specific challenges, particularly with regards to the strategic deployment of infrastructure for autonomous electric vehicles. The aim was to ensure reliable service while minimizing deployment and operational costs.

8.7.1 Optimization Model for Charging Infrastructure

To tackle this challenge, the EVA project went beyond descriptive models to use quantitative techniques. This approach integrated mathematical optimization methods with simulation results, demonstrating a way forward for data-driven decision-making within the MBSE framework.

The central question was how to determine the optimal number, location, and charging capacity of charging stations to ensure vehicles have sufficient charge to complete their journeys and remain available when needed. The optimization problem was characterized by the following elements:

- **Objective Function**: The optimization model aimed to minimize a weighted combination of two factors:
  - Total Infrastructure Cost: This reflects the capital expenditures required for establishing charging stations, including installation, purchasing, and maintenance.

  - Vehicle Downtime for Charging: The goal was to minimize the time vehicles spend in charging stations, ensuring faster user service and vehicle availability.
- **Decision Variables**: The model used a combination of binary (true/false) and continuous (real-valued) decision variables to represent:
  - Whether a specific location is chosen for a charging station.
  - The capacity of the station, e.g. how many vehicles could be charged simultaneously.
  - Vehicle-to-station assignment decisions, mapping which vehicle should recharge at a given station at specific times throughout the simulated scenarios.
- **Constraints**: Numerous practical constraints, informed by system and environmental considerations, were incorporated:
  - Total budget limitations for station deployment.
  - Geographic and infrastructural constraints for candidate station locations, such as access to the electrical grid and proximity to traffic.
  - Vehicle specifications and maximum range to ensure sufficient charge for expected route durations.
  - Usage patterns and demand, informed by simulation, to allocate vehicles to stations efficiently, avoiding unnecessary queues.

8.7.2 Mathematical Formulation

The optimization problem was formulated as an integer linear program, described by the following equations:

$$\begin{aligned} &\min Z = \sum_{i \in F} f_i y_i + \sum_{i \in F} \sum_{j \in D} c_{ij} x_{ij} \\ &S.T. : \begin{cases} \sum_{j \in D} x_{ij} \leq \gamma_i y_i, & \forall i \in F; \\ \sum_{i \in F} x_{ij} = 1, & \forall j \in D; \\ x_{ij}, y_i \in \{0, 1\}, & \forall i \in F, j \in D; \end{cases} \end{aligned} \tag{8.5}$$

In this model:

- x_{ij} is a binary variable indicating if vehicle j is served by station i.
- y_i is a binary variable indicating if station i is deployed.
- f_i represents the cost of opening station i.
- c_{ij} is the time required to charge vehicle j at station i.
- γ_i denotes the charging capacity of station i.

The objective function minimizes the total cost of deploying stations and the charging time for vehicles. The first constraint ensures that the number of vehicles served by a station does not exceed its capacity. The second constraint ensures each vehicle is assigned to exactly one station.

8.7.3 Simulation and Results

To demonstrate how well our model works, we created a situation with three types of charging stations. Vehicle arrivals were estimated using a Poisson distribution, and the departures used an exponential distribution with varying parameters for each type of station.

Table 8.2 lists the simulation parameters from our tests, and Figure 8.6 illustrates the best number of charging stations needed to meet users' needs. Specifically, this graph indicates that we don't need more than five charging stations to meet demand.

In conclusion, the integration of optimization techniques and data-driven decision-making within the MBSE framework enabled the EVA project to strategically design and deploy an efficient charging infrastructure. This approach not only minimized costs and improved service reliability but also demonstrated the transformative potential of combining simulation and optimization in complex systems engineering.

Table 8.2 Simulation parameters used in our mathematical formulation.

Parameter	Value
D	Set of vehicles.
F	Set of charging stations.
f_i	Cost of opening a new station i.
Simulation time	24 hours.
λ	Arrival rate of 41 vehicles per hour.
$\mu_1 = 12$	Necessary time (minutes/vehicle) to leave a station of type 1.
$\mu_2 = 10$	Necessary time (minutes/vehicle) to leave a station of type 2.
$\mu_3 = 7$	Necessary time (minutes/vehicle) to leave a station of type 3.

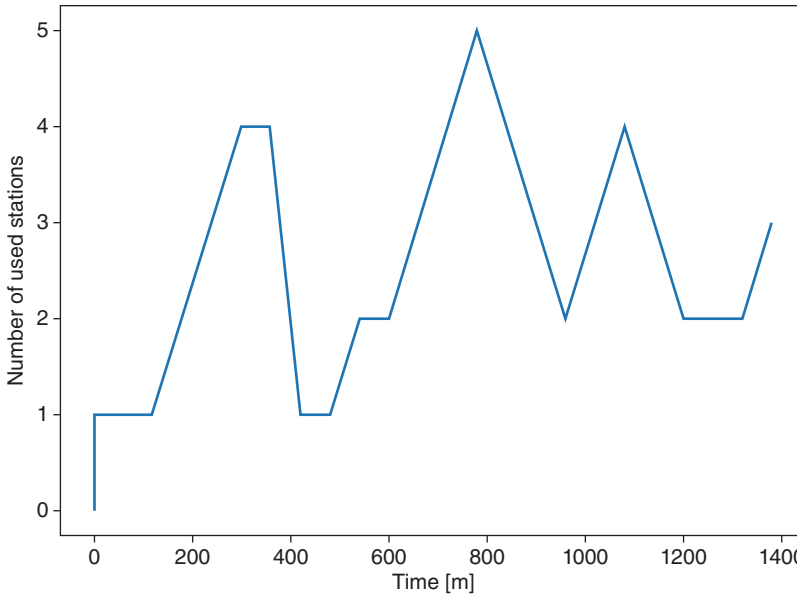


Figure 8.6 Optimal number of necessary charging stations.

