SYSTEMS ENGINEERING SERIES
Cuadernos de Isdefe

1

Introduction to SYSTEMS ENGINEERING in the 21st Century

CUADERNOS DE Isdefe

Isdefe

Introduction to Systems Engineering in the 21st Century

Volume 1

Systems Engineering Series

Cuadernos de Isdefe

AUTHORS

Dr. Alejandro Salado / Adolfo Sánchez Domínguez / Dr. Dinesh Verma

Thomas Allen McDermott / Víctor Ramos del Pozo

Dr. Ronald Giachetti / Juan Carlos Larios Monje

David Long / Belinda Misiego Tejada / Dr. Kaitlin Henderson

Christopher Delp / L. Miguel Aparicio Ortega / Dr. Joe Gregory

Dr. Kaitlynn Castelle / Miguel Ángel Coll Matamalas

Original Title: Introduction to Systems Engineering in the 21st Century

© Isdefe
C/ Beatriz de Bobadilla, 3 -28040 Madrid
www.isdefe.es

First printing: June 2024

ISBN: 978-84-695-7816-2
No comercial
Legal deposit: M-14809-2024

Publisher: Ingeniería de Sistemas para la Defensa de España SA SME MP
Coordinator: Juan Manuel García Montaño
Technical coordinator: Dr. Alejandro Salado
Editor: Dr. Alejandro Salado
Editing and style review team: Juan Manuel García Montaño, Mª del Rocío Manjón Pérez
Book design and layout: Iliana Aguilar Jiménez
Video recording and production: Favorit Comunicación

Printing by Byprint Madrid
Printed in Spain - *Impreso en España*

"[A system is] any portion of the material universe which we choose to separate in thought from the rest of the universe, for the purpose of considering and discussing the various changes which may occur within it under various conditions."

J.W. Gibbs

PROLOGUE

Around 30 years ago, Ingeniería de Sistemas para la Defensa de España S.A. S.M.E. M.P. (Isdefe) published the systems engineering “blue books.” The “blue books” were a series of sixteen monographs that captured the state of the art in systems engineering at the time, in a practical and concise manner. The series was a special feat because of two main reasons. First, it conformed the first systems engineering publications that had been ever written in Spanish; and was certainly the largest one in scope, still to this day. Second, and most importantly, it formally brought systems engineering to Spain at a time when, let’s be honest, just a few individuals were even aware of what systems engineering was.

The “blue books” covered a wide range of systems engineering topics, arguably the whole body of knowledge at the time: from General Systems Theory to Systems Analysis. The series was edited by an editorial board (drafting committee) composed by generals from the Spanish Armed Forces, government officials from Spain, and Isdefe employees, and was written by several national and international experts in the different topics that it covered, including authors such as the late Ben Blanchard, Donald R. Drew, and Jezdimir Knezevic, among others. While the “blue books” were really an impressive and unique source of knowledge, just a few people had access to them. The monographs served as the basis for Isdefe to introduce their employees, clients, and partners to systems engineering, and have remained so. (I am very lucky to treasure one copy of the entire collection in my office despite having never been an Isdefe employee!).

But time has passed and, while one could argue that it has only been a bit short of 30 years since then, the systems engineering landscape has significantly changed, both in the size of its body of knowledge and in its widespread and external awareness. Some examples, non-exhaustive, follow:

- The number of active members of the International Council on Systems Engineering (INCOSE) (founded in 1990 and becoming international in 1995) has grown exponentially since the 2000's. A similar trend exists for the number of new members that join INCOSE.

- A plethora of new methods that span the whole system life cycle has emerged. Systems engineering is not anymore just a process or a guide for good engineering, but systems engineers can resort to methods that are dedicated to its activities, such as capturing requirements, architecting systems, and so forth.

- Today, we have technology that is dedicated to supporting the systems engineer; we no longer must remain cornered to paper, word processors, and spreadsheets.

- There are more and more educational opportunities for systems engineers. We no longer have to limit our learning to on-the-job training, but many universities offer graduate programs, several offer PhD programs, and some undergraduate programs. INCOSE has established a formal certification program, and the number of consulting companies offering training and systems engineering publications (papers, books, reports, etc.) keep growing every year.

- People start to spell systems engineering for real. In Spain for example, systems engineering has been traditionally associated with Information Technology (IT). This is still the case for the most part, but since 2014 there is an INCOSE chapter in Spain (Asociación Española de Ingeniería de Sistemas, AEIS), a podcast, and a few events that get organized every year. In just the last 10 years, the amount of Spanish you hear at a systems engineering conference has dramatically increased. And it feels good!

In this new context, Isdefe is emerging again as a national leader in the field and is set to revise and/or complement the “blue books” with a contemporary look at systems engineering. The purpose of this monograph is to present a snapshot of the state of the practice of systems engineering, including topics that are nascent to practice. The monograph is targeted to practitioners in the Spanish government and industry (while hopefully reaching a wider, international audience), who are engaged in the development of engineered systems, both as customers and suppliers, in the defense, security, space, energy, and transport sectors.

The monograph has been developed as an edited collection of chapters to be read as a monolithic piece, with each chapter being written by one or two international experts in the field in tandem with an Isdefe employee. Each chapter can be thought of as an introduction to a topic that could become a full monograph in the future to tackle in detail the specific topics of the chapter. Our intention with this monograph is that after reading a chapter, the reader gains a solid awareness of the state of the practice and is left willing to learn more about that topic to implement it in their organization.

It has been our intention to write the monograph with an outreach style yet aiming at being technically concise and sound. We have tried to limit content that describes visions for the future of systems engineering, opinions about the state of affairs in systems engineering, or ‘sales’ pitches that are shallow and unsupported by practice or research. Our goal has been to keep the content as factual as possible, without overstating existing capabilities yet without ignoring modern advances. It is important to recognize however that, given the fluid state of the field right now, different organizations will be at completely different points of maturity with regards to the material presented in the monograph; I dare to state that some may not even be mature on the practices described in the old “blue books” yet. Such organizations should not take the content in this monograph as a utopia, but rather as evidence that there is a path for them to develop and mature their systems engineering capabilities.

The monograph contains six chapters:

Chapter 1 presents the current context in which systems engineering is applied in Europe, as well as some of the competencies of the systems engineer of the 21st century. Main topics include the transition from vertical integration to specialization, the complexity of contractual structures, the alignment of objectives across the supply chain, international teams, dual roles of customer/contractor, and market and political constraints.

Chapter 2 presents three aspects of modern and future systems that may jeopardize traditional systems engineering practices: highly cyber physical systems, distributed governance (systems of systems), and learning-based systems and human-machine teaming. Emphasis is given to current practices with doubtful effectiveness for such kinds of systems and the chapter presents current trends on how to address these unique aspects of these new systems.

Chapter 3 presents the need to evolve and adapt systems engineering development models to the current context of engineering projects. The chapter separates the discussion between traditional, dominantly plan-driven approaches to systems development such as the Waterfall and Vee models, and agile development approaches that emphasize responsiveness. Most projects would benefit from both kinds of models, and the authors discuss hybrid approaches and how to tailor the development models to the context of particular projects.

Chapter 4 moves the discussion from systems and processes to technology for the systems engineer. This chapter formally introduces MBSE, and it covers aspects such as the effects of formalizing systems engineering, the divergence and convergence of semantics, and authoring, conducting reviews, and configuration control in model-based environments.

Chapter 5 presents the novel capabilities enabled by digital models and high computational power of current workstations to support system development and integration. Topics include automated generation and evaluation of architectures, set-based design, and intelligent management of the supply chain and integration process. among others.

Finally, Chapter 6 addresses the same aspects as Chapter 5 but applied to deployment, operations and sustainment, and retirement. Topics include the use of digital twins to perform predictive maintenance, Virtual Reality environments to train users, and the use of Artificial Intelligence to define operational strategies, among others.

Personally, it has been an honor to edit this first monograph for Isdefe's new series, and to work with such a talented pool of authors. I feel fortunate that they all agreed to embark on this adventure with us. I am very grateful to Isdefe and their project management team for trusting and engaging me in this initiative, as well as for having always showed continued support.

On behalf of the authors, the project management team, and Isdefe, I hope that you find the reading educative, enjoyable, and useful.

Dr. Alejandro Salado
The University of Arizona

/Administration
/Human Resources
/Legal
/Accounting
/Finance
/Marketing
/Publicity
/Promotion
/Research
/Business
/Development
/Engineering
/Manufacturing
/Planning

TABLE OF CONTENTS

INTRODUCTION TO SYSTEMS ENGINEERING IN THE 21ST CENTURY

"This is where most of us are today: trained in some other field and picked up systems engineering as best we could, engineers with on the job training in SE."

A.W. Wymore

CHAPTER 1

Systems Engineering in the 21st Century

Dr. Alejandro Salado, *The University of Arizona (alejandrosalado@arizona.edu)*
Adolfo Sánchez Domínguez, *Isdefe (asdominguez@isdefe.es)*
Dr. Dinesh Verma, *Stevens Institute of Technology (dverma@stevens.edu)*

Abstract

This chapter presents the current context in which systems engineering is applied in Europe, as well as the ideal competencies of the systems engineer of the 21st century. Main topics include the transition from vertical integration to specialization, the complexity of contractual structures, the alignment of objectives across the supply chain, international teams, dual roles of customer/contractor, and market and political constraints.

Keywords

Keywords: systems engineering history, systems engineering evolution, skills and competencies.

1. A TRADITIONAL PERSPECTIVE TO SYSTEMS ENGINEERING

Traditionally, systems engineering has been conceived as the glue that connects disciplines in an engineering endeavor on the one hand, and the user or customer on the other. The systems engineer acts as a sort of technical coordinator, who makes sure that the decisions made by traditional engineers (e.g., electrical engineering, mechanical engineering, etc.) are well balanced and aligned towards a common goal, which is defined at the system level. A common analogy has been that of an orchestra director, who makes sure the harmonies, timings, and volumes played by each instrument are adequately aggregated to deliver a (hopefully) beautiful piece of music [1]. Without such a direction, the orchestra would likely yield a cacophony instead. The need to balance the desires of engineering disciplines has been captured by several people using cartoons such as the one in Figure 1, which caricaturizes the design of a smartphone as desired by different engineering disciplines and the resulting product once those have been balanced with the customer in mind.

While technical coordination and integration arguably remains the most common representation of systems engineering, such a view is incomplete though. Lifecycle is the other pillar under which the traditional systems engineering perspective is built. The systems engineer also oversees the evolution of the system throughout its lifecycle (that is, from conception to retirement), and in fact foresees those considerations to inform early decisions; the systems engineer thinks about the end before the beginning. If technical coordination or integration is important to balance the desires of engineering disciplines, being intentional about lifecycle considerations is important to promote consistency between the actual need to be satisfied and the final product that is released into operations. This paradigm has also been often represented by cartoons, such as the one in Figure 2, since at least the 60's.

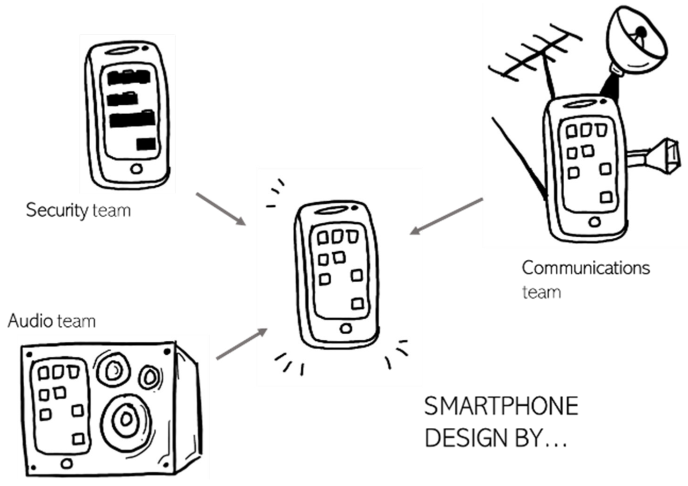
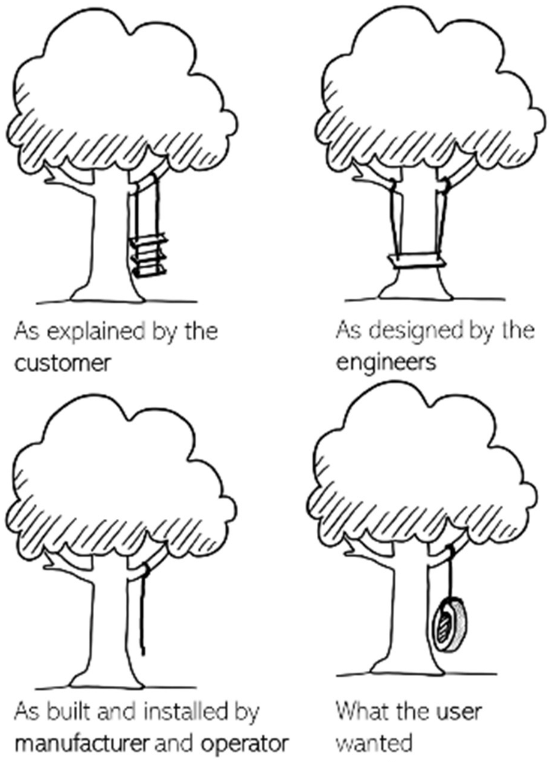


Figure 1. Balancing engineering silos [2]


As explained by the customer
As designed by the engineers
As built and installed by manufacturer and operator
What the user wanted
Figure 2. Integration of lifecycle considerations [2]

Several tenets are central to this perspective, which include among others big picture thinking (sometimes equated with a systems mindset), foresight, and communication and influence [3]. Big picture thinking is the ability to identify connections (or relationships) between parts and understand how those relationships yield emergent behaviors at a higher level (usually called the system level). Following the paradigm of the technical coordinator, the big picture thinking is exhibited when an engineer understands, for example, that the power consumption of a given part is not critical in isolation, but that it has effects on the sizing of the power supply, which may affect the required capability of the thermal control system, which may in turn affect the design of the housing structure, which may affect the electromagnetic performance of the system, etc. This applies as well to the paradigm of the lifecycle considerations, where big picture thinking is exhibited when an engineer understands, for example, that a given architecture may achieve an excellent performance but at the unbearable cost of a very intricate integration process or impossible future maintenance.

Probably unsurprisingly, technical coordination in the context of integrating lifecycle considerations is done with the purpose of promoting success in the development of the system. That is, one wants to avoid the unpleasant surprise that the system they developed is infeasible to deploy and operate or even unfit for purpose; this is regardless of whether this is the result of technical inconsistencies or unbearable operations. The ability to foresee and anticipate issues and problems during system development becomes therefore instrumental for a systems engineer. Because of the strong influence that early decisions can have in the success of the development effort (both in terms of development efficiency and solution effectiveness) [4], foresight and anticipation enable steering the development work early on in a way to avoid or easily mitigate the consequences of obstacles that could emerge during the system's lifecycle.

The ability to communicate and influence others is essential for anyone aiming to coordinate the work of other engineers, as well as for anyone working across lifecycle boundaries. For example, a systems engineer may have to convince an antenna engineer that their awesome antenna is unnecessary, and a worse-performing antenna is not only acceptable but what they actually need to yield a feasible solution at the system level. Similarly, a systems engineer may have to convince a thermal engineer that their thermal model does not need so much accuracy, and that just a coarse estimate is sufficient for a particular project. Coordination does not only occur between engineering disciplines though. A systems engineer may have to explain to a project manager why certain modifications are critical (and do so without resorting to technical jargon!), as well as translate some project management concerns to their engineering team (and do so without appearing to be a manager just cutting down their budget!).

Until very recently and probably in most organizations still today, these skills have predominantly been developed through experience [3] and have heavily relied upon talent [5]. Informally described as scar tissue, the systems engineer possesses a knowledge base acquired through suffering and learning from multiple mistakes and problems in several projects. Every unknown unknown that a systems engineer encounters during his/her career (that is, events that they are unaware of and still materialize), even before they became a systems engineer, is converted into a known unknown that is added to his/her personal set of heuristics or principles (that is, the engineer knows now that the event could occur), acquiring the ability to foresee and act on them before they occur again. Essentially, you've been there, you've seen it, you've lived through it. In addition, a natural byproduct of accumulating such experience is the building up of respect and/or reputation among peers [6]. Generally, these significantly improve one's position when it comes to exerting influence and persuading others [7], and are often associated with career promotions, which afford more opportunities to engage in diverse forms of communication. Furthermore, those individuals with curiosity to stretch beyond one's domain of expertise, can develop their breadth of knowledge as they exchange insights with others in multi- or interdisciplinary teams [8].

Therefore, systems engineering has been traditionally conceived as a step for some senior engineers in their career progression: those that were very good in their respective domains, developed a solid breadth of knowledge across the domains that build up the system of interest, and were able to communicate across silos (including to management, customers, and other individuals engaged in various areas of system development across the lifecycle) would become the systems engineer for a specific project [3]. Systems engineering would not be something you learnt in college, but rather something you would become and grow into.

Given the lack of scientific foundations for the discipline [9], processes became early on an effective vehicle to harmonize systems engineering work [10]. These became so prevalent in systems engineering practice that many started to refer to systems engineering as the systems engineering process, rather than considering it a discipline. Actually, this view is still prevalent in several spheres. (One just needs to search for the term on the internet, and they will find such treatment by corporations, federal agencies, and universities as of the time of writing this chapter!) Such a process is generally described (simplistically) as a top-down sequence of activities that transform and decompose needs and requirements into a set of components that are later integrated to form a system. This view reinforces the idea of systems engineering as a step in an engineer's career progression, since one only needs to learn and apply a process to an engineering effort.

Given the proliferation of different processes to support a systems engineering effort, such as the Spiral development or more recently Agile methodologies, there has been a trend in the systems engineering community to not consider systems engineering a process itself, but rather a methodological approach: the systems approach. In fact, this is the purview of the International Council on Systems Engineering (INCOSE), which defines systems engineering as "a transdisciplinary and integrative approach to enable the successful realization, use, and retirement of engineered systems, using systems principles and concepts, and scientific, technological, and management methods" [11]. While the approach paradigm certainly encompasses a broader understanding than what the process paradigm did, there is still an underlying implication of systems engineering not being a discipline itself, but something that can be learnt transversally to one's career.

This conception of systems engineering has remained virtually unchanged since the days of the Apollo program. In fact, the Apollo program has remained the main paradigm for the education of engineers for the most part. No doubt, Apollo is probably one of the most, if not the most inspiring and bold engineering endeavors humankind has witnessed. (We certainly believe so.) One may therefore argue that, if such is the case, why not use Apollo as the epitome of how engineers are to be educated, trained, and developed? The answer to that question may be in the fact that it is also fair to acknowledge that the context in which the Apollo program happened may not be representative of the contexts that most engineers face today.

2. THE CURRENT EUROPEAN CONTEXT FOR SYSTEMS ENGINEERING

2.1. Organizational, political, and industrial complexity

As stated, the Apollo program was an outstanding engineering undertaking. But it also benefited from certain contextual features that we do not see often if ever in today's engineering programs. Leaving aside the fact that the Apollo program was a race in the middle of a war (even it was the Cold War), let us look at how NASA's funding at the time compared with today's funding situation (ref. Figure 3).

The are three features to observe. First, one can easily see the peak of funding that NASA received during the Apollo days, which is around double of what NASA received after landing on the Moon. Second, one should note that NASA's budget during the Apollo program was mainly consumed by the Apollo program, while today's budget is spread through many ongoing and future NASA projects. This allocation of resources further amplifies the funding difference between the Apollo program and other NASA programs today (at least as consumed per year). The third aspect, which is probably the most important one with respect to context for engineering practice, was told to the first author of the chapter by a colleague, who supervised a PhD student researching government funding allocation during the Apollo program. Every budget request that NASA solicited to government was approved. That means that, while the funding in today's projects reflects NASA budget (what they have available to spend), the histogram for the Apollo years show what NASA needed to complete the project. Effectively, engineers during the Apollo program did not have cost constraints.

But financial resources are not the only contextual difference between then and now. Organizations adopt one of two primary paradigms to tackle the development of complex systems: vertical integration or horizontal integration. In vertical integration, a single organization owns all stages of the system development process, encompassing conception, engineering, manufacturing, and production. Conversely, organizations adopting horizontal integration rely on different external organizations, which specialize on different aspects of the engineering endeavor, allowing them to leverage their expertise and knowledge. For example, while a vertically integrated organization would design and manufacture every component needed for their system, a horizontally integrated organization would purchase a given component

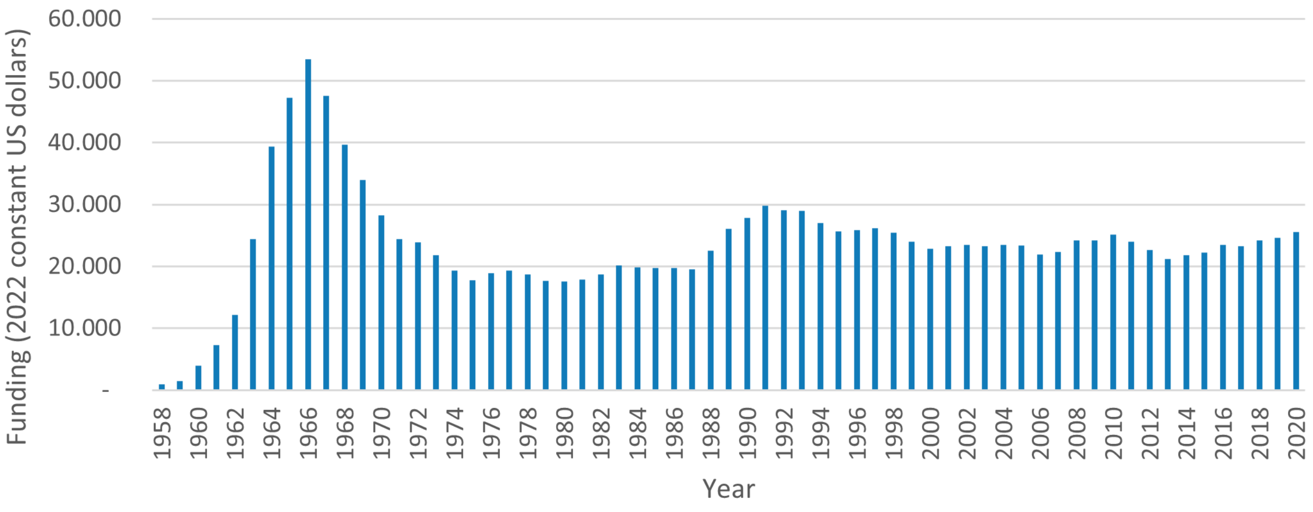


Figure 3. NASA funding over time

(e.g., a battery) to another company that specializes in such kind of technology (e.g., a company that *only* designs and builds batteries). While vertical integration offers greater control over system development, horizontal integration promises higher efficiency due to the allocation of different tasks to expert companies in those tasks. It should be noted though that vertical and horizontal integration should not be understood as binary, but rather a tendency towards more vertical integration or more horizonal integration.

Vertical integration was common in the early days of systems engineering, probably because existing companies would simply grow their existing product portfolio towards products of higher complexity and scale. However, horizontal integration was later favored in the hopes of higher financial efficiency and lower risk promised by specialization and became the norm for large scale engineering endeavors. In fact, horizontal organization is still the most prevalent approach today, particularly in Europe. However, organizations have started to accept that specialization in these areas (i.e., large scale systems such as those in the defense and space sectors) has not lived up to its promise, at least in the way in which it has been implemented.

Horizontal integration introduced the need for contractual structures to govern the relationships between the different organizations engaged in the development of a system (here, the roles of prime contractor, subcontractor, etc. emerge). The burden imposed by contractual constraints often leads to significant costs that, in many cases, have jeopardized any possible gains obtained through specialization[1]. Furthermore, this was an aspect that, while taken into account from a philosophical or conceptual perspective in systems engineering, was not embedded within systems engineering practice in a way that could be operationalized to effectively navigate and/or blend contractual and engineering aspects of an engineering project.

In Europe, this situation is exacerbated by the geopolitical constraints emerging from the need for various countries to not only work together, but to compete and capture European funding. It is not uncommon in European projects that the funding that an organization may receive to pursue an engineering endeavor is proportional and/or bounded by the financial contribution that the country made to the project through the European financial and political channels. This may result from a country's goal to manage a project, to keep control or leadership of a technology, or to develop new capabilities or technologies unavailable to the country at that time, among others. To compete in this context, many multinational companies replicate capabilities in different countries instead of fully implementing specialization, leading to internal competition that confounds with political constraints and competition with other external organizations.

Today, certain organizations are attempting to transition back to vertical integration, in the hopes to integrate their engineering teams more easily towards a common goal. SpaceX has shown with its rockets a notable example of how vertical integration in the 21st century has contributed (albeit not the only contributing factor!) to significantly lower the cost of a large-scale system with respect to horizontal integration.

1. While contracts existed before, the fact that budgets were virtually unlimited made contracting just a vehicle to communicate work, not a source of risks affecting the development effort.

The geopolitical constraints remain though in the European context. While in the USA the political and cultural union with a centralized government facilitates the execution of large-scale engineering programs with a high technological content, in the European context, the fragmentation into multiple states with diverse political and economic interests makes it difficult to tackle large engineering programs that, due to their size or complexity, would be impossible to undertake alone. Supranational organizations (such as the European Space Agency) or international programs (such as Eurofighter, A400-M, or the more recent FCAS (Future Combat Air System)) try to mitigate the effects of this fragmentation by securing stable and prolonged funding commitments from member countries. (The boxed text presents some historical notes and examples of supranational organizations in Europe engaged in the development of large-scale engineered systems.) However, these supranational organizations do not resolve the problems inherent to particular interests of each member country when it comes to the distribution of workloads, the assumption of responsibilities, and the protection of their national industries and security interests.

A clear example of these difficulties is evident in the establishment of the FCAS program, which involves Germany, France, and Spain as the main contributing members. This program faced several challenges. One such challenge was the prioritization of the interests of certain national industries over the needs of the end users. For instance, France, led by Dassault, opposed Airbus serving as the national coordinator for the other two member countries. Additionally, Germany expressed doubts about the project's viability, while Spain preferred Indra over the more international Airbus as its national coordinator. These issues resulted in multiple delays and disagreements among the member countries and their industry partners. Clearly, the fragmented execution of this kind of engineering endeavors incurs enormous transaction costs related to negotiation and decision making among its members. Some of the consequences are the need to offer returns to each of the partners, the different weight of each of them within the programs, the location of production in plants that are sometimes suboptimal, or the need to transport parts between them when all the assembly could be done in a single location. In the end, all this contributes to higher prices for the end customer, excessive development times and, in most cases, delays [12].

Some examples of supranational organizations engaged in the development of large-scale engineering systems in Europe.

In Europe, the treaty establishing the European Economic Community (Rome, 1957) and even the 1997 Treaty of Amsterdam excluded the arms and defense sector from the Community sphere, making it difficult for governments to collaborate and to adopt a common development framework such as might exist in the United States. One of the initial endeavors to foster European cooperation in defense affairs emerged with the establishment of the Independent European Programme Group (IEPG) in 1976. This initiative was conceived as a technical association in alignment with the principles of the Atlantic Alliance, emphasizing the preservation of each member country's national responsibilities. The resolutions produced by the IEPG, though lacking binding authority, primarily served as a mechanism for exchanging information regarding national armament and equipment procurement procedures. Furthermore, they facilitated the investigation and evaluation of potential frameworks for overseeing joint projects.

In 1992, the IEPG became the Western European Armaments Group (WEAG) and was integrated into the Western European Union (WEU) as the body responsible for armaments cooperation. Despite continuing the procedures and relations of the IEPG, it established a series of new objectives such as the search for competition between the different national markets, the reinforcement of the technological base, and cooperation in defense R&D. In 1996, the Western European Armaments Organization (WEAO) was established; a new WEU subsidiary body essentially dedicated to managing research and technology activities.

Given the limited impact of these initiatives, a group of countries began to make progress in this field through different agreements until 1996, when the Organization for Joint Armament Cooperation (OCCAR) was formed, the main organization in the field of industrial cooperation in armaments and the embryo of the European Defence Agency (EDA). OCCAR's Program portfolio currently includes 17 important armament programs with a total operational budget in 2023 of about 6 Billion €: A400M, BOXER, COBRA, ESSOR, FREMM, FSAF-PAAMS, LSS, LWT, MALE RPAS, MAST-F, MMCM, MUSIS, NVC, PPA, REACT, TIGER, and U212 NFS. The governance of OCCAR and the management of the OCCAR programs follow the OCCAR rules.

Meanwhile, in Spain, ISDEFE (Ingeniería de Sistemas para la Defensa de España), a state-owned company, was established in 1985. It was created to serve as a resource for providing technical support in complex projects, with a particular focus on the defense, aeronautics, and information and communications technology sectors. Additionally, ISDEFE was tasked with offering technical assistance to the Ministry of Defense in systems engineering efforts, particularly related to the Modernization Programs of the Armed Forces. Since its inception, ISDEFE has been actively implementing Systems Engineering methodologies in the development processes of various systems, including command and control systems, air traffic control, platform reengineering, logistics chain optimization, intelligence and electronic warfare systems, and surveillance and border control systems. Notably, ISDEFE has played a crucial role in numerous military programs, such as the F-110 frigates, the VCR 8x8 wheeled armored vehicle, the A400M military transport aircraft, the Tigre helicopter, the S-80 submarine, and many others.

In 2004, the European Defense Agency (EDA) was founded to help its twenty-seven member states (all EU countries) to develop their military resources by pooling national interests and catalyzing the operational, technological, and industrial aspects required to implement multinational weapons systems programs, delegating the actual management of the programs in their development and production phases to organizations such as OCCAR.

In the aerospace field, two initiatives set the European framework for collaboration in systems engineering: the creation in 1975 of the European Space Agency (ESA) and the establishment in 2000 of the aerospace company European Aeronautic Defence and Space (EADS), renamed in 2014 as Airbus Group.

The creation of the European Space Agency (ESA), like that of EDA, was preceded by the creation of other bodies such as the European Space Research Organization (ESRO) in 1962, the European Shuttle Development Organization (ELDO), the European Space Research and Technology Centre (ESTEC), which would be responsible for the development of satellites and space vehicles, and the European Space Operations Centre (ESOC), responsible for the control of satellite operations. In 1973, with the global agreement of all member countries, three projects were approved (Spacelab, the Ariane Program, and Marots) and a fundamental decision was taken: the creation of the European Space Agency (ESA).

The European Commission has sought to overcome the issues of fragmentation by establishing the following initiatives:

- The Directorate-General for Defence Industry and Space (DEFIS), which directs the European Commission's activities in these sectors. In the defense industry field. DEFIS is responsible for maintaining the competitiveness and innovation of the European defense industry by ensuring the evolution of a capable European defense technological and industrial base. In the space domain, DG DEFIS is responsible for the implementation of the EU Space Programme, consisting of the European Earth Observation Programme (Copernicus), the European Global Navigation Satellite System (Galileo), and the European Geostationary Navigation Overlay System (EGNOS).

- The European Defence Action Plan (EDAP) of November 2016, which seeks to promote a strong and competitive European Defense Technological and Industrial Base (EDTIB) based on the European Defence Fund (EDF), main EDAP funding framework. This plan is based on four pillars through which specific actions and programs are implemented in the fields of research and development, enabling European defense supply chains and building a single European defense market.

- The Permanent Structured Cooperation (PESCO), of December 2017, which integrates 26 countries and whose key objectives are the improvement of defense capabilities, cooperation in military operations, and the development of joint military capabilities. Its ultimate goal is to strengthen the Union's strategic autonomy in defense matters.

These initiatives add to the ever-increasing set of regulations imposed by governments and federal agencies that organizations and engineered systems must comply with. Public agencies in Europe frequently navigate rigorous and highly regulated tendering procedures. While these procedures aim to ensure transparency and equal opportunities for bidders, they can sometimes become intricate and time-consuming. This complexity may lead to delays in the selection and contracting of engineering organizations. Moreover, the holistic nature of systems engineering demands bidding for contracts of various types (e.g., supplies, software licenses, consulting, works, etc.) involving multiple suppliers and subcontractors, which adds to the challenge of management and contractual relationships among stakeholders. Additionally, in the case of extensive transnational consortia, country-specific regulations for public procurement, data protection, and intellectual property must be harmonized.

In the context of Spain, for example, Article 99.1 of the Public Sector Contracts Law stipulates that “the object of public sector contracts must be determined. The same may be defined in attention to the specific needs or functionalities that are intended to be satisfied, without closing the object of the contract to a single solution.” This requirement compels organizations to clearly define the functional needs of contracted systems from the outset, preventing the project from leveraging the knowledge of companies during its early phases and, in some cases, the adoption of innovative development models, processes, or approaches, like Model-Based Systems Engineering (MBSE), if these are not mandated by the customer.

Time will tell if these initiatives succeed, but past experiences do not seem to correlate the addition of more governing bodies and regulations with success in system development. Concerns for overreliance on processes have been claimed as early as in 1969 already during the Apollo program, “If I plot a graph versus time of what appears to be a recent rising tide of costs, cost overruns, unsatisfactory performance and unhappiness among engineers, I have reason to worry. […] If I plot on the same graph versus time the rise in talk, directives … I see high correlation between the two graphs” [10], and were reaffirmed in 2010 by former NASA Administrator Mike Griffin [5]. In simplistic terms, each new regulation adds a new constraint that the solution must satisfy. And we know that each constraint reduces the solution space, and that a reduction of the solution space generally reduces the affordability of the developed system [13].

International collaboration, essential for major European programs, as described, presents other challenges that span beyond those formally introduced by geopolitical constraints. Some examples are listed below:

- Effective communication between multicultural and multilingual teams is not straightforward. Communication problems stem not only from language barriers but also from cultural nuances in communication styles. For example, what may be considered direct and clear communication in one cultural environment, may be considered disrespectful or confrontational in another. At the same time, multilingual teams may find it difficult to convey technical information accurately and in a way that is understood by all members because of losses of information in translation, as well as decreased cognitive performance due to the increase demand in cognitive load when speaking a second language.

- Coordination of activities involving personnel working in different locations and using different tools and technologies in their local work environment can lead to delays in exchanging information and coordinating activities. In addition, there are legal and security concerns for organizations within large consortia as cross-border data exchange may be subject to restrictions, which can add complexity to information exchange and collaboration efforts.

- Cultural and work habit differences may need to be overcome in order to achieve the ultimate goal of a joint system that meets expectations. The way decisions are made can vary greatly from one culture to another. Some cultures may prefer a consensus-based approach, while others rely on hierarchical decision-making. These differences can affect the effectiveness of the decision-making processes in a collaborative project.

These issues should not be interpreted as stating that international collaboration should not be pursued. Not only is international collaboration necessary to develop some large-scale systems, as explained earlier, but it is also preferable in many situations because of several benefits that it can yield, such as leveraging expertise and knowledge. For example, in the case of the Spanish S-80 submarine program, the Spanish Ministry of Defense contracted with the US Navy and General Dynamics-Electric Boat, which allow them to rethink the program and implement better engineering processes and methodologies, including adopting the NASA Systems Engineering Handbook. The point made in this section is that this sets a new context in which systems engineering must be applied, and for which methods and/or approaches may need to be evolved.

2.2. Engineering and technological complexity: New systems, new methods

We have seen in recent years, and continue to see, the emergence of new kinds of systems that exhibit fundamental differences with respect to those with which systems engineering was born. Traditional systems were characterized by being predominantly hardware-based (software was an isolated *tiny* part in those where there was software at all), monolithic (their capabilities did not depend on the capabilities of other systems), often developed in a green field (no dependency on legacy systems), and lacking agency (high predictability on the command-and-control behavior of the system). Traditional systems are less and less common, if they exist at all. For example, when the first airplanes were developed, all supporting systems also had to be developed to support the airplane (hence their name). Today, new airplanes are developed within the constraints of the existing supporting systems; support systems are no longer in a supporting role but in an enabling one. In other words, the development of contemporary systems is significantly constrained by their need to interact with many systems that are already in place (i.e., legacy systems).

However, the strongest novelties come from the increase in importance and size of software components in contemporary systems, the transition into distributed governance structures, and the emergence of agency.

In the past, software was limited to some very specific functionalities that were otherwise too difficult to implement in hardware. Software was conceived as a last resort, not a preferred go-to solution. Codes could be entirely reviewed and tested. Today, software drives most of the functionality of existing systems or, at least, most functionalities depend on software. One only needs to look at how cars have evolved; even the handbrake is controlled by software! The increase in the reliance on software, together with the increase in its complexity, challenges many of the assumptions under which systems engineering has been traditionally practiced. Software can be evolved quickly, it can be deployed incrementally, it introduces security vulnerabilities of an unprecedented diversity and potential severity, it is difficult if not impossible to comprehensively test, and it can be deployed and upgraded on the fly during operations, among others.

Traditional systems were monolithic in the sense that they alone could yield their intended capabilities; assuming supporting systems were in place. For example, a television would work fine as long as you could connect it to the power grid and tune its antenna. Today we are living the proliferation of systems of systems (SoS): those that are heavily interconnected with and rely on the capabilities of other independent systems whose primary goal is not to serve in a supporting role. In other words, systems that each have their own purpose are somehow leveraged to yield unanticipated capabilities. In the example of the television before, the purpose of the power grid is to provide energy to household devices and the purpose of the content providers is to provide content to consumers. Hence, in both cases the systems' purposes are to serve the television. However, a smartphone affords you the capability of personal navigation by leveraging signals provided by the Global Positioning System (GPS), even though the GPS's primary purpose is to guide missiles, not to help you find your way somewhere. Recognizing that not every collection of systems is necessarily a SoS is essential to avoid falling into the trap of calling and treating everything a SoS, as the term is often abused and has become a buzzword. Differentiating both is important because traditional systems engineering practices are likely ineffective and sometimes even infeasible to tackle the unique aspects of SoS [14]. In fact, there are even system attributes or performance metrics, such as availability, that we do not even know how to compute and/or predict for a SoS [15]. Certainly, one cannot simply use traditional systems engineering to engineer and/or integrate a SoS and expect to be successful. The application of systems engineering must be adapted (not tailored!) but, in all honesty, the systems engineering community has only started to grasp how to do so. As a body of practice, we are confident that managerial and governance independence of the constituent systems that form the SoS seem to be the latent factors that must inform the evolution. As the ISO standard to apply systems engineering to SoS indicates, approaches based on command and control assumptions, availability of information (e.g., to support verification), existence of requirements, or guaranteed services may be powerless [14]. Some have even claimed that SoS cannot actually be engineered, but only integrated [16], which makes processes related to requirements and architecture inapplicable. Instead, novel systems engineering methods that are based on persuasion and influence, use of incentives (e.g., mechanism design), opportunistic federations, or blended development and operation are being developed.

If distributed governance and the complexity of an ever-increasing reliance on software were not enough changes to the systems we must work with, we have started to witness the incorporation of agency in cyber-physical systems, mainly through artificial intelligence. The exhibition of agency dramatically changes several core assumptions that are central to systems engineering practice. For example, whereas traditional systems are considered to exhibit invariant behavior, in the sense that the behavior is set once the system is built, intelligent systems are designed to change their behavior as needed.

This challenges the effectiveness of many traditional systems engineering practices, which need to be evolved. For example, and certainly not exhaustively:

- Since an intelligent system may change its behavior between the test environment and the operational environment, tests in test environment may no longer be good proxies to predict system behavior during operation [17].

- If an intelligent system is expected to learn from its experiences, individual systems may exhibit an uncontrolled variability of its behavior with respect to its class or family. As a result, product lines may no longer be able to bound a class of systems [18].

- Intelligent systems have a stable underlying/supporting functionality (that is, to optimize a reward function) and the main driver of system performance and behavior is the training data. Functional decomposition is no longer able to capture how the system works, since it is data that drives behavior. How does this impact functional-based analyses such as fault-tree analysis or Failure Modes and Effects Critical Analysis (FMECA)?

While some of these problems may be relevant for software-based intelligent systems (such as a function that predicts your purchasing behavior in an online store), their solutions may not be directly transferrable to general systems, because of the highly coupled effects of the physical world. A new evolution of systems engineering is therefore necessary.

3. SYSTEMS ENGINEERING OF THE PRESENT-FUTURE

Traditional systems engineering paradigms and methods are becoming ineffective to deal with the new contexts – organizational and technological – in which engineering endeavors are being undertaken. The role of a single senior engineer trying to coordinate technical efforts by managing information in countless documents, manually controlling configuration items, relying on free-style sketches to model different system facets, and gut-feeling most decisions is slowly fading away. Methods underpinned by science, formal decision making, formal modeling, digitalization of engineering artifacts, and a diversity of systems engineering competencies that can be distributed within a team are shaping the present and future of systems engineering.

An overview of some aspects that are relevant to the contemporary and evolving context of systems engineering is discussed in the following sections. Note that these sections are not intended to provide a vision for the future of systems engineering or a roadmap for its evolution; some proposals can be found in other publications (e.g., [19]).

3.1. Emerging theories and foundations

The origins of systems engineering and its development as an engineering practice in the 1960's were accompanied by several efforts to formalize the discipline. Pioneers such as Wymore (e.g., [20]), Warfield (e.g., [21]), or Mesarovic (e.g., [22]), among others, tried to develop mathematical foundations to support systems engineering practice, which, as stated earlier, was primarily driven by intuition, talent, and eventually processes. However, it is fair to state that their work did not succeed in traversing the academic realm into informing systems engineering practice at the time. For several years their work did not only remain unused but the very interest in discovering the foundations of systems engineering faded away against more applied and readily usable research that focused on developing methods.

Today, there is a growing recognition, both in academia and industry, for the need of scientific foundations to inform systems engineering practice. Science is a core element of engineering. Absent of scientific principles, we would probably talk of craftmanship rather than engineering. Certainly, humans built bridges and manufactured products well before Newton formulated the first laws of motion. People resorted to their intuition, heuristics, and experience and were successful with them; here we are today, writing a book chapter on a computer! But we would not say that they practiced engineering. Engineering a bridge or a product is an entirely different feat, since the underlying science enables us not only to understand how things work, but also to use that knowledge to better predict the results of our decisions to improve the effectiveness and efficiency of the products.

By the same token, we should probably be calling systems craft to what we do today, and not use the term systems engineering yet. We have not been able to even agree on or find a rigorous definition for what a system is! [23] Let alone more intricate concepts such as requirements, needs, specifications, architecture, ilities, etc. To move our systems engineering practice, which is built upon experience, gut feeling, good practices, and supposedly good ideas, into real engineering, where we can undoubtfully assess the goodness of a systems engineering method and have a common and consistent set of concepts and associated vocabulary, among others, we need scientific foundations to build upon.

The U.S. National Science Foundation (NSF) has been supporting and funding fundamental research in systems engineering for at least two decades. The Department of Defense (DoD) funded the Systems Engineering Research Center (SERC) 15 years ago as a network of collaborating universities with the goal of transforming systems engineering practice by creating innovative methods, tools, and processes and bringing academia and practice closer together. As of the time of writing this chapter, INCOSE has launched its Future of Systems Engineering (FuSE) initiative, which includes the Foundations of Systems Engineering as one of its fundamental tracks. While we are far from having a comprehensive and mature set of scientific principles to underpin systems engineering, research has already unveiled several of them, even if most have still not transitioned into practice.

Today, we know that most of the decision methods and/or processes that are used in practice are fundamentally flawed and their recommendations should not be trusted, and why. We know that risk matrices generally embed wrong orders of criticality, and why. We know that most categorizations of requirements lead to poor requirements, and why. We know that verification agreements between customers and contractors should be based on programmatic discussions, not on technical ones, and why. We know that existing MBSE tools are unable to model requirements and that flagging models as requirements leads to poor solution spaces, and why. We know that decisions in engineering should not be consensual in general, and why. We know that verification plans should not be baselined and contracted early in the system development, and why. And, moreover, we also know how many of those activities should be done, and why. The list keeps going but we are not trying to be exhaustive, just indicative.

Modern systems engineering needs to address these deficiencies, whilst recognizing the organizational and governance complexity inherent in conception and development of large-scale systems.

3.2. Systems engineering beyond technical coordination

The conceptualization of the systems engineer as a technical coordinator or technical manager has significantly expanded in recent years. In addition to these, the systems engineers of today can take other roles, including but not limited to requirements engineer, system designer or architect, system analyst, verification and validation engineer, interface engineer, operations engineer, and information engineer (including configuration management and metrics) [24].

Lately, we are even seeing a significant growth of job demands for system modelers (or unfortunately called MBSE engineers), engineers that specialize in the application of MBSE and support traditional systems engineers or systems engineering teams with modeling needs, from converting their ideas into formal models to taking care of model management tasks. In essence, as the field matures, systems engineers may specialize in the different processes or tasks that systems engineering encompasses.

To aid the personal development of the systems engineer, INCOSE has developed a framework that captures the competencies that a systems engineer can acquire and should acquire when targeting certain competency level [25]. The framework categorizes competencies between core (those that all systems engineers should have, such as systems thinking or critical thinking), professional (those that relate to the work of the systems engineer within an engineering team, such as technical leadership or negotiation), management (those related to technical coordination, such as risk management), and technical (those that relate to systems engineering processes, such as requirements or architecture). And it divides expertise between five competency levels, from awareness to expert. An example is shown in Table 1.

A core benefit of the framework is the recognition that a systems engineer does not need to impersonate the knows-it-all role, but that his/her competencies can be developed in line with the systems engineering role that he/she may take. This allows teams to move from every engineer executing systems engineering in their domain (for which strong systems engineering competency cannot probably be developed) to assigning high competency roles within the team, sharing different systems engineering tasks.

Awareness	Supervised practitioner	Practitioner	Lead practitioner	Expert
Describes different types of requirements	Assists with the elicitation of requirements from stakeholders	Elicits and validates stakeholder requirements	Defines and documents enterprise-level policies, procedures, guidance and best practice for requirements elicitation and management processes, including associated tools	Coaches lead practitioners in requirements elicitation and management
Explains why there is a need for good quality requirements	Describes the characteristics of good quality requirements and provides examples	Writes good quality, consistent requirements	Reviews and judges the suitability and completeness of the requirements set	Advises and arbitrates on complex or sensitive requirements-related issues

Table 1. Example of competencies for requirements definition at different levels of competency [extracted from [25]]

Specialization in different systems engineering areas is meaningful. And, while we are going in a good direction, there is still work to be done. For example, it is highly unlikely that someone would assign an electronics engineer the responsibility to perform the structural analysis of an airplane. For such a task, you would choose an engineer that had graduated college with a mechanical engineering degree, had several years of experience conducting structural analyses of growing complexity, and had potentially pursued advanced degrees in structural engineering. However, we find it absolutely sensible to take that electronics engineer and assign him/her the task of writing requirements for a multimillion-dollar system after just offering him/her a 2-day seminar on requirements engineering. Yet, most of the research and reports of most frequent causes of project failure do not list incorrect structural analyses as one of the main causes; eliciting the right requirements right is recurrently listed though (e.g., [26, 27]).

