



Architecting Engineering Systems

Designing Critical Interfaces

Marija Jankovic and Andreas M. Hein

Contents

Introduction	2
System Architecture Definitions	3
System Architecting	8
Overall System Architecting Process	8
System Definition Phase	9
Identification of Stakeholder Needs	10
High-Level Requirement Definition	13
System Architecture Modeling Principles	14
Implications and Challenges of Systems Architecting for Engineering Systems	16
Different Company Strategies Articulated around System Architectures	16
Integrating Systems: Notion and Challenges Related to System Architecting of System of Systems	18
Toward Product Service System Architectures	20
Conclusions	21
References	22

Abstract

System architecture is one of the key concepts in designing engineering systems. It relates business strategy and socio-technical system development. System architecture is critical in designing engineering systems as it is a focal point where novel designs are discussed, often in the form of integrating new technologies into existing system architectures. A key aspect of addressing system architecture is identifying, modeling, and managing critical interfaces. Many studies underline that the success of a development project is based upon managing critical interfaces successfully. Several research domains have been actively contributing to supporting the system architecting phase, developing different system architecture modeling approaches, integrating critical interface

M. Jankovic (✉) · A. M. Hein

Centrale Supélec, Laboratoire Genie Industriel, Université Paris Saclay, Gif-sur-Yvette, France

e-mail: marija.jankovic@centralesupelec.fr; andreas-makoto.hein@centralesupelec.fr

modeling, and proposing different system architecture decision support methods and tools. The objective of this chapter is to give an overview of overarching objectives and difficulties in system architecture design and to discuss existing methods and tools both in the literature and in practice. Due to novel challenges in design, such as autonomous vehicles, discussions on new types of architectures have begun, and we provide an overview of existing challenges and potential new domains.

Keywords

Engineering systems · Engineering system design · Interfaces · System architecting · System architecture · System modeling · System of systems

Introduction

In early 1960, the USA decided to pursue the Apollo Program, an engineering system of undeniable historical relevance. Its objective: To put a human on the Moon by the end of the decade. However, the whole program might have failed as soon as in 1961, neither because a particular technology failed nor due to an accident. It was because of a crucial decision related to how the system elements would interact with each other. For landing a human on the Moon, there were essentially three different proposals:

1. Direct. Use a big rocket launcher, which still needed to be developed, to directly launch a spacecraft to the surface of the Moon and directly return from the surface of the Moon to Earth.
2. Earth-Orbit Rendezvous. Use two launchers, smaller than for the Direct proposal, and dock the two pieces of the spacecraft in an orbit around Earth, and then launch the spacecraft to the Moon and perform the same maneuvers as for direct.
3. Lunar Orbit Rendezvous. Use only one launcher, smaller than for Direct, and achieve this by drastically reducing the payload by leaving the return spacecraft in lunar orbit, and land a separate, smaller spacecraft on the Moon; and then go back to lunar orbit, dock the spacecraft to the one in lunar orbit, and fly back to Earth with the spacecraft which remained in lunar orbit.

At that time, NASA pursued option 1 and 2 and rejected 3. NASA rejected 3, as the rendezvous in lunar orbit had not been done before and was considered too risky. However, a NASA engineer, John Houbolt, was advocating for option 3. He showed via detailed calculations and estimates that the rendezvous was not as risky as it seemed. He also showed that options 1 and 2 would require the costly and time-consuming development of a very large rocket launcher, which would make it impossible to land a human on the Moon by the end of the decade. After facing significant internal resistance, Houbolt, in a desperate act, wrote a letter to NASA Deputy Administrator Seamans and was able to convince him that option 3 was the

only feasible option for a crewed lunar landing by the end of the decade. This act of desperation ultimately turned the tides, and by 1962, the Lunar Orbit Rendezvous plan was approved.

Why this example at the beginning of a chapter on system architecture? The Lunar Orbit Rendezvous illustrates nicely the importance of system architecture. The architecture in this case includes several components: the rocket launcher, the lunar spacecraft, the rendezvous technology, the ground infrastructure, etc. It also includes relationships between the systems. The rocket transports the spacecraft into space. The spacecraft transports the crew to the Moon and back to Earth. It also shows the importance of system architecture, which is often defined at the early stages of system development. The “right” architecture might lead to a successful system, whereas the “wrong” architecture might lead into inevitable failure.

In this chapter, we will first present definitions for system architecture, a process for system architecting, and modeling principles. Furthermore, we will present some emerging topics in system architecture such as its link to company strategy, system of systems, and product service systems, which are relevant for engineering systems.

System Architecture Definitions

System architecture can be understood as the fundamental structure of a system (ISO/IEC/IEEE 2011). One can identify three major definitions of the system architecture that have been largely used in the literature, which essentially differ in what comprises “structure.”

According to the ISO/IEC/IEEE 42010:2011 standard (ISO/IEC/IEEE 2011), a system architecture comprises the “(system) fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution.” This definition is generic and would apply to various domains such as software architecture and general system architecture.

According to Crawley et al. (2016), system architecture “is the embodiment of *concept*, the allocation of physical/informational *function* to the elements of *form*, and the definition of *relationships* among the elements and with the surrounding *context*.” This definition considers the allocation of function to form as essential to system architecture.

According to Emes et al. (2012), the “architecture of a system is its fundamental structure” which may include principles applying to the structure as well as specific structures. This definition is generic and is coherent with the general definition of “architecture” as an object’s structure.

Table 1 provides an overview of these three definitions and their main characteristics. It can be seen that each definition considers something different as architecture (concepts and properties, embodiment of concept, structure). Looking at the elements of architecture, all three definitions stress the importance of system elements and relationships, i.e., its “structure.” Crawley et al. (2016) further develop and add allocation as a particular form of relationship between functions and form, both elements of a system.

Table 1 Comparative table on definitions of system architecture

	ISO/IEC/IEEE (2011)	Crawley et al. (2016)	Emes et al. (2012)
What it is	Fundamental concepts or properties of a system in its environment	Embodiment of <i>concept</i>	Fundamental structure
What it consists of in detail	Elements, relationships, and the principles of its design and evolution	The allocation of physical/informational <i>function</i> to the elements of <i>form</i> and the definition of <i>relationships</i> among the elements and with the surrounding <i>context</i>	Principles applying to the structure as well as specific structures

To summarize, although all three definitions are different, they all agree that architecture is some form of the structure of the system, in other words, an abstract property of a system, which is captured in the way elements are related with each other. In the following, we will use the definition by Crawley et al. (2016), as it provides more specific elements for what a system architecture is than the other two definitions.