8.8 Evaluation and the Path Forward

The EVA project, by pioneering an AI-enhanced MBSE methodology for the design and optimization of an autonomous mobility service, demonstrated substantial advantages over conventional MBSE approaches.

8.8.1 EVA's Contribution to MBSE Practice

By integrating AI techniques into the traditional MBSE framework, the EVA project significantly improved effectiveness and broadened the scope of analysis:

- **Capturing Collective Wisdom**: The automated knowledge extraction through NLP, using a diverse mobility corpus, provided a far richer and more comprehensive view of the domain compared to solely relying on manual expert elicitation. The resulting ontology acted as a shared resource, aggregating information that would have otherwise been fragmented across different stakeholder perspectives, and improving decision-making beyond the limitations discussed in Chapter 2.
- **From "What If" to "What's Most Likely"**: Traditional MBSE approaches struggle to effectively address the exponential complexity of scenario-based design in complex domains. By automating scenario generation, EVA provided a systematic way to create a wide range of plausible situations and then

intelligently selected those most relevant, facilitating a more comprehensive assessment of the envisioned mobility system, as highlighted by Chapter 7.
- **Enabling Effective Collaboration**: MBSE already promotes a collaborative environment, but by using formal models alongside visual representations (as highlighted in Chapter 4), we successfully bridged the communication gap between technical and nontechnical stakeholders within the extended enterprise. This shared understanding facilitated rapid design iterations, enabled faster feedback cycles, and enabled informed discussions on trade-offs, risks, and resource allocation based on data generated through simulation.
- **Quantitative Assessment and Optimization**: Through formal verification of workflows using Maude and the comprehensive simulations conducted with SUMO and Webots, the EVA project yielded quantifiable data that validated key architectural decisions. This data also formed a foundation for strategic resource allocation and design optimization, driving greater value by minimizing infrastructure costs and vehicle downtime while maximizing user service and environmental considerations.

8.8.2 Remaining Challenges

While offering numerous advantages, our novel approach also highlighted some challenges:

- **Managing Complexity**: The complexity of the overall MBSE model, including managing dependencies between models representing different aspects (structure, behavior, extra-functional properties, environment), continued to pose challenges, as discussed in Chapter 7. We primarily relied on hierarchical decomposition and modular design approaches (as discussed in Chapter 4), which helped in breaking down the complexity, but the development of new methods and tools to effectively handle interconnected models is essential.
- **Ensuring Data Quality and Availability**: Our project was heavily reliant on data availability, both for building the initial knowledge base through NLP, and for setting up simulations that accurately represented the real world. Future work should explore methods for validating ontology quality and techniques for addressing data limitations by incorporating statistical techniques or leveraging expert knowledge.
- **Limited Tool Support and Adaptability**: While we successfully combined existing tools (Capella, BPMN, Maude, SUMO, Webots), integrating environmental aspects (e.g. weather data), fully incorporating cybersecurity models, and managing the complex data flow between different toolsets remain active areas for improvement. Greater emphasis should be placed on evaluating the integration capabilities and environmental modeling support of various MBSE platforms when choosing toolsets for projects like EVA.

8.8.3 The Path Forward

The EVA project has laid a solid foundation for the future of MBSE, demonstrating the potential of integrating AI and advanced optimization techniques to improve system design and decision-making. Moving forward, several key areas merit further exploration:

- **Enhanced AI Integration**: Continued research into AI-driven scenario generation and analysis can further improve the comprehensiveness and accuracy of system models, enabling more precise predictions and optimizations.
- **Toolchain Development**: Developing more integrated and adaptable toolchains that seamlessly incorporate various modeling, simulation, and optimization tools will enhance the efficiency and effectiveness of MBSE processes.
- **Cross-domain Applications**: Applying the lessons learned from the EVA project to other domains, such as healthcare, aerospace, and smart cities, can demonstrate the versatility and scalability of AI-enhanced MBSE methodologies.
- **Sustainability and Resilience**: Incorporating sustainability and resilience considerations into MBSE frameworks will ensure that future systems are not only efficient and effective but also environmentally and socially responsible.

In conclusion, the EVA project has highlighted the transformative potential of AI-enhanced MBSE, providing a roadmap for future innovations in systems engineering. By continuing to refine and expand upon these methodologies, we can unlock new possibilities for designing and managing complex systems in an increasingly interconnected world.

8.9 Conclusion

The EVA project has demonstrated the transformative potential of integrating AI into MBSE to tackle the complex challenges of designing autonomous mobility solutions. By leveraging AI for scenario generation and optimization, the project highlighted how MBSE can be enhanced to provide comprehensive, data-driven insights that improve system design and stakeholder satisfaction.

8.9.1 Key Achievements

Throughout this chapter, we explored how the EVA project successfully applied an AI-enhanced MBSE approach to achieve several critical objectives:

- **Enhanced Scenario Management**: By automating the generation and prioritization of scenarios, EVA transitioned from an opinion-based to a

data-driven approach, allowing for a more exhaustive exploration of potential system states and interactions.
- **Improved Collaboration and Communication**: The use of formal models facilitated effective communication across diverse stakeholders, ensuring a shared understanding and alignment of project goals.
- **Optimization and Decision-making**: The integration of optimization techniques with simulation data provided actionable insights for infrastructure deployment, balancing cost and performance to meet user expectations.
- **Validation through Simulation**: Comprehensive simulations using MASs and formal models validated design choices, identified emergent behaviors, and supported a robust, iterative design process.