To fill this need, academia is stepping up, both through research (as explained in the previous section) and education. The first undergraduate program in systems engineering started in the early 1960's at the University of Arizona, with a heavy focus on math, and one of the first master degrees (if not the first one) in the late 1960's at Virginia Tech, with a heavy focus on technical coordination and integration. However, it was not until the mid-2000's that the Stevens Institute of Technology reimagined systems engineering education, with master degrees that focused on the different specialization areas and competencies that are required in modern systems engineering. Such a paradigm has triggered immense growth of master level offerings in the United States of America.

This has been driven by the demands of industry; it is just too risky to rely the future of multimillion dollar ventures on individuals that ignore good systems engineering practices and methods. Formal systems engineering is also slowly making its way into Europe, although maybe too slowly. Systems engineering degrees are rather scattered in different countries, instead of being regularly offered by most universities. For example, as of the time of writing this chapter, only the European University of Madrid offers a postgraduate program in systems engineering in the country, and it is not even a masters degree after the last educational reform. To clarify, it is not the degree that matters, but the competencies and expertise that formal education can yield. This is something not even discussable for traditional engineering disciplines, and hopefully a future paradigm for systems engineering.

The growth and development of formal systems engineering education at all levels (undergraduate, graduate, and even recently apprenticeship) is also changing how systems engineering exists as a career. If traditionally systems engineering has been a career progression, as described earlier in the chapter, we start to see systems engineering as a career choice itself, where one can progress from junior positions to more senior positions. For example, a junior systems engineer might be responsible for deriving and managing uncritical requirements for the system, while the senior systems engineer can focus his/her attention to just the subject of critical ones.

The emergence and adoption of MBSE is also contributing to this paradigm change, as it is becoming increasingly easier for an organization to recruit MBSE talent directly out of college than train and change the ways of its more senior personnel. In fact, some community colleges have started to offer associate degrees/apprenticeships for system modelers that would take jobs similar to those that drafters had with respect to mechanical drawings.

3.3. Novel methods and tools

The advancement of affordable computational power, together with the formalization of systems engineering (as discussed in section 3.1), have enabled the development of computer-based methods and tools to support systems engineering activities in unprecedented ways. Probably, the most impactful innovation has been the development of MBSE.

In a nutshell, MBSE promotes the use of formal machine-readable models to capture systems engineering artifacts that used to be captured in narrative form or informal models using general office tools (e.g., word processors, spreadsheets, or free drawing tools). In principle, there is no limitation on what systems engineering artifact is captured as a model (e.g., requirements, use cases, functional architectures, physical architectures, or verification plans, among others). But in addition, MBSE also captures the relationships between those artifacts (e.g., allocation of requirements to components, allocation of functions to components, traceability between requirements and their verification activities, etc.). The transition into a model-based environment is expected to provide several benefits, including among others better communication, improved consistency of information, reduction of development errors, and improved time and cost efficiency [28]. While there seems to be informal agreement within the community of practice about these benefits, it is important to mention that there is a general of lack evidence that these benefits can be actually realized [28].

Today, MBSE is experiencing rapid growth in availability and capabilities of tools, underlying modeling languages, and industry and government adoption that we could only dream of 10 years ago. In this sense, the adoption of Computer-Aided Design (CAD) in mechanical engineering has been often used as an analogy for how MBSE could transform SE. Several organizations are pursuing a similar transition path, opting to acquire software licenses, provide a short training to their employees on tool usage, and let them run with MBSE. However, MBSE is not just a technological evolution, as CAD was [29]. MBSE has implications at the process and methods levels, in addition to the adoption of dedicated computer tools, which also need to mature for the technological adoption to be successful in the long term. In other words, MBSE does not improve poor systems engineering. For example, in terms of modeling, users can now choose between SLD (the proprietary language of Vitech Corporation, Inc.), the Systems Modeling Language (SysML, managed by the Object Management Group and for which its v2 is currently being developed), Capella (an open language developed by Thales), the Object Process Methodology (a language underpinning an ISO standard), and the Lifecycle Modeling Language (LML). These languages or modeling frameworks do not only have different strengths and weaknesses, but their underlying structure has significant implications to the practice of systems engineering. Furthermore, the languages do not offer in most cases a direct translation to guarantee compatibility between them. Failing to realize that the three facets need to be considered for adopting MBSE leads to failure [30, 31].

MBSE adoption has primarily focused on the early development activities, mainly supporting requirements management, system architecture, and change propagation assessment, and is predominantly descriptive. That is, systems engineering models are used to describe aspects of the system (for example, how two components interface with each other) rather than to support quantitative analyses (e.g., evaluating the system level performance given the performance of its components). There are certainly exceptions to this, which have proven the feasibility of using MBSE in later phases of the system development (such as, for example, to plan test and integration activities [32, 33]), and the value of extending descriptive models to perform quantitative analyses [34] or executable simulations [35]. But these are not widely available or employed yet.

Furthermore, MBSE discussions are shifting towards a more general, digital engineering paradigm. In a digital engineering environment, all engineering artifacts are captured in computer models that are semantically connected with each other. This means, for example, that the power consumption attribute in an electronics model is connected with its corresponding power dissipation attribute in a thermal model and with its corresponding power consumption attribute in a power budget system model. This does not imply that all models use the same data and/or values for each parameter. Rather, digital engineering enables the formalization of authoritative sources of truth, which ensure the validity and truthfulness of the data sources. Digital engineering does not invalidate or substitutes MBSE, but it is intended to expand its ideas beyond the scope of systems engineering and throughout the entire lifecycle, from problem formulation to manufacturing to operational support.

Differently than with the adoption of MBSE, which has been facilitated through hybrid top-down and bottom-up efforts (that is, led in some cases by teams of engineers and in some cases supported by corporate leadership or customer mandate) [30], the adoption of digital engineering is being strongly mandated from leadership. The U.S. Department of Defense (DoD), for example, has established a dedicated Digital Engineering strategy [36], which has been rapidly flown down into its service branches. A similar action has been taken by the Ministerio de Defensa in Spain [37]. This approach surfaces a conflict between the urgency with which the customer or organizational leadership desires to advance their capability and the readiness of the community of practice to provide effective solutions in terms of processes and tools. Today, the envisioned capability is still far away from technological readiness.

Computational power has also enabled the acceleration of otherwise time-demanding tasks. Traditionally, the exploration of the solution space during conceptual design or system architecture efforts has been limited to trading-off a handful of alternatives in search of the most preferred solution. However, structured system models can be used now to increase the size of the solution space that can be explored, called tradespace, and automatically evaluate thousands if not millions of solutions *at once* [38]. Sensibly, the larger the solution space that is explored, the more likely it will be to find a better solution. This changes the focus of conceptual design from refining a reduced set of individual possible solutions to construct modular structural models that enable breadth and depth of exploration.

Furthermore, system development has been traditionally limited to a single solution that is chosen and iteratively refined until it goes into production and is later deployed. This means that a system concept or system architecture is chosen, and then it goes through detailed design, where iterations may occur, as the solution is refined into a working one with sufficient maturity to be manufactured and eventually deployed.

This paradigm has been called point-based design, since the design is based on choosing a single solution (point) in the solution space early on. However, computer models can significantly reduce the effort that it takes to maintain and refine a solution. The novel paradigm of set-based design proposes to leverage this advantage to avoid anchoring to a specific solution early in the system development [39], when knowledge is limited [4]. In set-based design, a set of solutions (points) in the solution space are chosen and refined together. Unattractive solutions are only discarded from the set once there is sufficient knowledge to do so with sufficient confidence. This provides flexibility in the development process without requiring changes to solutions, as acceptable solutions are retained.

The way in which past experiences and organizational knowledge may be used in systems engineering is also dramatically changing thanks to the use of machine-readable models and computational support. Cognitive assistants are machines (usually software systems) that aid humans in cognitive tasks [40]. Amazon's Alexa or Apple's Siri are good examples of cognitive assistants that we have started to use frequently, and ChatGPT is further amplifying the capabilities that this kind of systems can provide to humans. A key aspect is that the engineer can use natural language to dynamically and iteratively develop systems engineering artifacts with the support of the assistant. In systems engineering, for example, we have now the capability of not only asking a cognitive assistant to develop conceptual solutions or systems architectures for a given set of parameters, but also to explore the tradespace in several directions and explain its choices and recommendations [41-43].

The evolution of methods and tools has not been limited to those enabled by advances in computational capabilities though. We are also witnessing the emergence of new development processes, models, and paradigms that have emerged out of necessity to cope with the peculiarities of the new context and kinds of missions in which and to which systems engineering is applied now, as discussed in the previous sections.

Development processes that are more iterative, less scaffolded, and more rapid to field systems are becoming more and more frequent (e.g., agile, DevOps) and integrated with more traditional ones (e.g., Waterfall, Vee). Considerable progress has been made to develop the notion of systems of systems engineering, which has been lately redefined as *mission engineering.*

In mission engineering, the focus is deliberately put on the planning and integration of current and future system capabilities towards fulfilling current and future operational needs.

The main distinction here is that those system capabilities are provided by independently governed and/or managed systems.

This discussion is not exhaustive. There are other advances in systems engineering that are probably worth noting, but these should provide an idea of what the landscape of the current practice of systems engineering is (even if still futuristic for some organizations).

4. CONCLUSIONS

The systems engineering landscape has changed in recent years, and seeds have been planted to promote further changes in the years to come. This chapter has hinted at what those changes are and why they have been or are necessary. Resources are not unlimited anymore; rather, they are generally scarce. Systems engineering endeavors trespass geopolitical boundaries and must balance the consequences of industrial consolidations, rapidly evolving technologies, and the emergence of agile startups and rapidly changing market needs. The nature of the systems we must develop and work with now has also changed. Most developments are no longer in a green field, and we must often have to reconcile the constraints of large legacy systems with novel aspects related to increased cyber nature of systems, distributed governance, and even intelligence.

The systems engineer must outgrowth themselves from just being a technical coordinator to a systems engineering expert. Growing into a systems engineer on the job is not sufficient to effectively deal with today's systems engineering efforts. The systems engineer must learn novel methods and techniques, many of which start to be underpinned by research; many good practices have turned out to be not that good and many mature processes are no longer relevant. Technology is taking the center stage of the systems engineering effort, starting with the provision of digital assets that can connect across domains and moving into the use of cognitive assistants to augment the capabilities of the human engineer. While still nascent, these capabilities are rapidly evolving.

Systems engineering has arrived in the 21st century, and it will only keep maturing and evolving. Is your organization ready to modernize its systems engineering practices?.

REFERENCES

1. Ryschkewitsch, M., D. Shaible, and W.J. Larson, The art and science of systems engineering. Systems Research Forum, 2009. 03(02): p. 81-100.

2. Salado, A., Systems engineering, in The Engineering Management Handbook, B. Mesmer, et al., Editors. 2023, The American Society of Engineering Management: Huntsville, AL, USA. p. 361-370.

3. Pyster, A., N. Hutchison, and D. Henry, The Paradoxical Mindset of Systems Engineers. 2018, Hoboken, NJ, USA: John Wiley and Son, Inc.

4. Blanchard, B.S. and W.J. Fabrycky, Systems engineering and analysis. Vol. 4. 1990: Prentice Hall New Jersey;.

5. Griffin, M.D., How do we fix systems engineering?, in 61st International Astronautical Congress. 2010: Prague, Czech Republic.

6. Zinko, R., W.A. Gentry, and M.D. Laird, A development of the dimensions of personal reputation in organizations. International Journal of Organizational Analysis, 2016. 24(4): p. 634-649.

7. Manzoor, E., et al., Influence via Ethos: On the Persuasive Power of Reputation in Deliberation Online. Management Science. 0(0): p. null.

8. Delicado, B.A., A. Salado, and R. Mompó, Conceptualization of a T-Shaped engineering competency model in collaborative organizational settings: Problem and status in the Spanish aircraft industry. Systems Engineering, 2018. 21(6): p. 534-554.

9. Collopy, P.D. Systems engineering theory: What needs to be done. in Systems Conference (SysCon), 2015 9th Annual IEEE International. 2015.

10. Frosch, R.A., A new look at systems engineering. IEEE Spectrum, 1969: p. 24-28.

11. INCOSE, Systems Engineering Handbook. A Guide for System Life Cycle Processes and Activities. 5th ed. 2023, Hoboken, NJ, USA: John Wiley and Sons, Inc.

12. Villanueva, C.D., El Programa FCAS y la Industria Española de Defensa: una apuesta equivocada, in Revista Ejércitos. 2022.

13. Salado, A. and R. Nilchiani, A Research on Measuring and Reducing Problem Complexity to Increase System Affordability: From Theory to Practice. Procedia Computer Science, 2015. 44: p. 21-30.

14. (ISO), I.S.O., Systems and software engineering — Guidelines for the utilization of ISO/IEC/IEEE 15288 in the context of system of systems (SoS). 2019.

15. Salado, A. Abandonment: A natural consequence of autonomy and belonging in systems-of-systems. in System of Systems Engineering Conference (SoSE), 2015 10th. 2015.

16. Madni, A.M. and M. Sievers, System of Systems Integration: Key Considerations and Challenges. Systems Engineering, 2014. 17(3): p. 330-347.17.

17. Shadab, N., A.U. Kulkarni, and A. Salado, Shifting Paradigms in Verification and Validation of AI-Enabled Systems: A Systems-Theoretic Perspective, in Systems Engineering and Artificial Intelligence, W.F. Lawless, et al., Editors. 2021, Springer International Publishing: Cham. p. 363-378.

18. Shadab, N., et al. Product Herding for Intelligent Systems. in Conference on Systems Engineering (CSER). 2023. Hoboken, NJ, USA.

19. INCOSE, Systems Engineering Vision 2035. 2023.

20. Wymore, A.W., A mathematical theory of systems engineering: The elements. 1967, New York: Wiley.

21. Warfield, J.N. and J.D. Hill, A unified systems engineering concept. Vol. Monograph 1. 1972, Columbus: Batelle Memorial Institute.

22. Mesarovic, M.D. General systems theory and its mathematical foundation. in IEEE Systems Science and Cybernetics Conference. 1967. Boston, MA.

23. Salado, A. and A.U. Kulkarni, An Assessment of the Adequacy of Common Definitions of the Concept of System. INCOSE International Symposium, 2021. 31(1): p. 510-521.

24. Sheard, S.A., TWELVE SYSTEMS ENGINEERING ROLES. INCOSE International Symposium, 1996. 6(1): p. 478-485.

25. INCOSE, Systems Engineering Competency Framework. 2018.

26. GAO, DHS Annual Assessment: Major Acquisition Programs Are Generally Meeting Goals, but Cybersecurity Policy Needs Clarification. 2023.

27. GAO, Space Acquisitions: DOD Faces Significant Challenges as it Seeks to Accelerate Space Programs and Address Threats. 2019.

28. Henderson, K. and A. Salado, Value and benefits of model-based systems engineering (MBSE): Evidence from the literature. Systems Engineering, 2021. 24(1): p. 51-66.

29. Henderson, K. and A. Salado, Is CAD A Good Paradigm for MBSE? INCOSE International Symposium, 2021. 31(1): p. 144-157.

30. Henderson, K., T. McDermott, and A. Salado, MBSE adoption experiences in organizations: Lessons learned. Systems Engineering. n/a(n/a).

31. Henderson, K. and A. Salado, The Effects of Organizational Structure on MBSE Adoption in Industry: Insights from Practitioners. Engineering Management Journal, 2023. In press.

32. Salado, A., 5.5.2 Efficient and Effective Systems Integration and Verification Planning Using a Model-Centric Environment. INCOSE International Symposium, 2013. 23(1): p. 1159-1173.

33. Gregory, J. and A. Salado, Model-Based Verification Strategies Using SysML and Bayesian Networks, in Conference on Systems Engineering Research (CSER). 2023: Hoboken, NJ, USA.

34. Hecht, M. and J. Chen. Use of SysML for Quantitative System Reliability and Availability Analysis. in 2022 Annual Reliability and Maintainability Symposium (RAMS). 2022.

35. Karban, R., et al., Creating system engineering products with executable models in a model-based engineering environment. SPIE Astronomical Telescopes + Instrumentation. Vol. 9911. 2016: SPIE.

36. DOD, Department of Defense Digital Engineering Strategy. 2018, Office of the Deputy Assistant Secretary of Defense for Systems Engineering: Washington, DC, USA.

37. Defensa, M.d., Plan de Acción del Ministerio de Defensa para la Transformación Digital. 2020.

38. Ross, A.M. and D.E. Hastings, 11.4.3 The Tradespace Exploration Paradigm. INCOSE International Symposium, 2005. 15(1): p. 1706-1718.

39. Shallcross, N., et al., Set-based design: The state-of-practice and research opportunities. Systems Engineering, 2020. 23(5): p. 557-578.

40. Salado, A. and D. Selva, Asistentes cognitivos en ingeniería, in UEM STEAM Essentials. 2021.

41. Martin, A.V. and D. Selva, Explanation Approaches for the Daphne Virtual Assistant, in AIAA Scitech 2020 Forum. 2020.

42. Martin, A.V. and D. Selva, From Design Assistants to Design Peers: Turning Daphne into an AI Companion for Mission Designers, in AIAA Scitech 2019 Forum. 2020.

43. Martin, A.V.i. and D. Selva, Daphne: A Virtual Assistant for Designing Earth Observation Distributed Spacecraft Missions. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, 2020. 13: p. 30-48.

DR. DINESH VERMA

Dr. Dinesh Verma is the founding Executive Director of the Systems Engineering Research Center (SERC), and a Professor of Systems Engineering in the School of Systems and Enterprises (SSE) at Stevens Institute of Technology.

He was the founding Dean of SSE for 10 years from 2007 through 2016. During his fifteen years at Stevens he has successfully proposed research and academic programs exceeding $175m in value. Verma served as Scientific Advisor to the Director of the Embedded Systems Institute in Eindhoven, The Netherlands from 2003 through 2008. Prior to this role, he served as Technical Director at Lockheed Martin Undersea Systems, in Manassas, Virginia, in the area of adapted systems and supportability engineering processes, methods and tools for complex system development.

Before joining Lockheed Martin, Verma worked as a Research Scientist at Virginia Tech and managed the University's Systems Engineering Design Laboratory. While at Virginia Tech and afterwards, Verma continues to serve numerous companies in a consulting capacity. He served as an Invited Lecturer from 1995 through 2000 at the University of Exeter, United Kingdom.

His professional and research activities emphasize systems engineering and design with a focus on conceptual design evaluation, preliminary design and system architecture, design decision-making, life cycle costing, and supportability engineering. In addition to his publications, Verma has received three patents in the areas of life-cycle costing and fuzzy logic techniques for evaluating design concepts.

Dr. Verma has authored over 100 technical papers, book reviews, technical monographs, and co- authored three textbooks. He received the MS and the PhD in Industrial and Systems Engineering from Virginia Tech, and was honored with an Honorary Doctorate Degree (Honoris Causa) in Technology and Design from Linnaeus University (Sweden) in January 2007; and with an Honorary Master of Engineering Degree (Honoris Causa) from Stevens Institute of Technology in September 2008. He is a Fellow of the International Council on Systems Engineering (INCOSE).

DR. ALEJANDRO SALADO

Dr. Alejandro Salado is an associate professor of systems engineering with the Department of Systems and Industrial Engineering and the director of systems engineering programs at the University of Arizona. In addition, he provides part-time consulting in areas related to enterprise transformation, cultural change of technical teams, systems engineering, and engineering strategy.

Alejandro conducts research in problem formulation, design of verification and validation strategies, model-based systems engineering, and engineering education. Before joining academia, he held positions as systems engineer, chief architect, and chief systems engineer in manned and unmanned space systems of up to $1B in development cost.

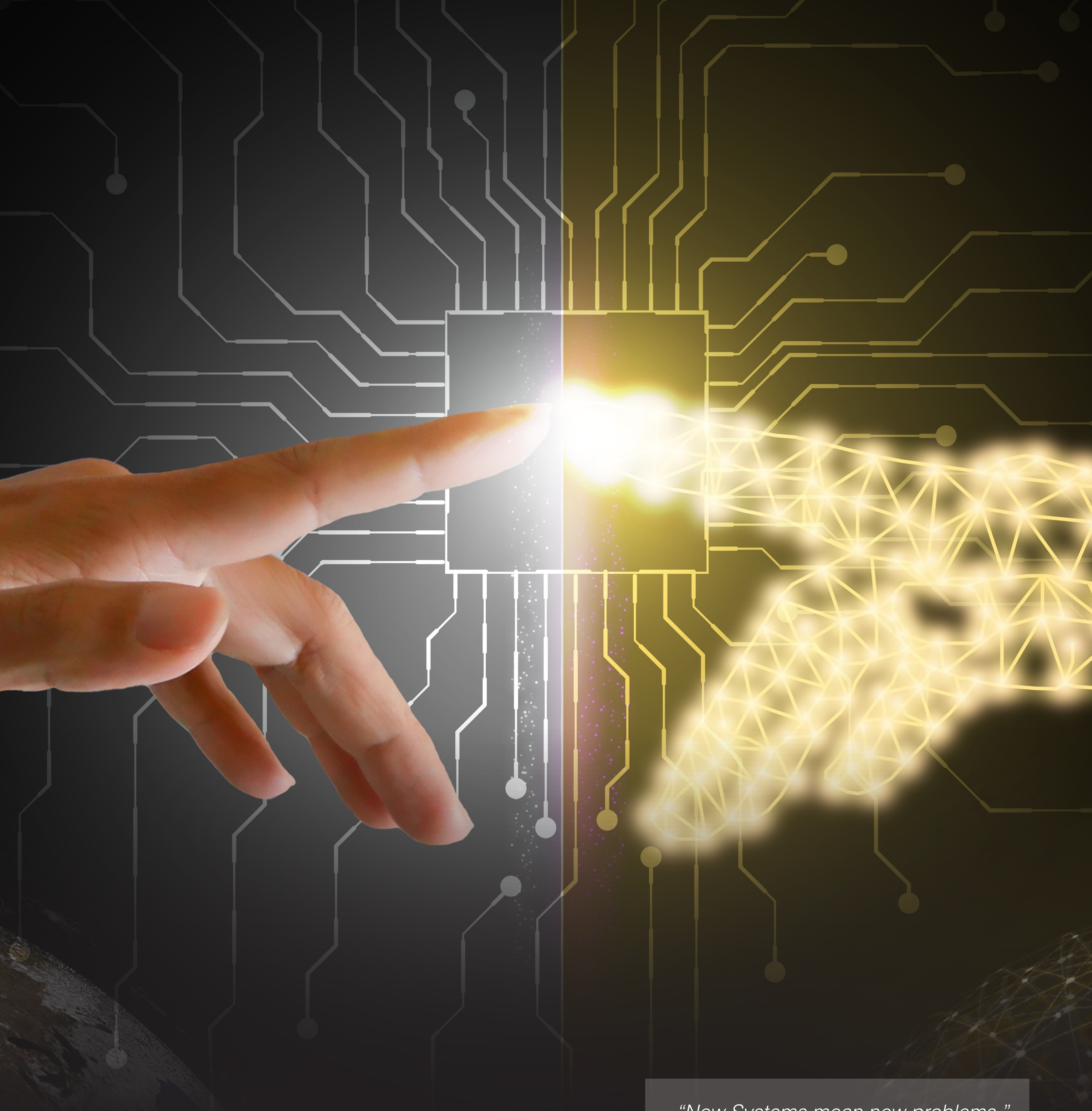
He has published over 150 technical papers, and his research has received federal funding from the National Science Foundation (NSF), the Naval Surface Warfare Command (NSWC), the Naval Air System Command (NAVAIR), and the Office of Naval Research (ONR), among others. He is a recipient of the NSF CAREER Award, the International Fulbright Science and Technology Award, and several best paper awards.

Dr. Salado holds a BS/MS in electrical and computer engineering from the Polytechnic University of Valencia, a MS in project management and a MS in electronics engineering from the Polytechnic University of Catalonia, the SpaceTech MEng in space systems engineering from the Technical University of Delft, and a PhD in systems engineering from the Stevens Institute of Technology.

ADOLFO SÁNCHEZ

Adolfo Sánchez holds a master’s degree in Computer Science Engineering from the University of Zaragoza and a master’s degree in History from the UNED.

He has participated in the definition and development of a modular avionics architectures evaluator, in the maintenance of the Eurofighter software, in quality assurance software developments made in strategic electronic warfare programs; in the acquisition, deployment, and implementation of the CIS Program of the Military Emergency Unit, and currently works supporting the Subdirectorate General of Digital Transformation of the Ministry of Defense and as an in-house trainer of various systems engineering courses in ISDEFE.



“New Systems mean new problems.”

J. Gall

CHAPTER 2

New Kinds of Systems

Tom McDermott, *Stevens Institute of Technology (tmcdermo@stevens.edu)*
Víctor Ramos, *Isdefe (vramos@isdefe.es)*

Abstract

This chapter presents four aspects of modern and future systems that are shifting traditional systems engineering practices: adaptability in systems, highly interconnected cyber-physical systems, learning-based systems with human-machine teaming, and distributed governance in systems of systems. Emphasis is given to current practices with doubtful effectiveness for such kinds of systems, then the chapter presents current trends on how to address these unique aspects of these new systems.

Keywords

Cyber-Physical Systems; Artificial Intelligence; Adaptive Systems; Agile Development; Learning Based Systems; Distributed Governance; Complexity.

1. INTRODUCTION

There are several trends that are changing the nature of the systems we develop and interact with today:

1. The pace of technological innovation continues to increase. With the availability of basic research on the internet, science and technology have become more of a commodity and the introduction of technical innovations into existing missions and platforms drives us to continuous adaptation and change. Systems are much less stable than they used to be, and competition favors those who can adapt the quickest to technology-driven change, not necessarily those who can invest the most to bring technologies to the mission.
2. The behavior of systems is and will continue to be defined more by software than hardware. All systems are or will be largely digital and will be supported by connected digital infrastructures. Hardware systems remain important but trend further towards more general purpose, more affordable and distributed even in critical applications like defense.
3. Data is increasingly collected and available for analysis. Computer storage and processing power continues to grow exponentially. Modern systems and future systems more so, adapt to changing internal and external conditions, as informed by changing data. The way we develop systems ought to change to better harvest and act on this data, and to take advantage of data and models to improve efficiency in development and management of programs.
4. Everything is becoming interconnected, and complex global systems of systems abound. Increasingly cyber-physical systems (CPS) are connected to other systems to share data and other resources as larger systems of systems. This opens both new opportunities and new vulnerabilities in system capabilities. Systems engineering (SE) rigor is necessary to balance the openness of these systems with other concerns such as safety, security, privacy, assurance, and alike.
5. Automation and user customization make systems more efficient, configurable, and adaptive. In particular, human tasks are much more dependent on teaming with automated or partially automated systems. This will continue to advance with artificial intelligence (AI) and machine learning (ML) to a new class of system known as a Learning Based Systems (LBS). An LBS is a new class of computing systems that achieves its function and performance through ML techniques. In the near future, human operators and LBS will both interact and adapt to jointly complete complex missions, a concept known as Human-Machine Teaming (HMT).

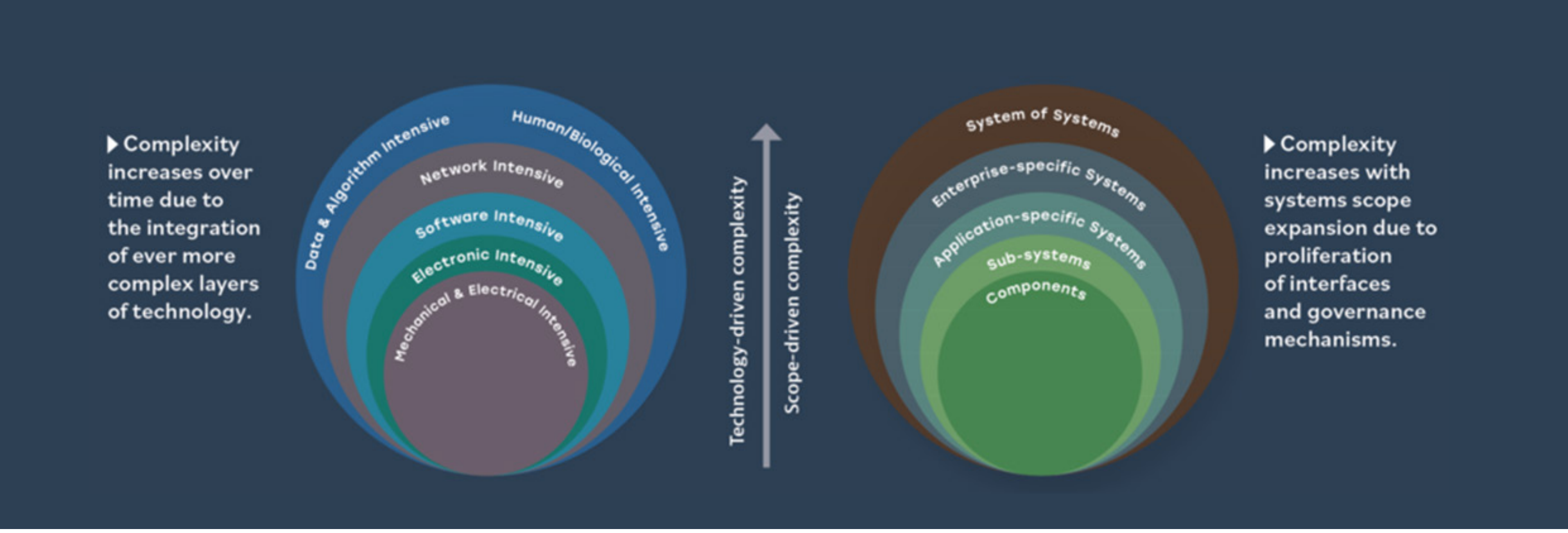
These trends are creating new kinds of systems, which we have categorized into four evolutionary and interrelated types:

- **Highly Adaptable Systems:** Modern systems make extensive use of software for their user-valued functionality as it is easier to change and minimizes production costs. Systems are becoming more and more tailorable in function to individual users. As a result, there is a trend toward systems that are easier to adapt to changing needs. In addition, highly adaptable development and support practices have permeated system business models to emphasize more frequent and consistent delivery of value in terms of capability introduction and flexibility. Learning and iteration have become more valued than up-front requirements understanding and stability, even in critical systems.

- **Highly interconnected Cyber-Physical Systems:** Modern systems are highly interconnected and fully dependent on software defined functionality. The Internet of Things (IoT) as an infrastructure is leading to large scale CPS systems such as smart cities, interconnected transportation, ubiquitous sensing, and manufacturing 4.0. Highly interconnected CPSs raise concerns over safety, security, and trust in domains where these have not previously been priorities. Large scale interconnected CPS increase the complexity and effective surface area of the systems we interact with. Digital twins (digital models of CPS connected to and operating with physical CPS) are now a rapidly growing domain of SE.

- **Learning-based systems and human-machine teaming:** Emerging systems will have increasing composition with LBS, which are systems whose behaviors are learned instead of programmed. These will be teamed with human users to augment human intelligence and agency (HMT). The impact of these systems on human tasking and situational awareness will also be an emergent property of future systems. LBS are expanding definitions of trustworthiness in systems, from primarily dependability concerns to human concerns such as ethics and fairness. Uncertainty management as well as test, verification, and validation of LBS are emerging challenges for SE.

- **Distributed governance in systems of systems:** The performance and capabilities of a system depend on information and shared functions with external systems that cannot be fully controlled. This leads to complex emergent behaviors and different methods of governance. Systems of systems (SoS) methodologies have been established to help manage governance, but as with other historical SE practices these are not keeping up with the rapid adaptation of SoS and shared governance.



▶ Complexity increases over time due to the integration of ever more complex layers of technology.
Human/Biological Intensive
Data & Algorithm Intensive
Network Intensive
Software Intensive
Electronic Intensive
Mechanical & Electrical Intensive
Technology-driven complexity
Scope-driven complexity
System of Systems
Enterprise-specific Systems
Application-specific Systems
Sub-systems
Components
▶ Complexity increases with systems scope expansion due to proliferation of interfaces and governance mechanisms.

Figure 1. Characteristics of increasing complexity in systems. [12]
Copyright © 2021 by INCOSE

The underlying theme across all four kinds of systems is the rapid growth in system complexity. Figure 1 highlights the related complexity concerns being driven by these new kinds of systems [12].

SE has always been viewed as a methodology to help manage system complexity, but the types of systems and their related system concerns are shifting, and the established methods are due to be updated. That is the subject of this chapter.

The next section evaluates the effectiveness of historically established SE methods in dealing with these new kinds of systems and SE concerns. Following that, we discuss some of the trends that are driving changes in the ways we practice SE as a result of these new kinds of systems. These discussions are not derived from exhaustive literature review, but from ongoing research and research roadmapping in the Systems Engineering Research Center (SERC) at the Stevens Institute of Technology in Hoboken, New Jersey, USA.

2. EVALUATION OF THE EFFECTIVENESS OF TRADITIONAL SE PRACTICES APPLIED TO THESE NEW KINDS OF SYSTEMS

2.1. Highly adaptable systems

Adaptability in systems is defined as the ability of a system to adapt itself efficiently and quickly to changed circumstances. An adaptive system is therefore a system that is able to fit its behavior according to changes in its environment or in parts of the system itself [28]. The Systems Engineering Body of Knowledge (SEBOK) further discusses system adaptability as the system's ability to satisfy externally driven mission and requirement changes with or without modification, as measured by some value indicator such as cost, time, or resources [29]. This definition implies that context-driven change is an intentional process. This does not effectively capture the nature of today's LBS, which are continuously updating themselves based on learned behaviors or outputs that are responsive to changing external context. For this kind of systems, it is no longer appropriate to segregate a system from its external context; both must be understood and modeled.

Highly adaptable systems are software-intensive, highly connected, and have extensive automation and user configuration capabilities. At their core are sets of data that drive the behavior of the system, which is defined by software logic, algorithms, and control and data management functions. The underlying mechanism is a connection between the sets of data in the system and the external environment that allows the behaviors of the system to rapidly change in response to external context changes. This can be an intentional data-driven learning and modification process responding to changing mission, or can be highly automated in response to changing external context as with emerging LBS.

However, SE, strongly influenced by aerospace and defense needs, has continued to be linked with the physical realization of large complex systems and other critical capabilities that are intended to persist for many years. The need for rigorous definition, analysis, and testing of these critical systems will always exist, but the lifecycle processes we choose to use should be tailored to the system's actual use and life. For example, SE for modern systems tends to be more model-based, agile, and responsive to user needs, which may be accomplished with more adaptable and efficient lifecycle processes by leveraging data and models [16]. Software systems engineering (SSE), information technology and enterprise architecture, distributed modeling and simulation, and automated manufacturing systems must all be leveraged in a convergent fashion to address lifecycle management of highly adaptable systems.

Modern SE technical and management processes transform data into views through models, which support analyses leading to decisions. This digital process flow supports "Data Transformed into Models then Analyzed through Views to make Decisions documented in Digital Artifacts." This process flow is not new, but is evolving from a largely manual, inefficient process flow to a highly automated process flow driven by rapid, consistent, and value-driven adaptation cycles. (Software development practices have evolved to manage this automation.) Figure 2 redraws the widely depicted "Define -> Realize -> Deploy&Use" stages of the SE Vee-model lifecycle process in a circular process to represent it as a:

1) set of data at the core, interpreted by model transformations, leading to design decisions,
2) layered across disciplines and engineering tasks to produce decision artifacts, and
3) in continuous iterative processes that could be entered from any point.

The challenge in highly adaptable systems is maintaining appropriate SE rigor and associated process definition to ensure these systems remain scalable, resilient, safe, secure, and usable by human operators. New SE lifecycle processes address shared and authoritatively managed sets of digital data and models associated with the system's entire lifecycle, not just a single engineering or program lifecycle [16].

2.2. Highly connected Cyber-Physical Systems (CPS)

Highly adaptable systems are evolving from software-only systems to fully connected software/hardware systems known as CPS. The U.S. National Science Foundation (NSF) defines CPS as "engineered systems that are built from, and depend upon, the seamless integration of computational algorithms and physical components" [19]. A CPS has computers and networks that control physical processes, often characterized by feedback loops that affect computations and the physical outcomes of those computations. Figure 3 provides a general layered depiction of a CPS framework [11]. As shown in the figure, the design of CPS must address these control activities in a device of interest but also the interconnected human and machine systems that interact with it, which may occur at long ranges over cyber networks [10].

Furthermore, CPS transform the way in which people interact with systems [19]. For example, humans can now interact with engineered systems over cyber networks instead of directly (such as controlling a thermostat from a mobile phone) or even across multiple interconnected CPS and large-scale software

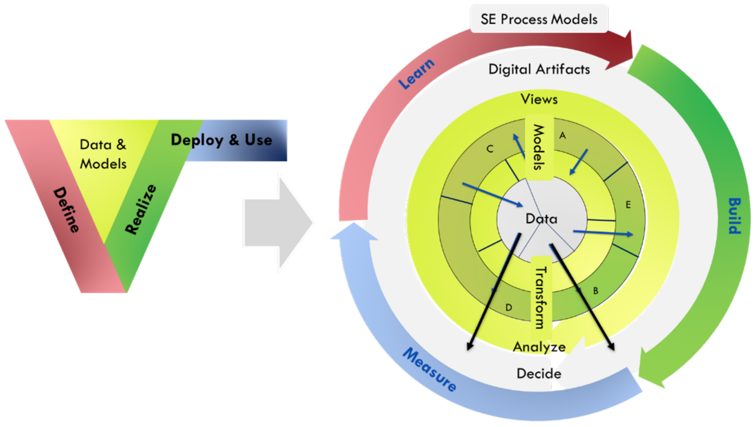


Figure 2. Circular Processes with Data at the Core [16]

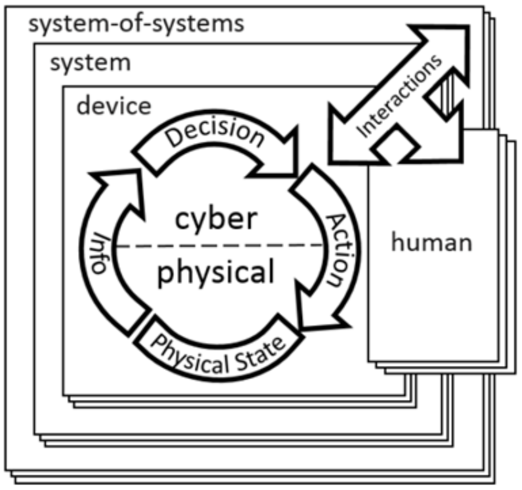


Figure 3. A general framework for CPS [11]. Republished courtesy of the National Institute of Standards and Technology

systems. These interactions are enabled by higher degrees of automation and autonomy [19], which increase however the complexity of the system control methods and the speed of evolution of the system in response to its external domain.

Traditionally, SE has distinguished how a system is constituted internally (i.e., its structure) from how the system manifests itself externally (i.e., its behavior) and deduced the function from the structure. This notion is the foundation underlying hierarchical construction of systems with defined input and output interfaces across multiple modular components. It has resulted in a focus on structural system representations in SE, supporting physical trades in primarily physical systems. Highly connected CPS, however, are heterarchical in nature. They are comprised of numerous, heterogeneous elements acting both independently and interdependently. Because of this complexity, traditional structural decomposition and physics-based models are insufficient. In complex systems, form (structure) and function (behavior) are intrinsically linked and not separable. The complexity of the system control methods (behavioral) and the evolution of the external domain that interacts with the system (adaptive) cannot be ignored. These changes are a product of the computer/network interactions and increased use of digital data in the control functions, along with the connected nature of the external environment and users.

As an example, Figure 4 shows a functional and structural view of the highly cyber-physical systems emerging in transportation systems today. In practice these systems are more often built from general purpose programmable hardware while behaviors are programmed by software. Their function in the full system may evolve incrementally over time, as we are seeing in vehicle automation today. Mechanisms that determine the qualities of these system, such as safety, adaptability, or resilience, are intentionally supported in the hardware and software designs but also realized by investment in both system structure/function today and architecture evolution over time. This is a concept known as technical debt, where decisions made in the current design gradually limit the system's ability to support new capabilities in the future. Informing decisions that drive both the immediate and long-term qualities of the systems they support are thus critical.

The complexity of these networks of CPS results in emergent behaviors that cannot be fully modeled and predicted, or even structurally decomposed in traditional ways. The technology traditionally employed by the SE is also going through an evolution towards digital engineering (DE) and simulation-based design practices. Time invested in up-front digital simulation of these emergent behaviors is essential to both short-term and long-term design decisions. These simulations can also be deployed to monitor the CPS behaviors after installation to more rapidly and accurately detect undesired behaviors in use. This is the driver for and concept behind digital twins, which will be expanded in Chapter 6. In this paradigm, both the realized system and the virtual simulated system and interconnected products are holistically planned and managed with full SE life cycles. As

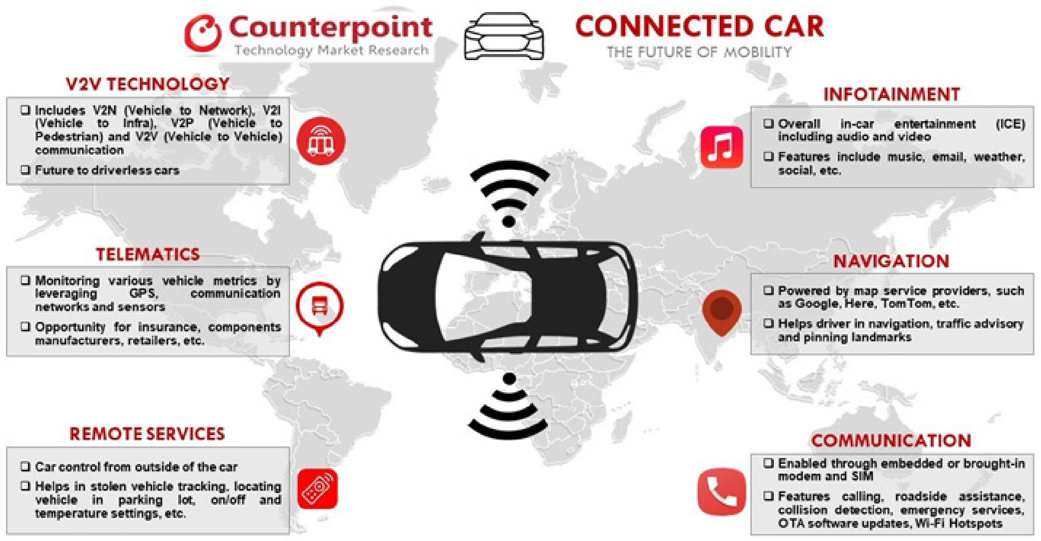

Counterpoint
Technology Market Research
CONNECTED CAR
THE FUTURE OF MOBILITY
V2V TECHNOLOGY
❑ Includes V2N (Vehicle to Network), V2I (Vehicle to Infra), V2P (Vehicle to Pedestrian) and V2V (Vehicle to Vehicle) communication
❑ Future to driverless cars
TELEMATICS
❑ Monitoring various vehicle metrics by leveraging GPS, communication networks and sensors
❑ Opportunity for insurance, components manufacturers, retailers, etc.
REMOTE SERVICES
❑ Car control from outside of the car
❑ Helps in stolen vehicle tracking, locating vehicle in parking lot, on/off and temperature settings, etc.
INFOTAINMENT
❑ Overall in-car entertainment (ICE) including audio and video
❑ Features include music, email, weather, social, etc.
NAVIGATION
❑ Powered by map service providers, such as Google, Here, TomTom, etc.
❑ Helps driver in navigation, traffic advisory and pinning landmarks
COMMUNICATION
❑ Enabled through embedded or brought-in modem and SIM
❑ Features calling, roadside assistance, collision detection, emergency services, OTA software updates, Wi-Fi Hotspots

Figure 4. The automobile as a highly connected CPS [2].
Image from https://telecom.economictimes.indiatimes.com/tele-talk/connected-car-opportunity-propels-multi-billion-dollar-turf-war/2971

can be seen, the concerns of a modern systems engineer expand in this context from just the structure/behavior and domain-driven non-functional qualities of the system itself, to also include its interaction in the larger connected context and environment, and finally to how both the realized system and its companion virtual twins are managed and evolved over the full lifecycle. This has become a multi-disciplinary challenge requiring a shift in SE skillsets to integrate across the principles, foundations, and characteristics listed in Table 1 in a holistic manner.

The SE concerns of CPS can also be stated in terms of trustworthiness, related to the ability of the CPS to withstand instability, unexpected conditions, and gracefully return to predictable but possibly degraded performance [11]. This is truly a system concern in highly connected CPS. These characteristics of trustworthiness, which include dependability, safety, reliability, privacy, security, and resilience, have for the most part evolved within distinct disciplinary and education silos. Historically, SE practice has treated these as disparate sub-disciplines. Large SE and integration projects often have property-specific leads, who represent discrete viewpoints within the trade-off process overseen by the chief systems engineer/integrator. Functional requirements often have caused engineers and designers to prioritize each property differently, based on domain-specific requirements and perspectives (e.g., energy, manufacturing, transportation, etc.). Achieving a certain level of success in each property is typically vital to the overall success of the system. Trends in the engineering of highly connected CPS suggest that SE disciplines are converging toward increased interdependency.

This is particularly important for highly connected CPS, in which systems-based holistic thinking is critical to supporting trustworthiness objectives and avoid problems arising with respect to one property, or protections inserted to address one dimension of concern, do not compromise other primary system objectives or cause deleterious unintended effects [17].