The importance of the system architecture is manifold. Most arguments for its importance belong to the class of arguments that explain why the early stages of design are important (Fricke and Schulz 2005). The main point is that in the early stages of design, there are still large degrees of freedom for the design of the system. For example, an automobile manufacturer might still be able to choose which type of vehicle to develop (car, motorcycle, tricycle). Once a decision has been made, e.g., development of a car, certain parameters are now fixed and can no longer be changed. A car has a very different set of parameters (e.g., parameters related to the passenger cabin) than a motorcycle (two-wheeled vehicle with no cabin). This process goes on to more and more detailed levels of the design. For example, a car can be a sedan, limousine, and SUV and have a gasoline, diesel, electric, or hybrid powertrain. Choosing one of these alternatives will again fix a set of parameters (e.g., market, size of the car, cargo compartment, etc.). Each subsequent decision will further constrain the parameters, and there are less and less degrees of freedom. From this example, it seems clear that the decisions at the beginning have a more significant impact than the later ones. Later in this chapter, we will look at more complex examples of engineering systems, which go beyond a single monolithic system such as a car and how that affects system architecture decisions.

System architecting as a sequence of decision-making has been proposed by Simmons (2008). However, the question is not only how to better choose between alternatives but also how to innovate or invent, in other words, how do we actually generate new alternatives we then choose from. This is exactly what the Lunar Orbit Rendezvous case illustrates. The question was not only to decide (choose between three alternative architectures) but also to innovate on different levels of the architecture and consider solutions that have never been previously considered.

A historical example for an invention that has its origin in the system architecture is the airplane. An airplane has three basic functions for flying: generate lift, control

the airplane, and propel the airplane. The main inventive element for airplanes was how these functions were allocated to the components of an airplane. In 1810, William Cayley published a book called *On Aerial Navigation*. It revolutionized the field of airplane design. What was his invention? Before Cayley, airplanes were essentially airplanes with flapping wings, called ornithopters. A famous example is Leonardo da Vinci’s ornithopter design, shown on the top of Fig. 1. In an ornithopter, the three functions of an airplane are all allocated to one component: the wings. The flapping wings at the same time generate lift, propel the airplane, and are used for control. The objective was to mimic birds. However, it turned out that building an ornithopter was challenging. It was unclear how to improve it, as it was unclear how the behavior of the wings was related to its functions. Hence, the result was rather trial and error, without a clear direction of improvement. For centuries, no substantial breakthrough in airplane design took place.

Here, Cayley comes in. Cayley proposed to allocate each of the functions to different system components. Lift was allocated to the wings, control to flaps, and propulsion to a separate engine. Today, we would call this a modular architecture. Each function was allocated to a different component. By doing so, suddenly, each component could be separately improved by building and testing prototypes. For example, Lilienthal was able to do experiments with gliders (without propulsion and limited control) and measure aerodynamic values, as he could build dedicated experimental setups, measuring a selected number of parameters. Ultimately, the Wright Brothers continued the research program by innovating each of the three components separately and integrating them gradually to build the first motorized, crewed airplane, capable of sustained flight.

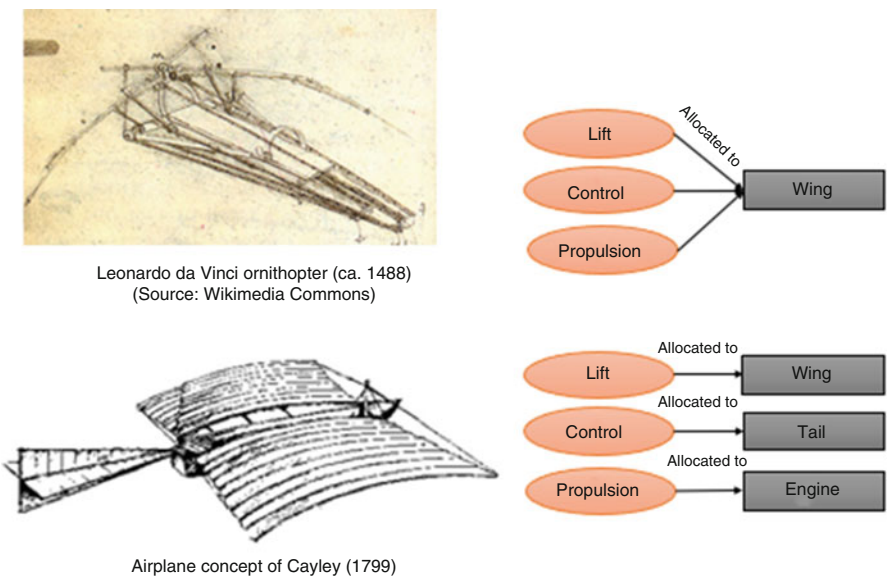


Fig. 1 Cayley, G. (1810). *On aerial navigation*. William Nicholson

As shown in Fig. 1, one can see that for Cayley's airplane concept, the relationship between function and form was substantially different from its precedent. This invention, together with Cayley's deeper understanding of the physics of flight, would open up a whole new field of research in aeronautics. It demonstrates the power of reasoning around the allocation of function to form.

Another example is the Mars Direct mission architecture, proposed by Robert Zubrin (2011). The Mars Direct mission architecture profoundly changed the way how we can transport humans to Mars and has since then dominated NASA's thinking about crewed Mars missions. The innovation comprised two parts, each involving a different allocation of function to form than previous architectures. First, instead of launching one single spacecraft with a human crew to Mars, two spacecraft would be launched. The first would just transport the cargo to Mars, the second the crew. Splitting cargo and crew means that the cargo can be sent to Mars on a slower spacecraft, needing less propellant, while the crewed spacecraft needs to make the trip to Mars as quickly as possible, in order to limit radiation exposure. The second part of the innovation was to introduce a new technology into the architecture: in situ propellant production. Among other reasons, crewed Mars missions are so costly, as all the propellant for the return trip needs to be carried all the way to the target destination. If that propellant could be produced on-site, the propellant for the return trip no longer needs to be carried to Mars, reducing the propellant mass needed for launching the spacecraft from Earth to Mars. However, producing the necessary amount of propellant takes time, months to even years. Again, allocation played a key role in exploiting the potential of this technology. The cargo spacecraft would transport the in situ propellant production plant to Mars and start producing propellant until the crewed spacecraft would arrive. The resulting architecture is expected to lead to a reduction in cost of up to an order of magnitude. This example shows that typical system architecting activities such as partitioning and allocation of function to form (separation of cargo and crew spacecraft) and infusing the right technology at the right place (in situ propellant production, transported by the cargo spacecraft) can lead to breakthrough results.

System architecture also plays a major role in changing a whole industry. Figure 2 shows trends in complexity increase for integrated circuits, automobiles, and aerospace vehicles. It can be seen that the automotive industry has achieved a decrease in development duration by a factor of 3 between the 1960s and the 2010s, while the complexity of cars increased by five orders of magnitude. This remarkable fact is even more remarkable in light of the number of critical interfaces that has increased (interfaces between different engineering domains needed in system development, interfaces between system components, interfaces between functions). For example, while cars in the 1960s were essentially a set of mechanical components, cars in the 2010s are a combination of mechanical, electronic, and software components. The different engineering domains behind these components need to collaborate. Hence, this increase in critical interfaces adds to the complexity, although this increase in complexity is not even measured by part count and source lines of code in Fig. 2. Interfaces between parts as well as lines of code are simply not accounted for. Hence, the "real" complexity increase is much higher.