8.9.2 Challenges and Future Directions

While the EVA project achieved significant success, it also highlighted ongoing challenges and areas for future research:

- **Managing Complexity**: The complexity of integrating various models and tools remains a challenge, requiring further development of methodologies and toolchains that can seamlessly manage interconnected models.
- **Data Quality and Availability**: Ensuring high-quality data for model training and validation is crucial. Future work should focus on improving data collection methods and leveraging statistical techniques to address data limitations.
- **Tool Integration and Adaptability**: Enhancing the adaptability and integration capabilities of MBSE tools will be essential for incorporating factors such as environmental and cybersecurity considerations into the modeling process.

As systems become more complex, the insights from the EVA project will become increasingly important. By using MBSE and adapting standard systems engineering methods, we can create systems that are safer, more effective, and better at meeting user and societal needs.

The EVA project is a notable case of what can be achieved when we merge systems engineering with modern AI and data science. In the future, we must keep innovating, working together, and learning from projects like EVA, advancing the area of MBSE toward smarter and more adaptable systems.

9

Conclusion

In this book, we have looked into the basic concepts and real-world uses of model-based systems engineering (MBSE). We started with an overview of the key concepts and principles behind this important way of practicing systems engineering. We also looked at the history that led to MBSE, the problems created by the increasing complexity of modern systems, and the drawbacks of older document-based approaches.

As we progressed further into MBSE, we unpacked the complex ideas and practices that are fundamental to the field. We discussed how systems, languages, and models are interconnected and reviewed the various types of models used in MBSE, including structural and behavioral models, as well as hybrid and agent-based models. We highlighted the advantages of using formal languages and the importance of managing interfaces and viewpoints in designing complex systems.

We devoted a significant amount of time to the important processes of model verification and validation (V&V). We examined the theories and practical methods that ensure system models are correct, consistent, and complete. The evolution of V&V practices, the key issues they address, and the standards and best practices guiding their use were thoroughly covered. We also stressed the essential role of V&V in limiting design space exploration and reducing costs and delays in system design and development.

The principles of system architecting and the concepts of MBSE were important in our study. We examined the key ideas of system architecture, the different types of architectures used in complex systems, and the methods for creating and changing these architectures. We highlighted the links between architectural issues and MBSE practices, emphasizing the benefits of a holistic approach to system design.

Our study ended with a case study showing how MBSE principles work in practice. This case study revealed how combining traditional methods with new techniques, such as artificial intelligence and data-driven decisions, can be powerful. The findings highlighted the clear advantages of using an MBSE approach, such

as improved system performance, shorter development times and costs, and satisfied stakeholders.

Looking back at what we learned from our study of MBSE, it is clear that this approach is changing the way we think about, design, and manage complex systems. Using MBSE is not just a technical task; it needs a change in mindset, culture, and processes within organizations. It requires continuous learning, teamwork, and innovation, as well as a willingness to adopt new tools, technologies, and methods.

The future of MBSE looks very bright, as increasing complexity and interconnections of systems demand a more holistic and in-depth approach for their design and management. The combination of MBSE with new technologies, such as artificial intelligence, big data analytics, and digital twins, offers great opportunities to change the way we build and manage complex systems. These technologies can improve MBSE, allowing for better predictions, improved designs, and instant monitoring and control of system behavior.

The future success of MBSE will depend on our ability to solve the problems that limit its use. These include the need for consistent standards and compatibility between different modeling tools and languages, the creation of simpler and more user-friendly interfaces for making and managing models, and the training of a workforce skilled enough to effectively use MBSE.

As we close this book, I encourage you to keep exploring the interesting field of MBSE. The path we have traveled together marks just the start of a long journey toward knowledge acquisition and innovation in the field of complex systems engineering. The theories and practical advice presented in this book provide a solid foundation on which to build your expertise and take MBSE to the next level.

I suggest you engage with the MBSE community by attending conferences and workshops, and working with other practitioners and researchers. Share your experiences, problems, and achievements, and learn from the various viewpoints and methods of others in different fields. By creating an active and supportive community, we can work together to push the development of MBSE and realize its full potential.

As you start your own MBSE journey, keep in mind that the road ahead comes with many exciting opportunities and challenges. Adopt the attitude of curiosity, creativity, and persistence shown by the pioneers in this field, and allow it to motivate you to expand what can be done. The future of complex systems engineering is in your hands, and with MBSE at your disposal, you have the knowledge and tools to shape it in important ways.

With the insights you have gained from this book, you can tackle the major challenges we face today, such as sustainable energy, smart cities, space exploration, and more. Use the power of MBSE to create systems that are more dependable, efficient, and in tune with societal needs. And above all, always keep learning, growing, having fun, and challenging the boundaries of what is achievable.