Awareness	Supervised practitioner	Practitioner
Communication and Networking	Basic computing concepts, including software engineering	Security and privacy
Embedded systems, both hardware and software	Discrete and continuous mathematics	Discrete and continuous mathematics
Real time systems	Physical world computing, including sensors, actuators, and real-time control	Interoperability
Physical world computing, safety, reliability, security, performance, and risk management	Cross-cutting application of sensing, actuation, control, communication, and computing	Reliability and dependability, Safety, Stability and performance
Human interaction with CPS, including ease of use	Modeling of heterogeneous and dynamic systems integrating control, computing, and communication	Human factors and usability
	CPS system development (emphasizing concepts of resilience and safety, test and verification)	Power and energy management

Table 1. NAS CPS principles, Characteristics, and Foundations [18]

Thus, we have today a multi-disciplinary challenge associating together foundational disciplines of various schools, such as engineering, computing, and human interaction. We also have a multi-disciplinary challenge across traditional SE sub-disciplines related to (1) dependability in computing systems (availability, reliability, safety, integrity, and maintainability), (2) system security engineering (confidentiality, integrity, and availability), (3) information systems (management, communications, and privacy), and (4) cybersecurity (threats and protection). This is also a system governance challenge, both in the enterprise systems that develop highly connected CPS, and in the emergent behaviors of the CPS themselves [17]. There is a need for a more interdisciplinary approach to system design, founded on rigorous system functional modeling and simulation, evolutionary design, and rigorous evaluation. As digital engineering and Model-Based Systems Engineering (MBSE) become more prevalent, there is potential to transform traditional system design and evaluation processes to more holistic and more evidence-based forms using models (to be further elaborated in Chapters 4 through 6). Verification and validation of highly connected CPS through test and evaluation have historically been the gold standard but is significantly expensive and fraught with difficulty as systems become more complex, more expansive, and more inter-dependent on other systems to realize their intended capabilities. Again, bringing data and models into this process aims to relieve some of the expense and make the entire process more flexible and amenable to changes that can occur across a system's lifecycle.

2.3. Learning-based systems and human-machine teaming

Concepts of highly adapted systems, highly interconnected systems, and resulting distributed governance are placing SE in the midst of a digital transformation driven by advanced modeling tools, data integration, and the resulting digital twins. Data, models, and computing systems are all converging with machine learning (ML) techniques to enable a new class of computing system that achieves its function and performance through the use of ML – known as a Learning Based System. Applications of LBS are evolving exponentially into many domains.

LBS are types of AI models that use ML algorithms to make decisions and solve problems without being explicitly programmed. ML algorithms independently detect, analyze, and learn patterns directly from input data and modify their behavior accordingly to predict new output. LBSs differ from a longer legacy of rule-based AI systems in their ability to achieve scale, provide adaptability, and handle complex tasks. Rule-based systems are a type of AI model that uses a set of prewritten rules to make decisions and solve problems. Developers create rules based on human expert knowledge that enable the system to process input data and produce a result. Almost all systems of the future, and the tools we use to design them with, are evolving to include compositions of rule-based and learning-based components.

The number of systems with some level of learning capacity is increasing exponentially today. In some applications, these systems work closely with humans; in others, operations are largely autonomous. In these systems, distributed teams of humans work with distributed teams of autonomous systems, with relationships that can change dynamically. Likewise, SE is evolving to more of a technological discipline that uses LBS to define and manage digital data and descriptive models that link different disciplines together. The SE digital engineering transformation is being followed by transformational advances in the discipline of SE using AI and ML technology for automation of many engineering tasks, designed to augment human intelligence. At the same time, the application of AI, ML, and autonomy to many of today's complex and critical systems drives the need for new SE methods, processes, and tools.

A primary goal of SE is to ensure that the behavior and performance of complex engineered systems meet the expected outcomes driven by user needs, and that the configuration of the system is managed across its lifetime. Advances in AI and ML application means that future system components may learn and adapt more rapidly, and that behavior and performance may be non-deterministic with less predictable but manageable outcomes. This is the LBS challenge. The inability to explicitly validate system behaviors or the time it takes to do that will impact trust in these systems and is driving change in the way the SE community traditionally addresses system validation [26]. The uncertainty present in multiple AI/ML components that interact defies traditional decomposition methods used by the SE community, requiring new synthesis methods. Finally, as systems develop means for co-learning between human users and machines, traditional models that separate human behaviors from the machine will need to be revisited [14].

Thus, the emerging SE challenge is to produce systems that gain value from their ability to learn, which may be inherently non-deterministic, but that are also appropriately robust, predictable, and trustworthy in the type of critical and complex uses common to the application of SE practices today. This includes both human and machine behaviors in joint decision environments, highly reliant on good human-systems design and presentation of decision information. It also includes the adaptation of test and evaluation processes to co-learning environments [14].

The future SE challenge involves LBS that actually adapt and learn dynamically from their environments. These environments could be real, simulated, or a mix of both thanks to the generation of synthetic data, which offers new approaches for system training and development but, on the other hand, generates the need for synthetic data validation. In this rapidly emerging future, machine-to-machine and human-to-machine (and maybe machine-to-human) trust will be critical. In this future, systems will be expected to learn to modify or create new behaviors as the context changes and this may happen fairly rapidly. Methods that revalidate system performance extremely rapidly or "on the fly" are not part of the current SE practice set and must be developed along with these types of learning systems [14].

The future of LBS and SE is difficult to predict at this point. The SE discipline is entering a period of rapid change where it will be trying to catch up to the evolving use of ML algorithms and resulting LBS. The complexity of engineered systems is rapidly increasing, but the automated tools that systems engineer's use are also evolving to manage that complexity – albeit at a slower pace. The transformation of SE away from manual paper-based workflows to a fully digital and model-based set of practices is actually quite urgent.

2.4. Distributed governance in systems of systems

The distinguishing feature of a system of systems (SoS) is the behaviors of the "whole" come from individual constituent systems that act independently and autonomously [3]. Furthermore, SoS are generally sociotechnical systems: technology-driven systems that involve significant human and social participation, where that participation in turn influences the architecture and design of the technical system [13]. In other words, the human is a part of the system, not an external agent.

In order to determine appropriate architecting principles, SoS engineering (SoSE) literature defines a classification system for SoS linked to the degree of managerial control in the SoS. The four SoS classes are Directed, Collaborative, Virtual, and Acknowledged SoS. The degree of central control or governance over SoS changes is the primary distinction of each class [4]. In a directed SoS, the integrated SoS is created to serve a specific purpose and is governed centrally. The constituent systems remain independent, but their normal operations are directed through a central authority. In a collaborative SoS, the constituent systems volunteer to collaborate and there is not a central control agent with authority to direct their operations. However, there is still a governance function determining an agreed upon purpose for the SoS. In collaborative SoS standards, regulations, norms, and circumstances drive the operation of the constituent systems.

SoS literature additionally distinguishes the collaborative SoS classifications as virtual and acknowledged. Virtual SoS are collaborative but develop without a centrally agreed upon purpose or set of goals. Governance is fully distributed. Acknowledged SoS have a recognized objective set and a designated managerial component, have resources allocated for development, but changes are based on collaboration objectives. Governance remains distributed but conforms to centrally determined policies and outcomes. A good example of a collaborative SoS is automotive traffic on the interstate highways. Whether or not this highway traffic SoS is viewed as virtual or acknowledged is linked to stakeholder perspectives. Drivers would view it as virtual (its goals are determined by my individual need to get from point a to point b), while highway transportation officials would view it as acknowledged (its goals are to safely and efficiently manage interstate traffic). Today's and future adaptable and highly interconnected systems are more collaborative and struggle with managerial control, and individual perspectives and behaviors will affect system behaviors in ways that cannot be easily predicted by traditional SE decomposition.

The Wave model is an established framework for evaluating and planning evolution in systems of systems (ref. Figure 5). The model recognizes that evolution is continuously driven

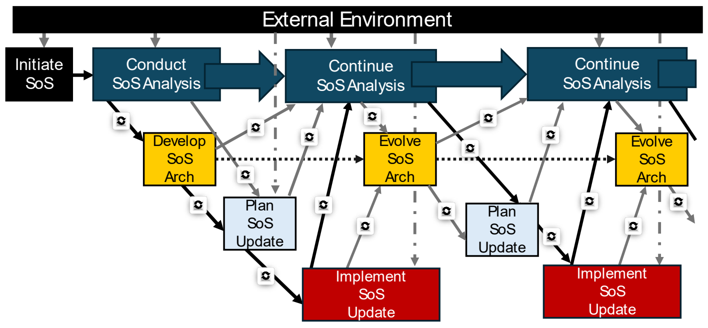


Figure 5. The wave model [7]

by input from the external context, and unlike traditional SE it views the analysis of system change as an ongoing process with multiple overlapping increments. In the Wave model, system evolution is a forward-looking process with feedback at each iteration, and managerial control strategies attempt to group multiple constituent changes into SoS level architectural changes to create efficiency in the test and validation process [7].

A different perspective is given in the innovation literature, called "transition management" [21]. Innovation literature counters the Wave model with a more bottom-up view of system evolution. Innovation system models recognize innovation as a complex adaptive process where lower-level innovations in constituent systems form niches of adoption, which over time produce broader changes in established SoS regimes, eventually resulting in transformation of the existing landscape (or context). Today one can view the public development of driverless automotive technologies as such an evolution "in-process." The primary aspect of this model is that innovation progresses through social layers and can be modeled as a multi-scale or multi-layer social phenomenon, as opposed to the more mechanistic view of the Wave model. Figure 6 shows this process as reflected in transition management literature.

The competing phenomena in these two views of SoS evolution are differing models of distributed governance. The SE perspective in the wave model assumes that SoS change can be planned and managed over time by some governance mechanism. The innovation perspective in the technology transition model notes that change will evolve from bottom-up technology evolution that cannot be governed but can be encouraged or guided. In defense systems, distributed governance of the SoS and constituent systems is a set of agreement processes between military organizations. In commercial innovation, distributed governance is determined by market factors. A SE challenge today in this regard derives from the commercial marketplace driving technology innovation and the defense and aerospace industries primarily driving SE methods and tools in a somewhat insulated form from that larger marketplace.

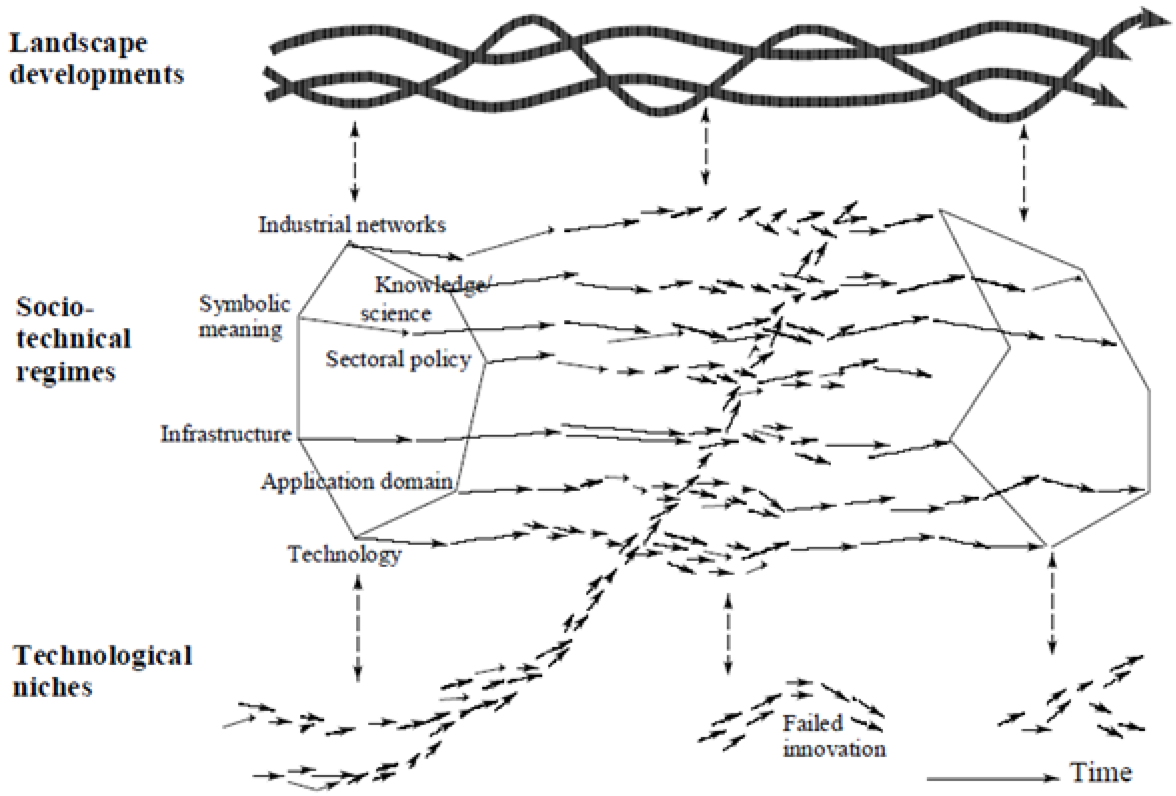


Figure 6. A dynamic multilevel SoS perspective on technology transitions [8]

This brings the concern of current SE methods being slow to change and not adapting well to the current dynamics of distributed governance in more adaptable, highly interconnected systems. Trying to fully predict emerging SoS requirements and rigidly plan SoS updates up-front runs counter to current commercial business markets. Commercial businesses do aim to centralize governance of SoS value but recognize actual SoS emergent capabilities require openness and experimentation ahead of rigid planning.

Technology is also driving the execution of distributed governance models. Recent technologies such as software orchestration and blockchain/distributed ledger technology (DLT) have led to the emergence of "leaderless organizations" where rules and system goals are distributed via software automation and people are allowed to self-organize their tasking. These same technologies will likely help to manage "authoritative data and models" in future SE, but the governance of these is an open question at this point. This is not just a commercial trend; emerging military command and control concepts envision future military SoS as an internet of things, ideally moving from rigidly connected platforms to more flexible distributions of capabilities managed by distributed control centers in line with the concept of collaborative combat. The same concepts of highly adaptive systems, highly interconnected CPS, and LBS/HMT emerge together to transform military operations. All these trends point to the need for SE methods and processes that center on agility and flexibility, along with improved integration of the business reasons for these traits.

3. TRENDS TO EVOLVE SYSTEMS ENGINEERING TO EFFECTIVELY ENGINEER THOSE TYPES OF SYSTEMS

3.1. Digitalization

The core change driving the discipline of SE today is the transition towards digital engineering, a paradigm where "shared and authoritatively managed data" can be transformed through "shared and authoritatively managed models" and related tools to create Digital Artifacts that can be used by various decision-makers and others needing digital access to the design and descriptions of the system across its lifetime [20]. These artifacts were almost always paper documents or drawings in the early years. Now they are generally based on digital technologies, but workflows still tend to be document driven instead of data driven.

The workflow view in Figure 7 shows conceptually how shared and authoritatively managed data should be transformed into digital artifacts in different life cycle stages in any SE process. Data, federated data models, and distributed data storage are the foundational infrastructure of modern SE. This is a distributed federation of data and models, governed across organizations, with authoritative processes to manage data and model provenance and configuration.

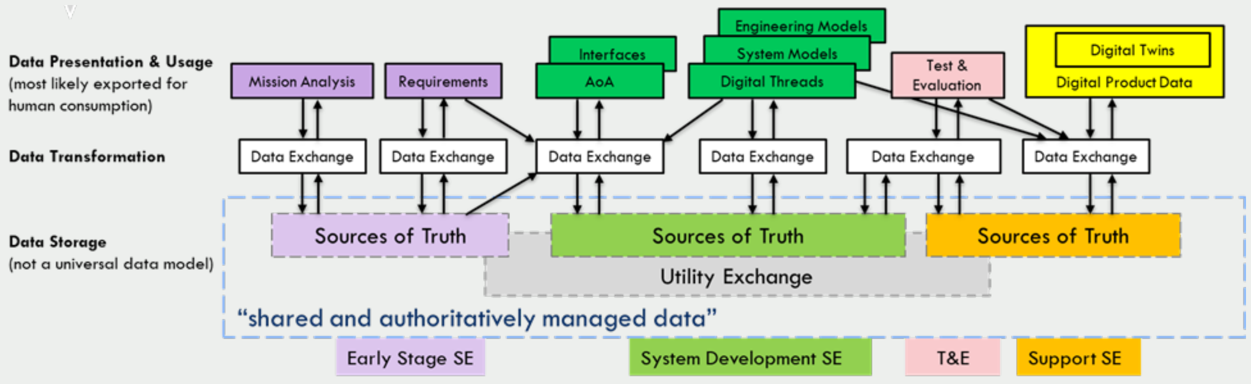


Figure 7. Data Transformation into the Life Cycle

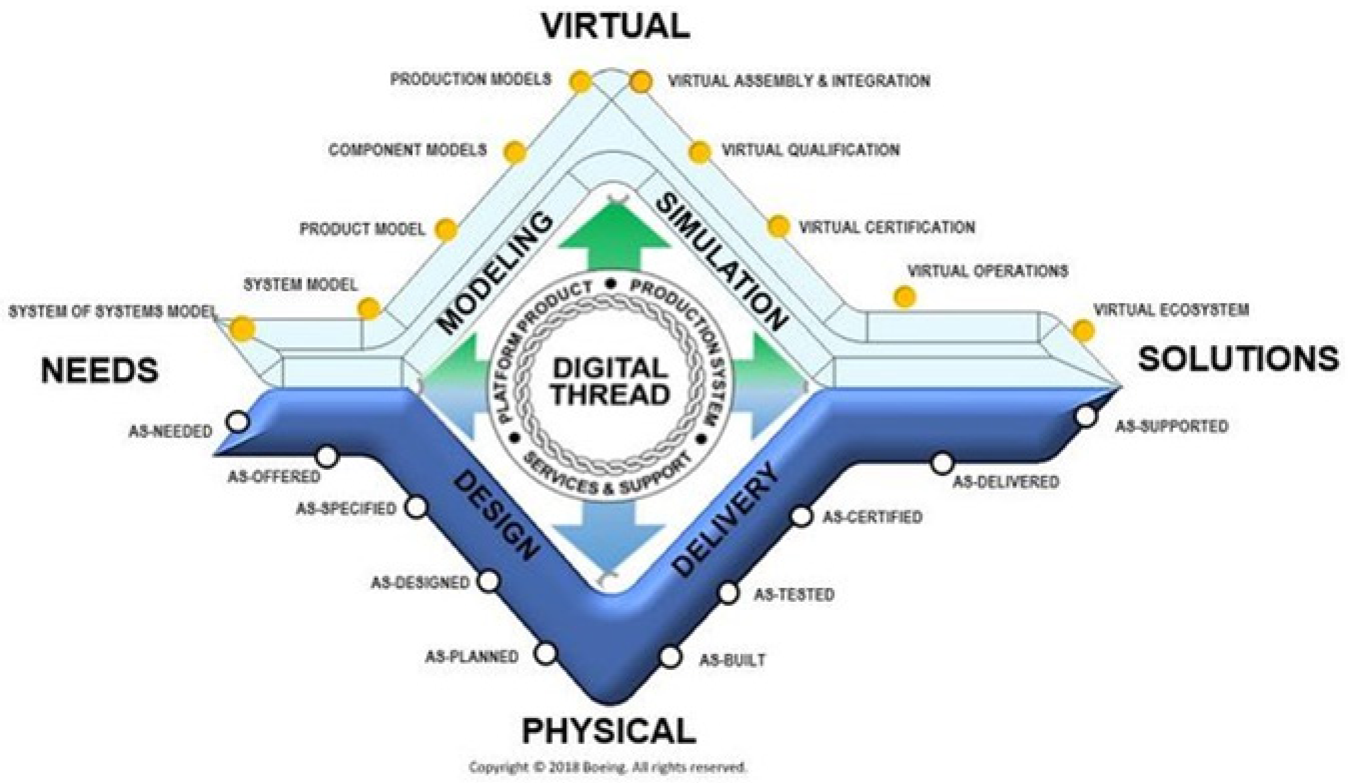


Figure 8. The Boeing "Diamond" showing the full lifecycle existence of virtual and physical (adapted from [5])

Figure 7 is particularly relevant to SE modernization, as "Data Management" is not currently defined as a disciplinary process in SE standards. Data, models, and data storage systems can each be conceived and treated as a separate system that must also be developed and deployed in support of the fielded system. These must be defined and built along with other system development aspects. These also have their own lifecycles. New SE lifecycle processes are evolving to address shared and authoritatively managed sets of digital data and models associated with the system's entire lifecycle, not just individual engineering or program lifecycles. This mental model has been depicted as a combination of a conventional Vee model with a mirrored Vee model to create a design diamond that incorporates digital counterparts of a product at all stages of its lifecycle (ref. Figure 8) [5]. The virtual system and the physical system are anticipated to evolve together in the long-term across the full system lifecycle.

Systems engineers have long used digital data and various modeling and analysis tools to produce digital artifacts for decision-making (such as Microsoft PowerPoint slides). However, the underlying data models have not been "seamlessly shared" in digital workflows and tools, or likely not shared at all. Furthermore, authority for that data has often been held by independent activities generally organized by discipline. Today, much of the transformation from data to models to decisions is still a manual interpretation of disparate data and analyses. This manual interpretation limits our ability to be iterative and responsive across disciplines and disciplinary tools. It is inefficient and non-holistic. The evolution of SE is a fully integrated, iterative workflow where the system is the focus, not the owner of the data or the particular element of a design. Today's primary challenge in digital engineering is not so much being "model-based," it is understanding, creating, and validating the underlying data model that integrates across requirements, design, test, disciplines, and disciplinary processes, with it being shareable and shared. The value of SE in the future can be realized through a more seamless and efficient transfer of data and models, starting from underlying performance drivers through models to decisions and ease of drilling back down from decisions to data. This is currently not the state of SE practice, but a target to define capability roadmaps.

3.2. The digital ecosystem

Collaboration between disciplines and organizations requires establishing high assurance interfaces between multiple engineering, development, and management applications in a digital engineering ecosystem. The future digital ecosystem for SE is envisioned as a service-oriented architecture to provide flexibility and adaptability between tool suites across and between organizations, as shown in Figure 8. Data exchange between applications may be technically envisioned as a set of collaborative APIs for data and information sharing, as well as query and response. Maturing standards, moving towards open systems, and building up experience implementing digital SE are necessary to facilitate the adoption of digital engineering in organizations [16].

3.3. The digital system model

One of the most impactful changes in SE in recent years has been the maturation and adoption of MBSE, which is presented with certain level of detail in Chapter 4. MBSE is the formalized application of modeling to support SE activities, including system requirements, design, analysis, verification, and validation activities beginning in the conceptual design phase and continuing throughout development, production, and later life cycle phases. MBSE has a particular value in the digital ecosystem as an approach to express and capture the relationships, interdependencies, and associated processes connecting systems level models and other disciplinary models as well as the life cycle process flow. For example, system models are useful for showing relationships among requirements, system functions, physical components, suppliers, acquirers, and users. Being machine-readable, formal models allow relationships between data elements to be established and to be processed by software thus enabling efficiency-improving capabilities.

In MBSE, the system model is used as the central repository for design decisions that span multiple engineering and business concerns; design decisions are captured as model elements in that digital system model [6]. This is the emerging digital product of the SE function.

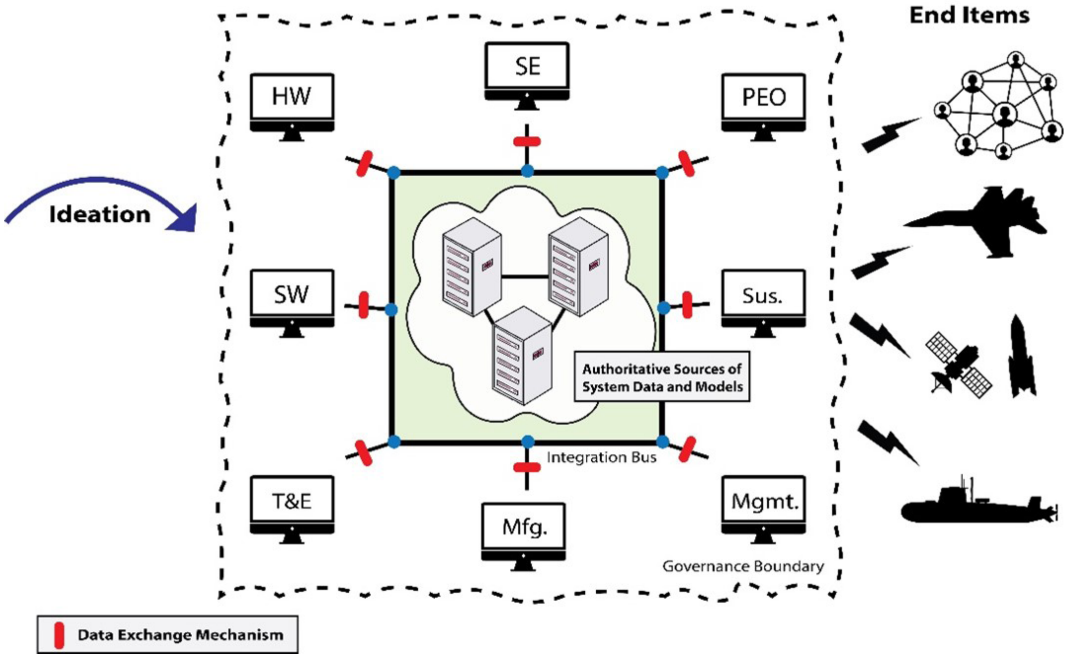


Figure 9. An Operational View of a Digital Ecosystem.
(SE: systems engineering, HW: hardware, SW: software, T&E: test & evaluation, Mfg: manufacturing, Mgmt: management, Sus: sustainment, PEO: program executives)

The practice of SE is still learning the most productive uses of MBSE. Initially the focus has been on a digital representation of the system specification, to include concept of operations, requirements, work breakdown structure, use cases and functions, and performance analyses. The concept of the digital system model as the central repository for design decisions (or at least an index to this) informed by executable simulations and/or quantitative analyses is the core use of MBSE in DE, and is still evolving, together with systems modeling languages and tools.

The use of a digital system model should eventually apply to all systems in all domains, but the concept is still in its infancy. Most systems are still developed and deployed without this formal but holistic set of models and views, leading to continued late lifecycle failures and rework. As new kinds of systems become more adaptable, interconnected, distributed, and learning-based, the importance of the MBSE activity should gain value. While there is still a way to go in terms of maturity of capabilities enabled by MBSE, initiating the transition is already feasible and valuable.

3.4. Data models, ontologies, and semantic web technology

As the digital basis for SE grows, it is relying more on defined and structured data (data modeling). SE practices now include machine usable languages and development of taxonomies, lexicon, and data ontologies to enable interoperability for computationally based data synthesis, analysis, and exploration. Connectivity across the digital system model starts at the data layer. Development and maintenance of an enterprise data model and environment has become part of SE work. The data model defines the specific structure and relationships of the data elements that inform higher level system models and analyses. A digital ontology is a reusable framework that models generalized data – general objects that have common properties and not specified individual entities. A data model that rests on top of a digital ontology supports greater interoperability and reusability of the data.

A recent SERC workshop report on digital ontologies summarized the need for and value of effective digital ontology development as one of the means to digital interoperability [25]. Traditional SE applications mainly rely on documents to capture and exchange information. Documents do not promote interoperability, efficiency, and automation, given their lack of formalism in both syntax and semantics. Ontology-enabled methods allow us to make inferences on data and information to determine new facts and discover previously unseen gaps and relationships. They also make it possible to uniquely identify data elements so that different data systems can refer to the same concepts without having to pass around or duplicate unwieldy data structures. These ontologies are a pragmatic means to formally describe the relevant entities and relations of a system, implement those descriptions and relationships, and be useful for our specific purposes and objectives. For example, the Common Core Military Ontologies represent data from different military branches in a common structure that allows data sharing and data aggregation in each domain using generic reusable data representations. The common ontology enables data interoperability in joint operations. In SE, syntactic interoperability across formal requirements, design, and test language is vital, but insufficient. Data interoperability is necessary to enable the digital infrastructure to exchange and aggregate requirements, design, and analysis information. The value of ontologies lies in what they allow us to communicate, and multi-disciplinary communication is at the core of SE and DE.

The SE community needs but does not yet have a broad understanding of the different ontological and related data modeling methods necessary to ensure data sharing and interoperability. The SE community is set to expand education and training to include those foundations of data architecting, data modeling, and data science specifically relevant for tomorrow's systems engineers.

3.5. Agility

Data transformations into and out of the shared and authoritatively managed federations of data and models described in the previous section are expected to happen iteratively and continuously across the entire life of a system. These data and models might originate in any phase of a system's lifecycle and any function associated with engineering and management. To respond to this new context, SE practices are becoming more agile and responsive. Increasing responsiveness does not mean eliminating critical SE processes, just increasing the number of iterations and shortening the cycle time between them. A recent workshop on agile practice in hardware-intensive systems captured a number of themes that must be better adopted into SE practice [1].

Commercial industry has adopted agile practices in software, hardware/software systems, and services to address rapidly changing threats to and opportunities in their business. A primary goal of agile practices in SE is to shift learning to as early as possible in the development process, that is, being intentional in the early design stages of systems to accommodate innovation in later stages of development. Agile practices augmented by digital models, prototypes, and test infrastructures help bring learning forward, reduce integration risks, and create more flexibility in long-term design decision points as a result. Choosing the appropriate systems, subsystems, or elements of the system to emphasize in this strategy helps anticipate evolution of parts of the system that have the most potential for change or those for which innovative change will have the most pronounced performance gain. Newer areas critical to SE, like software SE, information technology, enterprise architecture, distributed modeling & simulation, and automated manufacturing systems enable this transition.

The underlying premises of agile practices include direct collaboration between users and developers, encouragement of simplicity, and creation of continuous flow of value. In large scale defense systems, for example, the flow from warfighter need to capability passes through many organizations and processes before it becomes a program (of any type). This changes interpretation of needs and requirements, isolates the real customer from the capability development, and interrupts the flow of work from need to capability. On the contrary, commercial industry has pioneered many different ways to determine customer needs and responses using analytics embedded in the products themselves, accelerating customer understanding.

Similarly, agile practices challenge the single batch mindset and the belief that everything must be understood before implementation. This implies an enterprise-level shift to allow more frequent delivery of working systems (or system elements) through reconciliation of development and delivery cycles for best effect. Rather than compounding the effect of slower cycles that drive the pace of system-level delivery, a refactoring of the contributing streams of work can assure flow enabled by smaller batches of work. Milestone completion remains important but must be translated into buying down risk, not just criteria completion. Integrating both a consistent work cadence and milestone-driven goals are critical to agile in hardware-intensive systems. Meaningful movement of

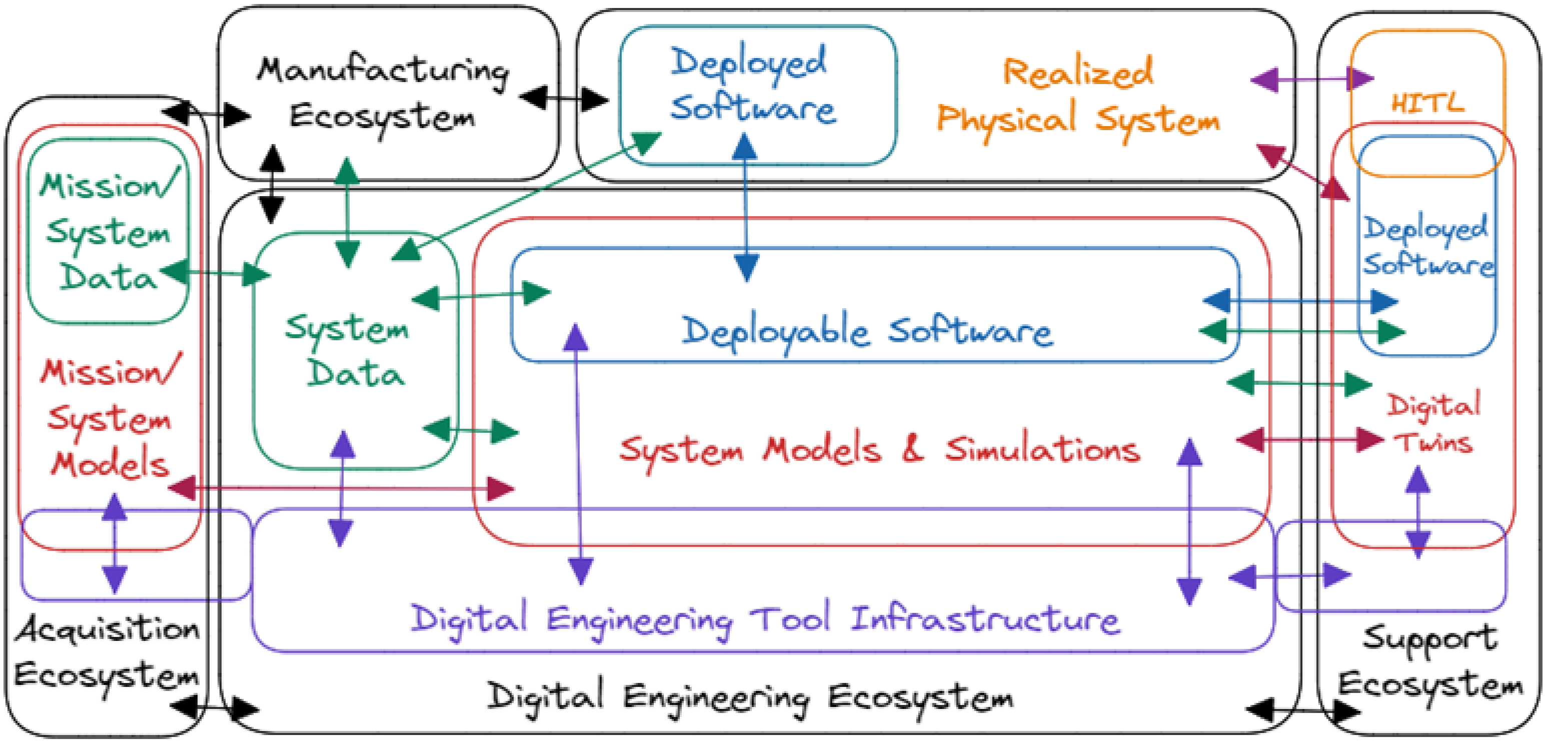


Figure 10. Today's systems should plan for deploying working software frequently into all aspects of program planning and development

prototypes from virtual environments to physical realizations to operational use has tangible benefits when the software is reused from one product to another. Programs should embed deployable software into simulation and training systems, allowing all developers and users to experience the operational use of the product and enabling Live Virtual Constructive (LVC) environments. Figure 10 depicts this approach.

What remains consistent with current SE practice is thoughtful decomposition and partitioning. Breaking down complex systems into smaller, manageable components allows for faster learning and better understanding of individual elements. Agile practices take advantage of modularity to architect systems that can be evolved over time. Control of interfaces and application program interfaces (APIs) is fundamental to both defining the work in the system and the team skills needed to do the work. Modular Open Systems Approaches (MOSA) precede agile development in both software and hardware systems. As a note, this partitioning must consider existing structural system decompositions as an area for innovation as blindly following traditional subsystem decompositions can limit agility.

Finally, agile practice in hardware-intensive systems requires front-end investment in test activities and infrastructure to buy down end-item risk. For example, SpaceX™ considers launch failures a data collection investment. Their mindset of corporate learning from multiple launch failures is a strong example of the culture and mindset required for innovation and continuous improvement in agile practice [9]. Investment in model-based engineering tools, multiple systems-level prototypes, and hardware-in-the-loop environments is critical for successful agile implementation in hardware-intensive systems. This is more difficult in hardware-intensive programs than software-only programs because the tools are more diverse and less well-integrated than in today's software development environments.

3.6. Modular and Open Systems Approaches (MOSA)

In highly adaptive and interconnected systems, as mentioned earlier, one needs intentionality in the early design stages to accommodate adaptation in later stages of development. Particularly in large scale critical systems like defense and aerospace systems, platforms may have lifecycles spanning decades. Choosing the appropriate systems, subsystems, or elements of the system to emphasize in this strategy helps anticipate evolution of parts of the system that have the most potential for change or those for which innovative change will have the most pronounced performance gain. Choosing the related business strategy is equally important to the success of the system over its life cycle. This is the driver behind a MOSA – an intentional design approach to maintain a technical and business basis for future flexibility and innovation in the system. Decomposition into smaller and smaller functional capabilities and hardware components reduces complexity at the component level, as well as the size of the associated work teams. However, modularity must be carefully designed and planned as an architectural strategy in a lifecycle planning activity. The skills to do this successfully are specialized and in high demand.

Decomposing to smaller modules increases agility but also increases integration risk. Standardization of interfaces, and particularly the use of open interfaces, can help reduce such risk. Open standards were almost always defined by documents in the past, making interpretation, compliance, and actual interoperability a challenge. With the introduction of the Internet Protocol (IP), "plug and play" capabilities into Microsoft WindowsTM products, and the AndroidTM operating system, open standards have been more often implemented as open-source software. The adoption of digital engineering and a digital system model facilitates formalizing the interfaces between modules in a digital baseline to eliminate interpretation and miscommunication [22].

3.7. AI4SE and SE4AI

It is difficult to predict exactly how AI and ML will impact SE today, but in the near future SE will undergo significant change in methods, processes, tools, and skills as part of the AI and ML megatrends. At an early 2019 Future of Systems Engineering (FuSE) workshop hosted by INCOSE, the terms AI for SE and SE for AI were first used to describe this dual transformation [15]. The "AI4SE" and "SE4AI" labels have quickly become metaphors for an upcoming rapid evolutionary phase in the SE Community. AI4SE may be defined as the application of augmented intelligence and machine learning techniques to support the practice of SE. Goals in such applications include achieving scale in model construction, confidence in design space exploration, and automated validation coverage. SE4AI may be defined as the application of SE methods to the design and operation of learning-based systems. Key research application areas include the development of principles for learning-based systems design, models of life cycle evolution, and model curation methods [14]. These are summarized from published SERC roadmaps on AI and Autonomy drivers for SE [14].

SE and software engieneering are fields that have language-based models that are very amenable to machine automation. Tools exist today that use large language models to author software code. AI-based tools that author SE requirements and descriptive models are beginning to emerge at the time of this writing. Some of the technologies and use cases to watch for include automated construction of models from features in semantic data, used in both creation of new models and correctness of developed models; automated search through data and models; automation of evidence-based models for assuring correctness and completeness of system requirements and design; automation of certification and accreditation processes via models and automation of quality assurance data; and eventually chatbots or cognitive assistants that automate many mundane data entry, exploration, engineering calculation tasks, and workflows.

Likewise, the emerging needs levied on SE by LBS will introduce significant changes in SE methods, processes, and tools. Most of these are reflected in the other trends in sections 3.1-3.6. Some of the more direct expectations include:

- Architecting new combinations of live and virtual digital architectures that dynamically adjust their structure and functionality.
- New methodologies to support human and machine situational awareness of impending risks and behavioral concerns.
- Methods, processes, and tools to connect system risk analysis results with AI software modules related to those risks.
- New analytical and evaluation methods that calibrate trust in LBS, beyond traditional availability and dependability concerns to concepts of ethics, fairness, etc.
- New mission level analysis and risk models arising from human and AI collaboration in shared tasks and functions.
- Methods for addressing AI-related system test and evaluation addressing these systems' ability to adapt and learn from changing deployment contexts.
- Computer-based simulation and training supporting non-static objectives and/or goals (games, course of action analysis) necessary to provide contextual learning environments for these systems.

This will also be a transformation of the SE workforce, with significantly more integration of software and human behavioral sciences at the forefront. As digital engineering evolves and traditional engineering models and practices rely more on the underlying data, many engineering tasks related to data collection and search, data manipulation, and data analysis will become automated. Also, the machine learning of modeled relationships and underlying data will become more powerful over time. This should be a positive change, automating many mundane engineering tasks leading to a greater focus on problem solving and design for the human engineer. Engineering speed and quality should improve as more engineering test and validation activities become automated. The idea of "cognitive assistants" that broadly support the engineer will evolve but they must evolve in a way that supports the problem solving and associated learning processes associated with engineering [14, 23].

4. CONCLUSIONS

This chapter presented four new types of systems that collectively are driving significant change in the SE community. These systems are highly adaptive, are highly connected including CPS and other systems that are the traditional core of SE, are governed in a highly distributed and flexible manner, and increasingly learn new behaviors and adapt on their own.

Seven trends that will significantly change the SE discipline are discussed, primarily from research and road mapping activities over the past 5 years in the SERC. These are digitalization, agile processes, the evolving digital SE ecosystem, modularity and open business models, digital system models, ontologies and semantic technology, and, of course, AI and ML. Many of these trends are occurring independently of each other, and it is the role of SE to be the integrator of these as well as the systems we impact. Some of these will be further elaborated in the following chapters.

There are numerous other trends that have not been included in this chapter but may also need to be incorporated into the evolution of SE practice. These have been chosen because, in our experience, are having the strongest impact in SE transformation at the time of writing this chapter.

REFERENCES

1. Acquisition Innovation Research Center (AIRC) (2023). Agile Development of Hardware-Reliant Systems. Retrieved September 1, 2023 from https://acqirc.org/events/agile-development-of-hardware-reliant-systems-workshop/.

2. Bhatia H (2018). "Connected car opportunity propels multi-billion-dollar turf war." Economic Times. Retrieved September 1, 2023 from https://telecom.economictimes.indiatimes.com/tele-talk/connected-car-opportunity-propels-multi-billion-dollar-turf-war/2971.

3. Boardman J and Sauser B (2006). "System of systems - The meaning of 'of,'" In 2006 IEEE/SMC International Conference on System of SE. IEEE.

4. Brose C (2020). The Kill Chain: Defending America in the Future of High-Tech Warfare. Hachette Books.

5. Brunton S et al. (2020). Data-Driven Aerospace Engineering: Reframing the Industry with Machine Learning. Retrieved September 1, 2023 from https://www.researchgate.net/publication/343877323_Data-Driven_Aerospace_Engineering_Reframing_the_Industry_with_Machine_Learning.

6. Delligati L (2013). SysML Distilled: A Brief Guide to the Systems Modeling Language. Addison-Wesley.

7. Dahmann J, Rebovich G, Lane J, Lowry R, and Baldwin K (2011), "An implementers' view of SE for systems of systems," 2011 IEEE International Systems Conference, Montreal, pp. 212-217.

8. Geels F (2002). "Technological transitions as evolutionary reconfiguration processes: A multi-level perspective and a case-study," Research Policy, December 2002, 31 (8-9), 1257-1274.

9. Gorman S & Eiras A (2023). "SpaceX rocket explosion illustrates Elon Musk's 'successful failure' formula". Reuters. Retrieved September 1, 2023 from https://www.reuters.com/lifestyle/science/spacex-rocket-explosion-illustrates-elon-musks-successful-failure-formula-2023-04-20/.

10. Griffor E (ed) (2016). Handbook of System Safety and Security: Cyber Risk and Risk Management, Cyber Security, Adversary Modeling, Threat Analysis, Business of Safety, Functional Safety, Software Systems, and Cyber Physical Systems. Syngress.

11. Griffor, E, Greer, C, Wollman, D and Burns, M (2017), Framework for Cyber-Physical Systems: Volume 1, Overview, Special Publication (NIST SP), National Institute of Standards and Technology, Gaithersburg, MD, [online], https://doi.org/10.6028/NIST.SP.1500-201. Retrieved September 1, 2023.

12. INCOSE (2023). SE Vision 2035: Engineering Solutions for a Better World. International Council on SE. Retrieved September 1, 2023 from https://www.incose.org/about-systems-engineering/se-vision-2035.

13. Maier M (1996). "Architecting Principles for Systems of Systems," Proc. of the Sixth Annual International Symposium, International Council on SE, Boston, MA, pp. 567- 574.

14. McDermott, T, Blackburn, M, & Beling, P (2021). Artificial Intelligence and Future of SE. In: Lawless, W.F., Mittu, R., Sofge, D.A., Shortell, T., McDermott, T.A. (eds) SE and Artificial Intelligence. Springer, Cham. https://doi.org/10.1007/978-3-030-77283-3_3.

15. McDermott, T, DeLaurentis, D, Beling, P, Blackburn, M & Bone, M (2020), AI4SE and SE4AI: A Research Roadmap. INSIGHT, 23: 8-14. https://doi.org/10.1002/inst.12278.

16. McDermott T, Alexander K, & Wallace R (2023). The Supra-System Model. INSIGHT 26 (2), 15-21.

17. McDermott and Horowitz 2017 is "McDermott T, Horowitz B (2017) Human Capital Development – Resilient Cyber Physical Systems. Systems Engineering Research Center (SERC) Technical Report SERC-2017-TR-075, September 29, 2017.

18. National Academies of Sciences, Engineering, and Medicine (2016). A 21st Century Cyber-Physical Systems Education. Washington, DC: The National Academies Press. doi:10.17226/23686.

19. National Science Foundation (NSF) (2016). Cyber Physical Systems (CPS). Retrieved September 1, 2023 from https://new.nsf.gov/funding/opportunities/cyber-physical-systems-cps.

20. Office of the Deputy Assistant Secretary of Defense for SE (2018). Department of Defense Digital Engineering Strategy, Washington DC.

21. Rotmans J, Kemp R, & van Asselt M (1999). "More evolution than revolution: transition management in public policy," Foresight 3 (1).

22. SAE (2023). OnQueTM Digital Standards System. Retrieved September 1, 2023 from https://www.sae.org/onque-digital-standards.

23. Salado A and Selva D (2021). "Asistentes Cognitivos en Ingeniería," UE Essentials, Universidad Europea de Madrid, Madrid, Spain.

24. Systems Engineering Research Center (SERC) (2017). Human Capital Development – Resilient Cyber Physical Systems. Technical Report SERC-2017-TR-075. Stevens Institute of technology, Hoboken NJ.

25. SERC (2023). Information Models and Ontologies to Enable Digital Engineering. Retrieved September 1, 2023 from https://sercuarc.org/event/information-models-and-ontologies-to-enable-digital-engineering-research-workshop/.

26. Shadab N, Kulkarni A, Salado A (2021). "Challenges to the Verification and Validation of AI-enabled Systems: A Systems-Theoretic Perspective," in W.F. Lawless, R. Mitty, D. Sofge, T. Shortell, T. McDermott (Eds.), Systems Engineering and Artificial Intelligence, (pp. 363-378), Cham: Springer, 2021.

27. von Bertalanffy L. (1969). General Systems Theory. New York: George Braziller.

28. Wikipedia contributors. (2021, September 29). Adaptability. In Wikipedia, The Free Encyclopedia. Retrieved September 1, 2023, from https://en.wikipedia.org/w/index.php?title=Adaptability&oldid=1047218141.

29. Zhu H and Arnold E, "System Adaptability." in SEBoK Editorial Board. 2023. The Guide to the SE Body of Knowledge (SEBoK), v. 2.8, R.J. Cloutier (Editor in Chief). Hoboken, NJ: The Trustees of the Stevens Institute of Technology. Accessed September 1, 2023. www.sebokwiki.org. BKCASE is managed and maintained by the Stevens Institute of Technology SE Research Center, the International Council on SE, and the Institute of Electrical and Electronics Engineers Systems Council.