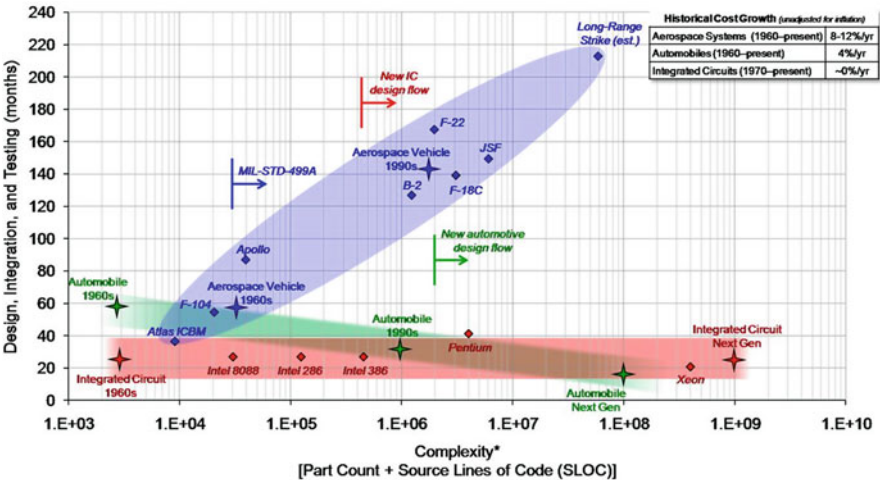


Fig. 2 Trends in complexity and development duration for different industries. (Eremenko, P. (2010, October 7). Adaptive vehicle make [proposer’s day briefing to DARPA for the Adaptive Vehicle Make Program]. DARPA Web Site [Online]. www.darpa.mil/WorkArea/DownloadAsset.aspx?id%2659)

We have talked a lot about interfaces but what is an interface? We briefly deal with this question in the following. It turns out that it is not easy to respond to this question and the definition of an interface has evolved over time and literature reviews have been addressing this evolution and their definitions (Parslov and Mortensen 2015). Roughly speaking, the majority of the literature considers interfaces as either functional or physical. A functional interface has a function such as transmitting energy out of a component. An example for a physical interface would be a USB port. As mentioned for the example of the automotive industry, interfaces do not only exist between technical components (hardware, software) but also between organizations (teams, departments, companies). Common interfaces between departments in a company are for transmitting data and information. For example, the design department in an automotive company would transmit design drawings to the manufacturing department.

Going back to Fig. 2, one important reason that cars today are developed about twice as fast as during the 1990s is the result of reuse or commonality. Commonality does not only indicate that already existing designs (carryover components) and production lines are reused but also deliberate planning for how designs and production lines can be used across a family of systems, which are under development at the same time. However, designing a family of systems and taking commonality into account is not trivial (Boas 2008). A general trade-off for commonality in a family of systems is between the performance of individual systems and their degree of commonality. The higher the commonality, the more similar the performance of the systems. Often one can observe that initially high commonality values in a family of systems can degrade over time. A reduction in commonality is

typically a consequence of using a new design, as a reaction to a lower than expected performance in one of the systems. This phenomenon is called “divergence.”

System Architecting

Overall System Architecting Process

Industry has been standardizing the collaborative development of system. This has been done in particular through the definition of system architecture frameworks such as the Department of Defense System Architecture Framework (DoDAF) (Department_of_Defense 2020), NATO Architecture Framework (NATO 2018), TOGAF proposed by the Open Group (The_OPEN_group 2020), MoDAF (Ministry_of_Defense 2020), etc. The aim of these standards is to allow for collaboration while system architecting by providing common concepts. Some frameworks such as TOGAF also provide a system architecting process. The proposed system architecture frameworks have been based upon enterprise frameworks such as Zachman’s enterprise framework (Zachman 2006). The main objective of these approaches is to provide a common understanding among different stakeholders in order to be able to collaboratively develop a system architecture. The NATO Architecture Framework (NAF) further specifies that “the purpose of the Architecture Definition process is to generate system architecture alternatives, to select one or more alternative(s) that frame stakeholder concerns and meet system requirements, and to express this in a set of consistent views” (NATO 2018).

Most of these standards propose a:

- Methodology – a process and a definition of how to design system architectures.
- Different viewpoints – these are different conventions of data definition and exchanging data related to the project at different times.
- Meta-model – a definition of a common reference data model that describes the system architecture.
- Glossary – a definition of common terms in order to clarify and enhance collaborations.

The military industry has been developing large systems using these frameworks since the 1940s. They have been also adapted and reworked so as to suit a variety of civil system developments (e.g., TOGAF). Although one could think that there is a wide variety in designing system architectures, the backbone process can be seen as the same (see section “[System Definition Phase](#)”) that can be refined to fit different purposes of system architecting.

A generic system architecting process may have the following steps:

1. **Identification of stakeholder needs.** Who are the stakeholders and what are their needs that need to be addressed by the system to be developed?

2. **High-level requirement definition** From the stakeholder needs, a set of system-level requirements is derived, which need to be satisfied, defining key functions and performance levels at a high level.
3. **Definition of functional architecture.** What are the functions that need to be executed to fulfill the system-level function(s) and performance? What are the interactions between the functions?
4. **Definition of physical architecture.** What are the system's components, to which the functions are allocated?

It is important that the perspectives advanced in these steps are interrelated. The system requirements are derived from the stakeholder needs, the requirements are satisfied (or not) by the system's functions and components, the functions are allocated to components, etc. A change in one of these perspectives likely leads to a cascade of changes in other perspectives. For example, changing the power of an engine in a car is likely to impact the acceleration of the car, which might lead to a non-satisfaction of the acceleration requirement, which might lead to a customer not satisfied, as the car does not speed up fast enough.

Some system architecting processes introduce an intermediate step between step 3 and 4, the definition of the logical architecture (e.g., OOSEM (Friedenthal et al. 2014)). The logical architecture is commonly used for grouping functions and allocating them to logical components. For example, all functions related to the conversion of power in a spacecraft could be allocated to the logical component "power subsystem," without specifying what technology the power subsystem is based on. The logical architecture introduces another level of abstraction when it is difficult to directly allocate functions to physical components.

A particularly important part of the process of system architecting is the definition of interfaces between functions and components. The interfaces "structure" the interactions between functions and components and influence the system's emergent behavior. These interfaces are so important that even whole industries have developed because of certain interfacing schemes, such as for personal computers (Baldwin et al. 2000). In particular, if the interfaces are defined "correctly," changes do not lead to uncontrollable cascades but are contained within specific areas of the system, which are often called "modules." Modules are of such importance, as they have a low number of well-defined interfaces and "hide" their complexity inside. In fact, industries often develop around such modules, such as industries for motherboards, hard discs, and screens in the computer industry.