Index

Note: Page numbers in *italics* and **bold** refers to figures and tables respectively.

a

A320 NEO case, 237–238
AADL. *see* Architecture Analysis and Design Language (AADL)
ABM. *see* agent-based modeling (ABM)
abstraction, 58, 128
  benefits, 77–78
  and computational models, 75–77, *75, 77*
  and design space exploration, 78–79, *79*
  high-level, 78
  low-level, 78
  mid-level, 78
  modeling benefits, 3–5, *4*
  paradigm shift, need for, 2
  role of, 2–3
Acceleo, 13
acceptance testing, 188
accuracy efficiency, 141
activity diagram, 149–152, *151*
aerospace industry, 16–17
agent-based discrete event simulation, 157
agent-based modeling (ABM), 35, 40, 46
  applications in, 163–166
  characteristics, 161
  structural and behavioral, 161–163
  traffic simulation, 164–165, *164*
agile MBSE, 281
agreement processes, 35–36, *36*
AI. *see* artificial intelligence (AI)
AI-enhanced MBSE
  achievements, 319
  challenges, 317–318
  EVA, contribution to, 316–317
AI/ML systems, formal verification of, 196
AI-powered scenario management
  domain knowledge extraction, 307, *308*
  multi-criteria decision-making, 308–310
  scenario generation, 308
allocation, requirements, 50, 62
Alloy Analyzer, 13
anomaly management, 186
Apollo program, 38–39

ARCADIA methodology
  end-product breakdown structure, 307
  logical architecture, 306
  objectives, 305
  operational analysis, 306
  physical architecture, 306–307
  system analysis, 306
architectural decisions, 240–241
architectural patterns. *see* patterns
architectural tactics. *see* tactics
Architecture Analysis and Design Language (AADL), 116
architecture optimization, 260–261
architecture selection, 55
Architecture Trade-off Analysis Method (ATAM), 241–242
artificial intelligence (AI)
  assisted V&V, 196
  driven integration, 281
  integration of, 118
assembly decomposition, 132–135, *135*
ATAM. *see* Architecture Trade-off Analysis Method (ATAM)
ATL. *see* Atlas Transformation Language (ATL)
Atlas Transformation Language (ATL), 13, 278
ATM system scenario analysis, 265–266
audit planning, 184
augmented reality, 281
automata theory, 201–202
automatic test case generation, 204
Automotive Open System Architecture Standard, 237
autonomous car system, V&V techniques for, 211–212
autonomous state machines model, *312*
availability testing, 198
axiomatic semantics, 98, 99

b

Backus–Naur form, 74
behavioral models
  continuous, 137–142
  discrete, 143–157
  hybrid, 157–161
  types of, 137
blackboard pattern, 239
Boeing, 787 Dreamliner, 16
bond graphs, 138, 140–141
Boolean dataflow, 155
bottom-up assessment, 304
bottom-up decomposition, 129–131
boundedness analysis, 147
BPMN. *see* Business Process Modeling and Notation (BPMN)
broker pattern, 239–240
Business Process Modeling and Notation (BPMN), 274, 311, 313

c

Cameo Systems Modeler, 12, 117
Capability Maturity Model Integration (CMMI), 267
case diagrams, 3
CI/CV. *see* continuous integration and continuous validation (CI/CV)
CIM. *see* Computation Independent Model (CIM)
class diagrams, 3
cloud-based modeling platforms, 118
cloud computing, 136
CMMI. *see* Capability Maturity Model Integration (CMMI)
coherence
  integral, 32
  pursuit of, 24
  types of, *32*
  universal, 32
  virtual, 32–33
collaboration, 11
  and communication, 10, 18, 41, 52, 62, 64, 112

real-time, 47
colored Petri Nets (CPNs), 79, 148, *149*, 311, 313
combined discrete-continuous simulation, 157
comma-separated values (CSV), 117
commercial aircraft design, 59–60
communication, 41, 104, 127
and collaboration, 10, 18, 41, 52, 62, 64, 112
developing system vision, 47
and documentation, 53
domain, 76
engagement and, 44
gaps, 51
secure, 57
stakeholder engagement and, 44
complexity management, 58, 127
complexity theory, 202–203
computability theory, 202–203
computational intensity, 158
computation domain, 76
Computation Independent Model (CIM), 245
computation tree logic, 201
configuration domain, 76
consistency checking, 184
constraint languages, 12
constraint verification, 93
context diagrams, 119
continuous and discrete dynamics, 157
continuous behavior models
bond graphs, 140–141
differential equations, 138–140
example, 142
numerical simulation techniques, 141–142
continuous improvement process, 264–265
continuous integration and continuous validation (CI/CV), 195
continuous verification and validation, 10
control systems, 139
cost escalation curve, 217–218
CPNs. *see* colored Petri Nets (CPNs)
CSV. *see* comma-separated Values (CSV)
cybersecurity, 56–57
cyclo-static dataflow, 155

d

data domain, 76
data exchange formats, 116–117
deadlock-free execution, 154
decision space, 3
decomposition
assembly, 132–135, *135*
bottom-up, 129–131
functional, 131–132, *133*
hierarchical, 129–131
service-based, 135–136
top-down, 129
DEMs. *see* discrete behavior models
denotational semantics, 97
descriptive models, 99–100
design optimization, 127
design space, 257
exploration, 215–216, *215*
design space exploration (DSE), 51, 54–55
design structure matrix (DSM), record in, 133
deterministic *vs.* stochastic systems, 27
deterministic finite-state machine, 145
development space, 4
development view of "4+1" model, 263, *263*
differential equations, 138–140
digital models, 101
discrete behavior models (DEMs)
activity diagram, 149–152
discrete events, 156–157
finite state machines, 143–146
Kahn Process Networks, 152–153
Petri Net, 146–149
synchronous data flow, 153–156, *155*

discrete events, 156–157
disorganized complexity, 34
distributed discrete event simulation, 157
document-centric approaches, 6, 14, *15*
domain knowledge extraction, 307, *308*
domain-specific languages (DSLs), 12, 106
DSE. *see* design space exploration (DSE)
DSLs. *see* domain-specific languages (DSLs)
DSM. *see* design structure matrix (DSM)
dynamic complexity, 33, 35
dynamic models, 100
dynamic simulation, 313
dynamic systems, 27
dynamic testing, 185