BIOGRAPHIES

THOMAS ALLEN MCDERMOTT

Tom McDermott is the Chief Technology Officer of the Systems Engineering Research Center (SERC) and a faculty member in the School of Systems and Enterprises at Stevens Institute of Technology in Hoboken, NJ. With the SERC he develops new research strategies and is leading research on digital transformation, education, security, and artificial intelligence applications. He previously held roles as Faculty and Director of Research at Georgia Tech Research Institute and Director and Integrated Product Team Manager at Lockheed Martin.

Mr. McDermott teaches system architecture, systems and critical thinking, and engineering leadership. He provides executive level consulting as a systems engineering and organizational strategy expert. He is a fellow of the International Council on Systems Engineering (INCOSE) and recently completed 3 years as INCOSE Director of Strategic Integration.

VÍCTOR RAMOS DEL POZO

Víctor Ramos del Pozo is a systems engineer at Isdefe, currently delivering technical assistance for the Spanish National Programme Office for the Next Generation Weapon System (NGWS) within a Future Combat Air System (FCAS), especially in the fields of the Combat Cloud, Collaborative Sensors and Simulation. He previously delivered technical assistance for the Spanish National Armaments Directorate in the fields of defence industrial base analysis, defence acquisition planning and major defence acquisition programmes management.

Before joining Isdefe, he worked as a systems engineer at EADS-CASA (current Airbus Defence and Space) for the Multi-Role Tanker Transport and Future Strategic Tanker Aircrafts programmes, and at Indra for the Identification Friend or Foe Department.

He is a member of the International Council on Systems Engineering (INCOSE) and delivers internal training at Isdefe in the fields of systems engineering and programme management.

"We have developments that follow all of the rules, but fail."

J. Frosch

CHAPTER 3

Evolution of systems engineering development and execution models

Dr. Ronald Giachetti, *Naval Postgraduate School (regiache@nps.edu)*
Juan Carlos Lario Monje, *Isdefe (jclario@isdefe.es)*

Abstract

This chapter presents the need to evolve and adapt systems engineering development models to the current context of engineering projects. The chapter separates the discussion between traditional, dominantly plan-driven approaches to systems development such as the Waterfall and Vee models, and discussion of agile development approaches that emphasize responsiveness and are characterized as highly iterative and incremental. Most projects could benefit from both, and we discuss hybrid approaches and how to tailor the development models to the context of particular projects. The chapter discusses other emergent methods including the merging of development with operations through DevOps.

Keywords

Systems engineering, development models, agile, DevOps, hybrid usage.

1. INTRODUCTION

A system development model defines an approach to develop a system to satisfy some needs expressed by one or more stakeholders. The development of large scale and complex systems often entails multiple technologies and a large number of diverse individuals bringing different knowledge and skills to the effort. This chapter first describes the purpose and intent of system development models. The models are organized into two categories: plan-driven models and agile models. Plan-driven models assume we sufficiently understand the needs and solution such that we can plan the development project in advance and then develop the system according to the plan. Agile models take the position that we do not fully understand the needs or solution and therefore the project execution must be adaptive or flexible, allowing for changes in the needs and/or the solution. After introducing multiple representative development models, the chapter then discusses how to tailor a development model to a particular project considering all the contextual factors contributing to a successful outcome.

1.1. System development context

The development of large-scale, complex systems places multiple demands of organizational nature (i.e., over the organization that develops the system, also called the realization system), which a system development model is intended to address. In this sense, the basic demands a team developing a new system will have to face are understanding the stakeholder needs, creating a concept addressing those needs, designing and integrating the concept with hardware, software, processes, and people, and then implementing the designed system so that it can be deployed.

System development is almost always constrained by schedule and budget, and system development models must be realizable in these conditions. An organization engages the work of a large number of people with different knowledge and skills during the system development, all of whom must be organized in an effective way to design and build the system. A development model provides a means to understand how to organize the work of all these individuals. Many systems include new or emerging technologies, which introduce risk into the development. Under these conditions, a system development model must provide for means to mature such technologies and to assess the associated risks to the development organization as well as to the system itself (e.g., risks that may materialize during system operation).

Many systems, such as those incorporating greater autonomy or artificial intelligence, face the challenge of maturing a technology in parallel to system development. At the same time, safety-critical systems, such as aircraft or medical devices, must be developed and will operate in highly regulated environments. These aspects generally impose severe constraints on the generation of information and data during system development, which is enabled and/or facilitated by adopting a suitable development model.

Since software has become an essential component of most systems, system development must now contend with the generally high complexity introduced by software, potential security vulnerabilities previously unaccounted for, and rapid deployment at *almost the click of a button*. In the past, when systems consisted mostly of hardware components, the system capabilities generally remained unchanged once the system was deployed. Because software is not physical (it is a set of instructions that must be installed on hardware), it enables system owners to continuously update system capabilities during utilization. Furthermore, software components have also enabled the implementation of artificial intelligence, which can create scenarios in which the algorithms and/or decisions made by the system can change over time [26]. This creates challenges for system verification as well as for human trust in the system concerning safety as well as other issues. Collectively, these characteristics affect the development process of software-intensive systems and any software components found in larger systems.

This section has described the context surrounding modern large-scale and/or complex system development projects. System development models have evolved to contend with many of the issues presented above.

1.2. Definition of a system development model

A system development model provides a **structured framework** and set of guidelines for organizing the development activities and coordinating the technical development of a system. Development models are often governed by a set of principles and/or best practices. The use of a development model helps in organizing and managing the associated efforts, ensuring a certain quality level, reducing risks, and improving collaboration and communication among team members and other stakeholders.

Following a defined process is correlated with a higher likelihood of delivering successful system and capabilities within the constraints imposed on time, cost, and quality [7].

System development models generally divide the process into phases to describe the progress or evolution of the system through its life cycle. Milestones are *decision gates* marking the completion of a phase, at which time the development organization decides whether to continue the system development to the next phase, to wait for a complete accomplishment of certain identified pending actions, or, as a worst case, terminate the project. Entry and exit criteria for each milestone are generally well defined in advance in order to minimize any undesirable uncertainties. In this sense, the risk of going ahead is often estimated and weighed before making the decision.

1.3. System life cycle model

The development of a system must be understood in the context of the system life cycle. A system life cycle describes the phases a system goes through from its initial conceptualization to its retirement or disposal, of which development is just one portion. Figure 1 shows an example of how ISO/IEC 15288 defines them. The arrows indicate the existence of multiple potential paths through the different phases, which may involve iteration and repetition. A new, unprecedented system might go through the phases sequentially from concept to development, to production, and to utilization and support, before being retired. Instead, an existing system might be upgraded and cycled back from utilization to concept again. Some systems may be designed and deployed incrementally and would transit through multiple iterations of the phases for different aspects of the overall system.

It should be noted that systems engineering activities such as the elicitation of stakeholder needs or system integration are not *exclusively* associated with any of the individual phases of the system lifecycle. Furthermore, some system engineering activities may be executed iteratively and recursively at multiple levels of the system hierarchy. Consequently, systems engineering spans all the phases of a system's life cycle.

1.4. System development models

As the body of knowledge surrounding process models evolves, the International Council on Systems Engineering (INCOSE) has revised their classification of system development models several times [20]. We describe two categories as:

1) Pre-specified or plan-driven models.
2) Evolutionary or agile models.

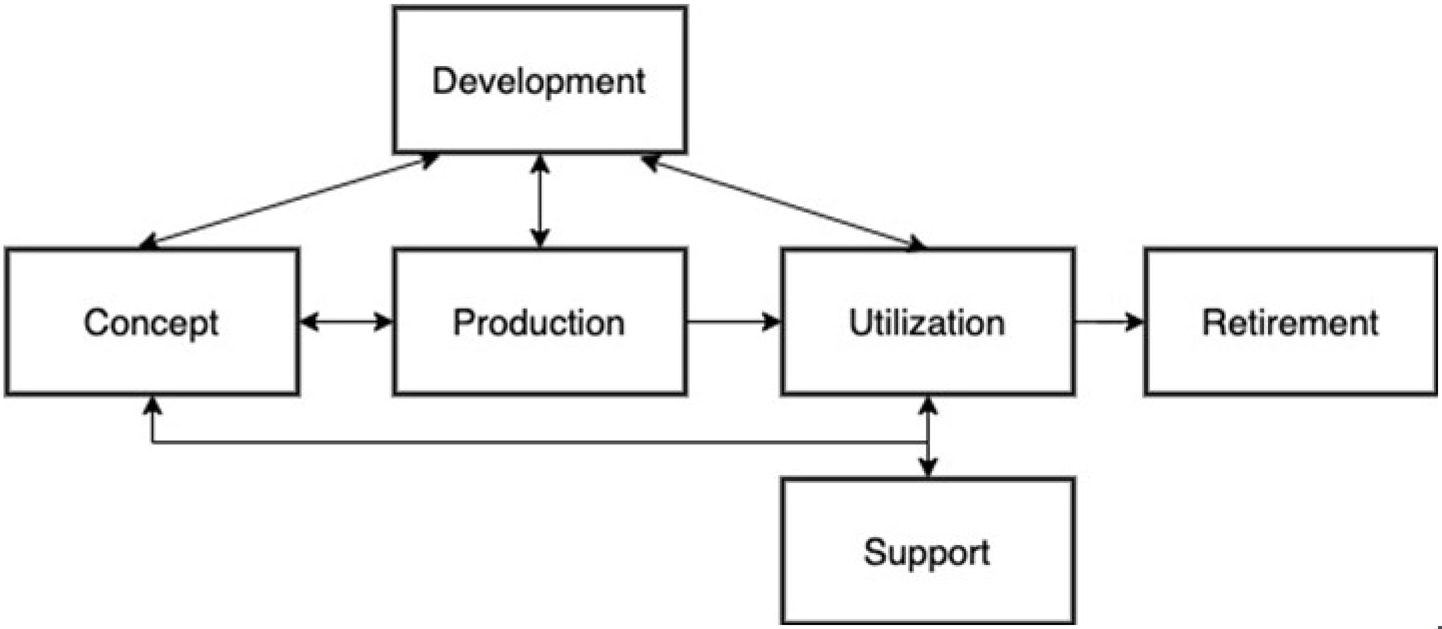


Figure 1. System life cycle (adopted figure from INCOSE Handbook, 2023)

The first category includes traditional models such as the *Waterfall model* and the *Vee model*. The second category describes an evolution away from strictly plan-driven models and includes the *Spiral model* based on iterative development cycles. The second category has come to be called agile methods and borrows heavily from the software engineering community. The categorization is according to the *dominant* characteristics of the corresponding models. Nevertheless, it should be noted that certain overlap between the categories exist because most models share characteristics. For instance, many authors present the Waterfall model as the archetypical plan-driven model with strictly sequential execution of activities. Yet, the original article by Royce acknowledged iteration among the activities. In fact, iteration and recursion are always inherent to any development process, within and between its different phases.

2. PLAN-DRIVEN DEVELOPMENT MODELS

Plan-driven models describe a class of system development models that prioritize the predictability available through adhering to a plan laying out a structured development approach. This section reviews the archetypical plan-driven development models of the Waterfall and the Vee models.

2.1. Waterfall model

The Waterfall model is a plan-driven model in which all engineering activities are defined and planned at the start of the project. Each activity in the Waterfall model is supposed to be executed completely before the next activity begins. Moreover, the output of the preceding activity becomes input to the next activity. Iteration of activities is not planned in the Waterfall model but is expected to happen only when problems are not fixable at one stage and, in such a case, the project *moves* just to the previous stage. Figure 2 shows the obvious *sequential* structure of the activities in the Waterfall model.

Although the Waterfall model is often criticized and almost considered irrelevant in systems engineering circles these days, the model is still well suited and useful for projects in which:

- it is possible to confidently know the requirements at the beginning,
- those requirements will be stable and unchanging,
- there is little technical risk, and/or
- there are hazardous risks that require careful progression of activities.

Under these conditions, the Waterfall model's sequential and structured approach for system development provides the benefits of predictability and easier coordination of the work activities.

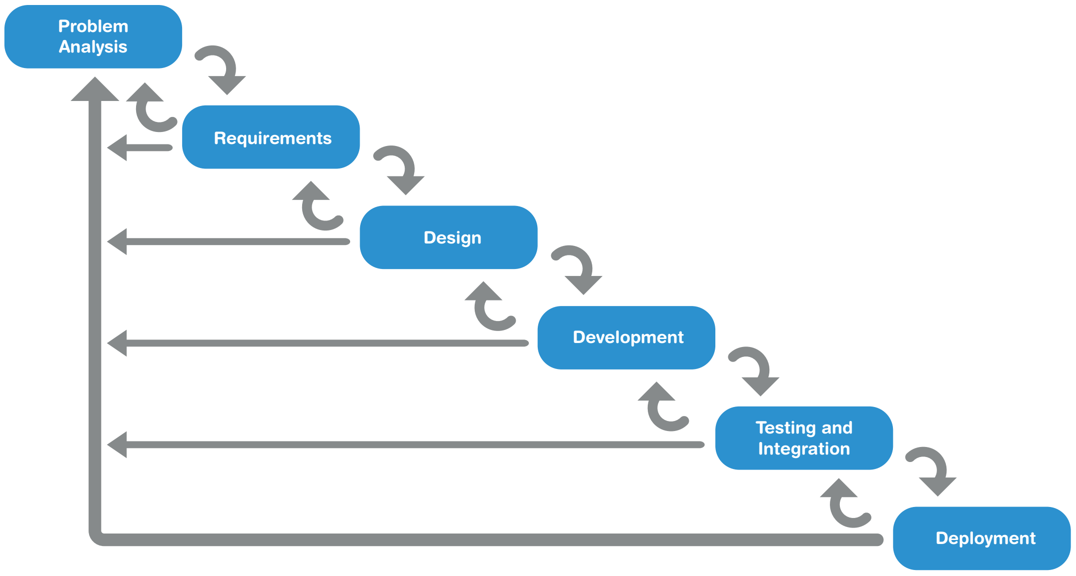


Figure 2. Waterfall model

The Waterfall model becomes a challenge to implement and can cause extensive and expensive rework under scenarios where:

- the requirements are poorly known,
- the requirements are likely to evolve or change, and/or
- it is possible that new requirements emerge during development.

Any issues discovered during later activities such as testing and/or integration can mean a significant amount of rework, since the team may need to go all the way back to requirements and redesign the system. Such rework usually causes significant budget overruns and schedule slips. Furthermore, the Waterfall model does not explicitly incorporate stakeholder feedback during the lifecycle, since the activity is timeboxed within its own stage, which limits its ability to validate concepts before doing too much work or to adapt to changing needs. Moreover, the tightness of the development process eliminates any opportunity for incrementally deploying the system. These aspects make the Waterfall model unattractive for most engineering projects.

2.2. Vee model

The systems engineering Vee model is a plan-driven model that leverages the hierarchical nature of most complex systems. Development occurs in a top-down fashion in which a system is recursively decomposed into lower-level systems (which may be assigned names as subsystems, components, etc.). Importantly, the Vee model recognizes uncertainty and technical maturity are not uniform across all elements of the system, and hence allows for different system elements to evolve at different paces. This is actually its key tenet, that while all activities must end in sequence (as was the case for the Waterfall model), activities may begin in any order. That is, within the Vee model, the activities might not begin in the same order for every system element, but their completion does occur per the constraints of the process such that integration cannot be completed until all system elements are manufactured. In fact, the Vee model emphasizes the validation and verification at each level and between levels, which can quickly lead to iteration as the team resolves any issues [32] to reduce the financial and schedule impacts of rework. For example, the Vee model encourages to begin validation in parallel with the elicitation of stakeholder needs, start verification in parallel with derivation of requirements, start integration as soon as system architecture starts, etc.

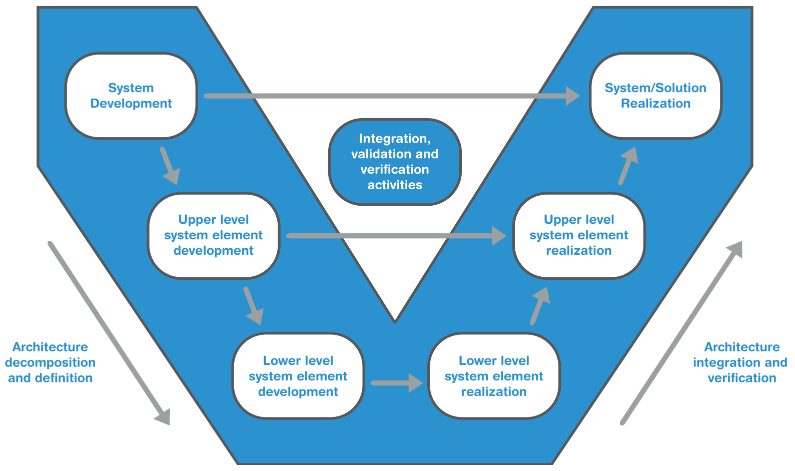

System Development
System/Solution Realization
Integration, validation and verification activities
Upper level system element development
Upper level system element realization
Lower level system element development
Lower level system element realization
Architecture decomposition and definition
Architecture integration and verification

Figure 3. Vee Model

Figure 3 shows the original, canonical Vee model [14] of which there are now many variations (many of them are inconsistent with the concepts expressed in the original Vee model). The left-hand side of the Vee depicts the decomposition effort as stakeholder needs being mapped into system design concepts and eventually derived into a system specification. At the bottom of the Vee model, those system elements are designed and built, and give way to the right-hand side of the Vee model, which depicts the synthesis or integration effort as those elements are integrated up the system hierarchy with verification and validation occurring at each step. The symmetry between the left- and right-hand sides of the Vee model highlights the interdependencies between each stage on the left with the corresponding one on the right. For instance, the concept of operations on the left side defines measures of effectiveness and performance that are the basis of the system validation occurring later in the project.

The Vee model is particularly suited for system development projects that require a systematic and structured approach, where:

- systems in question are large,
- involve many building components, which have different levels of maturity and therefore risks, and
- have stringent safety and reliability constraints.

The Vee model, or minor variations of it, is probably the most common mode used in industries such as aerospace, defense, and automotive.

The Vee encounters some of the same shortfalls as experienced with the Waterfall model. The Vee model suffers from poor responsiveness to significant changes in requirements or poor initial assumptions that are not discovered until later in the process. The result is costly rework with the concomitant delays in schedule.

3. EVOLUTIONARY OR AGILE DEVELOPMENT MODELS

Evolutionary or agile models describe a class of highly iterative and incremental approaches to system development. Agile models carry out iteration and incremental development based on short development cycles called *sprints* or iterations. Iteration describes the repetition of the same activities during each sprint such as analysis, design, test, and build for a small portion of the system. We see the incremental aspect through the delivery of value or system capabilities at the end of each sprint. Agile models emphasize the end of a sprint should result in a measurable outcome. For software systems, each sprint usually ends with functional code that can be delivered to stakeholders. For hardware-intense systems, a sprint might not result in actual hardware because of the longer development time required, and instead a sprint may result in risk reduction, an engineering specification, or completion of an engineering analysis. This type of model emphasizes value delivery as an incremental process whether such delivery is some visible portion of the system or advancement of the project at the end of each sprint.

Unlike plan-driven models, the agile models do much less planning and requirements analysis during the beginning of system development. Agile models are highly adaptive and flexible approaches because as the project team plans each new sprint, they will make course corrections based on what they learned during the previous sprint. Hence, agile methods are updating their plans on a regular basis driven by the cadence of the sprints.

The agile models are based on a set of principles informing a mindset and then implemented by multiple, different methods. The principles were described in the Agile Manifesto [2] and are listed in Table 1.

1.	Early and continuous delivery of value to the customer
2.	Acceptance of changing requirements, important for competitive advantage
3.	Frequent incremental delivery of value to stakeholders
4.	Users and developers working together
5.	Focus on the people, self-organized teams
6.	Face-to-face communication is most effective for teamwork
7.	Working software is primary measure of progress
8.	Development speed should be sustainable indefinitely
9.	Continuous attention to good design
10.	Keeping the design as simple as possible and avoid doing unnecessary work
11.	Best architectures emerge from self-organized teams
12.	Regular meetings for teams to reflect on lessons learned

Table 1. Agile Principles

A differentiating aspect of the agile models is that they comingle principles on how to develop a system with principles on how to organize the development team. This is a differential aspect with respect to many of the plan-driven models, which provide little guidance about the project management considerations of the system development. The principles dealing with continuous delivery, acceptance of changing requirements, using working software as the measure of progress, steady development speed, constant attention to good design, and simple design are principles all dealing with system development. Agile models uniformly prescribe self-organizing, multidisciplinary teams working in close collaboration with end users (generalizable to general stakeholders) to deliver value incrementally. Furthermore, they place greater value on face-to-face communication and regular meetings as a means for better coordination and faster recognition and adoption of lessons learned, which are intended to improve the management of development teams.

Agile methods have been widely adopted and routinely practiced in software development projects to improve team collaboration, communication, and flexibility in responding to changes in customer and/or user requirements. The application of agile principles to systems engineering processes, particularly in the development of complex systems with hardware components, remains an area of active research and discussion. Several studies have shown the benefits of applying agile principles to systems engineering processes. Dove et al. [11] studied the application of agile principles in several companies and identified eight principles for agile systems engineering consistent with the agile manifesto including attentive situational awareness, attentive decision making, agile operations concept, product line architecture, shared knowledge management, continual integration and test, common mission teaming, and iterative and incremental development. The International Council on Systems Engineering (INCOSE) has a team that has been looking at how agility can be infused or adopted by systems engineering organizations [38]. Their work is part of the Future of Systems Engineering (FUSE) initiative of INCOSE and emphasizes how development organizations can become more agile through process and workforce development. Therefore, while agile models are proven and the dominant approach in the software, systems engineering is still examining and trying to understand how best to adopt agile principles given the constraints of physicality imposed by hardware.

Next, we present the Spiral model, which was introduced in the 1980s, as an example of an evolutionary model. Then, Scrum is presented because it is widely used in the software industry and the iterative process and concepts from Scrum are heavily borrowed by other evolutionary models. Finally, we discuss two system development models that adopt agile concepts from Scrum for larger projects including both hardware and software: the Scaled Agile Framework (SAFe) and the Disciplined Agile Delivery (DAD).

3.1. Spiral model

The Spiral model was introduced for software development and emphasizes reducing risk through incremental commitment from stakeholders throughout system development [3]. A project following the Spiral model does iterations to progressively develop the definition of a software system and eventually deliver it to a customer. The cycles consist of the four activities (ref. Figure 4):

1) determining objectives, alternatives, and constraints,
2) evaluating the alternatives, identifying and resolving risks,
3) developing and verifying the product, and
4) planning the next phase.

Each iteration or spiral results in a prototype or build of the system, albeit often a partial build. Consequently, the project develops and deploys the system in increments.

The Spiral model is suitable in system development contexts characterized by uncertain or evolving requirements and projects where risks are significant and need to be continuously assessed and mitigated throughout the project lifecycle. However, the use of the Spiral model is generally only possible when the system can be deployed in increments. Software systems was one of the original domains for which

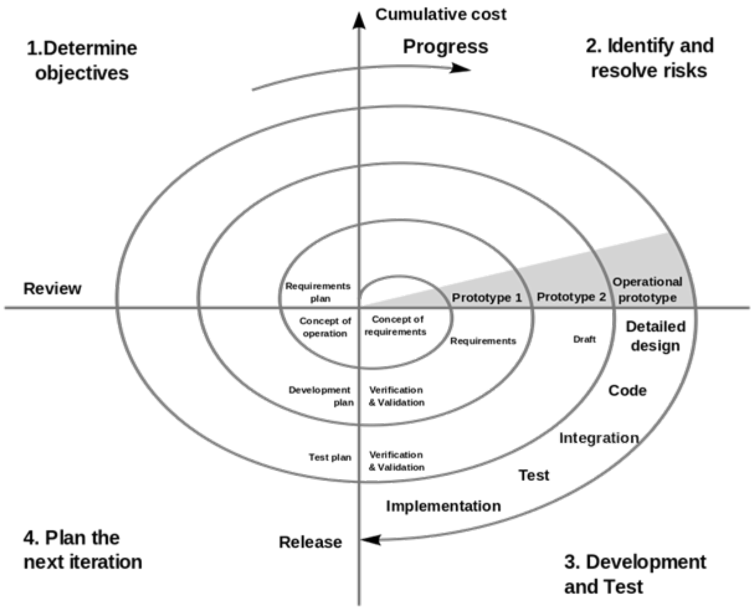


Figure 4. Spiral Model

the Spiral model was intended [4], but it has also been applied to other domains. For example, the ship design process is usually shown as a Spiral model because ship design involves multiple technical specialties with a high degree of interdependence between them. In the ship design spiral, the designers make decisions on hull form, hull size, power, hydrostatics, and so on and must iterate through these decisions on they converge on an acceptable design [13,15]. The spiral model has also been applied to the development of the Global Hawk unmanned aircraft, which was able to deploy capabilities incrementally with each spiral [17].

3.2. Scrum

Scrum is an agile software development model for managing self-organized teams in short iterations called sprints delivering working code at the end of each sprint [34]. The popularity and success of Scrum for software development has motivated people to adapt the Scrum development model to systems involving both hardware and software [e.g., 3, 10]. Figure 5 shows the Scrum process. Teams work from a backlog of work items that were identified in conjunction with a Product Owner representing the voice of the customer. The work items are requirements for features or functions often expressed as user stories. During the sprint planning session, the team selects work items to analyze, design, develop, and test in the next sprint. The sprint is the actual development iteration and is typically of two weeks duration for software development but may be longer and even vary for hardware development. The team will have a regular meeting to discuss progress of the sprint. A sprint retrospective meeting is held at the end of the sprint for the team to discuss ways in which they can improve. Scrum is a learning process and focuses on the people and interactions between people in developing software.

Scrum works well in dynamic environments where requirements change (because new and/or changed requirements can be worked into the backlog and be addressed in future sprints), as well as with systems that can be built and deployed incrementally. Adoption of the Scrum development model requires implementing a corresponding organizational culture, as with other agile models, because it intertwines development processes with team management and communication principles. An organization that does not subscribe to such working principles as established by the team would likely not be successful in performing a Scrum process.

Because Scrum was originally applied to small projects, there were some initial doubts about its suitability to guide large projects. However, variants such as Large-Scale Scrum (LeSS) [24] were proposed to address this potential shortcoming. While Scrum has been adapted and applied for systems development, it is not widely practiced, and many questions remain as to its applicability in different system development contexts.

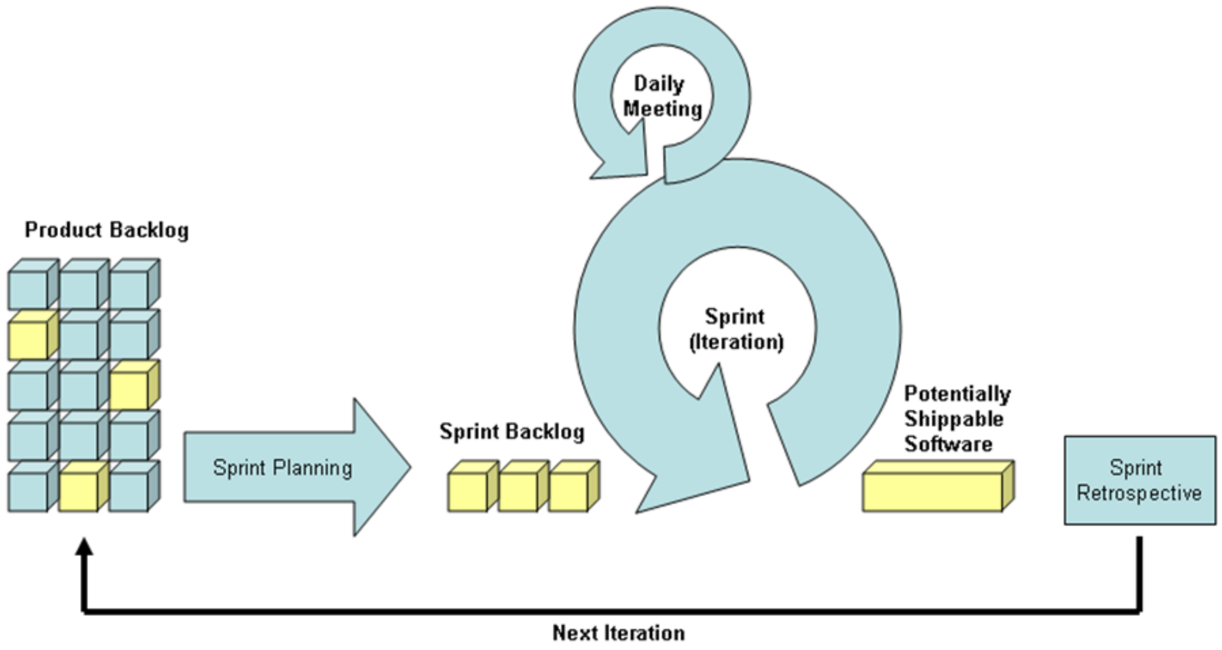


Figure 5. Scrum Process

3.3. Scaled Agile Framework

The Scaled Agile Framework (SAFe) describes a structured and scalable approach to implementing agile principles and practices in organizations with multiple teams working on complex projects [25]. The teams consist of individuals who have the ability to do the analysis, design, build, and test of their work. They work on the systems development through Agile Release Trains (ARTs), which are defined time frames (typically 8-12 weeks) during which multiple teams synchronize their work to develop and deliver a capability of value to the stakeholders. Hence, ART cycles are strictly time-boxed and their scope is allowed to vary.

The teams plan the development goals of an ART through an event called Program Increment (PI) Planning. The PI Planning brings together all teams within an ART to plan and align their work for the upcoming program increment. During PI Planning, teams collaborate to define objectives, break down work, estimate effort, and establish dependencies. SAFe includes a lot of guidance on management of the development team and how it should be organized.

SAFe was developed to address the criticism that agile methods such as Scrum, do not scale to large and more complex system development projects. SAFe does this through the guidance provided for scaling agile from the team level to the program and portfolio levels. It offers different configurations, allowing organizations to tailor their implementation based on their specific needs. In providing the additional guidance on how to structure the development, SAFe is sometimes criticized as sacrificing the agile principles for the sake of greater discipline [31].

3.4. Disciplined Agile Delivery

Disciplined Agile Delivery (DAD) is an agile method that builds on other methods such a Scrum but is intended for the entire life cycle, including operations, for example, unlike other agile methods that focus primarily on development [1]. DAD defines three phases: inception, construction, and transition. During inception, the project identifies the project vision, stakeholders, and initial requirements. In construction, the project develops the system solution on an incremental basis. In transition, the project puts the design through production and engages stakeholders for validation. The DAD applies an iterative and incremental development approach, characteristic of agile models, during the construction phase.

DAD is mainly intended for larger projects by being enterprise aware, meaning DAD recognizes the enterprise context and takes into consideration governance, compliance, and organizational standards. DAD emphasizes the importance of validating the architecture in the earlier sprints and consequently reducing risk. The DAD is particularly appropriate to software-intensive systems in which the development team can establish and react to feedback loops between operations and development (i.e., DevOps, which will be described later).

3.5. Agile methods with hardware

Agile methods, originally developed for software development, may encounter certain difficulties when applied to hardware development due to the inherent physicality and manufacturing aspects involved. We group these constraints here because they apply to all of the agile models previously discussed. Some of the constraints that can limit the application of agile methods for hardware development are:

- **Longer Lead Times:** Hardware often requires longer lead times for designing, ordering materials, and building components compared to software development. This can make it challenging to adhere to the short iteration cycles typically associated with agile methods.

- **Manufacturing Complexity:** Manufacturing hardware involves intricate processes, specialized tools, and strict quality control measures. Unlike software, which can be *easily* modified and updated, modifying and updating physical hardware components often require different tools, facilities, specialized personnel, materials, and so forth. This can limit the flexibility and ability to adapt quickly, which is a key aspect of agile methods.

- **Costly Iterations:** Iterating on hardware designs can be expensive, especially when it involves tooling, materials, and production processes. Unlike software, where changes can be made relatively *easily* and at a low cost, hardware iterations often require additional investments. This can make frequent iterations and experimentation challenging from a cost and resource perspective.

- **Physical Prototyping:** Physical hardware typically requires the creation of prototypes for testing and validation. Building physical prototypes can be time-consuming and costly, which can impact the ability to iterate rapidly and embrace quick feedback loops, as typically done in agile methods.

- **Supply Chain Dependencies:** Hardware development often relies on complex supply chains, involving multiple vendors and lead times for procuring components and materials. This introduces additional dependencies and challenges in managing schedules, coordination, and ensuring timely availability of required resources.

- **Regulatory Compliance:** Hardware products often need to comply with industry-specific regulations and standards. Ensuring compliance can involve extensive testing, certification processes, and documentation. This can add complexity and time to the development process, potentially impacting the agility of the iterative cycles.

- **Law of Physics:** The functioning and performance of hardware systems are bounded by the laws of physics. As a result, strong coupling generally exists between design variables and across engineering disciplines. This makes it very difficult to allocate work in independent tasks.

Despite these constraints, agile principles and practices can still be applied to certain aspects of hardware development. For example, while sprints may not be as short as with software, the principle of frequent scoping and assessment of work can be helpful in keeping a healthy level of productivity and quickly react to misalignments between needs and solutions. This has been shown in practice. For example, Yang et al. [39] showed that the use of agile principles in the development of a hardware product led to improved communication, reduced project risk, and increased customer satisfaction. Similarly, Thakurta et al. [35] showed that the use of these principles in the development of an embedded system led to improved team collaboration, faster development cycles, and reduced project risk. Paasivaara and Lassenius [29] describe Ericsson's long journey of adopting agile to the design and development of their products.

Overall, while the physical constraints of hardware development pose challenges for the direct application of agile methods, careful adaptation, and a tailored approach can help leverage agile principles to enhance collaboration, flexibility, and customer satisfaction in in this type of projects.

4. COMPARISON BETWEEN PLAN-DRIVEN VERSUS AGILE DEVELOPMENT

This section compares the category of plan-driven models versus the category of agile models. We first consider how the two categories of models address the triple constraint of project budget, schedule, and scope.

Plan-driven development models tend to assume a fixed scope-budget-schedule triad (and are often implemented by fixing the scope only), and only act upon them reactively when issues are encountered, such as due to the uncertainty of requirements or the unfolding of technical risks. On the contrary, scope, and to a lesser extent budget are intentionally in flux when using *agile models*. This happens as a result of organizing the work in iterative time-boxed sprints, as explained earlier, so that the project can adapt its tasks and targets as deemed necessary.

It should be noted that, while agile models are generally implemented to fix budget and schedule (letting scope change), and plan-driven methods are generally implemented to fix scope (letting cost and schedule change), these are not constraints imposed by either of the development methods. In fact, either kind of development model can implement techniques that target the fixing of a specific dimension (e.g., Cost as an Independent Variable (CAIV) fixes cost at the expense of schedule and scope [6]). The key difference is whether *changes* are made reactively to problems or intentionally embedded in the development process.

Table 2 provides a brief comparison between how plan-driven models and agile models typologies address core aspects of the business environment, the system to be developed, and the development organization in which they are applied. The table is not intended to be exhaustive, but to broadly hint at the main differences between the two classes of models. As an example, one consideration is the type of business environment the organization and system will operate in. Organizations working in regulated environments and/or dealing with safety critical systems will generally need more planning, documentation, and traceability of requirements. Plan-driven models address these concerns directly. However, this does not mean agile models cannot be used in these environments. For example, Hanssen et al. [16] propose a variation of Scrum, called SafeScrum, demonstrating agile principles in the development of safety-critical systems such as avionics despite the strong belief that agile is inconsistent with the need to comply with stringent safety regulations.

Contingent Factors		Plan-Driven Models	Agile Models
Business Environment	Stability of market and certainty of requirements	Better for stable and unchanging requirements	Better in dynamic environments with changing requirements
	Clarity and certainty of requirements	Better when requirements are clear and known	Better able to deal with ambiguous and unclear requirements
	Regulatory or safety critical environment	Better for complying with strict regulations and policies; ensure traceability with documentation and plan	--
System	Part of a system of system with lots of Interconnections with other systems	Better for planning and controlling those interfaces	--
	Long lead-time items or must plan for scarce resources (e.g., test range)	Better for planning the acquisition of such items and aligning budget, schedule, and resources.	Greater risk of not having such items on hand when needed or available due to limited up-front planning
	Many needed quality attributes (-ilities such as reliability, maintainability, cybersecurity, etc.)	Plan-based models identify and define these requirements early on, which is useful because the quality attributes are often met through multiple aspects of the overall system design	If quality attributes are identified during sprints (especially later sprints), then risk of rework to have the system comply with those requirements because multiple aspects of system might have to change
	Technological risk present	Attempt to develop plan to mature the technology prior to insertion in the system	Rapid learning and risk reduction through iterations
Organization	Highly geographically and organizationally distributed team	The planning and discipline of these models are able to accommodate these scenarios well	Some agile models are less able to handle such organizational structures without significant modification to principles (e.g., difficulty of face-to-face communication). Better for smaller co-located teams

Table 2. Comparison Plan-driven and Agile Models

5. TAILORING DEVELOPMENT MODELS

It is convenient to keep in mind that there is not an ideal or *default* preferred model for a system development. We view the best system development model as one that best fits:

- the organization, its people, and its culture;
- the system to be developed, its complexity, its connectedness to other systems, and the extent of new technologies that it uses; and
- the business environment in which it is developed, the dynamism of the business, and the degree of uncertainty surrounding the development.

For this reason, *tailoring* of the development models should be often called for. Tailoring can take the form of establishing a canonical development model and adapting it to the particulars of the project, it may involve a hybrid approach blending aspects from multiple different development models, or it may be approached by using different development models for different parts of the system (e.g., adopting a Vee model for the hardware components and an agile model for the software). Tailoring relies upon the systems engineering team having deep knowledge of the advantages and drawbacks of each development model and understanding the factors affecting the development, as mentioned earlier (e.g., business environment, system, and organization).

The agile approaches have been demonstrated as being very successful in the software industry, yet there are limitations as aforementioned in applying them to hardware. For this reason, there is interest in hybrid approaches that blend agile and traditional plan-driven development models to strike a balance between flexibility and the need for rigorous development processes. When tailoring, it is essential to address two questions [33]:

1) To what degree is agility demanded by the market, technology, and other environmental factors?.

2) To what degree can the organization be agile?.

In essence, it is important to coordinate the demands or needs that the system must satisfy with the ability and/or feasibility of the organization to adopt a particular development model [5]. Tailoring, in this sense, should not be seen as something to be done when *ideal* models cannot be used. Instead, tailoring should be conceived as a required step when devising what type of model to adopt. Furthermore, *ideal* models should serve as paradigms. In reality, a development model will generally need to be tailored.

Figure 6 refines Table 2 by showing plan-driven models and agile models not as a dichotomy, but as continua of adequacy between the two for different variables.

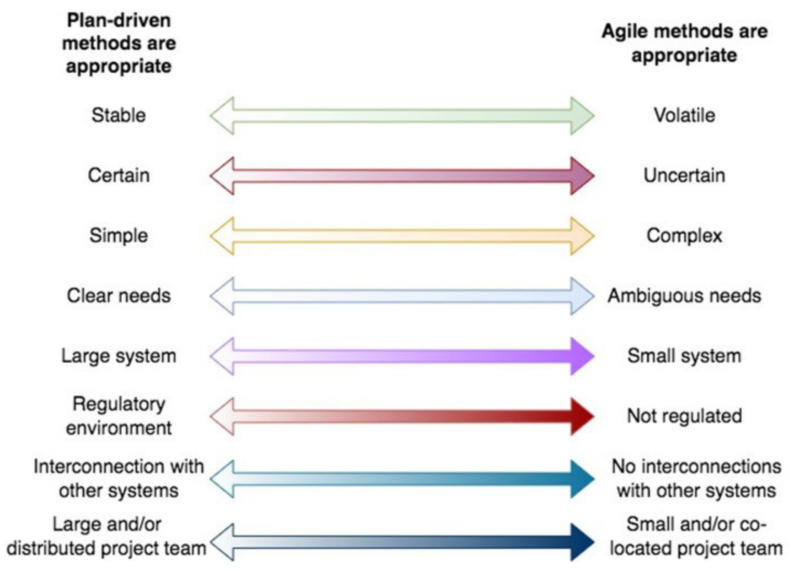

Plan-driven methods are appropriate
Agile methods are appropriate
Stable
Volatile
Certain
Uncertain
Simple
Complex
Clear needs
Ambiguous needs
Large system
Small system
Regulatory environment
Not regulated
Interconnection with other systems
No interconnections with other systems
Large and/or distributed project team
Small and/or co-located project team

Figure 6. Factors for deciding between plan-driven and agile models

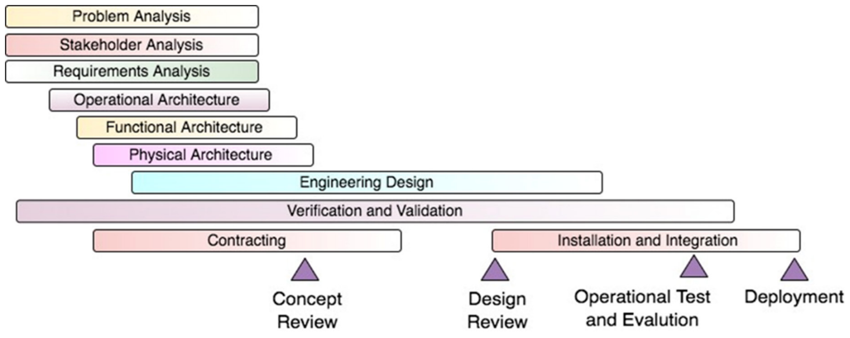


Figure 7. High level plan

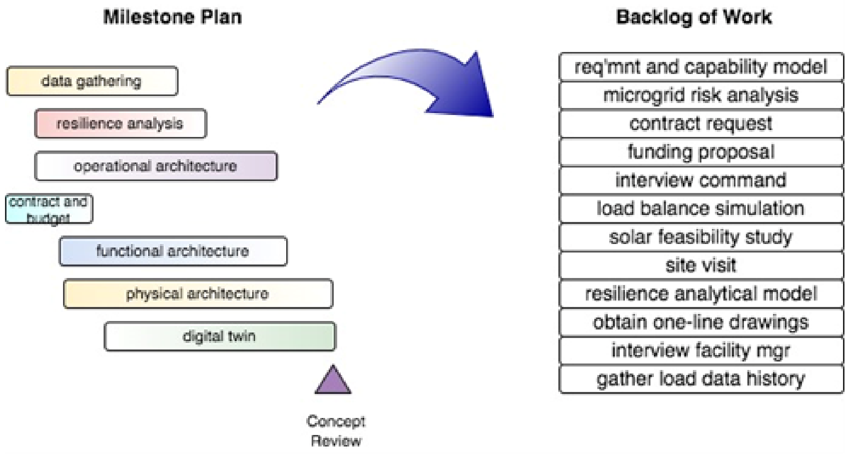


Figure 8. Milestone plan and backlog

One means to combine plan-driven models with agile principles is to adopt and adapt important agile concepts at multiple levels of development. A hybrid method can iteratively and incrementally evolve the system according to agile principles while maintaining some of the predictability available from plan-based models.

As an example, consider hardware-based or hardware-intensive systems. This type of system requires some planning because hardware often requires parts with long lead-times, it cannot be refactored, and customers want to know when the system will be first deployed. Additionally, larger systems often have many interfaces and interactions with other systems that must be planned for and controlled. Interactions in this context also include performance dependencies between the different components. These issues are addressed by macro-planning of the overall development process and by using a *top-down* approach starting with an initial system architecture. Figure 8 shows a high-level view of the system development activities and Figure 8 shows the more detailed planning prior to the next milestone. Both figures show there is extensive parallelism of the activities, but they do not depict how the intensity of effort changes with each activity. For instance, the problem analysis activity might occur through to the concept review milestone, but the amount of manpower and effort will be greater earlier in the project compared to just before that milestone.

Figure 7 shows verification and validation (e.g., testing) occurs *continuously* throughout the process. The frequent testing enables permanent design maturation and risk reduction. Within each phase, there are iterations of understanding, designing, building, and testing ideas through the use of models. Additionally, there are feedback loops and iterations between phases. For instance, as capabilities are analyzed and defined, the team might rethink how they framed the problem and revise their associated analysis. As a result, this process progressively analyzes, designs, and evaluates the stakeholder needs, requirements, and mission to build the architectural products.

6. MERGING DEVELOPMENT AND OPERATIONS WITH DEVSECOPS

System development models have been traditionally designed to end when the system transitions to the utilization phase of its life cycle. While systems may certainly cycle back to development from utilization, development models tend to assume that, in such a case, a new project would be specified. The sharp boundary between development and utilization (also referred to as operations) is being challenged by a novel development model called DevOps.

DevOps emerged from the software engineering community and aims to break down the barriers often found between *system developers* (Dev) and the *system operators* (Ops) [22]. DevOps implements a continuous cycle of developing software, testing it, and pushing the software out to operations who use the software, monitor its performance, and provide feedback to development. DevOps is based on having open and close communication between developers and operators, continuous feedback, continuous integration, and steady operational flow.

DevOps has been and continues to be almost exclusively applied to software-intensive systems. The challenges to adopting DevOps to hardware are evident. Unlike software, new features or functions cannot be pushed out over a network, and moreover, changes to a hardware's performance usually entails changing the hardware itself. DevOps has been challenging to implement even with software for embedded systems [27]. Only recently, do we see DevOps being applied in system domains involving both hardware and software, but with the continuous integration and verification occurring only with the software components [40].

In military systems, the need for security is paramount, and DevOps has morphed to include security considerations, creating a new model concept known as DevSecOps [28]. This approach involves integrating security practices into every stage of the software development process, from design and coding to testing and deployment, to ensure that security is built into the system from the outset. By adopting a DevSecOps approach, military organizations can help to mitigate the risks of cyberattacks, protect sensitive data and systems, and ensure that military operations are not disrupted by software vulnerabilities or failures. Overall, the DevSecOps methodology has emerged as a key approach for developing secure, reliable, and scalable software systems, and its importance is likely to continue growing in military and other high-stakes environments.

7. CONCLUSIONS

The chapter has reviewed the need for, and role system development models fulfill in systems engineering. The life cycle of systems has been described as consisting of six stages of concept, development, production, utilization, support, and retirement. The systems development process covers primarily the stages of concept, development, and production – although recent approaches seek greater integration between development and utilization (or operation). Development models can be broadly categorized as either plan-driven or agile. The plan-driven models are the more traditional systems engineering approaches and include the Waterfall and Vee models.