System Definition Phase

There are numerous system lifecycle processes and therefore system development processes that are related to the system type, system development context, and different development environments (Haskins et al. 2006). Several very good comparative analyses of system development processes exist and can be found (Haskins et al. 2006). However, whatever the existing process, first stages are entirely

dedicated to the understanding of the system stakeholders' needs, system requirements, and, respectively, to the definition of the system perimeter. This exploration (addressing in general step 1 and 2 given in the previous section) is mostly focused until the concept development phase (e.g., exploratory research and concept definition phase (ISO/IEC 15288) or pre-phase A – producing a feasible design by exploring several alternative architectures – to phase C, refinement and completion of build-to designs (Shishko and Aster 1995)). This process is rather generic and stems from identification and definition and stakeholder needs. It extends to mission design and to ConOps (concept of operations) design. These are highly iterative processes and might be differently devised in different contexts. The *NASA Systems Engineering Handbook* gives an overview of this iterative process and necessary relationships (see Fig. 3).

The interesting point here is that in order to define the system perimeter and thus system architecture that is capturing this perimeter, the *NASA Handbook* identifies the processes stakeholder expectations, requirement definition, logical decomposition (which is in essence system architecture definition), and design solution definition. However, in different contexts, it might be that one starts with stakeholder expectations, to explore requirement definition, to define the ConOps in order to refine the initial system architecture definition (e.g., *INCOSE Systems Engineering Handbook* v3.2 p. 69). Oftentimes, even though these processes can be represented as sequential, they are done iteratively and need to ensure the consistency between them.

Identification of Stakeholder Needs

Identification of stakeholder needs is a critical point for good definition of system architecture. The main objective of this process is “to identify who are the stakeholders and how they intend to use the product” (NASA Handbook). Freeman (1984) underlines that this term was first defined in “those groups without whose support the organization would cease to exist.” This definition further evolved to “any group or individual who can affect or is affected by the achievement of the organization’s objectives,” which is also the definition used in both NASA and INCOSE handbooks. Moreover, the MIT System Architecture Group defines stakeholders as groups that (1) affect directly or indirectly the focal organization’s activities, or (2) receive direct or indirect benefits from the focal organization’s activities, or (3) possess a significant, legitimate interest in the focal organization’s activities (Crawley 2009; Sutherland 2009). The typology that is related to this definition and discussed in this work is the following:

1. “Stake” holders: those who have a direct stake in the project.
2. Beneficiaries: those who derive benefits from the project.
3. Users: the ultimate consumers or users of the project’s outputs.
4. Agents: those who act on behalf of other stakeholders in the model.
5. Institutions: official bodies or organizations that directly impact the project.

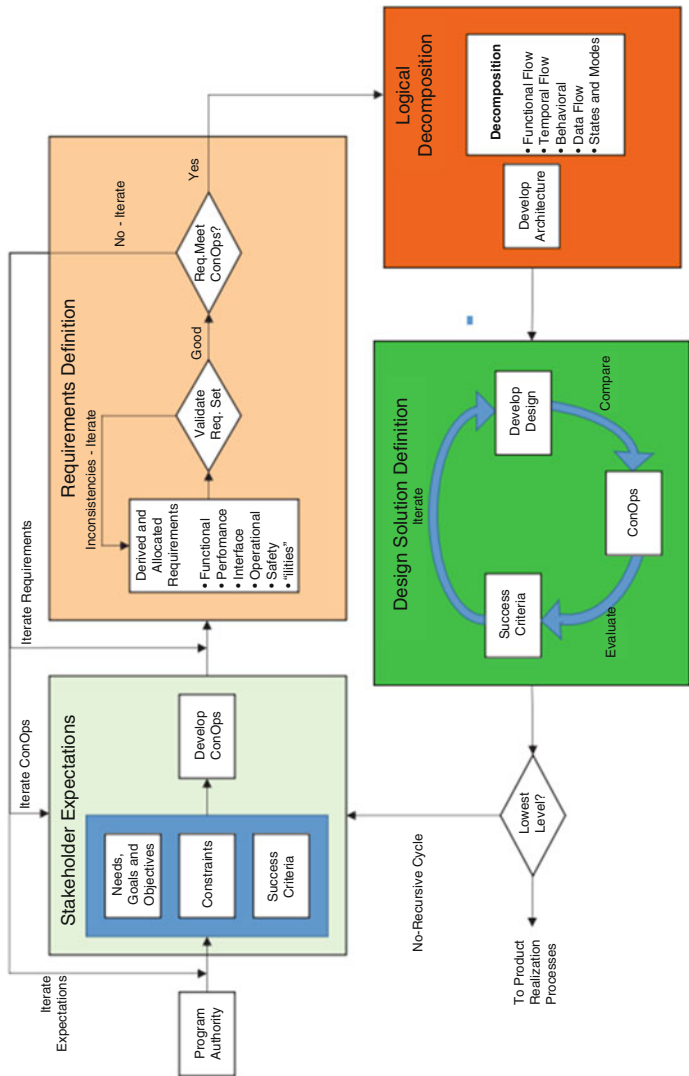


Fig. 3 Interrelationships between early definition processes (NASA Systems Engineering Handbook, SP-2016-6105 Rev2, p. 44)

6. Interests: those with a significant, legitimate interest in the project's outputs, who may not be considered a direct stakeholder in the traditional sense.
7. Project: relatively, the focal organization itself is also a stakeholder in the eyes of other stakeholders.

Several methods and tools are used in order to identify and define stakeholder needs (e.g., marketing questionnaires and surveys, focus groups, prototypes, serious games, virtual reality/simulations). These techniques allow either to capture the needs that stakeholders express or to observe what they want and how they use a system in a given situation. Novel approaches to refine stakeholders needs appear such as stakeholder value network modeling (Feng 2013) allowing for modeling and exploring possible value exchanges between stakeholders based on graph theory.

Oftentimes, in order to refine the understanding of stakeholder needs, **concept of operations** is used (ConOps). ConOps represent high-level scenarios of use describing how the system will be used in order to satisfy stakeholder needs (NASA Handbook). The objective of ConOps is to describe and prescribe the intended operation of a system, supporting the understanding of a system context (SE Handbook). In order to be exhaustive, ConOps need to consider all phases of the system lifecycle integrating the knowledge related to off-nominal solutions as well as to develop a as complete as possible set of possible malfunctions and degraded modes. Several modeling techniques are used for ConOps modeling: scenario modeling, activity modeling (activity diagrams in SysML), process modeling (business process modeling, swim lane modeling), state machine modeling/event modeling, etc. Other than initially discussed objectives, ConOps are also used (SE Handbook) to manage the traceability between operational needs and requirements, as a basis for verification planning, and to generate models to test the validity of external system interfaces, as a basis for calculating system capacity and mission effectiveness.

Stakeholder expectations as well as ConOps are used for the definition of the top-level **requirements** defined in order to understand the technical problem to be solved and establish the "design boundary" or "system boundary." The "design boundary" or "system boundary" is actually the perimeter of the system that needs to be developed. In the system engineering literature, it is addressed as "design boundary" making reference to system design, while in system thinking and modeling, it is addressed as "system boundary" or "system perimeter." This boundary is progressively refined by understanding constraints and limits that the system needs to adhere to, elements and parameters that are endogenous or exogenous (meaning parameters/elements that are within or out of the control and possible change within system design), and external and enabling systems in order to establish possible physical and functional interfaces (mechanical, electrical, thermal, human, etc.).