e

Eclipse Process Framework, 13
e-commerce system, 136, 243–245, 250–251
Éco-mobilité par Véhicules Autonomes (EVA) project, 302–303
  achievements, 319
  AI integration, 305
  AI-powered scenario management, 307–310
  capabilities, 303–304
  challenges, 319–320
  decision-making, 313–316
  evaluation, 316–318
  formal checking, 310–311
  future research, 319–320
  "meet-in-the-middle" approach, 304–305
  motivation, 303
  optimization, 313–316
  path forward, 318
  simulation, 311–313
EDA. *see* Event-driven Architecture (EDA)
emergent properties, 41
  and behaviors, 25–26, 61
empirical validation, 188
encapsulation, 128
engineered systems, 27
enterprise architect, 12, 117
enterprise processes, 36, *36*
enterprise software, 136
environment
  collaborative modeling, 118
  connecting system with, 112–113
  impact assessment, 56
  representation, defined, 162
Epsilon Transformation Language, 278
EVA. *see* Éco-mobilité par Véhicules Autonomes (EVA) project
Event-driven Architecture (EDA), 231
extended finite-state machine, 145
extra-functional models, 167–168
  properties domain, 76–77

f

F-35 Joint Strike Fighter, 17
face validation, 188
Failure Mode and Effects Analysis (FMEA), 167, 187
Fault Tree Analysis (FTA), 167, *169*, 187
feedback, 51
  loops, 40–41
finite difference method, 141
finite element method, 141
finite-state machines (FSM), 143–146, *146*
first-order logic, 201
flexibility, 242
fluid dynamics, 139
FMEA. *see* Failure Mode and Effects Analysis (FMEA)
formalization, 37
  principles, 90–94, *93*
formal methods, 195, 206, 311
  advantages, 311
formal models, 101
  emergence of, 6

formal semantics, 153
formal specification languages, 86
formal verification, 111, 158
  languages, 86
"4+1" view model, 263
FSM. *see* finite-state machines (FSM)
FTA. *see* Fault Tree Analysis (FTA)
functional decomposition, 131–132, *133*
functional languages, 75

g

gap analysis, 259, *260*, 304
geographic distribution, 29
global air traffic control system, 30
global transportation systems, 29
granularity, 133, 136
graph theory, 92, 202

h

hardware-in-the loop (HIL) testing, 195
hazard analysis, 57
Hazard and Operability Analysis (HAZOP), 167, 187
HAZOP. *see* Hazard and Operability Analysis (HAZOP)
hierarchical activity diagram, 151
hierarchical decomposition, 129–131
hierarchical design, 58
hierarchical Petri Nets, 148
hierarchical structure, 133
HIL testing. *see* hardware-in-the loop (HIL) testing
human-system integration, 55
hybrid models
  challenges, 159
  characteristics, 157–158
  example, 159–161, *160*
  power of, 159
  verification and simulation, 158

i

IBM Engineering Requirements Management DOORS, 13
IDLs. *see* interface definition languages (IDLs)
IEEE. *see* The Institute of Electrical and Electronics Engineers (IEEE)
IFE system. *see* in-flight entertainment (IFE) system
incident response and recovery, 57
INCOSE. *see* International Council on Systems Engineering (INCOSE)
Industrial Revolution, 193
in-flight entertainment (IFE) system, 132, *133, 135, 146*
informal semantics, 97, 99
inheritance, 129
The Institute of Electrical and Electronics Engineers (IEEE), 80, 106, 118
integral coherence, 32
integral whole, 31
integrated air defense networks, 29
integrated system views, 10
integration, verification, validation, and qualification (IVVQ), 55
integration frameworks, 274
interdisciplinary approach, defined, 42
interface
  defined, 135
  diagrams, 110
interface definition languages (IDLs), 62, 110
interface design and management
  challenges and practices, 111–112
  connecting system with environment, 112–113
  defined, 109
  lifecycle, 110
  modeling techniques for, 110
  Rover case, 113–115
  V&V of, 111
International Council on Systems Engineering (INCOSE), 7, 10, 105, 117

Internet of Things (IoT), 136
interruptible activity diagram, 151
IoT. *see* Internet of Things (IoT)
ISO standards, 106, 118
  ISO/IEC/IEEE 15288 standard, 24, 35–36
IVVQ. *see* integration, verification, validation, and qualification (IVVQ)

k

Kahn Process Networks (KPNs), 152–153
Key Performance Indicators (KPI), 282
KPI. *see* Key Performance Indicators (KPI)
KPNs. *see* Kahn Process Networks (KPNs)

l

languages
  abstraction and computational models, 75–77, *75, 77*
  expressing models, 73–74, *74*
  expressivity and tractability, 74, *74*
  formalization principles, 90–94, *93*
  heterogeneous systems, 87–90
  modeling, 115–116
  selection, 87
  spectrum of, 86
  tools, 94
  to viewpoints, *105*, 106
layered architectures, 229–230
library management system, 3
lifecycle cost analysis, 56
lifecycle management, 56, 168–169
  technology and, 44
linear *vs.* nonlinear systems, 27
linear temporal logic (LTL), 201
liveness analysis, 147
logical view of "4+1" model, 263, *263*
loose coupling, 135, 230
LTL. *see* linear temporal logic (LTL)