The Waterfall and Vee models predominantly move through the system development activities in a sequential fashion. Projects following these models start with planning and generally understanding all the system requirements prior to moving onto design activities. However, it has been long recognized that actual implementation of these models involves extensive iterations and recursion of activities.

Agile models have emerged primarily from the software engineering community. Software and its development differ in several important ways from hardware, which helps explain why agile methods become popular there first. Among these differences is that software is intangible and can be deployed over a network. Consequently, software is easy and relatively inexpensive to modify, even after the software is deployed to users. Moreover, the market for software is highly dynamic with changing needs and frequently introducing new technologies. The agile methods exploit the characteristics of software making it easy to change in order to thrive in such a dynamic business environment. Agile models are founded on principles of iterative and incremental development, continuous verification and validation, self-organized teams, continuous integration, and close interaction with users. The Spiral model was one of several other models that started to more formally and explicitly describe the systems development process as being iterative and incremental. Other novel agile development models emerged later, such as Scrum, SAFe, and DAD, which implement these principles.

The chapter has also provided a comparison between plan-driven and agile models along different dimensions to highlight their strengths and weaknesses. In general, plan-driven approaches are suitable for development projects where requirements are knowable, technology is mostly mature, and the environment is stable. They are also useful when audit trails and documentation are necessary such as for safety-critical systems or many defense systems. Agile models, on the contrary, are suitable in environments where there is high uncertainty and dynamism. The chapter observes that many projects might have characteristics suitable for plan-driven and other characteristics for agile models. In such cases, the chapter suggests a hybrid plan-driven and agile model attempting to combine the strengths of each process model. Such tailoring of process models is very important in systems engineering because there is no single model suitable for every project. Instead, informed and knowledgeable systems engineers should tailor or customize the system development models to fit the needs of the organization and the development project.

The chapter described emerging development trends affecting most system development projects such as DevOps. The DevOps concept interleaves development with operations using continuous feedback for the evolution of the system throughout its service life. DevOps as well as other emerging development trends can be incorporated into any of the system development models, and indeed the development models may need to be modified to better exploit these new capabilities.

REFERENCES

1. Ambler, S. W., & Lines, M. (2012). Disciplined agile delivery: A practitioner's guide to agile software delivery in the enterprise. IBM press.

2. Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., ... & Thomas, D. (2001). The agile manifesto.

3. Boehm, B. W. (1988). A spiral model of software development and enhancement. Computer, 21(5), 61-72.

4. B. Boehm, Spiral Development: Experience, Principles, and Refinements, CMU/SEI-2000-SR-008, July 2000.

5. Boehm, B., and Turner, R. (2004). Balancing agility and discipline: A guide for the perplexed. Addison-Wesley Professional.

6. Brady, J. (2001). Systems engineering and cost as an independent variable. Systems engineering, 4(4), 233-241.

7. Chrissis, M. B., Konrad, M., & Shrum, S. (2011). CMMI for development: guidelines for process integration and product improvement. Pearson Education.

8. De Weck, O. L., Roos, D., & Magee, C. L. (2011). Engineering systems: Meeting human needs in a complex technological world. Mit Press.

9. Dingsøyr, T., Falessi, D. and Power, K. (2019). "Agile development at scale: the next frontier". In: IEEE software 36.2 (2019), pages 30–38.

10. Douglass, B. P. (2015). Agile systems engineering. Morgan Kaufmann.

11. Dove, R., Lunney, K., Orosz, M., & Yokell, M. (2023, July). Agile Systems Engineering–Eight Core Aspects. In INCOSE International Symposium (Vol. 33, No. 1, pp. 823-837).

12. Elssamadisy, A., Abu-Tayeh, G., & Brown, T. (2018). Scaling agile and lean development in the enterprise. Pearson Education.

13. Evans, J. H. (1959). Basic design concepts. Journal of the American Society for Naval Engineers, 71(4), 671-678.

14. Forsberg and Mooz, 1998 à Forsberg, K. and Mooz, H. (1998), 7.17. System Engineering for Faster, Cheaper, Better. INCOSE International Symposium, 8: 917-927. https://doi.org/10.1002/j.2334-5837.1998.tb00130.x.

15. Gale, P.A., ''The ship design process,'' in Ship Des. Construction, vol. 1, T. Lamb, Ed. Alexandria, VA: SNAME, 2003, ch. 5.

16. Hanssen, K. G., Stålhane, T., & Myklebust, T. (2018). SafeScrum®–agile development of safety-critical software. Springer Nature Switzerland AG.

17. Henning, W.A., and Walter, D.T., (2005). Spiral development in action: a case study of spiral development in the Global Hawk Unmanned Aerial Vehicle program. Master's Thesis. Naval Postgraduate School, Monterey, CA.

18. Jim Highsmith and Alistair Cockburn. "Agile Software Development: The Business of Innovation". In: Computer 34.9 (2001), pages 120–127.

19. Ibarra H., and Scoular, A. (2019). "The Leader as Coach". In: Harvard Business Review (November - December 2019).

20. INCOSE (2023). SE Vision 2035: Engineering Solutions for a Better World. International Council on SE. Retrieved September 1, 2023 from https://www.incose.org/about-systems-engineering/se-vision-2035.

21. ISO/IEC/IEEE International Standard - Systems and software engineering -- System life cycle processes, 2015-05-15.

22. Kim, G., Humble, J., Debois, P., & Willis, J. (2016). The DevOps handbook: How to create world-class agility, reliability (2nd ed.). IT Revolution Press. https://itrevolution.com/book/the-devops-handbook/.

23. Knaster R. and Leffingwell, D. (2017). SAFe 4.0 distilled: applying the Scaled Agile Framework for lean software and systems engineering. Addison-Wesley Professional.

24. Larman, C., & Vodde, B. (2016). Large-scale scrum: More with LeSS. Addison-Wesley Professional.

25. Leffingwell, D.: Scaling Software Agility: Best Practices for Large Enterprises. Pearson Education, Boston (2007)

26. Lockey, S., Gillespie, N., Holm, D., & Someh, I. A. (2021). A review of trust in artificial intelligence: Challenges, vulnerabilities and future directions. Proceedings of the 54th Hawaii International Conference on System Sciences.

27. Lwakatare, L. E., Karvonen, T., Sauvola, T., Kuvaja, P., Olsson, H. H., Bosch, J., & Oivo, M. (2016, January). Towards DevOps in the embedded systems domain: Why is it so hard? In 2016 49th Hawaii international conference on system sciences (hicss), pp. 5437-5446.

28. Miller, A. W., Giachetti, R. E., & Van Bossuyt, D. L. (2022). Challenges of Adopting DevOps for the Combat Systems Development Environment, Defense Acquisition Research Journal, 29(1).

29. Paasivaara M. and Lassenius, C. (2019). "Empower your agile organization: Community- based decision making in large-scale agile development at Ericsson". In: IEEE Software 36.2 (2019), pages 64–69.

30. Pahl G. and Beitz, W. (2013). Pahl/Beitz Konstruktionslehre: Grundlagen erfolgreicher Produktentwicklung. Methoden und Anwendung. Springer-Verlag, 2013.

31. Putta, A., Paasivaara, M., & Lassenius, C. (2018). Benefits and challenges of adopting the scaled agile framework (SAFe): preliminary results from a multivocal literature review. In Product-Focused Software Process Improvement: 19th International Conference, PROFES 2018, Wolfsburg, Germany, November 28–30, 2018, Proceedings 19 (pp. 334-351). Springer International Publishing.

32. Scheithauer and Forsberg, 2013. à Scheithauer, D. and Forsberg, K. (2013), 4.5.3 V-Model Views. INCOSE International Symposium, 23: 502-516. https://doi.org/10.1002/j.2334-5837.2013.tb03035.x.

33. Stelzmann, E. (2012). Contextualizing agile systems engineering. IEEE Aerospace and Electronic Systems Magazine, 27(5), 17-22.

34. Schwaber 1997 à Schwaber, K. (1997). SCRUM Development Process. In: Sutherland, J., Casanave, C., Miller, J., Patel, P., Hollowell, G. (eds) Business Object Design and Implementation. Springer, London. https://doi.org/10.1007/978-1-4471-0947-1_11.

35. Thakurta, R., Mukhopadhyay, D., & Das, D. (2020). Applying agile practices in embedded system development: An industrial case study. Journal of Systems and Software, 168, 110660.

36. Ullman, D. G. (2019). Scrum for hardware design: supporting material for the mechanical design process.

37. Wade A Henning and Daniel T Walter. Spiral development in action: a case study of spiral development in the Global Hawk Unmanned Aerial Vehicle program. Technical report. Naval Postgraduate School, Monterey, CA, 2005.

38. Willett, K. D., Dove, R., Chudnow, A., Eckman, R., Rosser, L., Stevens, J. S., ... and Yokell, M. (2021, July). Agility in the Future of Systems Engineering (FuSE)-A Roadmap of Foundational Concepts. In INCOSE International Symposium (Vol. 31, No. 1, pp. 158-174).

39. Yang, J., Lappas, T., Egan, M., & Bagaria, N. (2019). Applying Agile methods to hardware product development. Journal of Systems and Software, 154, 1-14.

40. Zaeske, W., & Durak, U. (2022). DevOps for Airborne Software: Exploring Modern Approaches. Springer Nature.

BIOGRAPHIES

DR. RONALD GIACHETTI

Dr. Ronald Giachetti is a Professor of Systems Engineering at the Naval Postgraduate School (NPS) in Monterey, CA. He teaches and conducts research in model-based systems engineering, system architecting, and system of systems engineering. He has 30 years of engineering experience in academia and industry. He has published extensively with over 50 technical papers including a textbook on the design of enterprise systems. He has lectured internationally in Europe, Latin America, and Asia. He has held multiple leadership positions including Chair of the Systems Engineering Department, Dean of the Graduate School of Engineering at NPS, and Chair of the Corporate Advisory Board (CAB) for the International Council on Systems Engineering (INCOSE). He holds engineering degrees from Rensselaer Polytechnic Institute, New York University – Polytechnic, and North Carolina State University. Outside of engineering, he is an avid sailor and races in local and national regattas.

JUAN CARLOS LARIO MONJE

Juan Carlos Lario Monje is an aeronautical engineer with a multidisciplinary experience in several engineering fields such as safety, quality, programs, offsets, contracts, definition & handling of requirements, integration, V&V of systems and Certification & Qualification. He has acquired a global perspective on aeronautical international programs (civil/military), since its development phase to later serial production and final entry into service, with specific involvement on aerodynamics, Electronic Warfare and Electro-Optical systems. With direct participation in the main European aeronautical programs, both civil (A380, A350) & military (A400M, EF2000, Tiger-HAD, NH90), he has reached a profound knowledge of Spanish aeronautical Industry core businesses (Airbus, Indra), defence programs end-users (SP Army, Air Force) and European Contract agencies (NETMA, OCCAR). With and endless curiosity, he is an enthusiastic professional of continuous learning and teaching. CSEP certified by INCOSE.

"All models are wrong, some are useful."

G.E.P. Box.

CHAPTER 4

Model-Based Systems Engineering

David Long, *Blue Holon (david.long@incose.net)*
Dr. Kaitlin Henderson, *Radiance Technologies (kaitlin.henderson@radiancetech.com)*
Belinda Misiego, *Isdefe (bmisiego@isdefe.es)*

Abstract

This chapter presents Model-Based Systems Engineering (MBSE) as an evolution of the discipline that leverages the power of computer models. It identifies the main elements that MBSE should address to be successful and demystifies some common misconceptions that lead to weak and/or poor practices in organizations. After presenting some unprecedented capabilities by adopting MBSE approaches, the chapter provides some guidance to facilitate the successful adoption of MBSE.

Main topics include the effects of formalizing systems engineering, divergence and convergence of semantics, and authoring, review, and control of configuration in model-based environments.

Keywords

Model-Based Systems Engineering (MBSE), ontology, metamodel, Systems Modeling Language (SysML).

1. INTRODUCTION

Model-Based Systems Engineering (MBSE), Model-Based Engineering (MBE), Digital Thread and Digital Twin (DT), Digital Engineering (DE), and Digital Transformation (yet another DT). There is an explosion of concepts and the corresponding acronym soup as the power of digital is applied to systems engineering and the greater engineering lifecycle. This chapter will try to clarify the differences between the various concepts and how they interrelate. It will move beyond the marketing, myth, and misconception to a practical understanding of what digital transformation means for systems engineering, the fundamentals needed to be known, and the expected value to be achieved.

Before diving into discussions about what MBSE is, let us look at the environmental context from which it has emerged. In recent decades, the development and capability of products have evolved. First, base products built on electro-mechanical technologies have largely moved to smart products leveraging electronics and software to bring new capabilities and an enhanced user experience. Next, sensors and networks have been introduced to create smart, connected products with capabilities such as knowing exactly when the next train will arrive and how many seats remain available. After, these products needed to be more and more connected until becoming a product system, with all the elements coordinated. Today, traditional and intelligent systems are collaborating as systems of systems (SoS) to meet the needs of society (e.g., airport operations in which the flight information system, the transportation system, the operational safety system, and all the other systems involved collaborate in a coordinated and connected manner).

In the days of electro-mechanical products, the interactions between parts, components, and systems were somewhat limited. Given this low coupling between components and between the design teams creating them, documents represented an appropriate solution for capturing architecture and design data. In a document-based world, data is dispersed throughout a multitude of documents. However, as new technologies are introduced and systems become larger and more complicated, the number and complexity of interfaces increases dramatically. Data dispersed between artifacts could generate inconsistencies between them. Plus, keeping hundreds of documents updated for a complex system can be an arduous task. Design inconsistencies may occur due to interpretation discrepancies of the information from use of approaches, language, or diagrams without common semantics between stakeholders. Discovering inconsistencies in the design (and therefore significant rework) can lead to potential delays and extra costs in development, and undiscovered inconsistencies can result in system failure [1].

Looking at the rapidly evolving technological progress in the world, it is fair to say that the needs from engineering design processes have exceeded what the capabilities of traditional systems engineering can provide. In fact, in 2014 INCOSE published the "System Engineering Vision 2025" [2], with a statement of where the industry had to move to in order to solve society needs. The publication highlighted a number of challenges, including:

- Mission complexity is growing faster than our ability to manage it.
- System design emerges from pieces, rather than from architecture.
- Knowledge and investment are lost at project life cycle phase boundaries.
- Knowledge and investment are lost between projects.

What worked for electromechanical systems in the 1950s, 1960s, and 1970s is not sufficient to address today's needs and technologies. Systems engineers started to make the leap to model-based approaches to respond with agility and efficiency to the complex and changing world.

In 2021, INCOSE published the "System Engineering Vision 2035" [3]. One of the headlines of this publication is "The future of systems engineering is predominantly Model-Based." By 2035, it is expected that a family of unified, integrated MBSE-Systems Modeling and Simulation (SMS) frameworks will exist.

But what is MBSE? The challenge is the meaning of MBSE is very muddy, which traces to the breadth and ambiguity of what constitutes a model. One technical definition of model is "A physical, mathematical, or otherwise logical representation of a system, entity, phenomenon, or process" [4]. A broader definition of model is "a graphical, mathematical (symbolic), physical, or verbal representation or simplified version of a concept, phenomenon, relationship, structure, system, or an aspect of the real world" . A computational fluid dynamics representation of air flowing over a wing satisfies these definitions, but so does a drawing of a process flow and a traditional interface control document. Because of the breadth of what constitutes a model, different practitioners will have very different interpretations of what MBSE is. Most people will cite INCOSE's definition of MBSE, which is "the formalized application of modeling to support the activities of systems requirements, design, analysis, verification and validation starting in the design phase conceptual and continuing in the development stage and subsequent stages of the life cycle," but the ambiguity around "model" and "modeling" remains.

Simply put, MBSE represents a new way of performing systems engineering; one that uses model-based techniques to perform systems engineering tasks instead of traditional document-based ones [5]. It is about moving from capturing data in natural language documents to richer representations that reflect and communicate the phenomenology, architecture, and design of systems, ideally in a machine-readable form. The central aspect is the system model, from which all other artifacts are derived. It serves as the connective tissue that binds together the greater digital enterprise that enables engineering. If MBSE were to emerge today, it would have been most likely referred to it as digital systems engineering.

MBSE should make system descriptive and analytical models explicit, coherent, consistent, and actionable by both humans and computers. It should reflect an evolution from low-fidelity representations in documents to higher-fidelity, richer representations that machines can read and interpret. It should improve granularity of knowledge capture for knowledge management, analysis, and learning. It should enable one descriptive architectural model connecting multiple analytical models, which together represent design with the requisite degree of rigor for the problem and solution at hand.

MBSE should also leverage models for communication and analysis, represent "authoritative data" for system design and specifications, ensure consistent design and specifications, and provide an explicit system model to engineering teams. In short, MBSE should be an evolution, not a revolution, in thinking and approach leveraging modern technologies and capabilities to better represent data, information, and knowledge. While evolutionary, this change offers transformative results. MBSE is expected to improve system quality, reduce costs, shorten development times, integrate new technologies and give digital continuity with manufacturing and operations.

2. NECESSARY (AND IDEAL) ELEMENTS OF MBSE

There is a fundamental difference between models in systems engineering and model-based systems engineering. Engineers have always used models to understand and reason. The output of engineering has always been some type of model that helps analyze and advance the understanding of a problem and solution. MBSE leverages modern techniques to capture and represent the fundamental information required to engineer a system. It has traditionally been more on the descriptive and architectural side, although not inherently limited to them, extending from a first expression of need through the whole system life cycle. These models help elicit, capture, and represent a system so that those computational techniques (analytical models, modeling & simulation, etc.) traditionally applied can continue to be used.

In the world of engineering design, architectural models connect the idea behind a design solution with its implementation as a real system. They are the way in which the current working team is aligned, communicate with a higher awareness to minimize misunderstandings, and capture knowledge over time. These models attempt to represent the entities of the engineering problem and their relationship to each other and connect them to the proposed solution or existing mechanism that addresses the problem. If one can properly characterize functionally (i.e., what the system does) and physically each part of the system and its interfaces, as well as the interactions and exchanges between them, then it becomes possible, at the right degree of fidelity and precision, to pass a detailed component description down to a subject matter expert to finish out the design in parallel. The model used in this way is the centerpiece of MBSE.

Four elements are critical to a model [6]:

- **Language:** The modeling language enables the clear expression and representation of the model, so that understanding and insight can arise. The language must be clear and unambiguous to depict the model accurately and understandably.

- **Structure:** Structure allows the model to capture system behavior by clearly describing the relationship between system's entities.

- **Argumentation:** A model must be capable of making the critical "argument" that the system fulfills the stakeholder's needs. The model must represent the system design in such a way that the design team can demonstrate that the system accomplishes the purposes for which it is designed.

- **Presentation:** A model must include some mechanism to present the argument in a way that can be seen and understood by the users, which may include engineers, customers, and other stakeholders.

These elements can be defined by using metamodels, semantics, and ontologies. Metamodels, semantics, and ontologies build the underlying formalities that make so many of the key benefits of MBSE possible (e.g., collaboration, modeling sharing, reuse, reduction of ambiguity, etc.). A metamodel defines the syntax, constraints, and patterns that make up the modeling language that is used when creating a model. An ontology can be thought of as a type of metamodel that defines a common set of terminology, relationships, and context for a given domain, incorporating precise semantics to the terms and relationships.

Figure 1 shows an example of part of a metamodel for systems engineering. It captures (and constrains) the different concepts that the systems engineer may use to support their work, aligning the understanding and interpretability that every member of the team has. For example, an engineer could not distinguish between a requirement and a specification, as the metamodel does not offer different concepts for those two terms. At the same time, the metamodel does not allow components to directly exchange inputs and outputs but forces the engineer to think about the functions that the component performs to think about inputs and outputs. The impact of this is not only one of fostering understanding and guaranteeing certain good practices when reasoning about systems engineering information, but it also allows to construct machine-readable models to enable better accessing and processing such information. In this metamodel example, one could see how an engineer could query the underlying model to identify all verification requirements related to a function (through a relational path through requirements, components, and finally functions), or enforce rules to guarantee that every component must execute a function, or every function must have at least one input and one output to be consistent with systems theory.

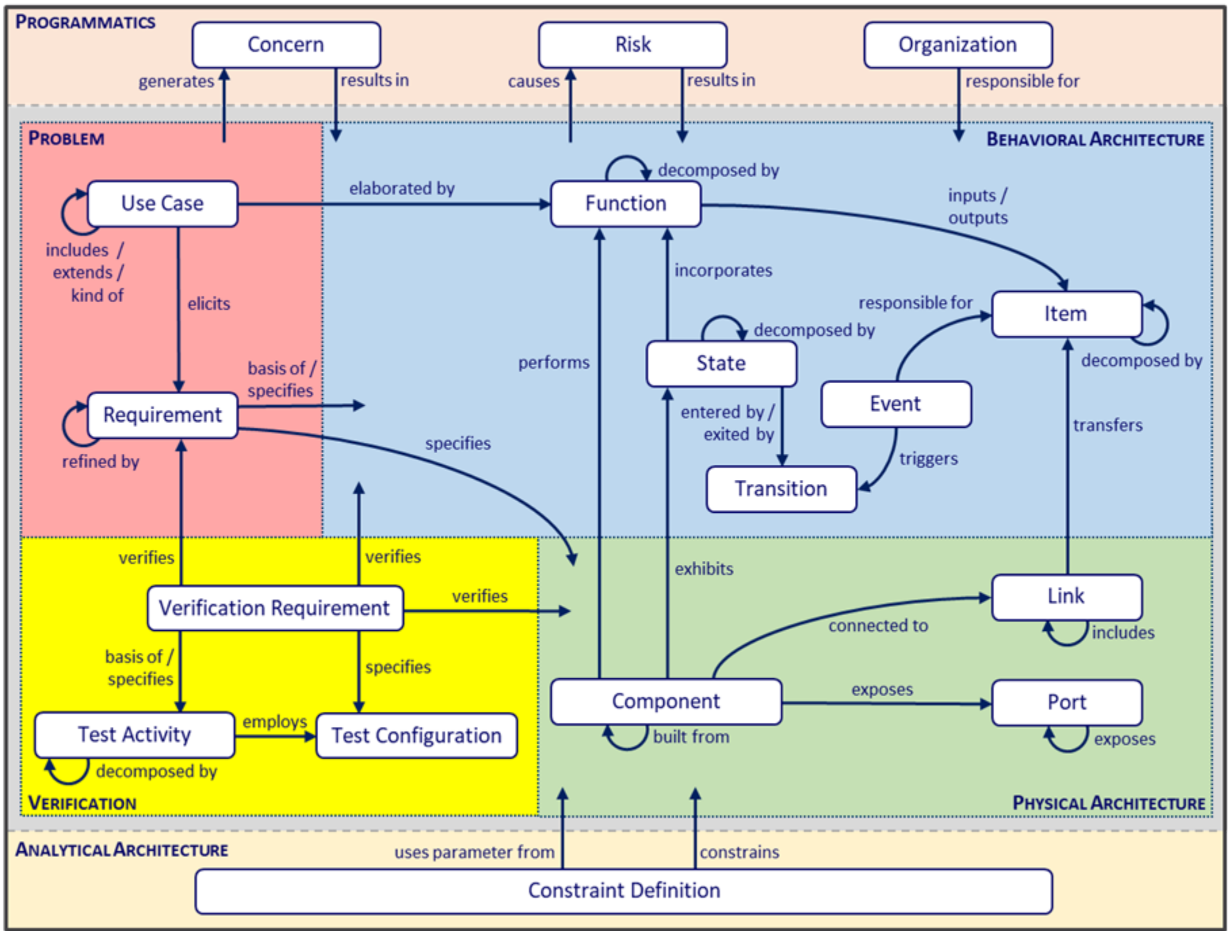


Figure 1. Example of a systems engineering metamodel

Adding precise semantics to a metamodel enables the use of axioms, which afford reasoning capabilities. Whereas a regular metamodel can only guarantee syntactic correctness, an engineer can leverage the semantics of a model to infer aspects of the model that stem from the meaning embedded in the model. For example, whereas a regular metamodel can only check that every Component executes a Function, an ontology affords the possibility of checking that a specific instance of a Function actually represents a function. Imagine that an engineer defines *snake* as a Component and *fly* as a function, and creates the relationship that the *snake* performs the *fly* function. This is syntactically correct, and a regular metamodel would consider it valid. However, an ontology could identify that the model is not sensible because snakes cannot fly.

There is no unique design for metamodels or ontologies; different organizations may prefer different ones to better accommodate their specific context and needs. However, the metamodel and/or ontology should not be arbitrary. Concepts, relationships, semantics, and axioms must be meaningful, internally consistent, and ideally consistent with systems engineering theory and principles, if one would want to benefit from its structure and machine-readability (if implemented as such).

George Box famously said “All models are wrong, but some are useful. The question is how wrong a model can be and still be useful.” Whenever engineers deal with models, they must be deliberate about defining and understanding the purpose behind that model so that the model is fit for purpose in its type, scope, and level of fidelity. MBSE can be valid at any point along the system life cycle, as long as the investment of effort into the model is aligned with the appropriate purpose. For example, if the model is being used in the front end of the lifecycle, then a high-fidelity representation is highly unlikely to be necessary. In place of precision, it may be better to seek elicitation: a better understanding and alignment with the stakeholders to elicit the real needs at the architectural phase. Whether it is fresh design or reengineering, early life cycle, or late life cycle, if the purpose is identified and kept in the forefront, then MBSE can deliver value. If it is not, it is easy to fall into the trap of modeling for the sake of modeling, which defeats the purpose of MBSE. The usefulness of a model is clearly tied to understanding its intended purpose.

The implementation of MBSE requires two pillars in addition to the modeling language, the tool that enables the creation and visualization of the models and the method with which MBSE is implemented [5].

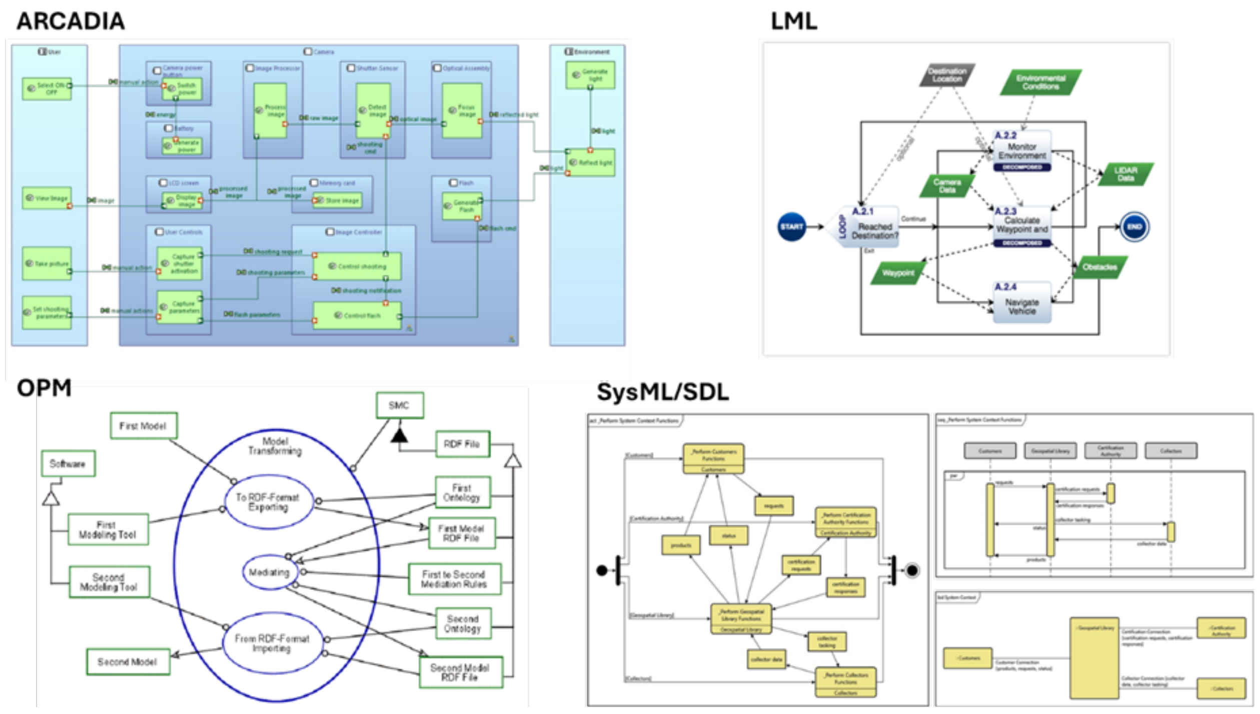


Figure 2. Examples of system models created in different modeling languages

The Systems Modeling Language (SysML) is the dominant language associated with MBSE, but it is important to note that SysML does not equal MBSE. In fact, SysML is not the only language, nor is it always the right language. SysML was developed as a profile of the Unified Modeling Language (UML) developed by the software engineering community to help close the communication gap between system and software engineers during the rise of software-intensive systems. In this sense, SysML is a general-purpose graphical language that is intended to model systems, not necessarily to model all aspects of systems engineering, despite being abused in this sense by the practicing community. Other modeling languages include the Lifecycle Modeling Language (LML), the Object-Process Methodology (OPM), and various languages tied to specific tools such as Capella/Arcadia and Vitech Corporation's Systems Definition Language (SDL). Some graphical representations are shown as examples in Figure 2.

Modeling tools are a special class of tools that are designed and implemented to comply with the rules of one or more modeling languages and enable users to build well-formed models in these languages. Good MBSE tools are far more than diagramming tools. The diagrams are not the model itself; they are merely views of the underlying model which contains a set of elements and relationships that are shown in the diagrams. Much as a computer-aided design (CAD) tool can present top, front, and side views of the geometric model of a part, MBSE tools produce a variety of visualizations of the underlying system model. Some examples of MBSE tools include Cameo Systems Modeler™ (Dassault Systemes), Capella™ (Eclipse open source), Enterprise Architect™ (Sparx Systems), GENESYS™ (Vitech Corporation), Innoslate™ (SPEC Innovations), and Rhapsody™ (IBM). Examples of some of their user interfaces are shown in Figure 3.

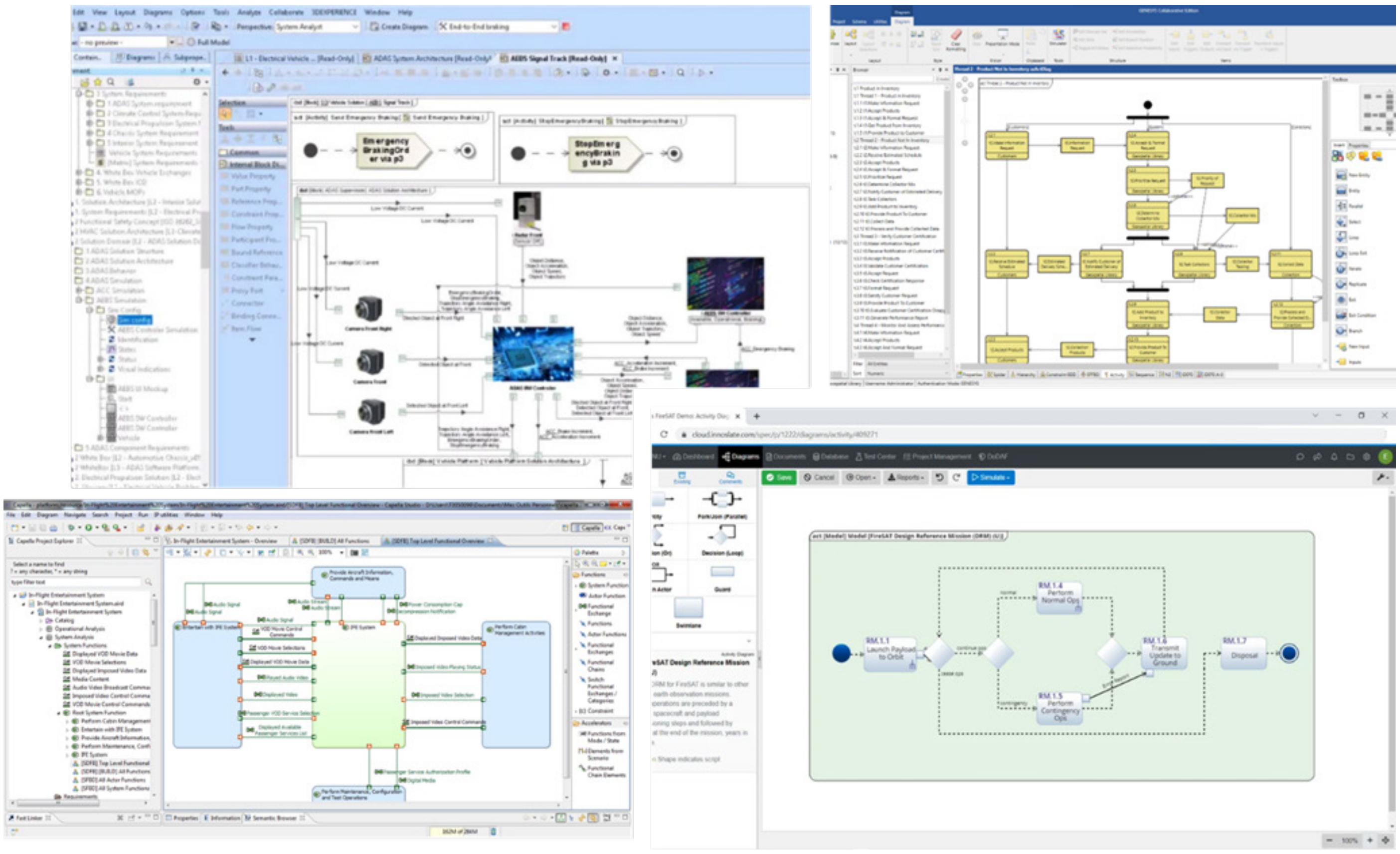

Emergency BrakingOrder via p3
StepEmergencyBraking via p3
RM.1.1 Launch Payload to Orbit
RM.1.4 Perform Normal Ops
RM.1.5 Perform Contingency Ops
RM.1.6 Transmit Update to Ground
RM.1.7 Disposal

Figure 3. Examples of the user interfaces of different MBSE tools

Modeling methods and/or methodologies can be seen as a roadmap with a set of tasks that ensures the entire team builds the model consistently and works towards a common purpose. A method helps to define the scope; how deep and broad the approach should be towards the model (also based on the opinion of the development teams and the time available). Some general methodologies include INCOSE Object-Oriented Systems Engineering Method (OOSEM), Systems Modeling Toolbox (SYSMOD), Object-Process Methodology (OPM), and the Integrated Systems Engineering and Pipelines of Processes in Object-Oriented Architectures (ISE&PPOOA). It should be noted that these methodologies, while developed with some level of generality in mind, may not fit the needs of every organization. In fact, the adoption of a particular MBSE methodology requires adopting a specific approach to systems engineering. Therefore, ad-hoc methodologies can however be implemented by an organization to better align their implementation of MBSE with their working processes and specific adoption of systems engineering practices. Furthermore, because of this strong connection between MBSE methodologies and systems engineering practices, consistency with systems engineering standards may constraint the implementation of MBSE or vice versa. The adoption of a specific MBSE methodology may require the change and/or tailoring of existing standards.

Finally, people are central to implementing MBSE. When putting together a team to perform MBSE, the team must collectively exhibit expertise in systems engineering, the modeling languages of choice, and the modeling tools of choice, besides any other specific expertise required to complete the project at hand (e.g., different analytical methods, application domain, etc.). Some teams opt to split the expertise between different team members, while some other teams opt to guarantee that all team members have expertise in the three areas. There are advantages and disadvantages to each approach (e.g., splitting expertise accelerates the learning curve but is fragile and less scalable), but explaining them is outside of the scope of this chapter. It is not known where practitioners and professionals will be 15 years from now, but the state of practice and the evolution of MBSE is very fluid and volatile at the moment. Therefore, it seems fair to suggest right now looking for a team that has the characteristics fit for purpose given the business need and context of application.

3. MODELS ARE MORE THAN JUST DRAWINGS

When someone unfamiliar with MBSE looks at an MBSE tool, they will see a collection of drawings and wonder how this is different from using something like Microsoft PowerPoint or Microsoft Visio. The power behind MBSE comes from the underlying data structures, syntax, languages, etc. that make up the actual model (ref. Figure 4). What a diagram shows is considered a representation or view of the system model, as explained earlier. The diagraming capability of MBSE is a great tool to communicate with different stakeholders, but what makes it actually model-based is the connectivity between different components that are unambiguously represented due to the defined syntax and semantics of the modeling language.

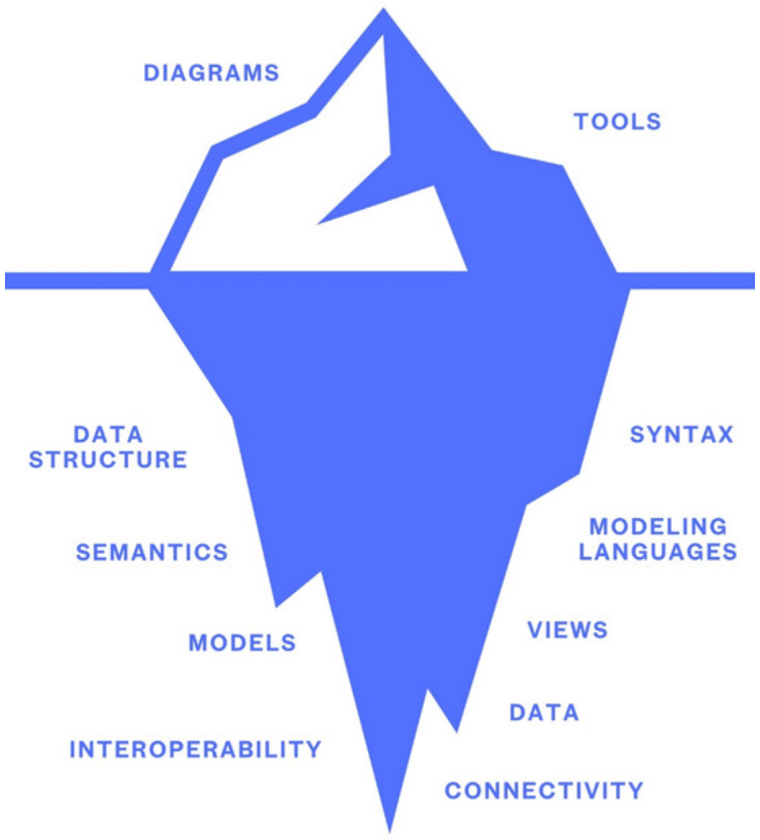


Figure 4. Iceberg model of what a user sees in a diagram versus the information contained in the model

A classic systems engineer will often state that a whiteboard is one of their greatest tools. They can go to the whiteboard, begin a sketch, draw bubbles and clouds and arrows, and label the components. This is useful at a high level and is very good for human-to-human alignment. But then, as the development progresses towards a more detailed design, that flexibility in language and symbology becomes a hindrance because greater precision is needed. An agreed upon meaning of symbols and terminology is necessary to avoid miscommunications and misalignments. There needs to be rigor in the language to ensure effective communication and reasoning by both humans and computers. Flexibility may be useful to understand concepts in the problem space, but if reusing components is desired, and eventually get to connecting models with other tools, there needs to be precision and rigor.

Discussing the precision and consistency of defined terminologies requires coming back to ontologies. Establishing an ontology is fundamentally saying that words, the interrelationships between the words, the concepts that they embody, and the context within which they are valid are defined. The ontology is the underlying knowledge architecture that enables capturing individual pieces of data unambiguously and reflect the interrelationships to represent information and knowledge. However, not all language models are built upon ontologies and/or the modeling language may hide meaning in its constructs that is unknown to the team members. An example from a research study that explored this differentiation between drawing and model follows.

A number of SysML experts were asked to evaluate the behavior of a system (specifically a car) given a system model and a starting condition [7]. The behavioral model was represented as a SysML state machine diagram, which is shown in Figure 4. Participants were asked to describe the behavior of the car when it was in the *Braking* state and the conditions *releaseBrake* and *speed = 0* occurred simultaneously. They were offered multiple choices to answer: the car will not experience those two conditions simultaneously when in that state, the situation is outside of the scope of the model, the system automatically defaults to one of the transitions, or this is a non-nominal situation that is not captured by the model. The responses were uniformly distributed; in other words, there was no agreement between the experts. The study showed that, while the experts could read the same diagram, their understanding of what the model conveyed was different, and this was due to a lack of understanding of what the modeling constructs embedded in the model beyond the graphical representation of some of its aspects.

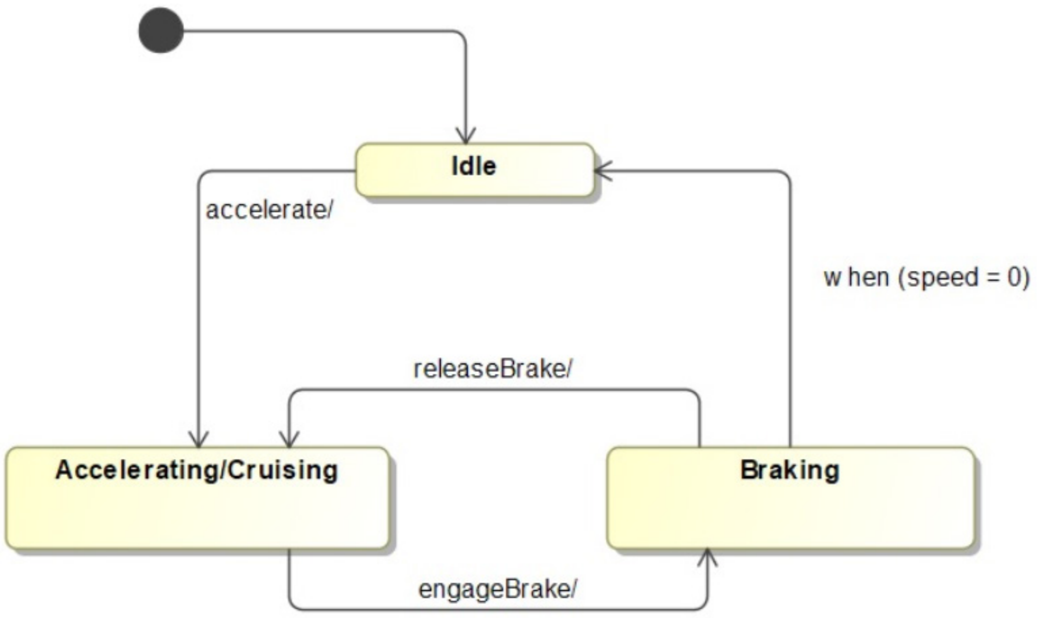


Figure 5. Behavioral model of the car system using a SysML state machine diagram [reproduced from [7]]

One of the challenges of ontologies is that there are multiple ontologies in play at the highest level of the system encapsulation. When the focus is on how systems engineering is performed and characterizing the resulting system of interest, there is a rather small core ontology (a general systems ontology) that enables to capture, communicate, and reason about concepts such as requirements, functions, exchanges, components, interfaces, and interrelationships between those concepts (e.g., satisfying a requirement with a function or allocating a function to a component). As the analytical and the phenomenology necessary to develop the system of interest are brought in, the scope of the required ontology explodes. The phenomenology for an aircraft, for an insulin pump, or for an IT system are very different, therefore, the language around them (the domain specific ontology) is different. The path to success in MBSE may lie in picking a base systems ontology that addresses the scope and language of the team, then specializing it to include key engineering concepts and concepts that align with the organization's methods.

A basic understanding of ontologies better justifies why models are more than just drawings. Returning to the analogy of CAD, the underlying ontology of geometry is points and vectors. If the canopy of a plane is shifted back by 50 centimeters, this implies that the underlying points and vectors in the data model are also being changed. The top and front views presented by CAD would then adjust to show the new position of the canopy. Similarly, in MBSE, if a component is added to the compositional diagram of a system, this is more than adding a box on a drawing. It is being specified that that system has a new subcomponent. Any representation of the system's physical architecture would then reflect this new subcomponent.

4. MBSE IS NOT A SILVER BULLET; GOOD SYSTEMS ENGINEERING IS A PRE-REQUISITE FOR GOOD MBSE

Despite its inclusion in the name, MBSE should not be about modeling. MBSE should be about doing systems engineering while properly leveraging models and a model-based approach. Without a proper understanding of the fundamental principles, processes, and methods for systems engineering, it is possible to spend a great amount of time and money modeling, but it will not be value added. Unfortunately, this is a common mistake that organizations make when adopting MBSE: purchasing several tool licenses and offering their engineers short training courses on the tool and/or modeling language, without guaranteeing a strong underlying expertise in systems engineering. This is exacerbated by a growth in professional modelers (expertise in modeling is relatively easy to build) at the expense of poor systems engineering practices (gaining expertise in systems engineering is hard).

The key to embracing MBSE is to focus on systems engineering and recognize that "model-based" is largely using computer-aided techniques to better execute what was previously done in a document-based environment. MBSE is not a point of departure for systems engineering, but rather an evolution to keep pace with and leverage the rapidly changing technological landscape. In fact, technology will continue to evolve, enabling engineers to better represent, communicate, and analyze data tomorrow than they can today. What is being today called MBSE is the beginning of a continuing evolutionary journey for systems engineering.

It is also important to recognize that technology is not always a blessing. A common trap of MBSE is that it may entice an engineer to be prematurely precise, particularly when heavily relying on professional modelers with little systems engineering expertise, which comes in at the cost of too much effort and over constraining the design envelope early in the design journey. Good SE practices and engineering judgement must be front of mind when progressing through the system lifecycle moving from higher levels of abstraction to more detailed understanding. The level of accuracy to be achieved should be driven by the purpose of the modeling effort. The same applies to precision. Determining the level of accuracy and precision that is necessary comes back to sound SE principles, engineering judgement, and knowing the purpose behind the modeling effort.

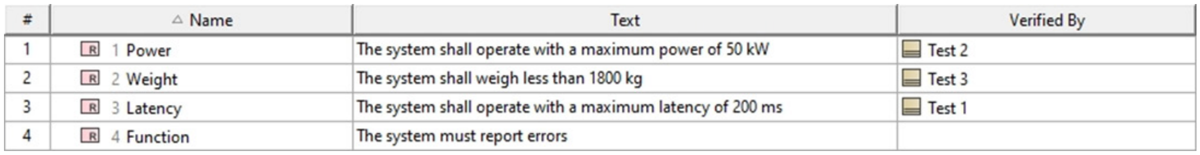
5. NOVEL CAPABILITIES ENABLED BY MBSE

There are numerous claims of benefits MBSE provides across literature sources and from observations made by practitioners. Some of these benefits include better communication, improved consistency, reduced cost, reduced time, reduced errors, and improved system understanding [8]. A key strength of MBSE lies in the ability to clarify communication and shared understanding across the team. It is the ability to better capture information over time and free it from so called drift. Recollections and understanding change, so formally capturing the system provides a more accurate, more precise representation over time. A clarified representation highlights where there are gaps in thinking or inconsistent understanding across the team. A higher fidelity descriptive architectural representation helps align the team – clarifying understanding, exposing assumptions, and reflecting the design journey behind the current solution. At some point in implementation there will be a question about a design decision, or a new technology, or a requirements change, so capturing the design journey is as critical as the resulting design.