High-Level Requirement Definition

Many definitions of system requirements exist. The ISO standard defines requirements as a “statement that identifies a product or processes operational, functional, or design characteristic or constraint, which is unambiguous, testable, or measurable and necessary for product or process acceptability” (ISO/IEC/IEEE 2011). For example, Holt et al. define it as “a property of a system that is either needed or wanted by a stakeholder” (Holt et al. 2012). Requirement engineering is a process of encompassing several engineering activities (Berkovich et al. 2014; Jiao and Chen 2006; Song 2017):

- Stakeholders’ requirements elicitation
- Stakeholders’ requirements analysis
- Requirements specification
- Requirements change management

One of the key activities in these engineering activities is requirement modeling (Albers et al. 2013; Holt et al. 2012; Scherer et al. 2017). According to the recommendation of the IEEE 1233 standards, a requirement model should contain, in addition to the requirement description, the following attributes: unique identifier, priority, criticality, maturity, impact, and possible others (ISO/IEC/IEEE 2011).

The objective of requirements elicitation is to elicit and define system requirements. This process consists of several activities: research, discover, identify, and elaborate client requirements. Zhang identified in the literature four types of requirements elicitation methods: conversation, observation, synthetic, and analytic methods (Zhang 2007). Conversation and observation represent classical requirements elicitation methods and are based on techniques such as interviewing, workshops, and user observations. Synthetic techniques use techniques such as storyboarding, prototyping, and scenarios development. As for the analytic approaches, they are based on more formal and documentation-based methods based often on requirement modeling techniques. Requirement reuse consists of extracting requirements defined and verified in previous projects and using them in a new one. The objective is to reduce development time and cost and increase the productivity and quality of products. Several studies underline that this is particularly useful to help stakeholder rapidly elicit system requirements (Pacheco et al. 2015; Toval et al. 2002). Recycling requirements is slightly different than requirement reuse. It consists in identifying system requirements from previous projects but adapting them to the new one by adjusting requirement parameters and attributes such as maturity, criticality, and flexibility. Alexander and Kiedaisch (2002) defined parameter recycling as keeping the suitable parts in the base of the requirement description, adapting the other parts to the new project’s context, and integrating the resulting requirement to the new requirements system (Alexander and Kiedaisch 2002).

System Architecture Modeling Principles

As discussed in previous sections, in order to start addressing system architecture, one needs to define the system perimeter or boundary well. In order to do so, the definition of interfaces that are external to the system is essential. One of the major reasons to build a system is the possibility of emergence. Crawley et al. (2016) emphasize that “the overall functionality is greater than the sum of its parts.” However, the other side of the coin of emergency is actually the complexity. The complexity of a system stems from the fact that the system is constituted of many connected, interrelated, and intertwined elements and entities. This leads to the first principle of system architecture modeling: decomposition of different system architecture domains and management of interrelationships, dependencies, and interfaces.

In general, if one looks at different definitions of system architecture, several domains can be identified: functional, structural, and behavioral. This is consistent with the definition of system architecture (for instance Crawley) as well as several underlying theories such as function-behaviour-structure (Gero and Kannengiesser 2004). Each of these domains needs to be decomposed and modeled as well as interrelations between these domains.

Definition of Functional Architecture

The first domain is the functional domain. A system function defines what a system *should do or does*. Modeling system function is a considerable field (Special issue on Function modelling (Vermaas and Eckert 2013)), and several modeling approaches have been identified (Erden et al. 2008). However, two major governing decomposition principles are used when it comes to the decomposition: function decomposition or function flow modeling. Functional decomposition is often identified in companies as functional breakdown structure (FBS) (Dehoff et al. 2009). Several modeling techniques support this decomposition. One of the oldest one is functional analysis system technique (FAST) (Bytheway 2007). The second decomposition principle is actually addressing the order in which the functions may be executed. The technique that is largely used in industry is functional flow block diagram, introduced by Frank Gilbreth (American Society of Mechanical Engineers ASME) in 1921. These two function modeling principles are complementary and not independent. In order to manage engineering system development, functional decomposition is critical for allocating functions to different system levels. However, for each system level, the order of function execution allows for in-depth understanding of how the system functions. The difficulty lies in building coherent and complementary models of functional decomposition and flow and updating them through the engineering system development process.

Definition of Physical Architecture

As for the second domain, **structural domain**, or commonly addressed as *physical architecture* or *form*, the objective of this domain is to identify and describe system components (Browning 2001; Eppinger and Salminen 2001; Yassine et al. 2003; Yassine and Braha 2003) and elements that can achieve certain functions. As in the

case of functional domain, one of the two major approaches to address this domain is to decompose. Hence, one of the key tools that govern engineering system development process is the product breakdown structure (PBS). The PBS is not only allowing for hierarchical decomposition of one product but also integrates and reflects the organizational structure of engineering system development (for instance, identification of components that are developed in-house or that are outsourced). Modeling techniques that allow for understanding the system form largely stem from what is called “concept design” or “conceptual design.” Matrix-based approaches have been used since the 1960s (Steward 1962; Steward 1981). Approaches and models using matrices to define form are numerous (Ziv-Av and Reich 2005; Bryant et al. 2005; Jankovic et al. 2012; Maurer 2007). Network-based approaches in a large sense have also been frequently used (Haley et al. 2014; Sarkar et al. 2014; Moullec et al. 2013; Wyatt et al. 2008). The objective of these approaches is to identify and manage interdependencies and in particular system interfaces (Pimmmler and Eppinger 1994). For example, Pimmmler and Eppinger (1994) define four types of interfaces: spatial, energy, information, and material.

However, considering the functional domain without the structural one often leads to errors and engineering system project failures. In order to understand how the function is fulfilled, allocation of components to functions is necessary to define. “Allocation” or N2 matrices are usually used to define and manage these allocations. Conjoint consideration of these models actually allows for identification and definition of two types of interfaces, which have already been mentioned before: functional ones and physical ones. Functional interfaces represent different types of dependencies between functions. They can stem from functional flows or they can actually be deduced through system structure. For example, if one subsystem or component contributes to the fulfillment of two functions, even though these functions are not connected initially, one can deduce that there is a functional interface to manage. Often this induces also that there are several teams that need to collaborate to make sure that the functions are realized correctly. Physical interfaces are the ones that are identified previously. In particular, the interface definition documents are issued in order to characterize and allow management of these interfaces. However, interface management, as it is also linked to the organization of engineering system management, remains one of the major issues and causes for engineering system project failures.

Behavioural domain is the reason for building systems. Both ConOps and functions are used as proxies early in engineering system design to identify and define expected system behavior. However, actual behavior can be defined, simulated, and calculated only later in the process when more precise models (mechanical, electrical, software) are constructed. However, gathering knowledge and expertise has allowed defining high-level models estimating future system behavior. Models proposed in order to characterize this domain are anchored in value engineering (Miles 2015) or value-driven design (Collopy and Hollingsworth 2011). Cheung et al. (2012) underline that the difficulty lies in analyzing and mapping the relationships between component models to product (system) models to the value model that allows for deciding the most promising system architecture alternatives.