m

machine learning (ML), integration of, 118, 142
MagicDraw, 12
maintainability models, 167–168
maintainability V&V techniques, 199
Mariner 1, 194
Mars rover
  project, 47–48
  system, 26
Mars Science Laboratory (Curiosity rover), 17
mathematical models, 101
MathWorks Simulink, 13
MBSE adoption, 285
  cultural resistance, 285–286
  integration with existing processes and tools, 286
  model management and complexity challenges, 287
  scaling, 288–289
  skills gap, 285–286
  V&V challenges, 287–288
MBSE and project management integration, 279
  areas of intersection, 280
  challenges, 280–281
  emerging trends, 281
  opportunities, 280–281
MBSE methodologies, 262–263
  continuous improvement and evolution, 264–265
  "4+1" view model, 263
  scenarios, 264
  tailoring, to project needs, 264
MBT. *see* model-based testing (MBT)
MCDM. *see* multi-criteria decision making (MCDM)
Mealy machine, 145
mechanical systems, 139
meet-in-the-middle approach, 257–259, *258*, 304–305
mental models, 41

metamodel (Rugby), 74, 75, *75*
metaphor, 91
methodological approach, 261–262
methodology considerations, 257
  architectural options, 260
  architecture optimization, 260–261
  decision-making, 260–261
  design space, 257
  meet-in-the-middle approach, 257–259
metonymy, 91
microservices architecture, 230–231
ML. *see* machine learning (ML)
mobility service Petri Net model, *312*
model(s)
  abstraction and design space exploration, 78–79
  abstraction benefits, 77–78
  checking, 11
  computational models and abstraction levels, 75–77, *76, 77*
  continuous quest for new perspectives, 96
  and diagrams, 102–105, *103*
  expressivity and tractability, trade-off between, 74, *74*
  historical significance, 73–74
  linking, 168
  powerful example, 94–95
  power of modeling in MBSE, 96
  requirements, 118–120
  selection, importance of, 101–102
  and semantics, 97–99
  seven bridges of Königsberg, 95–96, *95*
  standards and tools, 115–118
  types of, 99–101
  viewpoints, *105*, 106, *107*
model-based approaches, 195
model-based process management tools, 274
model-based systems engineering (MBSE), 1–2, 6–7, 183, 240–241
  abstraction, 2–5
  adopt, 52
  aerospace industry, 16–17
  alignment, need for, 9–10
  analysis and automation, 16
  approaches, 283–284
  automation/analysis, 16, 269–270
  automotive industry, 17
  benefits, 295, 301
  challenges, 16, 301–302, 317–318
  checking, 184, 195, 205–206, 210
  collaboration, power of, 11
  considerations, 17–20
  core tenets of, 10
  defense industry, 17
  diagrams in, 102–105, *103*
  document-centric approaches, 6, 14
  effectiveness, measuring challenges, 284
  formal models, 6
  future development, 322
  historical context, 6
  interface design and management, 109–115
  journey, 7–9
  Key Performance Indicators (KPI), 282
  leveraging, for early V&V, 218
  linking, to architectural concerns, 245–249
  metrics, 282–283
  modeling requirements, 118–120
  modeling standards organizations, 117–118
  paradigm shift, 14–15
  power of modeling in, 96
  process, migrating to, 289–291
  process modeling, integration with, 270–271
  solution, 10
  supporting design space exploration, 15–16
  systems engineer in, 47–48
  technicalities, 10–11

model-based systems engineering (MBSE) (*Cont.*)
tool ecosystem, 12–13
tools, 117
tool selection and integration, 14
transitioning to, 63–64
trends and technologies, 118
validating and verifying requirements, 119–120
V&V techniques, 205–211
model-based testing (MBT), 204
model-centric approach, 5
Model-driven Architecture (MDA)
benefits, 247–249
for e-commerce system transition, 250–251
hardware/software co-design, *248*
Y model, 245, *246*
modeling and simulation techniques, 58
modeling languages, 86, 116
modeling paradigms
agent-based modeling, 161–166
behavioral models, 137–161, 166–171
model-based safety analysis, 171–175
structural models, 127–136, 166–171
modeling tools, 12
model-to-model transformations, 12, 13
model-to-text transformations, 12
model transformations, 274–275, *276*
applications of, 276–277
benefits, 275–276
best practices, 278–279
challenges, 278–279
languages and tools, 277–278
limitations of, 277
model validation. *see* validation
model verification. *see* verification
Model-View-Controller (MVC), 239
mode switching, 157
modular design, 58, 136
modularity, 242
modular open systems approach, 237
monolithic architectures, 229
Monte Carlo methods, 208
Moore machine, 145
multi-agent system (MAS), system simulation with, 311–313, *312*
multi-criteria decision making (MCDM), 260–261, 308–310
multi-domain modeling, 138, 140
multi-scale modeling, 142
MVC. *see* Model-View-Controller (MVC)

n

NASA's Jet Propulsion Laboratory, 17
Natural Language Processing (NLP), 307
used, for ontology extraction, *308*
natural systems, 27
need space, 3
NLP. *see* Natural Language Processing (NLP)
nondeterministic finite-state machine, 145
numerical crash test simulation, 196–197
numerical simulation techniques, 141–142

o

object constraint language (OCL), 12
object flow activity diagram, 151
Object Management Group (OMG), 74, 117
object-oriented models, 128–129
OCL. *see* object constraint language (OCL)
ODEs. *see* ordinary differential equations (ODEs)
OMG. *see* Object Management Group (OMG)
open architectures
A320 NEO case, 237–238
advantages, 236
challenges, 236–237
considerations, 236–237