Many of the benefits cited above stem from MBSE establishing an Authoritative Source of Truth (ASOT) for systems [9]. Essentially, this is formalizing that "connective tissue" discussed earlier. In the early days of MBSE, there was a vision of MBSE being established as the single source of truth; a central element that guaranteed all data used in a project were consistent. In essence, data lived in one location and other models would point to that location. In this sense, MBSE helped propagate data changes throughout all models that relied on such data. However, a single source of truth is just one solution to the consistency and relevance issue, and comes with some drawbacks, particularly in terms of vulnerability and efficiency. Lately, the concept has morphed into the more general ASOT and authoritative data, where the key is not on the *uniqueness* of the location but on the *certification* of the data being used and/or sourced. MBSE in this sense enables the identification and tagging of data used across the modeling environment.

This idea of an ASOT is the enabler for Digital Engineering, Digital Thread, Digital Twin, and all related digital transformation artifacts. If the systems engineering team is the technical connective tissue that binds together the project team, MBSE is the digital connective tissue that enables Digital Engineering, which enables creating a Digital Thread, which allows for development of a Digital Twin. Particularly, one could conceive MBSE as the subset of DE that allows all the disparate domains involved in the engineering process to work collectively with that authoritative source of data [8].

Beyond these benefits, MBSE done well enables novel capabilities that accelerate and advance the greater systems lifecycle. At the heart of each of these is representing data and knowledge in a structure and manner that is computable by both human and machine. Freeing knowledge from documents and artifacts, MBSE enables better alignment across the enterprise providing the right data to the right place at the right time at the right level of abstraction in full context presented properly for the consumer to better understand, analyze, and decide. Contrast this with traditional methods that represent one set of data (likely both missing information and including superfluous data for the decision at hand) in a single presentation and lacking context. This dynamic query and presentation of information on demand ensures consistency with the underlying model [10-12]. Properly coupled with strong visualizations including documents, diagrams, tables, and modern options such as dynamic animations and Unreal gaming engines, MBSE unlocks the power of multiple perspectives from engineer to operator to subject matter expert throughout the lifecycle (ref. Figure 6).



#	△ Name	Text	Verified By
1	R 1 Power	The system shall operate with a maximum power of 50 kW	Test 2
2	R 2 Weight	The system shall weigh less than 1800 kg	Test 3
3	R 3 Latency	The system shall operate with a maximum latency of 200 ms	Test 1
4	R 4 Function	The system must report errors	

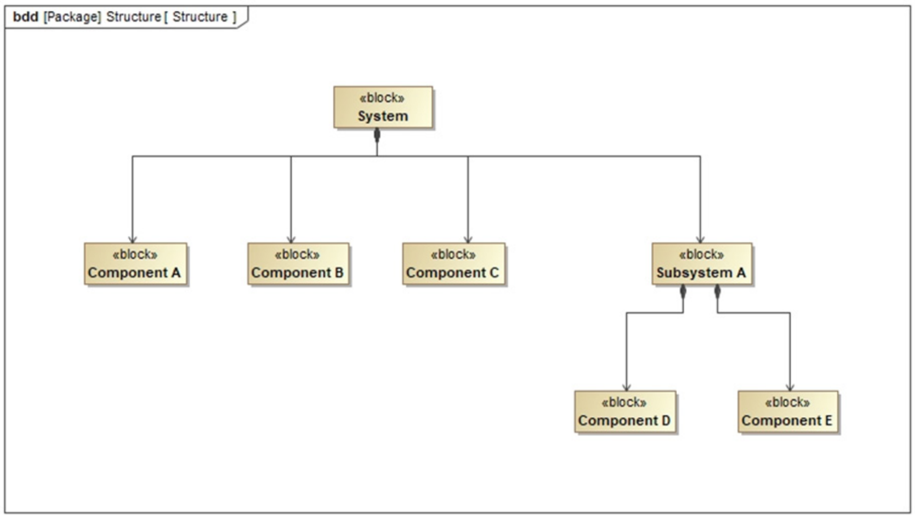
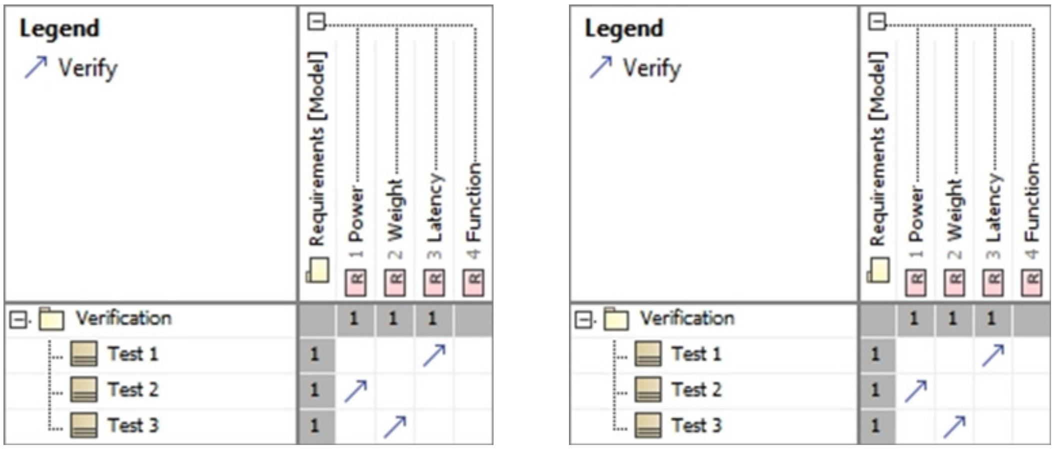


Figure 6. Example of querying a model

Done well, MBSE reflects not only the individual bits of data but also the relationships and dependencies between them helping to move from data to information to knowledge representation. This knowledge map enables classic traceability analysis, such as checking that all requirements are satisfied in the solution and that all functions are allocated to the physical architecture (ref. Figure 7). Moreover, it supports the rapid evaluation of change enabling the engineering team to trace the impact of a proposed change (be that a new requirement or a new component), identify the affected aspects of the solution architecture, and evaluate the change in the context of previous design decisions (ref. Figure 8). While humans are still responsible for the engineering and analysis, the scoping and context provided by the knowledge map improves the quality and accelerates the analysis.


Legend
↗ Verify

	Requirements [Model]	1 Power	2 Weight	3 Latency	4 Function
Verification		1	1	1	
Test 1	1			↗	
Test 2	1	↗			
Test 3	1		↗		

Legend
↗ Verify

	Requirements [Model]	1 Power	2 Weight	3 Latency	4 Function
Verification		1	1	1	
Test 1	1			↗	
Test 2	1	↗			
Test 3	1		↗		

Figure 7. Examples of traceability analysis visualization

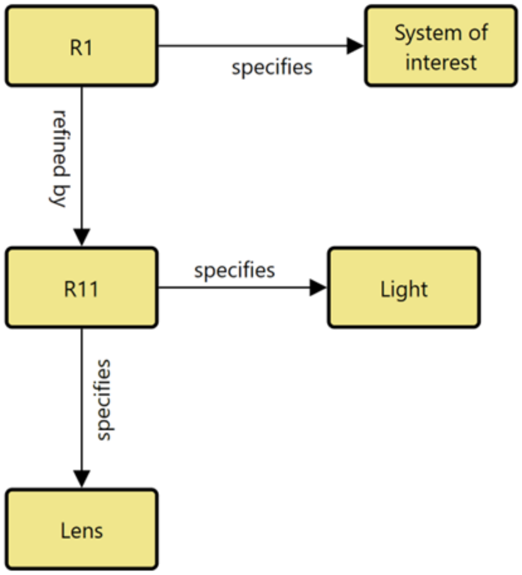


Figure 8. Example of a change propagation analysis visualization. (The diagram is automatically generated when requirement R1 is flagged because of change. It identifies all elements in the model that are associated with such a requirement.)

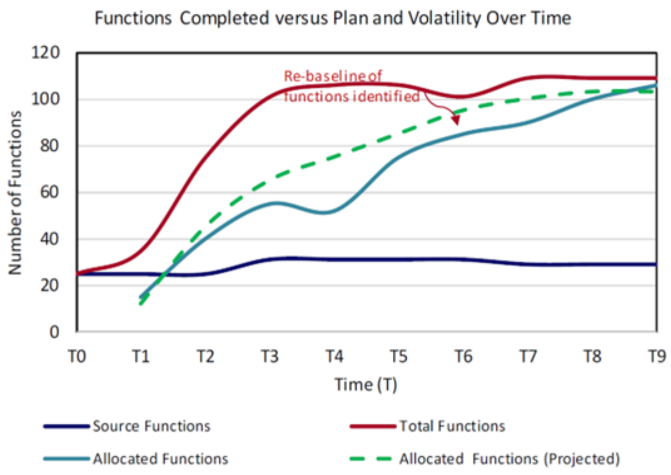


Figure 9. Example of completeness metrics visualization

This knowledge map generated through MBSE can be analyzed through computing techniques to identify completeness and design integrity issues. For example, traditional completeness checks such as ensuring requirements trace to solution elements, requirements are verified, and functions are allocated can be performed consistently and rapidly generating metrics to reflect the maturity of the design (ref. Figure 9). More sophisticated checks can identify design integrity issues such as accounting for all inputs, outputs, and interfaces during decomposition or ensuring interfaces of the right type transfer exchanges between components (ref. Figure 10). Moving beyond traditional computing techniques, machine learning algorithms could be applied to identify patterns and suggest design alternatives.

Best of all, the system model represents a virtual system prototype from day one, albeit at a high level of abstraction early in the project. The model can be dynamically simulated to identify issues and confirm system performance. As the system model is refined and analytic models are coupled with the descriptive architectural model, the level of detail and precision increases. This enables continuous evaluation and verification of the design, accelerating defect detection and enabling the engineering team to rapidly conduct trade studies.

These examples are not exhaustive, but they should give a glimpse of the state of the possible when systems engineering is moved into a modeling environment.

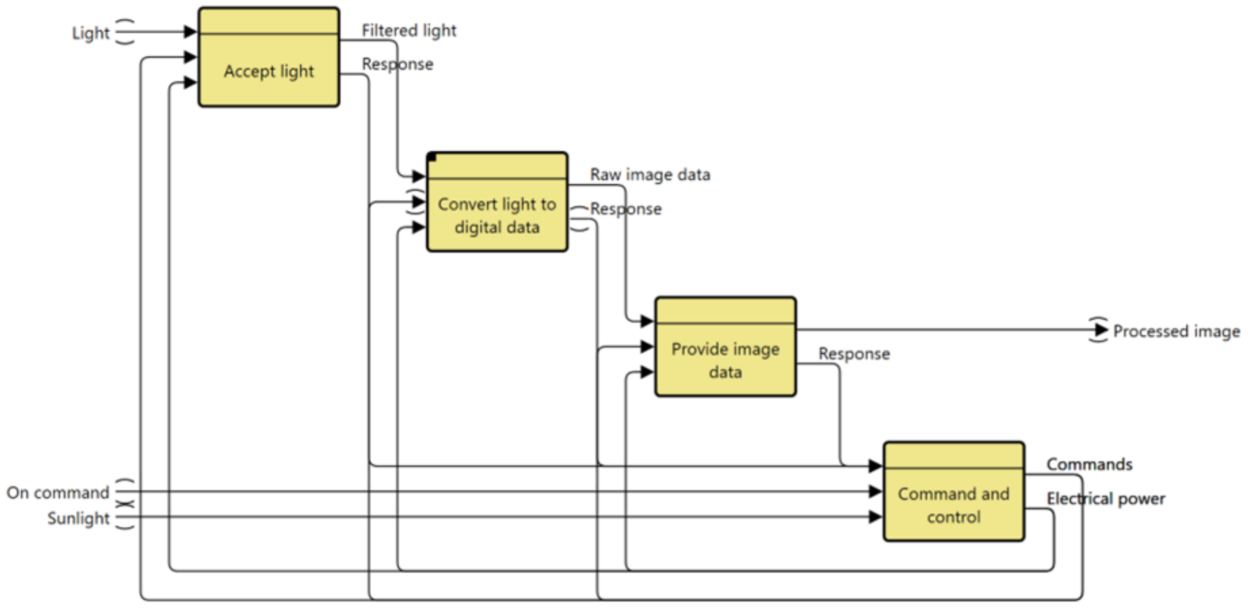


Figure 10. Example of automated design integrity check visualization (Note: Parentheses next to a signal on the sides of the diagram mean that such a signal is missing at the higher level of encapsulation; Parentheses on a signal next to a box indicate that such a signal is not allocated on the lower level of encapsulation.)

Digital Engineering: *an integrated digital approach that uses authoritative sources of systems' data and models as a continuum across disciplines to support lifecycle activities from concept to disposal.*

Digital Thread: *the communication framework that allows connected data flow and integrated view of an asset's data throughout its life cycle across traditionally siloed functional perspective.*

Digital Twin: *a computational model of a particular physical system with bidirectional communication with its physical counterpart. A digital twin coevolves with the physical system and reflects the state, status, and history of the system.*

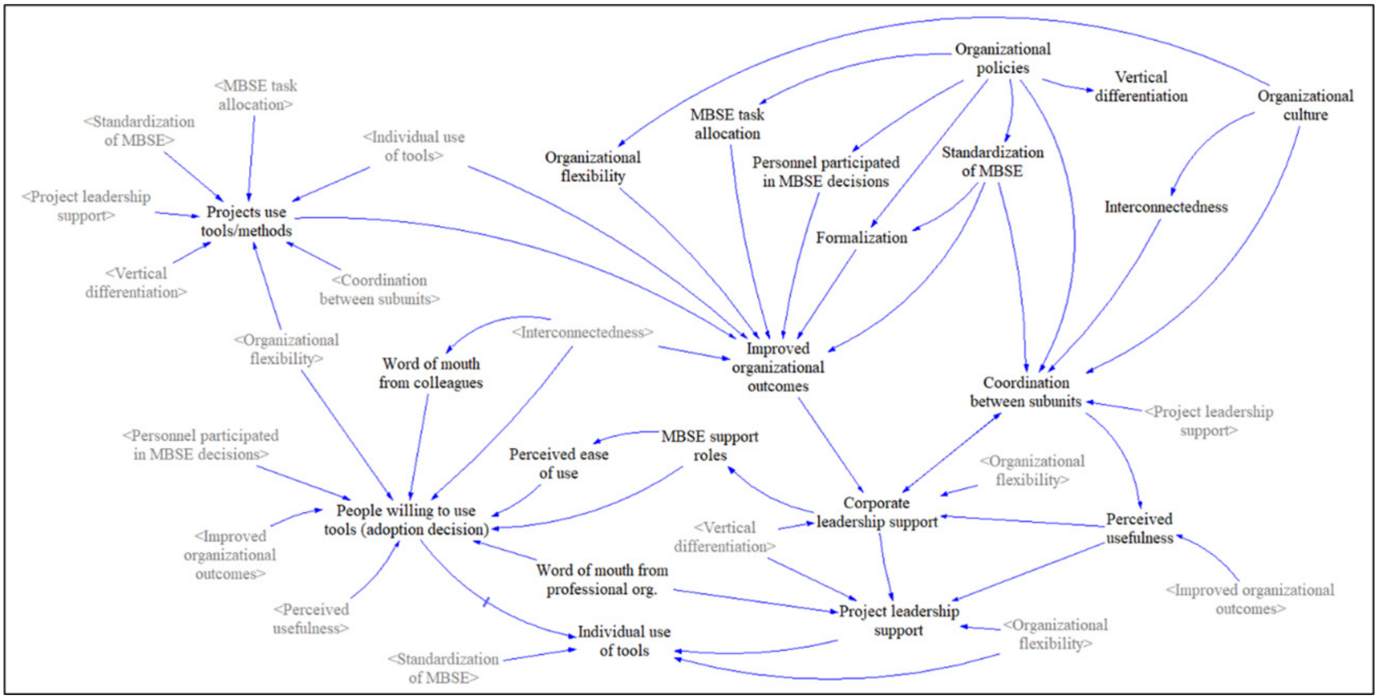
6. ADOPTING MBSE

At the time of writing this chapter, there is a growing number of organizations adopting and implementing MBSE around the world. However, organizations hold differing views of what MBSE is [13]. In some organizations, MBSE is defined in terms of the tools. Others define MBSE in terms of the models, model artifacts, methods, or processes they use. Some organizations even gloss over the 'SE' part of MBSE. Having the correct expectation of what MBSE is and what it can provide for an organization is critical for setting up an adoption effort to be successful in an organization [14].

As has been discussed throughout this chapter, it is critical to know that MBSE is *just* a different way of performing systems engineering. Therefore, when adopting MBSE, it is important to start from processes and practices and not from the tool to be used. The organization must focus on selecting where in their organization MBSE would be the most beneficial with the identification of business value, not simply technical benefit.

Lessons learned indicate that it is easier to adopt MBSE if the scope is limited at the beginning, as an early demonstration of benefits can be shown. In fact, improved organizational outcomes is central to adoption (ref. Figure 11). This comes down to a common element of change management: people want to know that something is going to help them before they commit to learning, applying, and supporting it.

Managers often sidestep this issue when adopting MBSE because it is difficult to show quantitative benefits [8]. However, it is still possible to rely on the anecdotal observations of others that have attempted to implement MBSE and learn from their shared experiences. Successful implementation and adoption seems to require a holistic approach, intentionally targeting the different aspects listed in Table 1 [13]. For example, organizational units that exhibited higher degrees of interconnectedness, standardization, and flexibility reported improved outcomes for MBSE adoption and implementation with respect to exhibited lesser degrees [16].



<MBSE task allocation>
<Standardization of MBSE>
<Individual use of tools>
<Project leadership support>
Projects use tools/methods
<Vertical differentiation>
<Coordination between subunits>
<Organizational flexibility>
<Interconnectedness>
Word of mouth from colleagues
<Personnel participated in MBSE decisions>
People willing to use tools (adoption decision)
<Improved organizational outcomes>
<Perceived usefulness>
<Standardization of MBSE>
Organizational policies
Vertical differentiation
Organizational culture
MBSE task allocation
Organizational flexibility
Personnel participated in MBSE decisions
Standardization of MBSE
Interconnectedness
Formalization
Improved organizational outcomes
Coordination between subunits
<Project leadership support>
Perceived ease of use
MBSE support roles
<Organizational flexibility>
Corporate leadership support
<Vertical differentiation>
Perceived usefulness
Word of mouth from professional org.
Project leadership support
<Improved organizational outcomes>
Individual use of tools
<Organizational flexibility>

Figure 11. MBSE adoption causal model [from [15]]

Organizational design	Organizational enablers/barriers	Organizational change management
Workforce knowledge/ skills	Leadership/ management support/ commitment	Application of MBSE methods/ processes and modeling practices
Integration	Training	Adoption/ implementation strategy and design
Demonstrated benefits/ results	Resources for implementation	Culture change management
Organizational structure	Tool infrastructure	Willingness to use tools (employee and stakeholder buy-in)

Table 1. Aspects necessary for successful MBSE implementation and adoption [adapted from [13]]

Interconnectedness: *people within the organization interact often and are willing to assist others with problems.*

Standardization: *the use of MBSE tools and methods are standardized across the organizational unit.*

Flexibility: *organizations can adapt easily to change, new technologies, and processes.*

These factors describe some ways an organization can set itself up for success when adopting MBSE. It is important to have some type of network established for people who are learning MBSE. This can take many forms: mentors, coaches, defined experts someone can turn to, or a network of experienced peers. It is also beneficial to have what tools and methods are used standardized across the organizational unit. This can make it easier for informal/formal networks of people to help everyone else who is learning. For example, if the same tool is used across the team, it is more likely that people will be able to help if a problem related to the tool arises. The connection with flexibility highlights the importance of change management. While this factor is more challenging to enact in a short period, there are steps can be taken to make sure people in the organization are prepared to adopt MBSE. Making sure the purpose of MBSE is clear and explaining how it can be beneficial to someone's daily work goes a long way towards having a workforce ready to accept MBSE.

While this type of research on MBSE adoption is helpful, it is important to know that there is no "one size fits all" approach to successfully adopt MBSE. As discussed with standards and methodologies, an organization's purpose behind using MBSE can vary widely. An adoption strategy needs to be in-line with that purpose, along with several other organizational factors (e.g. leadership buy-in, financial support, subject matter expertise of employees, etc.).

There are three main groups within an organization to consider when defining an adoption strategy [17]:

1. **The initiators and drivers of MBSE.** It is necessary to have some level of knowledge and expertise in the workforce to really drive the effective use of MBSE in projects. There are many modeling pitfalls that are easy to fall into, which could result in the creation of models that do not actually add value to the organization. This is a surefire way to make adoption of MBSE even more difficult. It is important (especially early on) to show the benefits of modelling, so it is key to have the right people on the team supporting the effort.

2. **The organizational units that should work *model-based.*** Building models is great, but if the models are not being used for anything then what is the point? For MBSE to fulfill its intended purpose, it needs to fit in an organization's process. In some cases, this could take adjusting workflows, job descriptions/responsibilities, or even the structure of the organization. It is more than simply translating document-centric approaches into a model-based world; it is a digital *transformation*. An organizational unit adopting MBSE needs to conscientiously plan out where in its process it makes sense to use MBSE to create the most value.

3. **The organization units responsible for the time and budget of the engineering projects.** MBSE represents a significant investment in money and time for an organization. This fact needs to be considered and used to manage expectations for leadership and others. The organizational unit that is adopting MBSE needs to be given the space and resources to do it successfully.

Preparing the workforce, setting up the infrastructure for adoption, and defining an adoption strategy are critical enablers for MBSE adoption. But once an organization gets to that point, they need to do the actual adopting. A key factor here is training. Many people find MBSE challenging to learn. This could be because they are actually learning multiple things at once: a tool, a modeling language, a method, and system engineering. According to practitioners, all stakeholders need some level of training, but the amount and what they need to learn varies [13].

Table 2 shows four categories of roles and their common MBSE training needs. Different roles may require only a subset of the components of MBSE (i.e., tool, language, method, SE concepts) that they need to be familiar with. For example, someone serving as a 'model reviewer' likely does not need training related to the specific tool or method. But they do need to be able to understand and interpret MBSE artifacts. In this case, some training in the modeling language may be all that is required. Two aspects are worth noting. First, the roles are not necessarily mutually exclusive, but a given individual may fulfill more than one role in different capacities at once. Second, whereas some teams may decide to split the role of a modeler and a regular systems engineer, some other teams may assign each systems engineer (or engineer in general) modeling tasks without depending on a dedicated team for modeling tasks.

Integration of work is another critical factor to the successful adoption of MBSE enterprise-wide [15]. Systems engineering in general involves different functional teams/disciplines to come together, and this is still the case with MBSE. Coordination between these two groups is key. A team can create a great model, but if it does not accurately represent the system then it is of little use. Additionally, system models are often broken up into a collection of smaller system models. In other words, multiple teams are responsible for modeling a component of the system that is ultimately integrated together. In these cases, having some level of standardization/consistency across those groups is critical since those disparate models will need to ultimately integrate. There are many different ways to correctly model the same thing, so having some guidelines for groups to follow relative to some of those decisions will help the final product be more cohesive and reduce integration effort.

Adoption of MBSE is difficult because there are many moving parts. But ultimately, there are many ways in which MBSE can provide value to the team. An organization *just* needs to find what that is and use that purpose to navigate through all the different decisions and components that need to be made.

Categories of roles who need training	
Model reviewers	Leaders, stakeholders, or customers who need to know how to use the models to make decisions
Developers (Modelers)	People who are building and maintaining the models in the tools, so they will need detailed knowledge of how the tool works
Other engineers	People who will be working in the model to some capacity. These people are often senior engineers or people from other disciplines who are helping with the content of the model
Administrators	People who work in IT or software who will be managing the relevant accounts, licenses, tools, etc.

Table 2. Categories of roles requiring MBSE training

7. CONCLUSIONS

Given the speed at which the world keeps accelerating, transitioning to model-based/digital approaches to engineering is inevitable. Traditional practices simply cannot keep up with the demand for rapid development and production of ideas into reality. Systems engineering itself is not changing; just the way it is done is. The key to embracing MBSE is to focus on good systems engineering and recognize that model-based is largely using computer-aided techniques to better execute what systems engineers have always done.

MBSE is gaining a foothold in industry and government as more and more people are starting to see the benefits of using it. At its core, MBSE is about capturing data in a better way using modern techniques enabling teams to better understand, communicate, reason, and retain knowledge. The cost of adoption in terms of time and money is high, but it is argued that the cost of not adopting it will be even higher in the long run.

BIBLIOGRAPHY

1. Blanchard, B.S. and J.E. Blyler, Systems Engineering Management. 5th ed. 2016, Hoboken, NJ, USA: John Wiley & Sons, Inc.

2. INCOSE, Systems Engineering Vision 2025. 2014.

3. INCOSE, Systems Engineering Vision 2035. 2023.

4. Defense, D.o., DoD Modeling and Simulation (M&S) Glossary. 1998.

5. Henderson, K. and A. Salado, Is CAD A Good Paradigm for MBSE? INCOSE International Symposium, 2021. 31(1): p. 144-157.

6. Long, D. and Z. Scott, A Primer For Model-Based Systems Engineering. 2nd ed. 2011, USA: Vitech Corporation.

7. Cratsley, B., et al., Interpretation Dscrepancies of SysML State Machine: An Initial Investigation, in Conference on Systems Engineering Research. 2020: Virtual.

8. Henderson, K. and A. Salado, Value and benefits of model-based systems engineering (MBSE): Evidence from the literature. Systems Engineering, 2021. 24(1): p. 51-66.

9. Blackburn, M. and T. West, Fundamentals of Digital Engineering, in Systems Engineering for the Digital Age. Practitioner Perspectives, D. Verma, Editor. 2023, John Wiley and Sons, Inc.: Hoboken, NJ, USA. p. 3-24.

10. Salado, A., 5.5.2 Efficient and Effective Systems Integration and Verification Planning Using a Model-Centric Environment. INCOSE International Symposium, 2013. 23(1): p. 1159-1173.

11. Dunbar, D., et al., Transforming Systems Engineering Through Integrating Modeling and Simulation and the Digital Thread, in Systems Engineering for the Digital Age. Practitioner Perspectives, D. Verma, Editor. 2023, John Wiley and Sons, Inc.: Hoboken, NJ, USA. p. 47-68.

12. Dunbar, D., et al., Driving digital engineering integration and interoperability through semantic integration of models with ontologies. Systems Engineering, 2023. 26(4): p. 365-378.

13. Henderson, K., T. McDermott, and A. Salado, MBSE adoption experiences in organizations: Lessons learned. Systems Engineering, 2024. 27(1): p. 214-239.

14. Henderson, K., et al., Towards Developing Metrics to Evaluate Digital Engineering. Systems Engineering, 2023. 26: p. 3-31.

15. Henderson, K., Exploring the Adoption Process of MBSE: A Closer Look at Contributing Organizational Structure Factors, in Grado Department of Industrial and Systems Engineering. 2022, Virginia Tech: Blacksburg, VA, USA.

16. Henderson, K. and A. Salado, The Effects of Organizational Structure on MBSE Adoption in Industry: Insights from Practitioners. Engineering Management Journal, 2024. 36(1): p. 117-143.

17. Weilkiens, T., Adoption of MBSE in an Organization, in Handbook of Model-Based Systems Engineering, A.M. Madni, N. Augustine, and M. Sievers, Editors. 2020, Springer International Publishing: Cham. p. 1-19.

DAVID LONG

David Long is the President of Blue Holon and a Research Scientist with the Systems Engineering Research Center (SERC). For over 30 years, he has helped organizations around the world increase their systems engineering proficiency while simultaneously working to advance the state of the art. He works with government and commercial organizations as they assess, adopt, and deploy new methods and tools to enhance their engineering enterprise. Throughout his career, David has played a key technical and leadership role in advancing and expanding the practice of systems engineering. David founded and led Vitech where he developed innovative, industry-leading methods and software (CORE™ and GENESYS™) to engineer next-generation systems. David is a frequent presenter at industry events worldwide delivering keynotes and workshops spanning the foundations of systems engineering, practical model-based systems engineering, digital engineering, and the future of engineering systems. His experiences and efforts led him to co-author the book *A Primer for Model-Based Systems Engineering* to spread the fundamental concepts of this key approach to modern challenges. An INCOSE Fellow and Expert Systems Engineering Professional (ESEP), David was the 2014/2015 President of INCOSE. David currently serves as INCOSE's Director for Strategic Integration and as a coach in INCOSE's Technical Leadership Institute. David holds a BS in Engineering Science and Mechanics and an MS in Systems Engineering from Virginia Tech.

DR. KAITLIN HENDERSON

Dr. Kaitlin Henderson is a Systems Architect at Radiance Technologies, where she supports various SysML modeling efforts. She has a PhD in Industrial and Systems Engineering with a concentration in Management Systems from Virginia Tech in Blacksburg, Virginia. She also earned her Bachelor's and Master's degrees in Industrial and Systems Engineering at Virginia Tech. Kaitlin has contributed to the fields of Model-Based Systems Engineering (MBSE) and Digital Engineering (DE) with her works on MBSE adoption, MBSE domain maturity, and MBSE/DE performance measurement. Kaitlin won the Best Presentation Award at the SERC Doctoral Student Forum in 2020 and has been recognized for her 2021 and 2022 publications in Systems Engineering.

BELINDA MISIEGO TEJEDA

Belinda Misiego Tejeda is the chief of innovation at Isdefe and coordinates Systems Engineering Campus in the company. Through this Campus, internal training courses are carried out with its own personnel in the different areas of this discipline: systems engineering initiation, fundamentals of systems engineering and various specific courses. She has more than 20 years of professional experience in the security and defense sectors, where she has developed and managed different systems engineering projects, mainly in the field of sensors and electronic warfare. Belinda, member of INCOSE and systems engineering professional (CSEP), was part of the board of the Spanish Association in Systems Engineering (AEIS), the Spanish chapter of INCOSE between 2017 and 2022. In addition, she served as Secretary of the EMEA Director of INCOSE in 2019-2022. Belinda has a BS/MS in Electrical and Computer Engineering and a graduate certificate in Research and Market Techniques from the University of Valladolid and a MA in Contract and Program Management in the Public Sector from the Spanish University of Distance Education (UNED).

"Is the universe really so loosely coupled? Or is this small dimensionality due to the fact that the humans who developed the equations controlled their experiments in accordance with their cognitive limitations?"

G. Friedman

CHAPTER 5

Digital Transformation in System Development

Christopher L. Delp, *Jet Propulsion Laboratory, California Institute of Technology (christopher.l.delp@jpl.nasa.gov)*
Dr. Joe Gregory, *The University of Arizona (joegregory@arizona.edu)*
Luis Miguel Aparicio Ortega, *Isdefe (laparicio@isdefe.es)*

Abstract

This chapter presents the novel capabilities enabled by digital models and high computational power of current workstations, cloud, and grid computing to support systems development, integration, and qualification. It introduces the concept of digital transformation and explains how it builds on the processes of digitization and digitalization. The key technologies that enable digital transformation are presented and discussed. These include formal languages and semantic web technologies. The chapter then looks at some examples of how digital transformation can lead to improved outcomes with regards to systems engineering practice. Examples include enhanced traceability, the automated generation and evaluation of architectures, set-based design, and the integration of physics-based models into systems engineering models.

Keywords:

Digital engineering, digital thread, tradespace exploration, set-based design, Multidisciplinary Design Optimization (MDO), semantic web technologies, ontologies, systems model.

1. INTRODUCTION

Interest in digital transformation has seen rapid growth in the last decade. In this chapter, we present an overview of this rapidly evolving field and discuss some of those relevant, emerging technologies that digital transformation aims to leverage and provide examples of how they are being applied to Systems Engineering (SE). First, to understand what we mean by Digital Transformation, there is a crucial distinction that needs to be made between two commonly used terms: digitization and digitalization.

Digitization refers simply to the change that occurs when an analog artifact, whatever it may be, is transformed into a digital artifact. If we take a physical document, such as a report or a photograph, and convert it into a digital format through scanning or simply by typing text in a word processor, we have performed digitization. This kind of process has been taking place for decades, and probably everyone understands the benefits of working with digital artifacts instead of with tons of paper. For example, digitization of information enabled vast quantities of information to be transmitted digitally and without loss. This process of digitization was enabled by advances in computer miniaturization and networking infrastructure, and it revolutionized the ways in which information can be managed and communicated, known as the Information Age or the Third Industrial Revolution.

Digital transformation, however, goes beyond digitization. In digitalization, engineers work with models and data that are digitally linked. Any information consumption is just a visualization of the underlying models and data. In other words, the information and its visualization are decoupled. Because of this, digitalization often involves rethinking traditional processes to take full advantage of the capabilities that have been made possible by connecting data and models in a machine-readable manner. Clearly, digitalization involves some deep thinking about how modern technologies can be effectively leveraged, particularly when applied within a business context. When it is to be applied within the boundary of an organization, digitalization requires careful consideration of the necessary cultural and organizational changes, relevant talent acquisition and development, data governance and security within and across organizational boundaries, as well as multiple other aspects. Together, this process of digitization and digitalization to improve processes within a business context is known as the Digital Transformation [1].

Before we look at some of the technologies that enable digital transformation, let us consider what an ideal 'digitalized' SE process might look like. In other words, to understand what is required to support our digital transformation, we must have a vision of what our digitally transformed enterprise should be capable of. In a nutshell, the information generated from all stages of the system lifecycle is in digital form, is linked through a data-driven architecture of shared resources, feeds descriptive and quantitative models, and can be used for real-time and long-term decision-making" [2] (ref. Figure 1). Practically speaking, this means that data can be seamlessly exchanged between the different tools that support modeling in each of system lifecycle phases. Not only does this provide the necessary traceability between design decisions, tests, requirements, and so on; it enables decision support analyses that were previously infeasible.

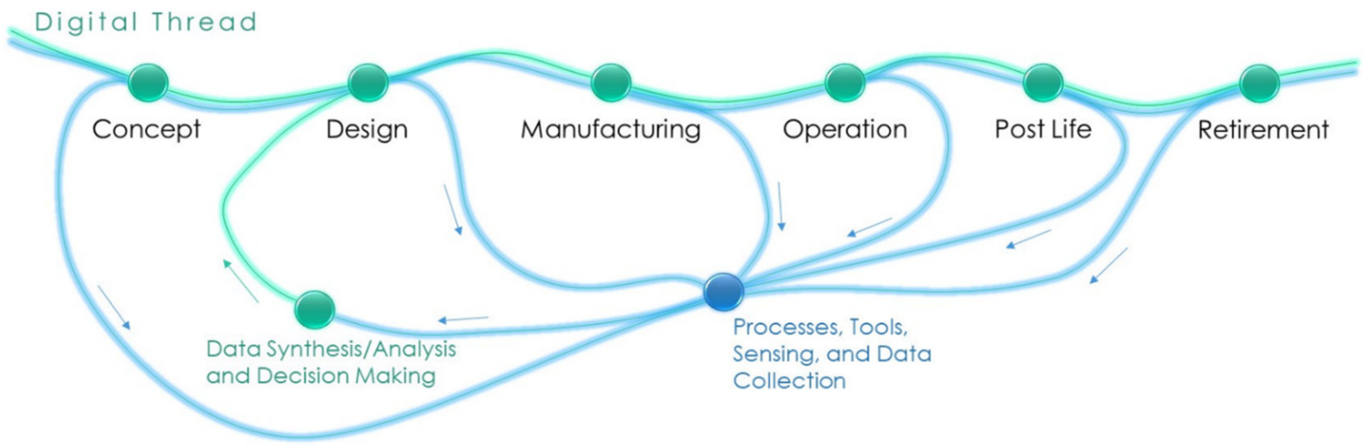

Digital Thread
Concept
Design
Manufacturing
Operation
Post Life
Retirement
Data Synthesis/Analysis and Decision Making
Processes, Tools, Sensing, and Data Collection

Figure 1. The Digital Thread related to Systems Engineering [adapted from [2]]

Furthermore, this capability extends beyond the processes of any single organization – an ideal digital thread enables collaboration and seamless data exchange across organizational boundaries to support critical aspects such as supply chain integration. Within the confines of appropriate security and privacy protocols, data may be rapidly accessible across the digital thread to support decision-making at the enterprise level. Appropriate change propagation pipelines ensure that downstream effects of a change to the data are identified and acknowledged.

2. ENABLERS OF DIGITAL TRANSFORMATION

2.1. Computer languages unleash the power of applied mathematics

It is the thrust of increasingly sophisticated systems coupled with the critical impacts of quality and cost that drive the need for systems engineering practices that are quantifiable and analyzable. Achieving this need often requires the development of an executable representation of the system. This intersection of need and the availability of a particular kind of system – the modern computational environment – is a rather poetic result: it is now possible to use a human-developed system to enter a recursive development cycle whereby improvements in the discipline of SE can deliver improvements in how we practice the discipline.

The core of this concept is rooted in the ability to mathematically model the world around us. In Computer Science, the essence of this innovation is the weaving together of computational instruction with what is arguably the most significant invention in human history – language.

Since its inception, systems engineering has focused on the specification of the systematic behavior of an element known as a 'System'. This element can take many forms as long as the central concept defining its existence is systematic behavior. Traditional SE focused on informal descriptions, human thought experiments, and heuristics to tackle the increasingly complex problems that SE was expected to deal with [3]. This approach often relied on expensive testing on real-world builds and rigid inflexible constraints on changes to meet unforeseen challenges.

The widespread development, deployment, and adoption of suitable modeling languages to support SE has been described as 'The Systems Engineering Challenge' [3]. In recent decades, increased computational power and the development of a multitude of declarative modeling languages have contributed to the advancement of SE. Some of these are displayed in Figure 2.

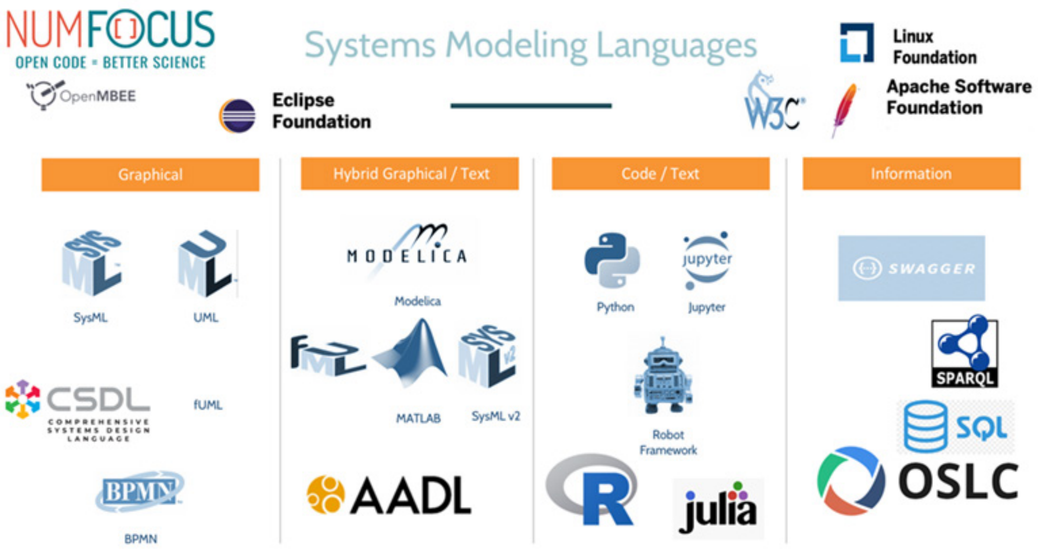


Figure 2. Examples of declarative modeling languages to support systems engineering and some of the standards bodies and open source software foundations that sustain them

Graphical languages such as SysML version 1 and CSDL provide the capability to visually model specifications of concepts and relationships. Text/code-based languages such as Julia and Robot expand the power across the lifecycle to capture detailed execution, analysis, and systems testing capability. The new generation of hybrid languages such as AADL, SysML version 2, or Modelica point to a powerful capability to specify, visualize, code, and analyze systems across the lifecycle. Information modeling languages such as OWL/SparQL and OpenAPI/Swagger also play a crucial role in modern systems modeling and Digital Engineering (DE). They provide a foundation for supporting the web-based interoperability required for model integration and interchange.

The languages presented in Figure 2 can all be considered 'Declarative Languages'. Declarative languages focus on semantics capable of being solved in a variety of ways besides traditional execution. Often, they can be solved using analytic techniques as opposed to procedural semantics which execute statements in the order they are read. Declarative models tend to have mathematical statements that can be solved in more than one way. It is worth noting as well that while languages such as MATLAB, Python, or Julia are not strictly declarative, they build libraries and capabilities used heavily in engineering that perform analysis on declarative models such as algebra solvers and other equation-based analysis. Examples include First-Order Logic (FOL) and Finite State Machine (FSM). FOL in particular has received significant attention, and not just in engineering, due to its ability to precisely capture knowledge and infer new knowledge based on a set of rules [5]. FSMs are a good example of this type of language. They declare states, behavior, and events. The a-causal nature of an FSM can be interpreted differently while still retaining the mathematical formality of the FSM. The most direct way to solve an FSM is to define a trajectory of events and test the FSM to see the output. A less obvious example would be to use a solver to analyze the entire FSM to find out if the event trajectory is valid. A third example would be to analyze the FSM to determine if there are any states defined such that they can never be reached.

While declarative languages tend to enable formal methods in DE, there is still value in informal descriptive and qualitative representations such as narrative text, graphics, diagrams, and pseudo code. They are an important part of the development cycle. As models are developed against informal material they undergo a process called the Model-Hardening Process. This process captures the relationships between the qualitative representations and the quantitative representations as the design matures.

With all the right ingredients, SE can produce precise, accurate, and complete digital representations of systems and the processes that will realize them. The 'right ingredients' are those that leverage the power of computation and the precision of formal languages to manage the huge amount of data that is associated with an SE project, automate SE processes through inference and reasoning, and preserve rigor within the system lifecycle. One area we can look to for inspiration in this regard is an area that has witnessed huge progress in the last 30 years in terms of information management and retrieval: the evolution of the World Wide Web.

2.2. Semantic web technologies

The Semantic Web is an extension of the World Wide Web that intends to make data machine-readable and provide a standard structure for data representation and reasoning [6]. At the core of this evolution is the transition from a web that "consists largely of documents for humans to read to one that includes data and information for computers to manipulate" [7]. An obvious first example might be the evolution of the search engine. Far from being a simple keyword-match web-crawler algorithm, modern search engines exploit this hyper-connectivity between data to understand the meaning behind your search query, employ complex algorithms utilizing machine-learning to understand user intent, and often generate a natural-language personalized response.

Semantic web technologies are the technologies that enable these capabilities. They provide standardized ways to represent data in triples of subject-predicate-object format that allows to describe knowledge such as 'Curiosity is a Mars Rover' or 'Curiosity executes Analyze Soil'. In this way, it is possible to build up complex networks of interconnected data that are entirely built on this simple triple pattern. By using an ontology, these networks can be leveraged to check data validity, as well as to infer new information that we have not explicitly declared. For example, consider the ontology in Figure 2. In this ontology, three classes are defined: System, Function, and Mars Rover. It has also been stated that Mars Rover is a subclass of System. An object property (a relation between two classes) called 'executes' specifies that the domain can only be a System, and the range can only be a Function. This ontology can be used to validate the dataset presented earlier (i.e., 'Curiosity is a Mars Rover' and 'Curiosity executes function "Analyze Soil"'), and to infer new information that that was not previously stated. Because Curiosity is a Mars Rover, and the ontology specifies that Mars Rover is a subclass of System, it is possible to infer that

Curiosity is also a System. Also, because Curiosity (a System) executes 'Analyze Soil', it can be inferred that 'Analyze Soil' is a Function. The validity of the dataset is confirmed because no rules in the ontology have been broken. If we had stated that 'Analyze Soil' executed 'Curiosity', our dataset would have been declared inconsistent, as the range of the relation 'executes' can only ever be a Function.

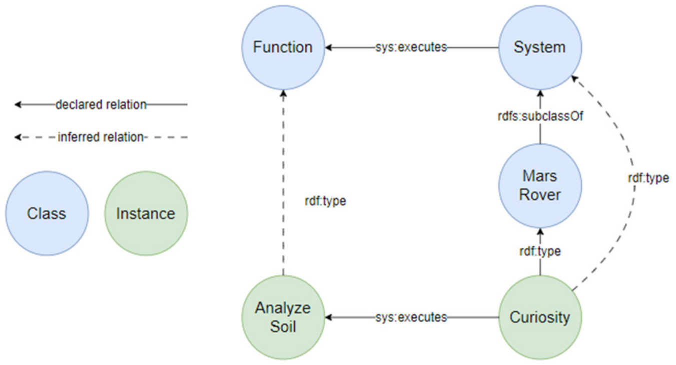


Figure 3. Example Knowledge Graph [Legend: rdf: Resource Description Framework; rdfs: Resource Description Framework Schema; sys: System Ontology]

As an example, Figure 4 shows a partial representation of the ontology to support the modeling of the harness design for a spacecraft.

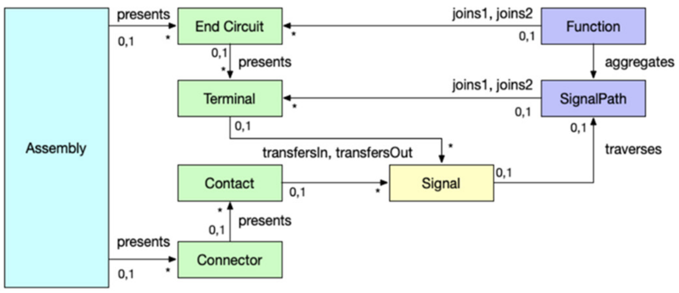


Figure 4. Partial Representation of Harness Design Ontology, from [8]

3. MODEL-BASED LIFECYCLE MANAGEMENT AND QUALIFICATION

In this section, we focus on how modern approaches to DE can support lifecycle management and qualification, e.g. verification and validation and certification. We present examples from the literature regarding traceability across design artifacts, early Verification and Validation (V&V), and model-based reviews.