The objective is to construct understanding models that go from component models and their design parameters to the overall system value model that allow for the decision. System value models can be also addressed as “-ilities” (overall system costs, system manufacturability, reliability, etc.).

Defining system architecture and system architecture alternatives can be a computationally expensive work. **System architecture** selection is based upon identification and definition of relevant parameters for system architecture selection which is not a trivial task and is often neglected (Moullec et al. 2015). Modeling and representation of possible system architecture performances and their relationship to different design parameter are considered as trade space exploration (Winer et al. 1998; Stump et al. 2002; McManus et al. 2004). The Pareto frontier is used in general in order to identify non-dominated system architectures and in particular to address system architecture trade-offs (Miller et al. 2014; Mattson and Messac 2005; Ross et al. 2004). This approach of multi-objective optimization allows for visualizing and identifying system architecture alternatives having the same trade-offs between two or more system performances.

Implications and Challenges of Systems Architecting for Engineering Systems

Different Company Strategies Articulated around System Architectures

System architecture has been studied and defined as one of the key elements related to enterprise strategy and operation (Fixson 2005). System architecture modeling and organization integrates and is related to different key enterprise strategies allowing for market positioning and share such as make or buy strategy, supply chain organization and management, reuse strategies, distributed development strategies, concurrent development strategies, and development alliances strategies. All these strategies are impacting directly enterprise benefits and success.

One of the determining tactical-level decision processes is related to determining the number and the type of processes that will be used to manufacture a product or a system (Fixson 2005). These processes are commonly addressed as commonality and diversity strategy, implying that if components can be reused across product families or multiple product generations, there is a possibility to manage and reduce cost through scale efforts. This is one of the reasons why system architecting is interesting for industry. In particular, two complimentary strategies (they are not the only ones but here we will focus on them) are used to define commonality and diversity: product family and related platform design and modularity.

A product platform represents a set of parameters or features or components that remain constant from product to product (Simpson et al. 2001). Although it can be features or parameters sharing defining one platform, in the majority of cases in industry product platforms are related to component reuse and sharing. This strategy allows also for product/system personalization while benefiting from reuse

strategies. In particular in engineering system design, this strategy is important as it allows also for spreading development costs across different projects and different market segments while maintaining or diminishing the development time. Research studies aiming at understanding, defining, and managing product platforms are many. Simpson et al. (2001) propose a method for product platform design starting from market segmentation to product platform design. Gonzalez-Zugasti et al. (2000) propose a method to architect product platforms starting from system requirements. De Weck et al. (2003) investigate the method to identify the optimal number of product platforms in order to reduce manufacturing costs.

Related to the definition of product platform strategy is the notion of product modularity. Product modularity arises from the possibility of decomposing products or systems into subsystems. The idea is the possibility for the reuse of these subsystems (Gershenson et al. 2003) and is related to the make or buy strategy. Companies developing engineering systems need to define a comprehensive make or buy strategy (i.e., what is made in-house and what is outsourced), hence creating design chains. In order to be able to outsource developments, there is a need to define subsystems and components in such a way that this is economically interesting and feasible as well as trying to manage interfaces between the module and the system. A considerable body of knowledge exists on the underlying principles for defining product modularity as well as indicators and methods supporting their definition. Gershenson et al. (2003) identify three categories of modularity based on interactions within a product:

1. “Component-swapping modularity, when two or more components can be interchanged in a module to change the functionality of that module
2. Component-sharing modularity, when two or more modules contain one or more of the same basic components as the core upon which they are built
3. Bus modularity, where a module with two or more interfaces can be matched with any number of components selected from a combination of basic components (Huang and Kusiak 1998)”

These strategies aim at managing development times, managing system costs, and increasing product variability while spreading across development efforts.

Although in general, the proposed approaches for system architecting address technical systems and are at the company level, they can be used to support reflections in policy making. One example is the application of stakeholder analysis using the stakeholder value network approach to large-scale infrastructure projects, such as in Feng (2013) and Hein et al. (2017), and space exploration programs (Cameron et al. 2008). Large infrastructure projects and space exploration programs are shaped by public policy, which is the result of finding a consensus between diverse actors (Cameron et al. 2008). Feng (2013) proposes a future work to combine the stakeholder value network approach with system architecture design. Such a combination of stakeholder analysis and system architecting has been proposed in Hein et al. (2018), where societal and public stakeholder concerns are factored into the system architecting process, illustrated by the case of a robotic taxi service. At an

even larger scale, system architecting approaches are proposed to address the COVID-19 crisis (De Weck et al. 2020). One of the most well-known research addressing policy making while using system modeling is related to the limits to growth on Earth (Meadows et al. 1972). Meadows et al. (1972) do not address the notion of system architecture explicitly but rather use system modeling, more specifically system dynamics. More recently, however, Hein and Rudelle (2020) have assessed the limits to economic growth on Earth by combining approaches from ecological economics and energy economics, with system architecting approaches such as technology infusion. Their objective is to derive recommendations for long-term decision-making in energy and climate policy making.

We expect that the tendency of using system architecting (or system modeling principles) in policy making is growing even more because of the interconnectivity of systems that give emergence to novel types of systems such as system of systems or product service system of systems (Hein et al. 2018). These systems not only have a technical aspect to address but also integrate societal challenges into system architecting processes.

Integrating Systems: Notion and Challenges Related to System Architecting of System of Systems

System architects are more and more confronted with the integration of individual systems, managed and operated independently, into a higher-level system. Examples are multimodal mobility, where different transportation systems such as trains, busses, and taxis are integrated to provide an integrated mobility service. The train system, bus lines, and taxis are typically operated and managed by different entities. Nevertheless, busses might get informed about a train arriving late and need to wait for passengers. A taxi might be redirected, as a passenger stepped out at the wrong train station. Hence, data is exchanged between the constituent systems. Such collaboratively integrated systems are called “system of systems” (Maier 1996). Although there is no consensus definition of a system of systems, several definitions have in common that each constituent system is independent and has its own purpose (Boardman and Sauser 2006). Some system of systems satisfies the criteria of engineering systems such as the electric grid, public transportation, and the healthcare system.

System of systems has characteristics which make their design challenging, and existing system engineering approaches have limited applicability (Sousa-Poza et al. 2008; Keating et al. 2003). As its constituent systems are managed and operated independently, forming a system of systems is also a collaborative challenge. This may require not only the consideration of the architecture of technical systems but may require the consideration of the architecture of collaborating organizations, also referred to as enterprise architecture (Cole 2009). Another hallmark of system of systems is that their constituent systems have their own purpose and independence (Boardman and Sauser 2006; Maier 1996). For example, a bus can be operated on its own and has its own purpose. This is in contrast to an engine in a car, which does not

provide value without being integrated into a car. Furthermore, as data and information need to be exchanged between constituent systems, common data formats, standards, and protocols need to be established. As with all systems, system of systems has emergent behavior, which might only be discovered through experiments of gradual interventions (Meilich 2006). Traditional systems engineering defines system boundaries at the beginning of the architecture design process. System of systems tends to have shifting boundaries, which change over their lifecycle, due to the introduction of systems, technologies, or changing requirements (Keating et al. 2003). For example, new modes of transportation could be introduced into a multimodal transportation system, such as electric scooters.