examples, 237
principles, 235–236
open communication and collaboration, 52
open standards, 14
Open Systems Interconnection Model, 237
operational constraints, 61
operational environment modeling, 61, 171
operational independence, 29
operational semantics, 97, 99
operational testing, 190
optimization model, 11
for charging infrastructure, 313–315, *316*
mathematical formulation, 314–315
ordinary differential equations (ODEs), 138
organized complexity, 34
overlap, 91

p

paradigm shift, need for, 14–15
parallel computing techniques, 142
parameterized dataflow, 155
parametric models, 171
Pareto front, 309, *310*
Paris analogy (viewpoints), 106–107, *107*
partial differential equations (PDEs), 139
patterns, 239–240
PDEs. *see* partial differential equations (PDEs)
PDM. *see* Platform Description Model (PDM)
performance testing, 190, 197
Petri Net, 79, 146–149, 311, *312*
Phonebloks project, 226–227
physical view of "4+1" model, 263, *263*
PIM. *see* Platform Independent Model (PIM)
pipes and filters pattern, 239
platform-based design, 54
Platform Description Model (PDM), 245
Platform Independent Model (PIM), 245
Platform Specific Model (PSM), 245
polymorphism, 129
polysemy, 91
port-and-connector notations, 110
power of iteration, 18
pragmatics, 91
pragmatic semantics, 98
prescriptive models, 100
probabilistic state machine, 145
process formalization, 267
automation in MBSE, 269–270
benefits, 268
elements, 267
methodology *vs.* process, 268–269
process modeling, 270
best practices, 271–272
challenges, 271–272
integration, with MBSE, 270–271
process orchestration, 272
benefits, 272–273
tools and techniques, 13, 273–274
process view of "4+1" model, 263, *263*
project processes, 36
PSM. *see* Platform Specific Model (PSM)
PTC integrity modeler, 117
Ptolemy's cosmic model, 83–84
pushdown automaton, 145

q

quality assurance, 44–45
quality control, 193
Query/View/Transformation (QVT), 277
QVT. *see* Query/View/Transformation (QVT)

r

reachability analysis, 147
refinement requirements, 54

regulatory environment modeling, 61, 171
reliability requirements, defined, 59
reliability testing, 198
reporting and continuous improvement, 186
ReqIF. *see* requirements interchange format (ReqIF)
requirements engineering (RE), systems engineering and
  challenges and strategies for, 51–52
  refinement and integration, 54
  Rover example, 52–53
  SE/RE interplay, 50–51, *51*
requirements interchange format (ReqIF), 13, 117
reusability, 54, 135
  of model artifacts, 17
Rover
  case, 113–115, 173
  example, 52–53
  model, 35
  subsystems, *25*, 26
Rugby metamodel, 74, 75, *75*

S

safety management, 57–58
safety methods, 198–199
safety models, 167
safety requirements, defined, 57
scalability V&V techniques, 200
scenario analysis, 207
scenario-aware dataflow, 155
scenario-based models, 172, 263, *263*
  ATM system, 265–266
  in MBSE, 264
SDF. *see* synchronous data flow (SDF)
SEBoK. *see* Systems Engineering Body of Knowledge (SEBoK)
security models, 167
security testing methods, 198
semantics
  axiomatic, 98, 99
  denotational, 97
  formal, 153
  informal, 97, 99
  operational, 97, 99
  pragmatic, 98
  train and gateway example, 98–99, *99*
semi-automatic partitioning, 55
sensitivity analysis, 207
sentiment analysis, 92
service autonomy, 230
service-based decomposition, 135–136
service composition, 230
service identification, defined, 135
Service-oriented Architecture (SOA), 230
seven bridges of Königsberg, 95–96, *95*
shared system model, 10
simulation, 210
  advantages, 208
  agent-based discrete event, 157
  combined discrete-continuous, 157
  critical scenarios testing, 207
  design flaws, early detection of, 207
  distributed discrete event, 157
  emergent behaviors evaluation, 207
  limitations, 208
  Monte Carlo methods, 208
  parameters, used in mathematical formulation, **315**
  scenario analysis, 207
  sensitivity analysis, 207
  and testing, 111
  tools, 11, 13
Simulation of Urban MObility (SUMO), 312, 313
Sirius frameworks, 12
SOA. *see* Service-oriented Architecture (SOA)
socio-political complexity, 34, 35
software crisis, 195
software design process, modeling, 294–295

Software Development Process, 291–294, *292*
SoS. *see* systems of systems (SoS)
Space Race, 193
  catalyst, 37–38
Spotify, migration to microservices, 233–235
standard exchange formats, 13
standardization, 58
state machines, 3
static analysis, 111
static models, 100
static systems, 27
static testing, 185
stiff systems, 142
stochastic activity diagram, 151
stochastic colored Petri Nets, 79, 148
stochastic models, 172
structural complexity, 33, 35
structural models, 127–136
SUMO. *see* Simulation of Urban MObility (SUMO)
supporting design space exploration, 15–16, *15*
swimlane activity diagram, 150
synchronous data flow (SDF), 153–156, *155*
syntax checking, 183
SysML. *see* Systems Modeling Language (SysML)
system architecting principles
  abstraction, 227
  complexity management, 225
  encapsulation, 226
  modularity, 226
  scalability, 227–228
  separation of concerns, 225
  teamwork with stakeholders, 228
system architecture
  architectural patterns, 239–240
  architectural tactics, 240
  decisions, 240–241
  and design, 50
  evaluation, 241–243
  fundamental principles, 225–228
  and integration, 44
  MBSE, 240–241, 245–249
  selection consideration, 232–233
  types, 228–233
  updating, 241–243
systematic model development, 140
systematic process, defined, 42
system boundaries, defined, 41
SystemC Cycle Callable, 78
system composition, 55
SystemC Transaction-Level Modeling, 78
system dynamics, 41
system engineering principles
  considerations in, 53–63
  defined, 23–24, 42–43
  historical perspective, 37–39
  issues, 43–45
  managing system complexity, 33–35
  requirements, 49–53
  system consistency, 31–33
  system lifecycle, 35–37
  systems engineer, role of, 45–48
  systems of systems, 28–31
  system thinking, 39–42, 52
  taxonomy, 24–28, *28*
  transitioning to MBSE, 63–64
system environment, 60–61
Systems Engineering Body of Knowledge (SEBoK), 6–7, 90, 212, 228
  capitalization and reuse, 54
  complexity management, 58
  complexity types, 33–34
  cybersecurity and resilience, 56–57
  defined, 24
  design space exploration, 54–55
  human factors and ergonomics, 55–56
  integration, verification, validation, and qualification, 55
  ISO/IEC/IEEE 15288 standard, 24
  lifecycle process structure, 35–37