3.1. Traceability of system design artifacts

Traceability refers to the ability to establish explicit relationships between elements of the system design (e.g., a quantity value, a design decision) across the system lifecycle. In systems engineering, design traceability is a common example, whereby all system elements, design decisions, and test cases can be traced back to a particular requirement (or set of requirements). The purpose of this is to be able to ensure that the results of the design can be explained and checked for review and audit.

DE, and particularly the digital thread, enables unprecedented traceability across the system lifecycle. One of the main benefits of this approach is that data can be defined once in an Authoritative Source of Truth (ASOT), and then distributed across the digital thread to other models and artifacts that require access to the data. An ASOT is the source of a baselined version of data and should be accessible to authorized applications that intend to use the data in some analysis or decision, for example. In this way, engineers can be confident that applications across the entire digital thread (thus representing the entire system lifecycle) are using a valid set of data. The establishment of ASOTs within a digital thread is an effective way to avoid inconsistencies.

Similarly, traceability within a digital thread enables the identification of change propagation paths. If we consider a change to a system requirement, for example, traceability across the digital thread enables engineers to identify the system elements and tests that will be affected, among others.

To achieve this degree of traceability, there are two aspects to consider: data interoperability and technical interoperability [9]. Data interoperability ensures that there is a consistent understanding of the relevant terminology, and a standard data structure to which data can be mapped. Technical interoperability solves the problem of getting data from multiple sources into the same database in the first place – or at least provides point-to-point connections between relevant tools. Technical interoperability can be streamlined by using established integration protocols such as REST API. The Open Services for Lifecycle Collaboration (OSLC) is a community that develops and releases open-course standards aimed at improving integration through REST APIs [10].

The Dragon Architecture, under development by NASA JPL, is a DE environment that comprises multiple systems and software engineering applications and implements the digital thread by connecting the tools together in a graphoriented structure, emphasizing explicit relationships to form an integrated heterogeneous model [11] (ref. Figure 5). The use of ASOTs and technical interoperability are core aspects of the Dragon Architecture. It enables the user to define inter-tool relations, thus providing connectivity that integrates requirements, architecture, detailed HW and SW design, test cases, and so on. The Dragon Architecture achieves this through the application of Internet and World Wide Web based interoperability technology and standards. Interoperability through these technologies and standards is a fundamental principle of the Dragon Architecture.

Although not always necessary, ontologies can help with data interoperability. Interoperability in Dragon, however, does require the expression of semantics in order to integrate and inter operate especially focusing relationships. It is often the case that Ontology is associated with the Semantic Web and the Web Ontology Language. While this is certainly a valid means of modeling an Ontology, it is not the only way an Ontology can be specified. The application of ontologies to SE aims to "enable more automated sharing of information directly between models to ensure model consistency, improve the rigor of engineering process, and ultimately, reduce the effort needed to get a clear answer to engineering questions" [8]. The Digital Engineering Factory (DEF), for example, is a DE environment under development at the University of Arizona that enables users to integrate data from multiple tools and structures this data in accordance with the University of Arizona Ontology Stack (UAOS) [9]. An ontology stack is a structured hierarchy of ontologies, based on the same Top-Level Ontology (TLO), to support data interoperability between domains. The DEF uses a hub-and-spoke architecture to integrate data from multiple tools into a central database (hosted in the Violet tool). This enables users to define inter-tool connections, thus supporting traceability between requirements, system architecture, detailed HW and SW design, verification activities, and project management (ref. Figure 6).

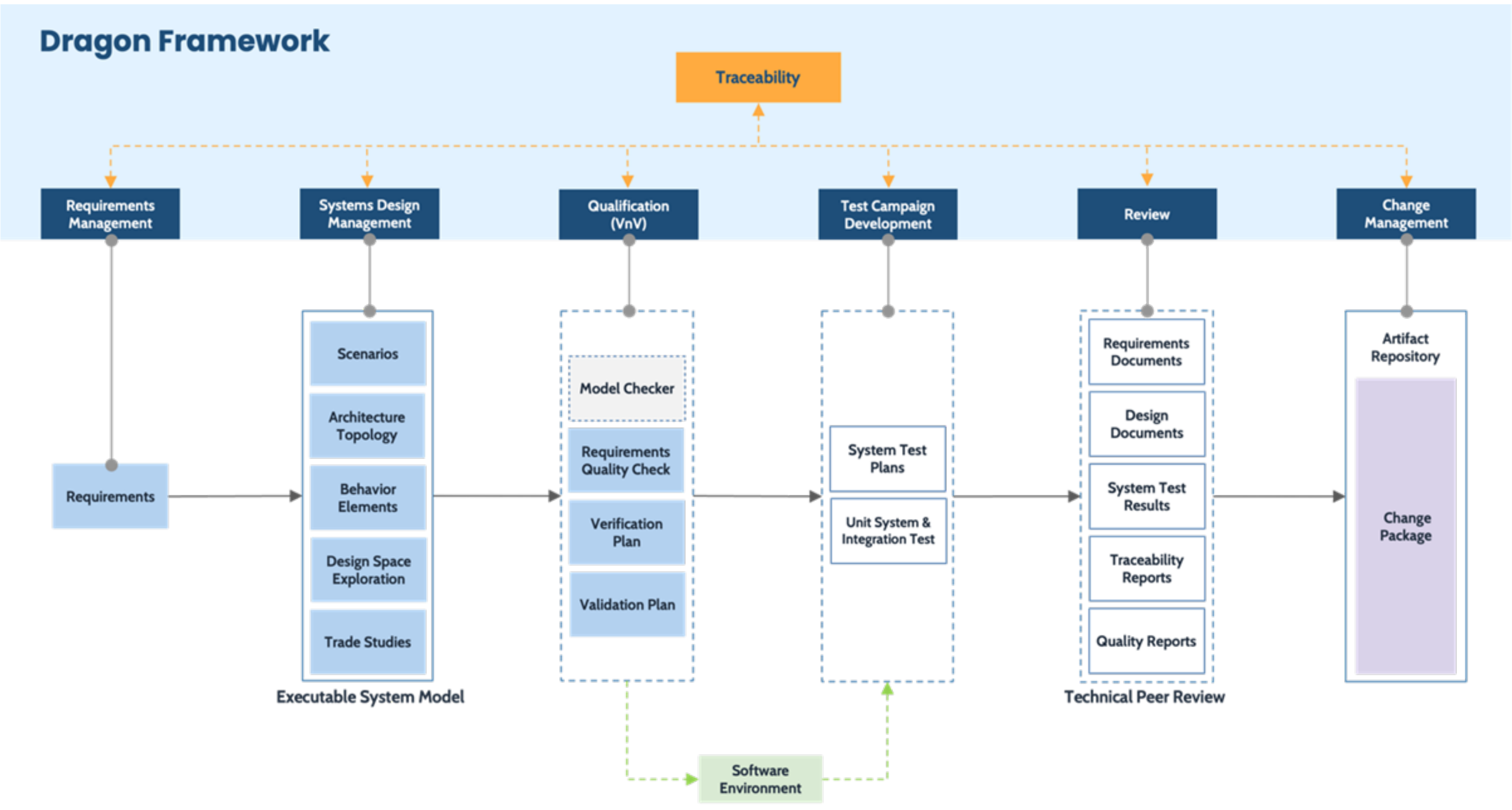


Figure 5. Dragon Architecture Functional View. This is an illustration intended to communicate the general flow of work and conceptual dependencies. It is not intended to describe a process or gates. All functions are realizable with off-the-shelf technology except for model checking

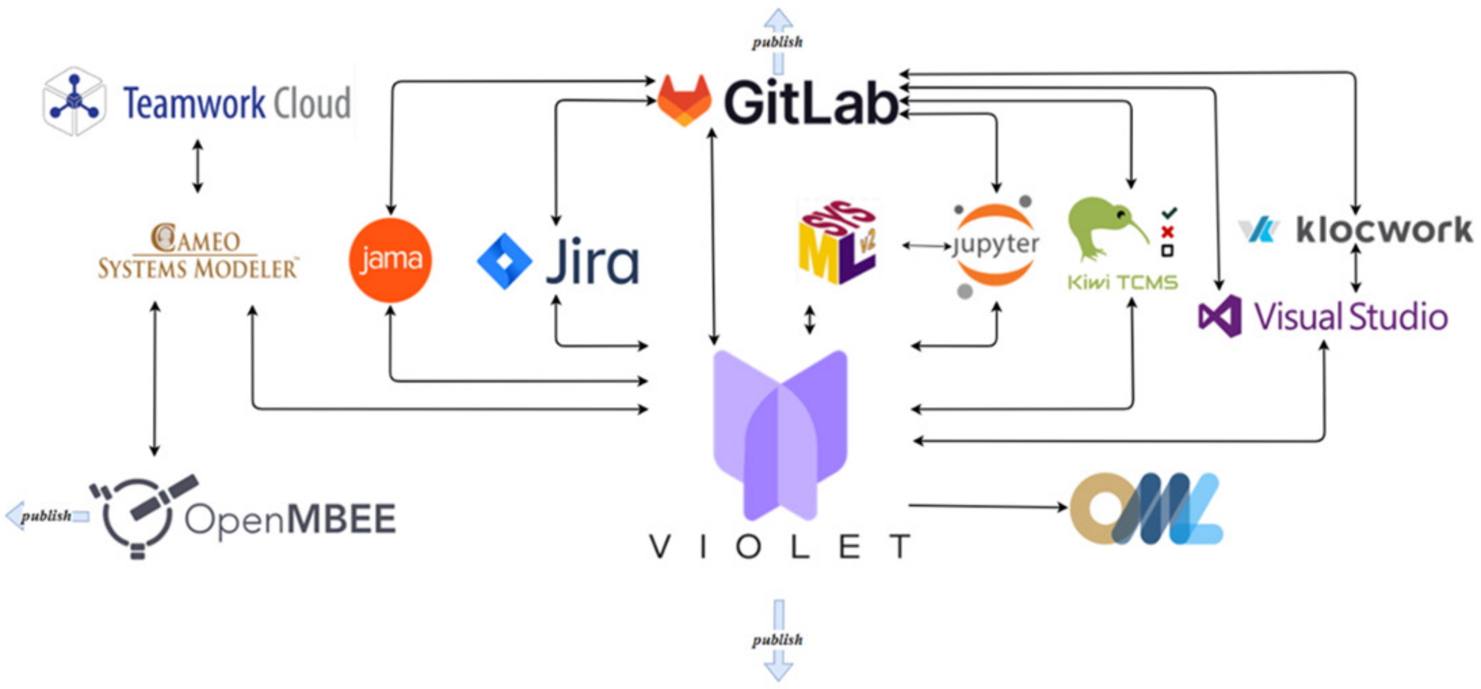


Figure 6. The Digital Engineering Factory (DEF), adapted from [12]

3.2. Early verification and validation

The potential connectivity offered by the digital thread and its enabling technologies also makes it possible to move V&V earlier in the lifecycle [16]. For example, modeling the system architecture in the early stages of design with an executable modeling language enables the execution of simulations and resulting quantitative analyses to assess compliance to requirements [15]. This is valuable because "finding problems early is a key enabler to DE providing full potential value" [13]. Several modeling languages, but not all, provide these executable capabilities. For example, in SysML, the behavior captured in activity and state machine diagrams can be simulated.

In addition to the use of formal methods to execute and simulate models, which involve a complete, mathematically grounded representation of a system, lightweight formal methods can also contribute to early V&V. Lightweight formal methods involve the mathematical representation of some part of the system specification. In this way, lightweight formal methods provide some of the benefits of formal methods, such as the ability to detect errors in the early stages of system development, but without the need to redefine the entire system specification [20].

"One of the strengths of using DE approaches to concept design is that the various models, tools, and ASOTs are all connected via a digital thread. They share the same consistent, authoritative data. During concept design, one useful application of this digital thread is to connect simulation models with system definition models…and unambiguously share requirements, requirements traceability, and system behaviors" [21].

3.3. Model-based reviews and digital signoffs

The digitalization of SE processes also extends to the major gates of the system lifecycle: technical reviews. Traditional document-based review processes have been used by systems engineers for many years to determine whether system development programs can progress to the next stages in their development. However, this traditional approach to technical reviews often leads to "lengthy evaluations of static, contractually obligated documents" that "represent snapshots of the systems as seen through the prism of the entrance criteria [to the next phase of the lifecycle], and do not represent a view of the system in its totality" [22]. Not only is this process inefficient, but it also often lacks the ability to holistically represent the necessary system-related information to stakeholders.

Model-based reviews have the potential to offer improvements in this regard (e.g., [24, 25]). First, reviewers can approve subsets of data/information instead of having to review and approve monolithic aggregations of information in disparate and often overlapping documents. This has been coined as digital signoffs, where even required signoffs can be assigned to different pieces of data for them to be considered baselined, accepted, and/or approved [24]. Second, reviewers can interact with the interoperable ontology stack to find the information that they consider relevant for the review, as opposed to having to dig and hunt for treasure in static documentation. For example, a reviewer might query the DE repository to identify all verification evidence related (i.e., traced) with a requirement, identify all functions that participate in a capability related to a specific need of a stakeholder, or identify the remaining integration and test activities for a specific component. In both cases, it is easy to realize how the use of DE to support technical reviews can result in an accelerated and more fluid process, while potentially improving its efficiency and reducing the likelihood of review gaps.

It should be noted though that model-based reviews come with challenges as well. Among others, models need to be comprehensive, not just limited to traditional 3D drawings; there is a significant learning curve, as the people involved do not only need to learn the new modeling languages and tools but also the new processes to operate in a model-based review and rely on the ability of the reviewers to identify the necessary information in the digital repository [23]. Adopting DE can help to mitigate some of these challenges. Data and technical interoperability provide engineers with a common data structure and database from which project information can be retrieved. Semantic web technologies can be leveraged to automatically validate this dataset and queries to retrieve, and conveniently present, key information can be standardized and automated (e.g., in an autogenerated document or dashboard).

4. BEYOND TRACEABILITY AND LIFECYCLE MANAGEMENT

So far, we have discussed how DE can be used to enhance traceability, enable early V&V, and support the model-based review process. However, DE is not only about traceability and management. In this section, we provide examples of how DE can support a set of unprecedented analysis that have been infeasible in document-based practices. They include tradespace exploration, set-based design, multi-disciplinary optimization, the integration of physics models within system models, and human-system integration.

4.1. The Tradespace Exploration paradigm

When developing a new system, it is crucial to focus efforts during the early stages of the life cycle on finding a preferred design solution. Since evaluating the entire set of possible solutions requires a significant investment, this analysis is not usually extensive due to the unavailability of the necessary resources. As a result, hasty and generally unjustified decisions are made to reduce the design options and only a few alternatives are evaluated. Consequently, alternatives that could be more valuable than the chosen ones are frequently overlooked.

Tradespace Exploration is a method that leverages digital models and the high computational power of current computers to enable a rapid and comprehensive in-depth analysis of the solution space [26]. Through this strategy, multiple design options are explored as candidate architectures and analyzed to determine the most suitable one. In essence, instead of focusing on finding specific design options, the engineer defines a generic model of the system that can be automatically instantiated as the enumeration of different components and values that their main variables can take. Each resulting solution is evaluated against a set of decision criteria. Among them, a Pareto Front, formed by those solutions that dominate the solution space (that is, there is no solution that performs better in all criteria) can be identified for choosing the desired solution within the set.

For illustrative purposes, consider the design of a new tank with the following desired needs: high transport, high tank protection, and fair mobility. The generic architecture of the tank includes Armor, Weapon System, and Propulsion as its main systems, with each having different design options (Armor: Conventional steel, Ceramics, or Composite; Weapon System: 120 mm cannon or Guided missile launcher; Propulsion: Hybrid electric motor or Internal Combustion Engine). Furthermore, each instantiation can have different performance values, such as having different masses. Considering mass as a surrogate for the overall performance of the different variants of the different technologies (in this case, a total of 71 lower level components), it is possible to enumerate all possible instantiated architectures (e.g., Conventional Steel – Performance 1/120 mm Cannon Performance 1/Hybrid Electric Motor Performance 1, Conventional Steel – Performance 1/120 mm Cannon Performance 1/Hybrid Electric Motor Performance 2, Conventional Steel – Performance 1/120 mm Cannon Performance 1/Hybrid Electric Motor Performance 3, … for a total of 3,408 architectures) and plot them according to their cost (the higher the cost the less preferred the alternative is) and value provided in operations (the higher the value the more preferred the alternative is; note that both have been calculated using notional models), identifying then the Pareto Front (ref. Figure 7, where each apparent line is actually the effect of many points close together).

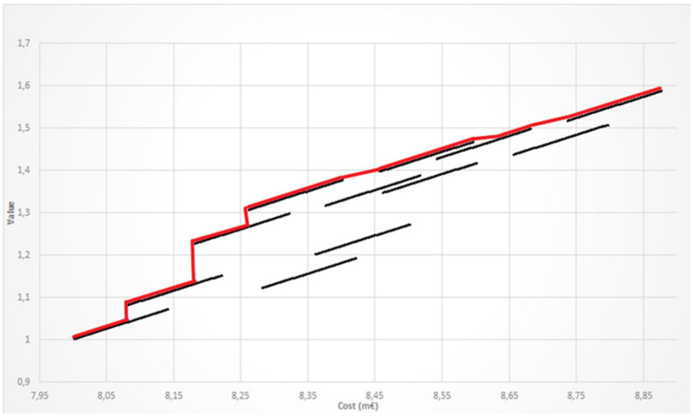


Figure 7. Solution Space for Tank Example relating Cost and Value (Pareto front highlighted)

4.2. The Set-Based Design paradigm

Traditionally, the Point Design Method (PDM) has been employed for system design. It begins with an analysis of alternatives, resulting in the selection of a single concept, which is refined as the development progresses. When a problem is encountered during development, the solution is modified as necessary, generally resulting in higher costs and delays the later the modification is accomplished.

In contrast to this rigid approach, the Set-Based Design (SBD) paradigm encourages maintaining multiple options open and evaluating the trade-offs and advantages of each design rather than quickly converging on a single design [30–32]. It facilitates thorough exploration of the design space and promotes innovation by deferring commitment to a single concept. The solution space progressively narrows as more knowledge and information about the system are acquired (ref. Figure 8). This reduces the risk of erroneous theories and assumptions of decisions early in the design process, which is delayed for critical decisions until sufficient information and knowledge is available to gain flexibility. Furthermore, as a byproduct of working with sets of solutions instead of with single solutions, SBD provides greater flexibility to adapt to unforeseen circumstances. The use of powerful computers with high computing capacity, and object-oriented modeling, play a significant role in simultaneously developing and evaluating various design concepts at a relatively low cost.

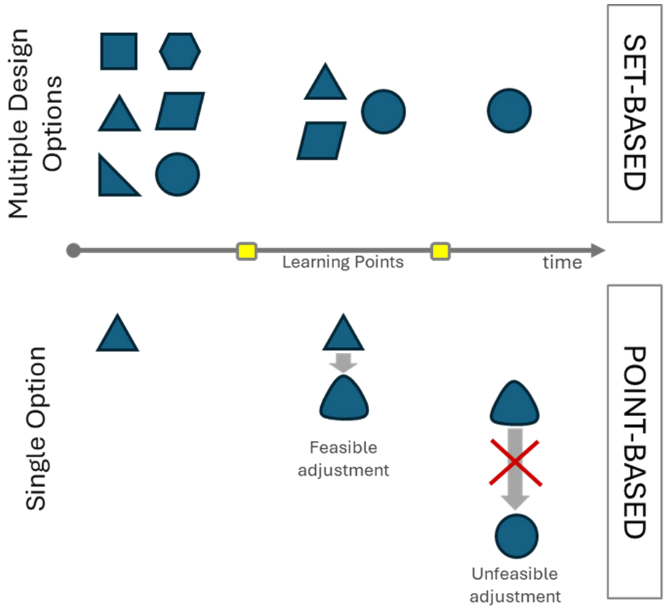


Figure 8. Set-based Design Maturation Process

An SBD essential process consists of three main effects (ref. Figure 9):

1. Definition of the design space, identifying a set of viable alternatives to be progressively developed independently, even by different engineering groups, thereby generating concepts from diverse perspectives. It is crucial to consider a large number of alternatives for evaluation.
2. Discover intersections among different sets of independent solutions to define the core of the new design progressively.
3. Gradually eliminate concepts that prove incompatible or contribute limited value.

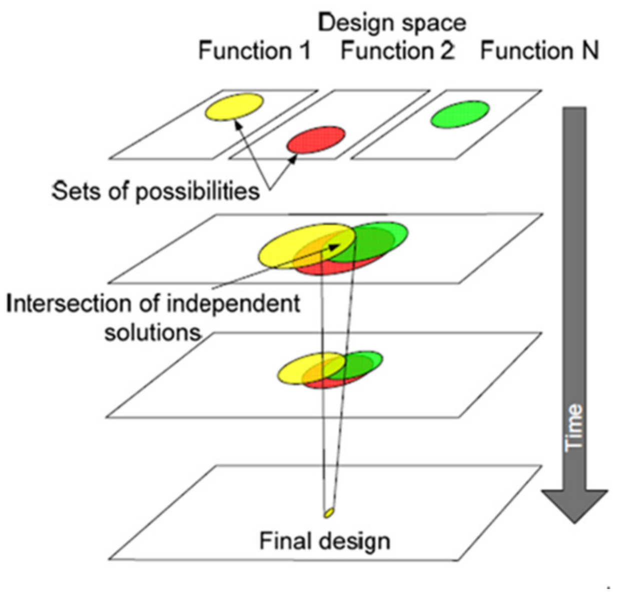


Figure 9. Discovering Intersections in Set-Based Design process [adapted from [33]]

4.3. The MDO paradigm

Traditionally, system decomposition is performed by manually fixing characteristics of the components that form the system and iterating between them until their integration leads to a satisficing system solution. This is due to the coupling that generally exists between the different components of a system. For example, a team designing a satellite may start by estimating the electrical power consumption of each component. They would then aggregate the power dissipated in the different structural panels of the satellite and use that to inform the design of the thermal system (for dissipation). Sizing of the thermal system may feedback into the sizing of the structural panels, and so forth. This iterative process is generally performed manually: propagating the different modifications until something seems promising to work.

Instead, in Multidisciplinary Design Optimization (MDO) the goal is to find an optimal combination of components by managing conflicting objectives not just a satisficing one [34]. MDO leverages computer models to execute optimization algorithms. Particularly, the method consists of integrating mathematical models of the components and executing a global optimization function at the system level. In this way, the design of the components would not lead to just a satisficing solution but to an optimal system. The work of the engineering team moves from iterating their coupled characteristics to developing accurate models of their components (and their integration), so that the computer can handle the iterations on its own.

MDO does not only identifies a solution that is optimal, but it does so more efficiently than human iteration and guarantees alignment between the decisions at component level and the objectives of the project at the system level. However, one should note that robustness of the solution becomes important, as the accuracy of the underlying models to perform the optimization is likely coarse at the stage when MDO is employed.

4.4. Connecting system models with physics-based models

The connection of system models with physics-based models is a common capability leveraged within a digital thread. In essence, behavioral and structural models of a system architecture can be connected to their design instantiations in physics-based models, propagating constraints (in one direction) and results (in the other direction) between the different models automatically, guaranteeing data consistency and trustworthiness [11].

For example, Figure 10 shows the use of the Dragon Architecture to support the architecture and design of a Rover. Adopting the application of computer languages allows the flow to occur from informal, qualitative descriptions to executable mathematically accurate representations in physics. In this case, the system architecture is modeled using SysML, which is connected to its physics-based model, which is established in Modelica. SysML can effectively represent the functional behavior and interfaces of the rover, and Modelica does the same for the physical behavior and interfaces in an executable manner, which is not possible when using natural language or informal diagrams or graphical representations. Collectively, both can be used to represent how signals can be passed to execute the physical response of the Rover based on the behavior in the systems model. This illustrates the explicit ability to unambiguously compare the verify and validate the Power and Drive Behavioral Specifications against a Physical representation of the product.

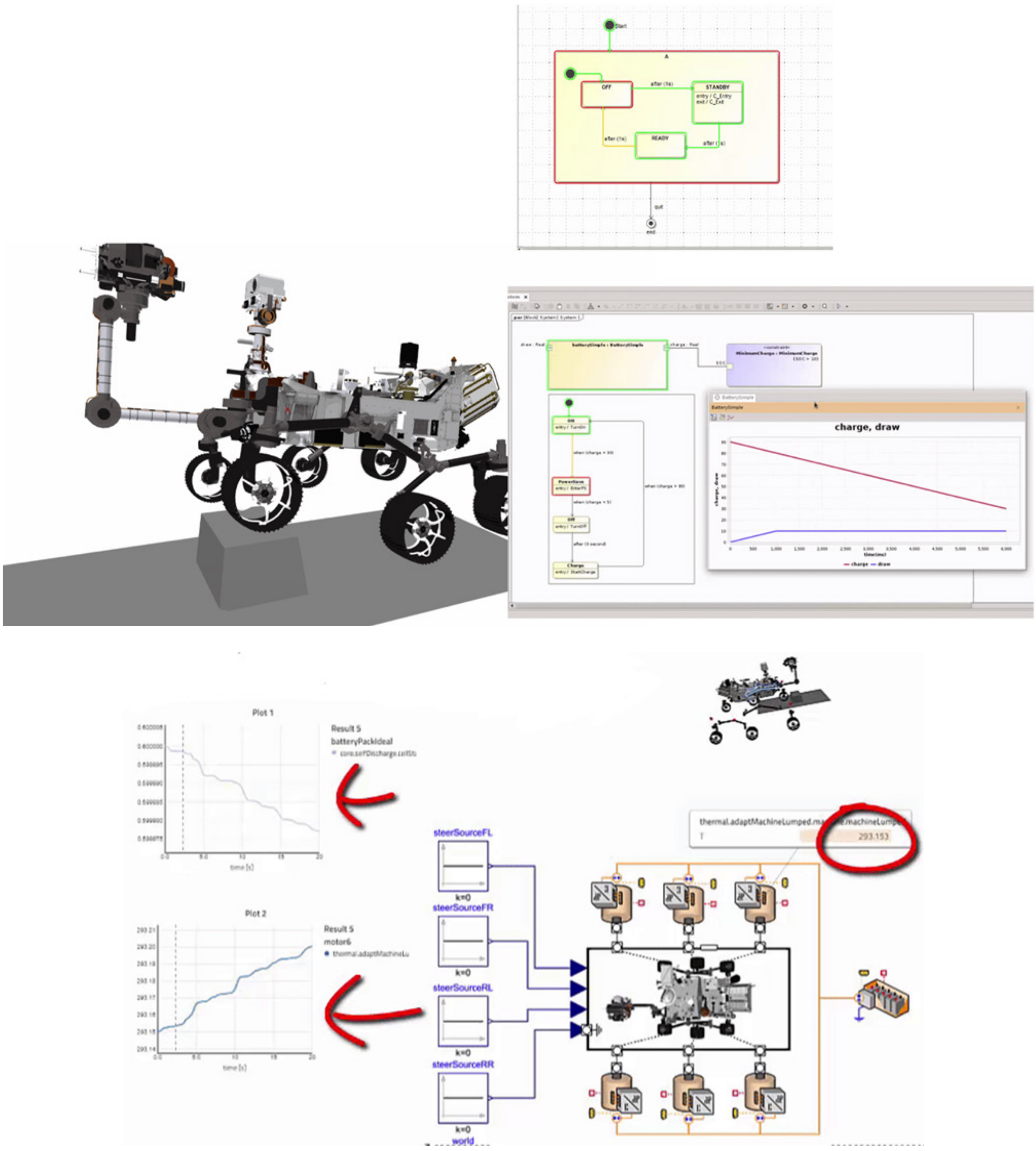


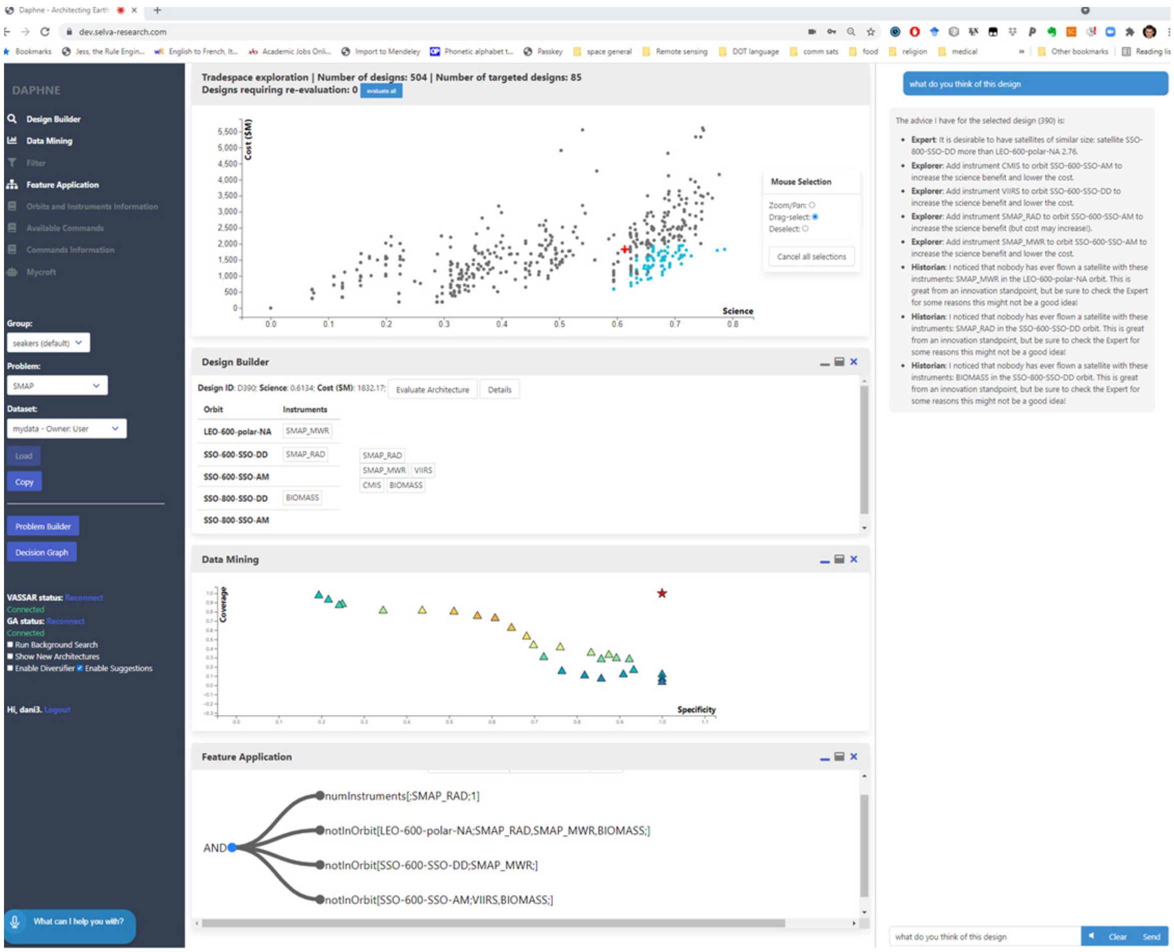
Figure 10. Iterative Phases of the Model-Based Development Process including Multi-Physics Simulation. Enabled by the Dragon Architecture, from [11]

4.5. Cognitive assistants

The developments in artificial intelligence and the computing power available today are enabling the development of cognitive assistants for systems engineering tasks. A cognitive assistant is a machine that acts as a virtual engineer, a senior companion with whom the human engineer works, and that performs engineering tasks that humans are not capable of carrying out effectively or efficiently.

With a cognitive assistant acting as a virtual engineer, the computer moves from being a tool in which we input data, create models, and run simulations, to a highly experienced work companion with whom we co-design, to whom we assign some of our tasks, on whom we rely to better understand what engineering decisions we should make, and who ensures that the knowledge generated in our project is stored within the organization so that other colleagues, and ourselves, can benefit from it in future projects [28]. Instead of evaluating design options for a system, we will ask our virtual companion, *What do you think of this design?, Are we missing a key requirement?, What risk do you think we assume if we do not conduct this test?,* or *Would it cost us much to replace this interface with just a single command?.*

Cognitive assistants allow for the extraction and analysis of large amounts of data from various sources almost instantly and can provide the human engineer with the answers they need directly in natural language. Cognitive assistants can also initiate conversations unilaterally when they believe they can be useful to the human engineer, such as providing suggestions to improve an architecture that is currently being worked on. An example user interface is shown in Figure 11. By interpreting system models and relying on ontologies, cognitive assistants can improve their recommendations and performance when executing these functions.



DAPHNE
Design Builder
Data Mining
Filter
Feature Application
Orbits and Instruments Information
Available Commands
Commands Information
Mycroft
Tradespace exploration | Number of designs: 504 | Number of targeted designs: 85
Designs requiring re-evaluation: 0
Cost ($M)
Science
Mouse Selection
Cancel all selections
Design Builder
Data Mining
Coverage
Specificity
Feature Application
AND
numInstruments[;SMAP_RAD;1]
notInOrbit[LEO-600-polar-NA;SMAP_RAD,SMAP_MWR,BIOMASS;]
notInOrbit[SSO-600-SSO-DD;SMAP_MWR;]
notInOrbit[SSO-600-SSO-AM;VIIRS,BIOMASS;]
What can I help you with?
Problem Builder
Decision Graph
Clear
Send

Design ID: D390; Science: 0.6134; Cost ($M): 1832.17;

Orbit	Instruments
LEO-600-polar-NA	SMAP_MWR
SSO-600-SSO-DD	SMAP_RAD
SSO-600-SSO-AM	
SSO-800-SSO-DD	BIOMASS
SSO-800-SSO-AM	

what do you think of this design

The advice I have for the selected design (390) is:

- **Expert**: It is desirable to have satellites of similar size: satellite SSO-800-SSO-DD more than LEO-600-polar-NA 2.76.
- **Explorer**: Add instrument CMIS to orbit SSO-600-SSO-AM to increase the science benefit and lower the cost.
- **Explorer**: Add instrument VIIRS to orbit SSO-600-SSO-DD to increase the science benefit and lower the cost.
- **Explorer**: Add instrument SMAP_RAD to orbit SSO-600-SSO-AM to increase the science benefit (but cost may increase!).
- **Explorer**: Add instrument SMAP_MWR to orbit SSO-600-SSO-AM to increase the science benefit and lower the cost.
- **Historian**: I noticed that nobody has ever flown a satellite with these instruments: SMAP_MWR in the LEO-600-polar-NA orbit. This is great from an innovation standpoint, but be sure to check the Expert for some reasons this might not be a good idea!
- **Historian**: I noticed that nobody has ever flown a satellite with these instruments: SMAP_RAD in the SSO-600-SSO-DD orbit. This is great from an innovation standpoint, but be sure to check the Expert for some reasons this might not be a good idea!
- **Historian**: I noticed that nobody has ever flown a satellite with these instruments: BIOMASS in the SSO-800-SSO-DD orbit. This is great from an innovation standpoint, but be sure to check the Expert for some reasons this might not be a good idea!

Figure 11. Daphne's user interface, a cognitive assistant to support the architecture of space systems [29]

5. CONCLUSIONS

This chapter has described how advances in computation and digital technologies provide organizations with the opportunity to digitalize (not just digitize) their processes in what is known as the ‘Digital Transformation’. DE is about creating interoperable, rapidly accessible datasets that can be efficiently integrated across the systems lifecycle in what is known as the ‘Digital Thread.’ Declarative languages and Internet based interoperability provide the foundations to manage the data interoperability aspects, and open standards support the technical interoperability. Semantic web technologies can be leveraged to provide the framework for its implementation.

Throughout the chapter, multiple examples have been presented. The digital thread can provide unprecedented traceability across the lifecycle, is an enabler of early V&V, and supports model-based design review. Furthermore, DE gives raise to unprecedented systems engineering capabilities such as tradespace exploration, SBD, MDO, the integration of physics-based models early during system architecture, and even the adoption of cognitive assistants thanks to advances in AI. These all contribute to increasing the effectiveness and efficiency of system development.

REFERENCES

1. "DoD Instruction 5000.89: Test and Evaluation," 2020.

2. V. Singh and K. E. Willcoxy, "Engineering design with digital thread," AIAA/ASCE/AHS/ASC Struct. Struct. Dyn. Mater. Conf. 2018, pp. 1–21, 2018.

3. Z. Scott, "Speaking in Tongues: The Systems Engineering Challenge," INCOSE Int. Symp., vol. 29, no. 1, pp. 836–849, 2019.

4. P. De Saqui-Sannes, R. A. Vingerhoeds, C. Garion, and X. Thirioux, "A Taxonomy of MBSE Approaches by Languages, Tools and Methods," IEEE Access, vol. 10, no. October, pp. 120936–120950, 2022.

5. P. Witherell, S. Krishnamurty, I. R. Grosse, and J. C. Wileden, "Improved knowledge management through first-order logic in engineering design ontologies," Artif. Intell. Eng. Des. Anal. Manuf. AIEDAM, vol. 24, no. 2, pp. 245–257, 2010.

6. A. Patel and S. Jain, "Present and future of semantic web technologies: a research statement," Int. J. Comput. Appl., pp. 1–10, 2019.

7. N. Shadbolt, W. Hall, and T. Berners-Lee, "The semantic web revisited," IEEE Intell. Syst., vol. 21, no. 3, pp. 96–101, 2006.

8. D. Wagner, S. Y. Kim-Castet, A. Jimenez, M. Elaasar, N. Rouquette, and S. Jenkins, "CAESAR Model-Based Approach to Harness Design," in 2020 IEEE Aerospace Conference, 2020, pp. 1–13.

9. J. Gregory and A. Salado, "A Digital Engineering Factory for Students," in Conference on Systems Engineering Research (CSER), Tucson, AZ, USA, 2024.

10. OASIS Open Projects, "Open Services for Lifecycle Collaboration." 2023.

11. R. Karban et al., "Towards a Model-Based Product Development Process from Early Concepts to Engineering Implementation," in 2023 IEEE Aerospace Conference, 2023, pp. 1–18.

12. J. Gregory and A. Salado, "The Digital Engineering Factory: Considerations, Current Status, and Lessons Learned," in INCOSE International Symposium, Dublin, Ireland, 2024.

13. A. M. Madni and M. Sievers, "Model-based systems engineering: Motivation, current status, and research opportunities," Syst. Eng., vol. 21, no. 3, pp. 172–190, 2018.

14. R. Karban, F. G. Dekens, S. Herzig, M. Elaasar, and N. Jankevicius, "Creating system engineering products with executable models in a model-based engineering environment," Model. Syst. Eng. Proj. Manag. Astron. VI, vol. 9911, p. 99110B, 2016.

15. M. A. Bone, M. R. Blackburn, D. H. Rhodes, D. N. Cohen, and J. A. Guerrero, "Transforming systems engineering through digital engineering," J. Def. Model. Simul., vol. 16, no. 4, pp. 339–355, 2019.

16. K. Giammarco and K. Giles, "Verification and validation of behavior models using lightweight formal methods," Discip. Converg. Syst. Eng. Res., pp. 431–447, 2017.

17. R. Stevens, "Digital Twin for Spacecraft Concepts," IEEE Aerosp. Conf. Proc., vol. 2023-March, pp. 1–7, 2023.

18. W. K. Vaneman, R. Carlson, and C. Wolfgeher, "Defining a Model-Based Systems Engineering Approach for Milestone Technical Reviews," 2019.

19. M. R. Blackburn and B. Kruse, "Conducting Design Reviews in a Digital Engineering Environment," Insight, vol. 25, no. 4, pp. 42–46, 2022.

20. V. Romero, R. Pinquie, and F. Noel, "An Open Benchmark Exercise for Model-Based Design Reviews," in IFIP International Federation for Information Processing, 2023, pp. 176–185.

21. R. Pinquié, V. Romero, and F. Noel, "Survey of Model-Based Design Reviews: Practices & Challenges?," Proc. Des. Soc., vol. 2, pp. 1945–1954, 2022.

22. A. M. Ross and D. E. Hastings, "The tradespace exploration paradigm," 15th Annu. Int. Symp. Int. Counc. Syst. Eng. INCOSE 2005, vol. 2, pp. 1706–1718, 2005.

23. N. Shallcross, G. S. Parnell, E. Pohl, and E. Specking, "Set-based design: The state-of-practice and research opportunities," Syst. Eng., vol. 23, no. 5, pp. 557–578, 2020.

24. D. J. Singer, N. Doerry, and M. E. Buckley, "What is set-based design?," Nav. Eng. J., vol. 121, no. 4, pp. 31–43, 2009.

25. C. Small et al., "A UAV Case Study with Set-based Design," INCOSE Int. Symp., vol. 28, no. 1, pp. 1578–1591, 2018.

26. D. Raudberget, "Practical applications of set-based concurrent engineering in industry," Stroj. Vestnik/Journal Mech. Eng., vol. 56, no. 11, pp. 685–695, 2010.

27. H. Chen, "Multidisciplinary Design Optimization (MDO)," Encycl. Ocean Eng., no. July, 2021.

28. A. Salado and D. Selva, "Asistentes Cognitivos en Ingeniería de Sistemas," UEM STEAM Essentials, pp. 1–8, 2021.

29. A. Virós and D. Selva, "From design assistants to design peers: Turning daphne into an ai companion for mission designers," AIAA Scitech 2019 Forum, pp. 1–12, 2019.

Disclaimer/Acknowledgements

Part of the research presented in this chapter was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration (80NM0018D0004).

Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by the United States Government or the Jet Propulsion Laboratory, California Institute of Technology.

CHRISTOPHER DELP

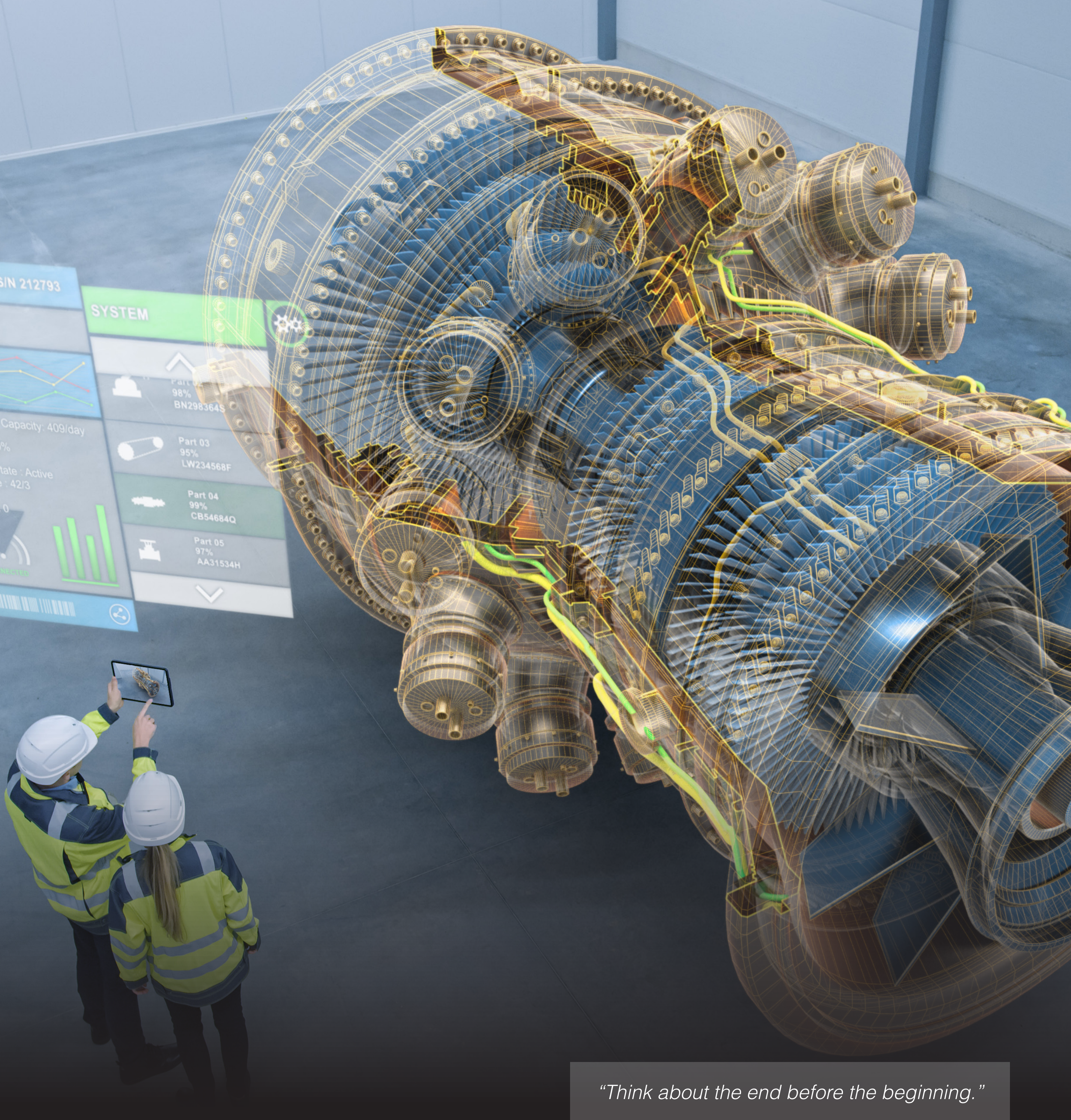
Chris Delp serves as the Technical Group Supervisor for Systems and Software Solutions Engineering, a group dedicated to implementing process-oriented methodologies for the creation of robust software environments designed to deliver advanced model based and digital engineering solutions. In addition, he holds the position of Manager of the Computer Aided Engineering Systems Environment at NASA JPL. This effort is at the forefront of engineering innovation, providing a state-of-the-art environment for system modeling, analytical assessments, and simulation development grounded in Open Engineering principles. In his previous role, Mr. Delp spearheaded the Model Environment Development for the Europa Clipper mission, establishing a Model-Based Engineering Environment and fostering the growth of the Open MBEE community—a collective focused on open-source Model-Based Engineering models, software and frameworks. With a diverse background in Systems Engineering and Software Engineering on JPL Flight Projects, his expertise encompasses system architecture and design, the development and testing of safety-critical software, and the application of Systems Engineering principles throughout the project lifecycle. Mr. Delp is esteemed as a leading authority in Model-Based Systems Engineering (MBSE) and Model-Based Engineering Environments. He holds an MS in Systems Engineering from the University of Arizona.

DR. JOE GREGORY

Dr. Joe Gregory is a postdoctoral research associate at the University of Arizona. His research interests include digital engineering, model-based systems engineering, and the application of semantic web technologies to support engineering. In 2022, he received his PhD in Aerospace Engineering from the University of Bristol for his development of the SysML-based 'Spacecraft Early Analysis Model'. He is the co-chair of the Digital Engineering Information Exchange (DEIX) Taxonomy Working Group.