System of systems is also rarely designed from scratch and is subject to interventions to existing systems. System of systems evolves over time and might be subject to constant change (Maier 1996). They might emerge (train lines and bus lines exist in parallel initially but are then integrated) or evolve gradually by integrating more and more systems. Hence, the architecting process of system of systems rarely follows the typical system architecting process but is rather iterative with incremental modifications, which may take place at different speeds. Although approaches for architecture design of system of systems have emerged, they seem to be rather mature in the area of defense and security (Meilich 2006). For other domains, architecting system of systems seems to be rather on an ad hoc basis, and it seems that there is a lack of systematic approaches (Keating and Katina 2011).

System of systems also evolves on different time horizons. For example, a multimodal transportation system is dynamically reconfigured within minutes to hours (train stops operating; replacement busses are put in place). However, the definition of a new bus line might take years and might start with a low frequency which is then gradually increased. These different time horizons require different architecting approaches. For example, dynamic reconfiguration during operation may be accomplished by semiautomatic system reconfiguration approaches (Qasim et al. 2019, 2021). For longer time horizons, such as for a new bus line, traditional architecting approaches, including stakeholder analysis, may be appropriate (Feng 2013).

These specific system of systems characteristics require architecture models which specifically represent the following critical interfaces:

- Collaborative relationships between actors managing/operating constituent systems.
- Defining data exchange standards, protocols, and interfaces has a disproportionate importance, compared to traditional systems.
- Evolutionary development. System of systems is usually not developed from scratch. They evolve over time. This requires the consideration of different lifecycles of constituent systems and how new systems can be integrated into the system of systems and how the exit of systems might affect the system of systems.
- Different timescales. The behavior and evolution of system of systems have different timescales which might range from microseconds to years. Different

behavior models are necessary for adequately simulating and taking into account these different timescales. These different timescales also impact the choice of the system architecting approach, e.g., dynamic reconfiguration of the architecture during the operations phase vs. architecting during the design phase.

Toward Product Service System Architectures

Another type of emerging engineering systems is system with the objective of delivering services. Examples for such systems are the healthcare system, multi-modal transportation systems, and the energy supply system. These examples are similar to those we have given as examples for system of systems, which is not a coincidence. However, for now, we focus on the delivery of services by these systems. Services are gaining more and more in importance, and traditional manufacturing industries such as automotive and aerospace are undergoing a profound transformation. For example, instead of selling airplane jet engines, today, jet engine hours in operation are sold to customers. In the automotive domain, mobility services such as car-sharing are starting to substitute personal cars, particularly in large cities. Hence, the challenge is to understand how to design such systems combining products (e.g., car) with a service (e.g., mobility). These types of systems are referred to as product service systems (PSS) (Tukker and Tischner 2006). To come back to the examples we gave at the beginning of the paragraph, here we want to address specifically emerging engineering systems that are PSS. These PSS are typically at the same time *system of systems* (Hein et al. 2018). In terms of modeling the system architecture of such PSS, existing models for system of systems need to be extended, to represent services in more detail (Hein et al. 2018).

PSS that are engineering systems add an additional service, business, and societal perspective to system architectures (Hein et al. 2018). As a consequence, traditional system architecting with its representations of stakeholder needs, requirements, functional architecture, and physical architecture needs to be extended. For example, the introduction of autonomous vehicles adds a service layer to the architecture, where transporting a passenger from A to B at a given cost and duration is a service in that layer (Hein et al. 2018). Furthermore, while traditional system architecting often deals with development “from scratch,” PSS which are engineering systems are rather designed around *interventions* into an existing system, for example, using existing infrastructure elements and services (Hein et al. 2018).

While in traditional system architecting, the system boundary needs to be defined early on, in order to define the system’s concept of operations, in engineering systems PSS, we know the service and the desired service quality, but we do not necessarily know the boundary of the system and might decide in run-time or during systems operation which solution we choose (cloud or on-site server). These shifting system boundaries can be modeled by defining alternative solutions that are used under certain conditions or rules. A typical approach for defining alternative solutions, extensively used in software engineering, which is suited for PSS is feature

models allowing to define features that represent customer-related characteristics of a product or a service.

While the PSS literature oftentimes represents services as functions, one of the challenges is that emerging services are collaborative, involving multiple actors, and involve multiple systems and infrastructure elements that are dynamically reconfigured (Fakhfakh et al. 2019). Such services rely on a system of systems (a set of independent systems) and combine multiple lower-level services. Hence, in addition to the interfaces which are created between individual systems in system of systems, engineering systems PSS further add interfaces between services as well as between services and systems.

Conclusions

System architecting has been a cornerstone activity in defining systems that are durable and sustainable and with long lifespans. System architecture is critical in designing engineering systems as it is a focal point where novel designs are discussed. A key aspect of addressing system architecture is identifying, modeling, and managing critical interfaces. System interfaces have been identified as a key point in designing, modeling, and managing system architecture. Best practices, methods, approaches, and tools have been developed actively by different communities, both academic and industrial, in order to support this process as it has been clearly identified as key to business development strategy as well as overall company success. A considerable body of knowledge has been developed in order to support system architecting, and this collaboration process involves numerous and different stakeholders.

Nowadays, systems are changing. They are becoming more complex, integrating new technologies such as Internet of things, cloud computing, big data, etc. These novel technologies add novel layers to systems that are to be developed and increase the notion of services. This extension is adding critical interfaces to system architecture, which need to be identified and managed. For example, in system of systems, individual systems are collaboratively integrated, introducing interfaces between actors and systems, as well as between systems. Product service systems (PSS) as another example of system of systems introduce interfaces of collaboration between actors but also introduce interfaces between products and services, as well as between services.

These challenges actually become even more critical as sustainability issues and challenges for sustainable design increase the need to foster and support system architecting in order to propose systems that are more durable, reconfigurable, upgradeable, or flexible. Engineering systems are essential in maintaining the functioning of our society, in a context of diminishing resources. Hence, system architecting seems yet to be discovered and supported to advance collaboration to better our future at a global level.