Systems Engineering Body of Knowledge (SEBoK) (*Cont.*)
managing complexity, 34–35
managing interactions with subcontractors and partners, 62–63
risk and change management guidance, 54
safety management, 57–58
sustainability and lifecycle management, 56
systems taxonomy, 26–28, *28*
systems engineer, role of, 45–48
system simulation, with multi-agent system, 311–313, 312
system sizing concept, 58–60
Systems Modeling Language (SysML), 7, 12, 86, 116, 117, 119, 195
systems of systems (SoS), 28–31, *302*
system thinking, 39–42, 52

t

tables (model), 103
tactics, 239, 240
technical processes, 36–37
technological environment modeling, 61, 171
temporal logics, 201
terrain navigation, 173
testing
advantages, 210
case generation, in MBT, 209
limitations, 210
role, in MBSE V&V activities, 209
thermal systems, 139
timed activity diagram, 151
time domain, 76
timed Petri Nets, 79, 148
timed state machine, 145
time-invariant *vs.* time-varying systems, 27
top-down analysis, 304
top-down decomposition, 129
trade-offs, 77–78
analysis, 55
between expressivity and tractability, 74, *74*
train and gateway example, 98–99, *99*
training and education, investment in, 18
transformation tools, 12–13
two-way deterministic finite automaton, 145

u

UAT. *see* user acceptance testing (UAT)
UML. *see* unified modeling language (UML)
UML MARTE, 78
unified modeling language (UML), 1, 74, 108, 115, 274
universal coherence, 32
universal whole, 31
UPPAAL (gateway/train model)
coordination, 98
efficiency, 98
safety, 98
usability testing, 199
usage space, 4
use case diagrams, 119
user acceptance testing (UAT), 189–190
user experience, 199
user interface design, 55

v

validating and verifying requirements, 119–120
validation
acceptance and deployment, 190–191
acceptance testing, 188
considerations of, 192
coverage of critical scenarios, 191
empirical, 188
extra-functional properties, 191–192
face, 188

operational and performance testing, 190
planning, 189
reproduction of actual operating conditions, 191
requirements, 49
tests, development of, 189
traceability and documentation management, 192
user acceptance testing (UAT), 189–190
vendor support, 14
verification, 183
anomaly management, 186
audit planning, 184
complexity and size, controlling, 186–187
compliance with norms and standards, 187
considerations of, 186–188
consistency checking, 184
dynamic testing, 185
emerging properties, 187
independence and objectivity, 188–189
model checking, 184
reporting and continuous improvement, 186
reviews and inspections, 184–185
robustness and resilience, 188
static testing, 185
syntax checking, 183
tools and automation, 13, 188
verification and validation (V&V)
advanced techniques, 195
automata theory, 201–202
automation and tool integration, 214
for autonomous car system, 211–212
challenges, overcoming, 288
complexity theory, 202–203
computability theory, 202–203
continuous, 10
costs reduction, 216–219
delays reduction, 216–219
design space exploration, closing, 215–216
documentation, 213
first-order logic, 201
foundations, 193
graph theory, 202
Industrial Revolution, 193
iterative and continuous practices, 214–215
landscape and future directions, 196
maintainability and scalability, 199–200
model-based approaches, 195
model checking technique, 205–206, 210–211
operation and compliance, 197
performance and efficiency, 197
quality control, 193
reliability and availability, 198
requirements, 50
safety, 57, 198–199
security, 198
simulation technique, 207–208, 210–211
software crisis, 195
Space Race, 193
standards and guidelines, 212–213
temporal logics, 201
testing, 208–210, 210–211
traceability, 213
usability and user experience, 199
verification and validation (V&V) models
automatic test case generation, 204
early verification, through model analysis, 204
requirements coverage, 204–205
traceability, 204–205
validation, by prototyping, 205
validation, by simulation, 205

verification tools, 13
version control and configuration management, 111
Vienna Development Method, 86
viewpoints, 105–109
views (diagrams), 103
virtual coherence, 32–33
virtual integration, 55
virtual reality, 281
virtual whole, 31
VIsual Automated model TRAnsformations, 278
V&V. *see* verification and validation (V&V)

W

well-defined process, importance of, 18
workflow optimization, 56

X

XMI. *see* XML metadata interchange (XMI)
XML metadata interchange (XMI), 13, 117
Xtext, 12

Z

zero behavior, 158
Z language, 86

WILEY END USER LICENSE AGREEMENT

Go to www.wiley.com/go/eula to access Wiley’s ebook EULA.