L. MIGUEL APARICIO

Luis Miguel Aparicio holds an BS/MS in Computer Science and Engineering from the Polytechnic University of Madrid and a graduate certificate in High Logistics Management from CESEDEN. He holds various certifications in logistics, systems engineering, and project management, including Lean Six Sigma, ITIL, INCOSE, PMP, and PRINCE. After gaining professional experience at Coritel, Telefónica, and in the public sector, he joined Isdefe in 1999, where he remains to this day. He has collaborated as an adjunct professor at Rey Juan Carlos University and currently at the European University of Madrid. His professional experience has always revolved around information systems, with a particular focus on logistics. He currently serves as the Head of Logistics Systems Area at Isdefe, leading projects to support the rationalization and modernization of the logistical systems of the Spanish Armed Forces. He also has significant experience in projects related to the NATO Codification System, providing consultancy services in this field and supporting the development and modernization of SICAD, the Defense Cataloging System, a tool used in the National Cataloging Bureaus of Saudi Arabia, Belgium, Colombia, Peru, and Poland. Among the portfolio of projects managed within his area, there are several international consultancy projects, for example, with the European Defence Agency, NATO, the Military Industries Corporation of Saudi Arabia, or the National Cataloging Bureau of Jordan. He has also collaborated with the Armed Forces of Peru, the United Arab Emirates, Chile, and Ecuador. Currently, it is noteworthy the active role he plays in supporting the introduction of disruptive new technologies into Defense logistical information systems, with projects related to digital transformation, artificial intelligence, big data, or blockchain.



"Think about the end before the beginning."

W. Fabrycky

CHAPTER 6

Digital Transformation in System Deployment, Operation, Sustainment, and Retirement

Dr. Kaitlynn Castelle, *University of Maryland Applied Research Lab for Intelligence & Security (kcastelle@arlis.umd.edu)*
Miguel Ángel Coll Matamalas, *Isdefe (macoll@isdefe.es)*

Abstract

This chapter presents the novel capabilities enabled by Model-Based Systems Engineering (MBSE) and digital engineering to support system deployment, operations, sustainment, and retirement. Topics include advancements in the application of digital twin and digital thread technologies to support sustainment of systems through the life cycle and considerations for pursuing digital transformation and the adoption of advanced model-based technology. The chapter discusses the importance of planning for the use of these technologies early in the system life cycle to establish a foundation and strategy conducive to the application of model-based methodologies, automation, virtual reality, artificial intelligence, and other advanced technologies.

Keywords:

System Deployment, Operation, Sustainment, Retirement, Digital Twin, Digital Thread, Automation, Model Based Product Support (MBPS), Proactive Maintenance (CBM), Predictive Maintenance (PdM), Prescriptive Maintenance (RxM), Remaining Useful Life (RUL), Overhaul, Obsolescence Management, Configuration Management, Digital Transformation.

1. INTRODUCTION

Chapter 5 covered the use of digital engineering technology during the system development phase of the system life cycle. This chapter will dive into how a system's digital origin and foundation, and investments made in digital engineering technology enabling digital twin and digital thread, may be used through the later phases of the system's life cycle. The chapter also discusses considerations while pursuing this technology, addressing opportunities and risks through the deployment, operations and sustainment, and retirement of the system, including supporting logistics.

Interest in digital engineering technology, particularly those enabling digital twins and digital thread, has increased in recent years across various industries, specially manufacturing and defense. Three main concepts lay at the core of this interest: digital engineering, digital twin, and digital thread.

Digital engineering is a discipline that leverages advanced technologies, such as Computer-Aided Design (CAD), simulation, and data analytics, to support the design, development, and management of complex systems and products throughout their entire life cycle. It involves creating and integrating digital models, simulations, and data-driven decision-making to optimize design, manufacturing, deployment, operation, sustainment, and retirement processes, including logistics.

A **digital twin** is essentially a virtual representation of a physical object, system, or process (its physical twin) that enables real-time monitoring, analysis, and simulation. The concept of a digital twin allows organizations to have a deeper understanding of their assets, operations, and performance by connecting the physical and digital worlds through digital thread.

The term digital twin has been significantly abused though. While every digital twin will be a digital model, not every digital model necessarily constitutes a digital twin. Ideally, you could substitute a digital twin by its physical twin and vice versa, without noticing the change. This duality becomes very valuable because (1) you can experiment with the digital twin both in development phase and operational testing fairly inexpensively without risking its physical twin (the actual system) and (2) a digital model allows for accelerating discovery of system properties, that is, software simulation often runs faster than physical testing.

Figure 1 illustrates that there are many types of digital twins (virtual product twin, virtual process twin, digital factory twin and digital twin), depending on the phase of the product life cycle.

The **digital thread** refers to the means used for connecting the flow of data and information throughout the entire life cycle of a system or product. It enables effective communication and collaboration among various stakeholders across aspects and stages of the engineering life cycle, to ensure consistency and traceability across activities.

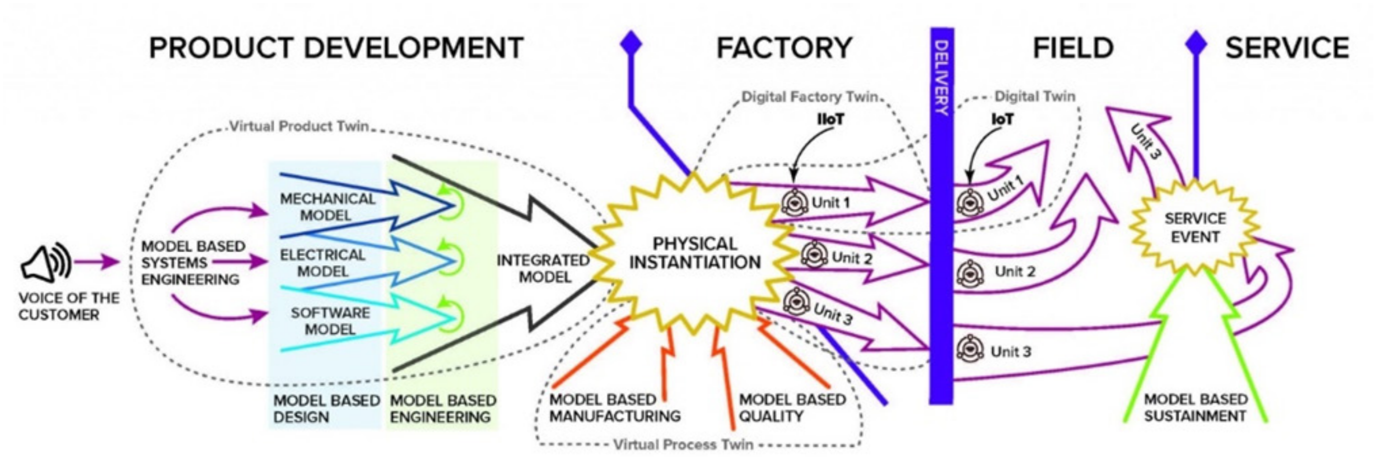


Figure 1. Digital Twins along the life cycle.

A more formal definition of digital thread, provided by the Defense Acquisition University (DAU), is as follows[1]:

> *"An extensible and configurable analytical framework that seamlessly expedites the controlled interplay of technical data, software, information, and knowledge in the digital engineering ecosystem, based on the established requirements, architectures, formats, and rules for building digital models. It is used to inform decision makers throughout a system's life cycle by providing the capability to access, integrate, and transform data into actionable information."*

Each organization's digital transformation journey is unique and may support greater innovation, cost savings, and reliability in their products and systems. However, the technology and applications described in this chapter may not be appropriate or necessary for all organizations or systems of interest (SoI). In addition, many of the expected outcomes may not be achievable if the organization is not prepared to leverage the technology in certain contexts, or if the data models are not accessible to the system operating organization. Furthermore, it should be noted that similar to the application of systems engineering in physical systems, direct involvement of users, operators, maintainers, and other active stakeholders is necessary for the development of the digital twins. However, although the needs of the system designers and developers for the digital twin and digital thread may differ from the needs of the operator during the service phase, the use cases for later stages of the life cycle should be considered early in the system development life cycle.

2. APPLICATION OF DIGITAL TWIN, VIRTUAL ENVIRONMENTS, AND DIGITAL THREAD TECHNOLOGIES IN LATER SYSTEM LIFE CYCLE PHASES

2.1. Digitally transforming system deployment

Overall, digital engineering technology has expedited system deployments, reduced costs, increased agility, and improved the overall quality of deployed systems [1]. Below is a description and examples of some capabilities enabled by digital engineering and its applications during the entry into service.

1. Find this definition through the following link: https://www.dau.edu/glossary/digital-thread (last accessed on 23/04/2024).

2.1.1. Streamlined deployments with automation, virtualization, and hardware-in-the-loop testing

The use of virtualization technology allows for increased predictability in system deployments. Virtualization involves creating virtual versions of hardware, software, storage, or network resources, which may leverage emulation to mimic their behavior. In this way, systems can be deployed first in a virtual environment, reducing the need for excess physical infrastructure. This allows for faster and more efficient validation and deployment, as virtual systems can be quickly replicated and scaled as required [2].

Later, as the actual system is deployed, digital technologies enable the automation of deployment tasks, such as software installation, configuration, and testing. Automation decreases the chances of human error and allows for consistent and repeatable deployments. Furthermore, by facilitating the adoption of DevOps[2] practices (including continuous integration and deployment), some system changes can be automatically built, tested, and deployed to production environments, ensuring a faster and more reliable deployment process.

2.1.2. Quality control

Digital twins and virtual environments embed detailed geometric information that, coupled with associated metadata, helps ensure compliance with design specifications and standards, enabling better quality control during manufacturing and assembly. Detailed geometric information provides the foundation for the development of accurate simulations, analyses, and visualizations. It enables a comprehensive understanding of the physical entity's shape, structure, and spatial relationships within the virtual environment, including additional information such as three-dimensional representation, dimensional accuracy, or material properties and textures, among others. Metadata associated with digital twins and virtual environments can be leveraged to enhance their understanding, management, and utilization, as it consists of descriptive information and properties that characterize and provide context to the digital representation of a physical object or system, such as life cycle information and accurate item identification (e.g., for configuration control).

2. DevOps is a software development methodology used as a set of practices and tools that integrates and automates the work of software development (Dev) and information technologies operations (Ops) as a means of improving and shortening the systems development life cycle.

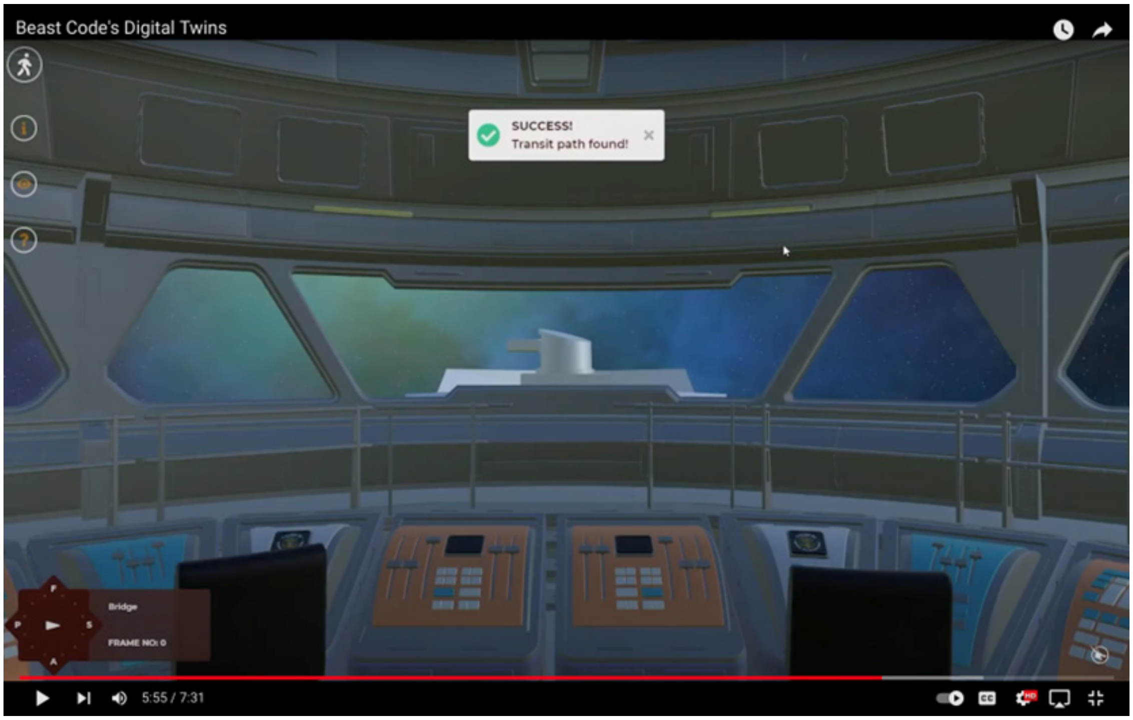


Figure 2. Example of a Digital Twin (link: https://youtu.be/G26mx4TnKyM?si=gR-fqlhmZw0B21-y).

2.1.3. System training and familiarization

Digital environments generated through Augmented Reality (AR) or Virtual Reality (VR) yield immersive learning experiences that can greatly enhance system training and familiarization, enabling personnel to better understand and manage complex systems in a lifelike environment with less reliance on being in proximity of the physical asset (e.g., see Figure 2). When successful, these virtual systems can provide technicians and operators with real-time, on-site maintenance assistance utilizing AR overlays to communicate visual instructions and reference manuals directly on the physical system, resulting in heightened efficiency and accuracy during maintenance activities.

An example of such a method is product model visualization in a digital mock-up, which proves valuable when linking design and logistics supportability information to the geometric information embedded in digital twins. This connection, established by integrating technical and logistics data, creates a spatially aware context that improves comprehension of maintenance tasks [3]. Product model visualization is a concept that involves creating and presenting visual representations of physical objects or systems, typically in a digital format. The primary goal is to convey information about the design, structure, and behavior of a product through graphical and interactive means. In the context of defense systems training and familiarization, for example, product model visualization plays a crucial role in providing an immersive and effective learning experience in a virtual environment. As an example, when coupled with the real system, an aviation maintainer can rely on digital artifacts directly projected and superimposed to the physical environment instead of relying on memorizing 2D diagrams and navigating dark, confined areas to locate specific parts [4]. Digital assets used in this way do not only improve training effectiveness and efficiency, but also actual task outcomes.

Despite their potential, challenges to adoption of these technologies remain, including but not limited to, ergonomic considerations and readily available and affordable VR/AR solutions [5]. As an intermediate alternative, digital artifacts can be provided to the operator or maintainer in the form of Interactive Electronic Technical Publications (IETP) or Manuals (IETM). These are digital formats of technical documents that provide comprehensive information, instructions, and guidance for the operation, maintenance, repair, and troubleshooting of complex systems or equipment [6]. These publications are designed to replace or supplement traditional paper-based manuals with electronic versions that offer various interactive and multimedia elements to enhance user experience and facilitate efficient learning and comprehension. Compared to printed manuals, IETMs are becoming more and more widespread because they are interactive and easy to use and maintain, as well as provide insight into the system that would otherwise be difficult to infer.

2.2. Digitally transforming operations and sustainment

2.2.1. Model Based Product Support and configuration management

Model Based Product Support (MBPS) consists of applying the principles and concepts of digital engineering to Integrated Life Cycle Support (ILS). In that sense, it refers to the modelling of any aspect related to ILS, including its supportability, reliability, and safety characteristics, both in terms of design and life cycle processes. MBPS leverages digital models, simulations, the interoperability of information systems and data to enhance the efficiency and effectiveness of supporting systems to improve operational readiness and to optimize sustainment and life cycle management throughout their useful life. Particularly, system models generated during system development are used during the operation and servicing phase to help understand the system functionality and its external and internal interfaces.

Data continuity, which is essential to support a coherent transition between the models generated and/or used in system development and those generated and/or used in MBPS, is enabled by the digital thread. On the other hand, maintaining an accurate configuration baseline of an asset or system remains critical in order to ensure it can be logistically supported. Applying MBPS by using authoritative sources of truth, it becomes no longer necessary to manually maintain two different configurations, one for design and one for logistics, throughout the life cycle, by incorporating all relevant logistics information directly onto the configured design elements themselves.

2.2.2. Proactive maintenance strategies enabled by digital twins and digital thread

Traditionally, both corrective and preventive maintenance have been the most widely used strategies of maintenance. In corrective maintenance, repairs are carried out after the problem has occurred. On the contrary, preventive maintenance focuses on preventing potential failures before they occur by executing scheduled maintenance actions, such as inspections, routine servicing, and component replacements (generally based on use patterns: running hours, cycles, number of starts, etc.). However, advancements in sensors and monitoring technologies and the resulting development of digital twins and the digital thread enable a turn towards data-driven maintenance approaches. These approaches rely on the use of algorithms to make predictions and recommendations to optimize system's reliability and the operational efficiency of the assets.

By moving from reactive to data-driven maintenance, organizations can minimize asset downtime and reduce maintenance costs, since potential failures modes can be detected and addressed before they occur, preventing operation interruptions and emergency repairs. Three main proactive maintenance strategies can be distinguished:

- **Condition Based Maintenance (CBM).** CBM focuses on monitoring the real-time condition of equipment by using various sensors and data collection tools to assess the current state of the equipment. CBM uses thresholds, or set conditions, to trigger alarms or maintenance actions when deviations from normal operating conditions are detected.

  CBM typically focuses on a specific set of parameters relevant to the equipment being monitored, and continuous monitoring of a few critical factors that directly impact immediate performance or reliability. The primary objective of CBM is to monitor the current condition of equipment and take timely actions to prevent imminent failures or issues.

  CBM is used in a broad variety of applications, including manufacturing, defense, industrial control systems, healthcare, to diagnose impending failure modes, thereby reducing maintenance costs, improving reliability, availability, and safety, extending time between overhauls (compared with traditional preventive maintenance), and reducing unnecessary downtime. With the ability to continuously monitor the asset's health and performance metrics, system data can be collected to develop models enabling proactive detection of failure modes.

- **Predictive Maintenance (PdM).** PdM also involves monitoring equipment condition, but it primarily focuses on using real time data analytics and historical data to predict when maintenance action is needed. PdM analyzes patterns and trends in data to forecast potential.

  Although CBM and PdM share similarities and often use similar technologies, there are some distinctions in them: CBM initiates maintenance actions based on the real-time condition of the equipment and focused on a specific set of parameters, while PdM aims to schedule maintenance activities just before the equipment is likely to fail, optimizing maintenance timing to prevent failures and minimize downtime.

  PdM often involves more comprehensive data analysis than CBM, using a broader range of data points and indicators to predict failures. It may incorporate advanced analytics, Artificial Intelligence algorithms, like machine learning, and other sophisticated algorithms to forecast failures, while CBM uses them less intensively.

- **Prescriptive Maintenance (RxM).** Prescriptive maintenance goes beyond PdM. It does not only predict potential equipment failures but also prescribes specific actions to prevent those failures or mitigate their impact. This approach integrates predictive analytics with automated decision-making systems to provide precise recommendations or prescriptions for maintenance actions. These recommendations can include detailed instructions on maintenance tasks, repairs, adjustments, or operational changes. RxM utilizes sophisticated algorithms of artificial intelligence and Big Data techniques to analyze extensive sets of data. It examines historical and real-time data to predict potential failures and determines the best course of action to avoid or address these issues. In the context of RxM, prognosis plays a significant role in determining the appropriate actions to be recommended. Prognosis refers to the estimation or prediction of future conditions or events based on the analysis of current and historical data. In RxM, prognosis involves forecasting potential failures, estimating the Remaining Useful Life (RUL), and predicting the future health and performance of assets or equipment [6].

Digital twins become central across maintenance approaches thanks to their capabilities to enable data-driven applications: real-time monitoring and analysis, predictive and prescriptive analytics, simulating "what-if" scenarios, health monitoring and prognosis, and optimization analysis. In predictive maintenance, digital twins continuously collect real-time data from sensors embedded in the physical assets and utilize historical data to analyze trends, patterns, and anomalies. By leveraging this information, digital twins facilitate predictive analytics and forecasting of potential failures or performance degradation. In prescriptive maintenance, digital twins play a crucial role, by providing not only predictive insights, but also actionable recommendations. By simulating different scenarios and analyzing data, digital twins can suggest specific maintenance actions or strategies to prevent failures or optimize asset performance. They also aid in prescribing the most effective actions to be taken based on predictive analytics and simulations.

Finally, it should be noted that digital twins can also be of value to support preventive maintenance, since they can assist in establishing asset's baseline conditions, continuously monitoring asset conditions, and identifying deviations from normal operating parameters by tracking equipment health.

2.2.3. Modernizing Maintenance, Repair and Overhaul (MRO)

As assets age, there may be a need for component replacements, upgrades, or redesigns. The geometric information embedded in digital twins, along with metadata on materials and manufacturing tolerances, aids in identifying compatible replacement parts and planning seamless upgrades. This can help in anticipating the integrability of new components with the existing system, avoiding compatibility issues and potential disruptions.

Integrating CAD enables effective collaboration among stakeholders, seamlessly incorporating modifications from an asset's life cycle into the CAD model for future reference. CAD models created during the design phase become the basis to provide a visual representation for the physical asset in the digital twin. This digital representation, with precise geometric data, later allows maintenance teams to visualize and understand the asset's components and assemblies, aiding in troubleshooting and planning maintenance activities, and additionally supporting communication of possible configuration changes to the deployed asset.

Alongside the geometric representation, digital twins incorporate essential metadata, such as material properties, manufacturing tolerances, and part numbers. These metadata provide crucial information for manufacturing, assembly, and future maintenance activities. Particularly, it aids maintenance personnel in quickly identifying the right components, accessing relevant information, and performing repairs more efficiently, reducing system downtime. For example, maintenance personnel can access the virtual twin to identify the exact components, understand their dimensions, and review historical maintenance records. This enables more efficient and targeted maintenance activities, minimizing downtime and extending the asset's useful life. Similarly, they can utilize the virtual twin to understand the system's physical layout, access metadata for part identification, and troubleshoot issues without direct access to the physical asset.

2.2.4. Obsolescence management

Obsolescence management is critical for ensuring the continuous operational readiness, sustainability, and effectiveness of certain systems over their extended service lives, which can often span several decades. Effective obsolescence management requires a proactive and holistic approach, integrating various strategies throughout the system's life cycle to ensure operational readiness, reduce risks, and manage costs associated with maintaining and supporting these systems over time. The use of digital models combined with some emergent manufacturing technologies, provides new tools for obsolescence management.

Geometric information embedded in digital twins makes it easier to reverse engineer certain parts and manage obsolescence, as they are exact replicas of their physical twins, enabling their manufacturing. Integrating CAD and Computer-Aided Manufacturing (CAM) systems streamlines product design and manufacturing processes, which is especially helpful for modernization and dealing with obsolete parts. This link can be further reinforced by novel manufacturing processes, such as additive manufacturing (also known as 3D printing), as the link between design and manufacturing can be entirely established as a digital thread.

2.3. Digitally transforming system retirement

Relative to other life cycle phases, system retirement is the least developed with respect to digital transformation. For some situations, it may be advantageous to preserve end use data from retired systems and components that have reached end of life, such as performance data, as the same component in the retiring system may continue to be used in other similar applications. As an example, if you retire a single aircraft, a certain actuator may still have usable life for use in other aircraft. Performance data may also be used for future simulations to influence future redesigns. The digital thread can also help in better identifying and dealing with components with special considerations for disposal, such as hazardous materials or classified hardware and software.

Furthermore, the retirement stage of a given system for one organization could be, simultaneously, the start of an acquisition program for another organization (e.g., when an organization acquires a second-hand aircraft from another organization). In this case, leveraging or even accepting the digital twin and digital thread developed with the system can be challenging for the organization who is receiving the assets, given that their digital transformation process may not be mature enough.

By leveraging digital twins and digital threads in the retirement stage, organizations can enhance sustainability, compliance, and efficiency in the decommissioning process. These technologies contribute to responsible and environmentally friendly practices while providing valuable insights for ongoing improvement in the development and life cycle management processes.

2.4. Examples of novel capabilities digitally transforming the system life cycle

2.4.1. Cloud-enable collaborative model development

Collaborative modelling tools facilitate cross-functional collaboration, which may be integrated with Product Life cycle Management (PLM) CAD suites to allow engineers, designers, and other stakeholders to work together seamlessly. Changes made in one tool can be communicated to the other, promoting communication and alignment between different teams.

The use of collaborative model development technology supports formalizing model planning, development, integration, curation, and using models for engineering activities and decision-making across the life cycle. As opposed to formal document-based approaches in legacy systems engineering practices, model-centric organizations leverage collaborative environments where teams can define and plan the creation of models to support engineering activities, ensuring a structured and auditable approach. This technology facilitates the formal development of models by providing tools incorporating various techniques and algorithms, integrating data from different sources, and curating models by refining and optimizing them over time. Significant productivity gains can be made with emphasis on the model-based definition and evolution from the legacy document-based approaches.

2.4.2. Data aggregation and integrated information systems

For most organizations, there is a significant labor burden associated with manual processes and disparate systems used to collect and use data. The modern information system enabled by digital threads may connect various disciplines and stages of the asset life cycle. Virtual product and process models, represented by digital twins and the digital thread, can enable analysis, performance optimization, decision-making, operations, by integrating data and information for downstream users.

Digital twin technology allows real-time (or near-real time) virtual representations of the current asset state, in which databases, repositories, and system models from multiple disciplines may be integrated. A well-architected digital thread can provide seamless integration and accessibility of relevant data across the asset's entire life cycle, from design to operation and sustainment to enable functionality in the digital twin.

2.4.3. Sensors and IoT

The emergence of novel capabilities in sensor technology and the implementation of the Internet of Things (IoT) have revolutionized the way in which digital twins are developed, offering a seamless integration of the physical and digital domains. In parallel, advances in microfabrication and nanotechnology have led to the creation of highly sensitive, compact, and energy-efficient sensors capable of detecting a wide array of physical phenomena, necessary to develop more realistic digital twins.

Concurrently, innovations in IoT technology, including enhanced connectivity options, robust data processing, and cloud computing, facilitate the reliable transmission and analysis of vast amounts of data collected from these sensors. This IoT infrastructure enables the real-time synchronization of physical assets with their digital counterparts, creating dynamic, virtual models that accurately reflect the physical world.

2.4.4. Cloud computing

The emergence of cloud computing has revolutionized system deployments. Cloud platforms provide on-demand access to resources, enabling organizations to easily scale their systems and deploy them globally. Cloud-based deployment models, such as Infrastructure as a Service (IaaS) and Platform as a Service (PaaS), simplify the deployment process for organizations by abstracting underlying infrastructure concerns. Specifically, they reduce the need to heavily invest in physical servers and data centers.

Infrastructure as a Service (IaaS) is a category of cloud computing services that provides virtualized computing resources over the internet. In an IaaS model, users can rent or lease various infrastructure components, such as virtual machines, storage, and networking, instead of investing in and maintaining their own physical hardware.

Platform as a Service (PaaS) is a cloud computing service model that provides a platform allowing customers to develop, run, and manage software applications without the complexity of building and maintaining the underlying infrastructure. In a PaaS model, the cloud provider delivers a comprehensive and integrated platform that includes development tools, runtime environments, and other services necessary for building, deploying, and scaling applications.

3. CONSIDERATIONS FOR PURSUING DIGITAL TRANSFORMATION ACROSS THE LIFE CYCLE

3.1. Overview

Digital transformation across the life cycle demands a comprehensive approach to technical data management strategy. This strategy should guide the acquisition, management, and maintenance of the technical data and software necessary to support a system from inception to retirement. Considerations such as safeguarding intellectual property and fostering competition should be central to the strategy. By securing the required data and rights, organizations can enhance system design understanding, optimize operations across varied environments, and unlock potential cost efficiencies in acquisition and sustainment.

The effectiveness of digital models and digital twin technology is deeply linked to their accessibility and usability by operational users. These stakeholders depend on intuitive tools that allow for the realistic simulation of digital twins, alongside seamless access to comprehensive logistics and technical data. This data should be open, interoperable, adaptable, and accessible across cloud and edge computing environments, necessitating a collaborative effort that integrates the insights of both technical teams and operational users right from the design phase.

Ideally, a single digital twin would suffice for all life cycle phases of a system, incorporating all necessary information, models, and behaviors. However, embedding operational and sustainment capabilities effectively in the digital twin requires attention during the concept, development, and production stages. The transition towards a unified digital twin demands a gradual integration of operational and product support models. Until such comprehensive integration is achieved, distinctions among design, operational, and sustainment digital twins remain essential. Without careful implementation, there is a risk of creating a digital twin that serves well for design and production but falls short in operations and sustainment due to an inability to replicate its physical twin's behaviors accurately. Thus, a true digital twin must effectively embody both design and operational and support functionalities.

The development of a holistic digital twin, equipped with the necessary capabilities and functionality to support desired use cases, can be regarded as a standalone software development project. This project should proceed in tandem with the development of the System of Interest, highlighting the need for a multidisciplinary approach. Achieving the goals of digital transformation necessitates a holistic view that encompasses people, processes, technology, data, and strategic objectives, all while maintaining a focus on establishing a durable digital thread that interconnects all elements of the life cycle.

Some guidance to implement and/or adopt digital engineering for later phases of the system life cycle is provided in the following sections.

3.2. When in doubt, start with a pilot

There is significant upfront work required in preparation for downstream use of digital twin and digital thread, including in the system concept and development phases. It can be overwhelming to figure out where to start, which use cases to prioritize, and what resources will be needed to mature the capability.

For many organizations, digital transformation efforts begin with a pilot project, followed by what is typically a slow, challenging transition from successful pilot projects to scaled operations [8]. Successful transformation at scale requires thoughtful planning and coordination to holistically implement the new technology within the context of the organization. The transformation should be aligned to strategic goals and commitment to from leadership to critically evaluate legacy processes, frameworks, and architectures and make appropriate investments in driving digital transformation objectives. It also involves leadership commitment to empowering innovative teams, and fostering a culture that is open to experimentation and capable of learning from failure.

3.3. Be cognizant of technical data rights

Organizations must plan in advance for the technical data rights needed to be acquired if the system is designed by an external entity. If this work is not done upfront, many digital transformation opportunities may not be achievable due to proprietary or incompatible formats, inaccessible data, or cost of rework for the system to produce desired data. In all projects, sponsors should anticipate technical data required for life cycle activities and establish rights or options to purchase data in contracts and service level agreements [9], as well as consult with experts regarding the appropriate data and technology standards to invoke in contract specifications.

3.4. Leverage standardization.

It is essential to adopt semantically rich, open, and accessible data standards, which facilitate interoperability, data linkage, and contextualization that all stakeholders can use and build upon for their needs. Emphasis on open data standards cannot be overstated. In order to have comprehensive data integration and offer a holistic view of the asset's history, current state, and simulate future states, the digital thread must support integration of data from different sources, such as design data, manufacturing data, sensor data, maintenance logs, and operational data.

Various standards development organizations (SDOs) have developed specifications to address interoperability in digital twins [10]. Some examples are provided in Table 1. Furthermore, the domain of digital twins has witnessed a growing expansion of open-source activities, particularly in the development of digital twin platforms and data management. These open-source initiatives contribute to collaboration, innovation, and the wider adoption of digital twins.

Standards Development Organization (SDO)	Scope
ISO/TC 184	Establishes industrial data standards across different domains, including manufacturing, industrial automation, and information systems to ensure the compatibility and interoperability of digital twins in the smart factory field.
IEEE P3144 Digital Twin Working Group	The Standard for Digital Twin Maturity Model and Assessment Methodology in Industry defines a digital twin maturity model for industry, including digital twin capability domains and corresponding subdomains. This standard also defines assessment methodologies, including assessment content, assessment processes, and assessment maturity levels.
The 3rd Generation Partnership Project (3GPP)	Focused on developing standards for 5G networks, which offer the high-speed and reliable communication capabilities required for digital twins.
The Open Geospatial Consortium (OGC)	Manages geospatial information standards, which are crucial for digital twins in smart cities and other domains.
IEC TC65	Focuses on interoperability standards, specifically in the smart factory context. Its efforts help to harmonize communication and data exchange between various components of digital twins within the manufacturing domain.
oneM2M	A global initiative that standardizes service layer IoT platforms, providing common service functions that are essential for the effective operation of digital twins.

Table 1. Standardization in digital twin technology.

3.5. Connect traceability and configuration management across the life cycle

Organizations must plan for traceability to ensure data is properly managed and easy to find, and that data flows and interfaces are maintained in operations and sustainment for the delivered systems. This may be difficult to invoke contractually even in a highly regulated industry, particularly in complex systems designed with disparate development processes. Furthermore, technical and logistics information of a system may reside and be maintained in disparate systems. Linking the Logistics Supportability Analysis Record (LSAR) databases and underlying data sets requires a blueprint for establishing and maintaining these connections, especially in distributed and disconnected environments.

3.6. Intentionally enable automation and analytics

An organization undergoing digital transformation may seek sensor technology and monitoring datasets to generate operational profiles or support development of behavior models. Monitoring technology may already be present in the system, displaying the condition of the asset or its behavior. Modelling and simulation of the remote asset or fleet is made possible by combining real-time data with historical and contextual information. In practice, this requires significant data capture and data usability, including careful planning to maintain linkages between authoritative data sources and robust methods to maintain links between authoritative data and automated methods to ingest multiple sources and formats in secure environments. However, lack of a pre-existing process foundation and the need to streamline or transform existing processes are significant barriers for organizations to adopt automation [11].

3.7. Actively manage knowledge to support operations & sustainment

When introducing new technology in a production system, the workforce usually undergoes training for a smooth system deployment. This generally requires unifying relationships, processes, and data across the production system operations, many of which cross organizational boundaries. Digital threads can facilitate the transfer of knowledge and lessons learned as an organization integrates and transitions from one asset to another. Historical data, maintenance actions, and failure analysis from similar assets can be shared and used to improve predictive maintenance strategies. Furthermore, digital threads can also enable collaboration and knowledge sharing. Authorized personnel, such as operators, engineers, or subject matter experts, can access the digital thread to share insights and knowledge, exchange information, maintain data libraries, or provide support. This collaborative environment enhances problem-solving capabilities and facilitates decision-making processes.

3.8. Implementation impacts on life cycle

Developing digital twins for complex systems presents multifaceted challenges, requiring a strategic approach from the outset. The development of digital twins generally begins with a focus on design and production, but this approach must evolve so that the digital twin can used throughout the entire system life cycle, from deployment through operations, maintenance, and eventual retirement.

The inherent value of digital twins lies in their ability to enhance the availability, accessibility, and accuracy of information. This is particularly true when data can be contextualized within the operational environment of users. Such capabilities, conceived at early life cycle stages, are instrumental in reaping long-term benefits. They facilitate not just a smoother integration of new technologies but also support the workforce as it navigates shifting paradigms. Adjusting traditional program and project management practices and making upfront investments in a robust digital infrastructure are essential steps in this direction.

A critical phase in this progression involves augmenting the digital twin traditionally employed for design and production with functionalities that are necessary to sustain the system once it has transitioned into operations. In line with systems engineering principles, engaging end-users, operators, and maintainers from the beginning in drafting operational concepts, defining scenarios, and setting requirements is key to success.

4. CONCLUSIONS

Digital engineering, underpinned by Product Life Cycle Management (PLM) and Model-Based Systems Engineering (MBSE) frameworks, plays a crucial role in the deployment, operations and sustainment, and retirement phases of system life cycle management. These methodologies facilitate the creation of digital twins and digital threads, which serve as dynamic virtual models mirroring real-world assets throughout their service life. Such digital foundations enable continuous evolution and improvement of systems by providing detailed blueprints for sustainment activities, such as maintenance. This comprehensive approach ensures that systems not only meet initial requirements but also adapt to future needs, thereby extending their utility and enhancing overall performance.

The implementation of digital engineering principles faces challenges, including organizational resistance to change, outdated infrastructure, and skill gaps. Overcoming these obstacles requires a focused strategy that highlights the transformation's unique benefits, such as improved efficiency and streamlined operations. Additionally, the rapid pace of technological advancement necessitates a flexible and collaborative digital transformation strategy, allowing organizations to explore innovative solutions and adapt to new challenges. Successful digital transformation also involves treating data as a strategic asset, establishing strong data governance, and ensuring data quality and security. By prioritizing these elements, organizations can maximize the benefits of digital engineering, leading to more reliable, cost-effective, and high-performing systems throughout their life cycle.

REFERENCES

1. Soybel, J. Designing a Make vs. Buy Strategy for Expendable and Attritable Aircraft Engine Development, 2021. Doctoral dissertation, Massachusetts Institute of Technology.

2. Singh, J., & Walia, N. K. A comprehensive review of cloud computing virtual machine consolidation, 2023. IEEE Access.

3. Marino & Castelle, 2021. https://www.amentum.com/blog/part-1-digital-engineering-supports-asset-availability-and-sustainment/.

4. Mcneely, J. R. X-Ray Vision: Application of Augmented Reality in Aviation Maintenance to Simplify Tasks Inhibited by Occlusion, 2022. Doctoral dissertation, Monterey, CA; Naval Postgraduate School.

5. Fingas, J. Microsoft's HoloLens headsets are giving US Army testers nausea. Engadget. Published 13 October 2022: https://www.engadget.com/microsoft-hololens-fails-us-army-tests-135010970.html.

6. Bolton, M. T. W., Waterworth, S. N., & McClurg, R. J. Enabling, Equipping and Empowering the Support Enterprise through Digital Transformation. (2018, October) In Conference Proceedings of INEC.

7. D. Galar, K. Goebel, P. Sandborn & U. Kumar. Prognostics and Remaining Useful Life (RUL) Estimation, 2022. Taylor Francis, 3: p. 89–134 (DOI: 10.1201/9781003097242).

8. Henderson, K., McDermott, T., Van Aken, E., & Salado, A. Towards Developing Metrics to Evaluate Digital Engineering. Systems Engineering, 2023. 26, p. 3-31. doi:https://doi.org/10.1002/sys.21640.

9. Thompson, G. E., & McGrath, M. Technical Data as a Service (TDaaS) and the Valuation of Data Options, 2019. Acquisition Research Program.

10. Song, J., Le Gall, F. Digital Twin Standards, Open Source, and Best Practices, 2023. In: Crespi, N., Drobot, A.T., Minerva, R. (eds) The Digital Twin. Springer, Cham. https://doi.org/10.1007/978-3-031-21343-4_18.

11. Lyjack, 2019. https://www.linkedin.com/pulse/how-digital-coach-can-help-improve-production-systems-craig-lyjak.

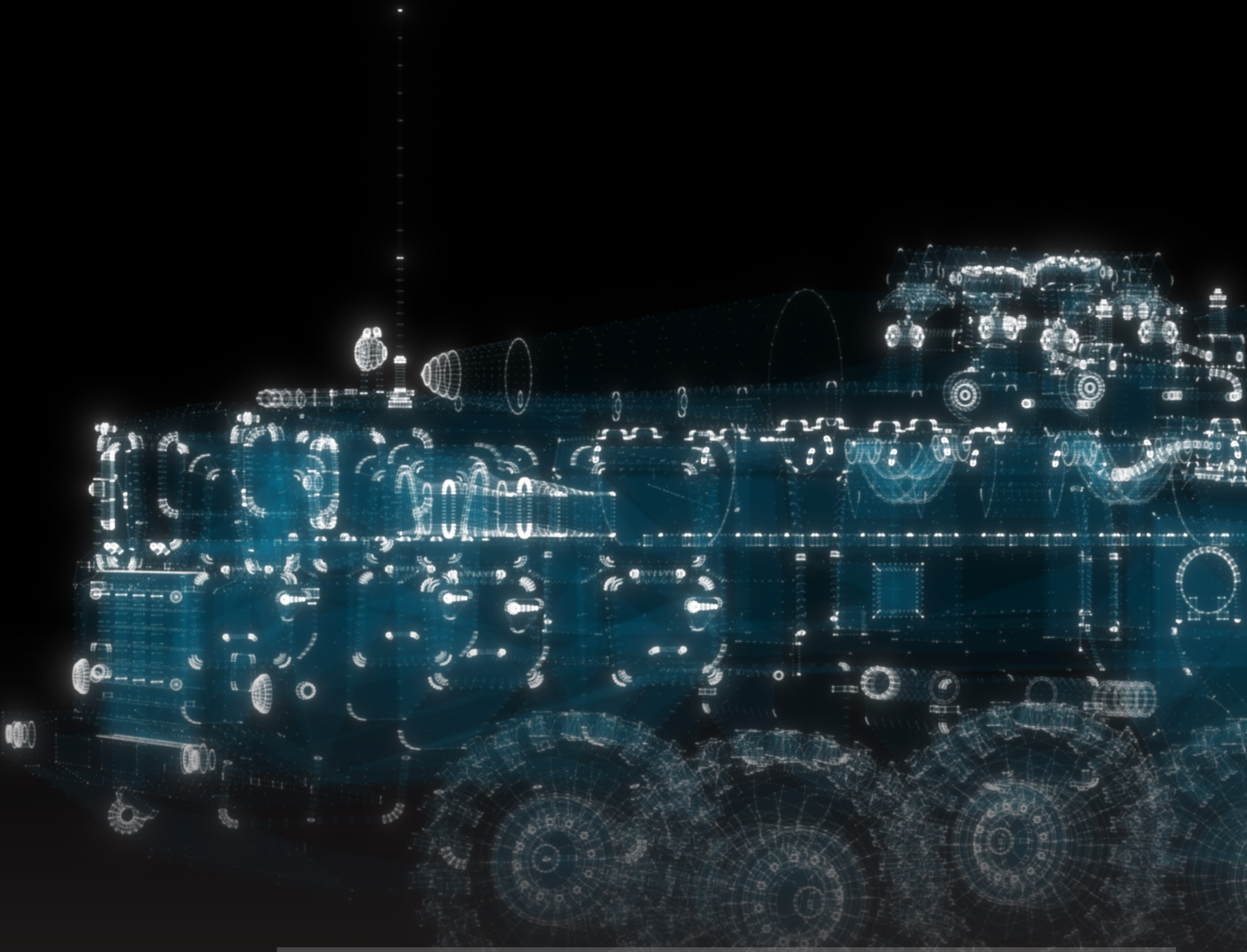
BIOGRAPHIES

DR. KAITLYNN CASTELLE

Dr. Kaitlynn Castelle is an Associate Research Engineer in the Acquisition & Industrial Security mission area with the University of Maryland Applied Research Laboratory for Intelligence and Security. Previously, she worked for the defense contracting industry, where she has served as a Data Scientist and agile product manager supporting the COLUMBIA Submarine Program Office. Her high visibility work in life cycle analysis leveraging Monte Carlo simulation supported investments in advanced construction in the Submarine Industrial Base. She is a co-founder and Program Manager of the Navy's Project Blue digital engineering innovation cell, focused on advancing life cycle capabilities in support of the sea-based strategic deterrence mission in collaboration with other DoD programs. Prior to her work in the defense sector, she served as an adjunct faculty member at Old Dominion University. She obtained her BS in applied mathematics and MS and PhD in engineering management from the Old Dominion University.

MIGUEL ÁNGEL COLL MATAMALAS

Miguel Ángel Coll Matamalas is a Systems Engineer at Isdefe. He is currently supporting the Logistics Support Command of the Navy in the development of logistic doctrine, and activities related to obtaining Integrated Logistic Support for ongoing Acquisition Programs. Specifically, he is involved in the development of the Digital Twin of the F-110 Program. He also participates in the NATO CNAD/LCMG/WG1, responsible for reviewing the ALP-10 standard (NATO Guidance for Integrated Life Cycle Support). Miguel Ángel holds an BS/MS degree in Naval Engineering from UPM (Universidad Politécnica de Madrid) and has completed an Executive MBA at IESE (Universidad de Navarra). In his previous professional stage, he has held various positions related to maintenance and asset management, both in the maritime sector and in water infrastructure. With the aim of applying this previous experience in different areas of sustainment, Miguel Ángel works on improving system supportability through design influence, digital twin development, and the applicability of Prescriptive Maintenance (RxM) and Model-Based Product Support (MBPS).



“Systems engineering requires knowledge and skill. It is impossible to acquire systems engineering skill in a short course or workshop! […] It is impossible to learn systems engineering from a systems engineering tool.”

A.W. Wymore

EPILOGUE

The previous six chapters are evidence of the profound transformation that the landscape of systems engineering is facing. Over the past three decades, we have witnessed a remarkable journey of growth, innovation, and maturation of the field. We have seen a considerable expansion of knowledge, the embracing of new methodologies, and the breaking of traditional boundaries that once confined our understanding and application of systems engineering principles. The field has changed in ways that were unimaginable at the time of Isdefe's "blue books'" publication.

The conclusion of this first monograph on modern systems engineering marks a new chapter in the dissemination of the discipline in Spain. This monograph marks the beginning of the new "blue books" series. It is both a testament to the progress made and a compass for the future. It encapsulates some key topics of the current state of practice in systems engineering, highlighting nascent areas and innovative practices that are shaping the future of the field. The diversity of topics covered—from the application of model-based systems engineering (MBSE) to the integration of artificial intelligence and digital twins in system lifecycle management—illustrates the multifaceted nature of systems engineering and its significance in a rapidly changing world. In the future, we intend to release additional monographs in the series to dive deeper into each of these topics. We also conceive the dissemination of these advances as opportunities for Isdefe engineers to succeed in the challenges they face, in the technically advanced programs of our society, particularly in these times of growing insecurity and uncertainty.

It is with a sense of pride and optimism that we release this monograph into the world. Our aim is for it to serve as a starting point for practitioners, guiding them through the complexities of modern engineering challenges and inspiring them to push the boundaries of what is possible in systems engineering. We extend our deepest gratitude to all who have contributed to this work, and to the readers, who have the challenging task of adopting and evolving the modern systems engineering practices we have presented here.

Let this epilogue not signify the end, but rather the commencement of a journey towards new frontiers in systems engineering and its wider adoption. May the insights contained within these pages inspire you, driving forward the development of engineered systems that are not only fit for purpose but also resilient and sustainable in the face of future challenges. Together, we stand on the threshold of a new era, ready to explore, innovate, and shape the future of systems engineering for the betterment of society and the world at large.

Dr. Alejandro Salado
The University of Arizona

Isdefe
C/ Beatriz de Bobadilla, 3
28040 Madrid
Tel.: +34 91 411 50 11
Email: general@isdefe.es
www.isdefe.es

Isdefe

CUADERNOS DE Isdefe