References

- Albers A, Klingler S, Ebel B (2013) Modeling systems of objectives in engineering design practice. DS 75-1: proceedings of the 19th international conference on engineering design (ICED13), design for harmonies, vol 1: design processes, Seoul, 19–22.08.2013
- Alexander I, Kiedaisch F (2002) Towards recyclable system requirements. In: Proceedings ninth annual IEEE international conference and workshop on the engineering of computer-based systems. IEEE, pp 9–16
- Baldwin CY, Clark KB, Clark KB (2000) Design rules: the power of modularity. MIT Press
- Berkovich M, Leimeister JM, Hoffmann A, Krcmar H (2014) A requirements data model for product service systems. *Requir Eng* 19:161–186
- Boardman J, Sauser B (2006) System of Systems-the meaning of of. 2006 IEEE/SMC international conference on system of systems engineering. IEEE, 6 pp
- Boas RC (2008) Commonality in complex product families: implications of divergence and lifecycle offsets. Massachusetts Institute of Technology, Engineering Systems Division
- Browning TR (2001) Applying the design structure matrix to system decomposition and integration problems: a review and new directions. *Eng Manage IEEE Trans* 48:292–306
- Bryant C, McAdams DA, Stone RB (2005) A computational technique for concept generation. In: ASME international design engineering technical conference & computers and information in engineering conference, Long Beach
- Bytheway C (2007) FAST Creativity & Innovation: rapidly improving processes, product development and solving complex problems. J. Ross Publishing
- Cameron BG, Crawley EF, Loureiro G, Rebentisch ES (2008) Value flow mapping: using networks to inform stakeholder analysis. *Acta Astronaut* 62:324–333
- Cheung J, Scanlan J, Wong J, Forrester J, Eres H, Collopy P, Hollingsworth P, Wiseall S, Briceno S (2012) Application of value-driven design to commercial aeroengine systems. *J Aircr* 49: 688–702
- Cole R (2009) System of systems architecture. In: System of systems engineering: principles and applications. CRC Press, Boca Raton, pp 37–70
- Collopy PD, Hollingsworth PM (2011) Value-driven design. *J Aircr* 48:749–759
- Crawley E (2009) Identifying value-reducing ambiguity in the system. Lecture Notes for ESD. 34 System Architecture
- Crawley E, Cameron B, Selva D (2016) System architecture: strategy and product development for complex systems. Pearson
- De Weck OL, Suh ES, Chang D (2003) Product family and platform portfolio optimization. ASME 2003 international design engineering technical conferences and computers and information in engineering conference. American Society of Mechanical Engineers Digital Collection, pp 175–185
- De Weck OL, Krob D, Lefei L, Lui PC, Rauzy A, Zhang X (2020) Handling the COVID-19 crisis: towards an agile model-based systems approach. *Syst Eng* 23:656
- Dehoff B, Levack D, Rhodes R (2009) The functional breakdown structure (FBS) and its relationship to life cycle cost. 45th AIAA/ASME/ASEE joint Propulsion conference, Denver
- Department of Defense (2020) DoD architecture framework Version 2.02
- Emes MR, Bryant PA, Wilkinson MK, King P, James AM, Arnold S (2012) Interpreting “systems architecting”. *Syst Eng* 15:369–395
- Eppinger S, Salminen V (2001) Mapping of interactions in the product, organization, process architectures. ICED proceedings
- Erden MS, Komoto H, Van Beek TJ, D’amelio V, Echavarria E, Tomiyama T (2008) A review of function modeling: approaches and applications. *Artif Intell Eng Des Anal Manuf* 22:147–169
- Fakhfakh S, Hein AM, Jankovic M, Chazal Y (2019) Towards an uncertainty framework for product service systems of systems. In: International conference on engineering design, Delft
- Feng W (2013) Strategic management for large engineering projects: the stakeholder value network approach. Massachusetts Institute of Technology

- Fixson SK (2005) Product architecture assessment: a tool to link product, process, and supply chain design decisions. *J Oper Manag* 23:345–369
- Freeman R (1984) Strategic management: a stakeholder approach. Pitman, Boston
- Fricke E, Schulz AP (2005) Design for changeability (DfC): principles to enable changes in systems throughout their entire lifecycle. *Syst Eng* 8, no–no
- Friedenthal S, Moore A, Steiner R (2014) A practical guide to SysML: the systems modeling language. Morgan Kaufmann
- Gero JS, Kannengiesser U (2004) The situated function–behaviour–structure framework. *Des Stud* 25:373–391
- Gershenson J, Prasad G, Zhang Y (2003) Product modularity: definitions and benefits. *J Eng Des* 14:295–313
- Gonzalez-Zugasti JP, Otto KN, Baker JD (2000) A method for architecting product platforms. *Res Eng Des* 12:61–72
- Haley BM, Dong A, Tumer IY (2014) Creating faultable network models of complex engineered systems. ASME 2014 International design engineering technical conferences and computers and information in engineering conference. American Society of Mechanical Engineers Digital Collection
- Haskins C, Forsberg K, Krueger M, Walden D, Hamelin D (2006) Systems engineering handbook. INCOSE
- Hein AM, Rudelle J-B (2020) Energy limits to the gross domestic product on earth. arXiv preprint arXiv:2005.05244
- Hein AM, Jankovic M, Feng W, Farel R, Yune JH, Yannou B (2017) Stakeholder power in industrial symbioses: a stakeholder value network approach. *J Clean Prod* 148:923–933
- Hein A, Jankovic M, Chazal Y (2018) A methodology for architecting collaborative product service system of systems. IEEE 13th system of systems engineering conference, June 19–22, Paris
- Holt J, Perry SA, Brownsword M (2012) Model-based requirements engineering. IET
- Huang C-C, Kusiak A (1998) Modularity in design of products and systems. *IEEE Trans Syst Man Cybern Part A Syst Humans* 28:66–77
- ISO/IEC/IEEE (2011) Systems and software engineering – architecture description 42010:2011
- Jankovic M, Holley V, Yannou B (2012) Multiple-domain design scorecards: a method for architecture generation and evaluation through interface characterisation. *J Eng Des* 23:743–763
- Jiao J, Chen C-H (2006) Customer requirement management in product development: a review of research issues. *Concurr Eng* 14:173–185
- Keating CB, Katina PF (2011) Systems of systems engineering: prospects and challenges for the emerging field. *Int J Syst Syst Eng* 2:234–256
- Keating C, Rogers R, Unal R, Dryer D, Sousa-Poza A, Safford R, Peterson W, Rabadi G (2003) System of systems engineering. *Eng Manag J* 15:36–45
- Maier MW (1996) Architecting principles for systems-of-systems. *INCOSE Int Symp* 6:565–573
- Mattson CA, Messac A (2005) Pareto frontier based concept selection under uncertainty, with visualization. *Optim Eng* 6:85–115
- Maurer MS (2007) Structural awareness in complex product design. Ph.D Thesis, Technical University of Munich
- McManus HL, Hastings DE, Warmkessel JM (2004) New methods for rapid architecture selection and conceptual design. *J Spacecr Rocket* 41:10–19
- Meadows DH, Meadows DL, Randers J, Behrens WW (1972) The limits to growth. *N Y* 102:27
- Meilich A (2006) System of systems (SoS) engineering & architecture challenges in a net centric environment. 2006 IEEE/SMC international conference on system of systems engineering. IEEE, 5 pp
- Miles LD (2015) Techniques of value analysis and engineering. Miles Value Foundation
- Miller SW, Simpson TW, Yukish MA, Stump G, Mesmer BL, Tibor EB, Bloebaum CL, Winer EH (2014) Toward a value-driven design approach for complex engineered systems using trade space exploration tools. ASME 2014 International design engineering technical conferences and

- computers and information in engineering conference. American Society of Mechanical Engineers Digital Collection
- Ministry_of_Defense (2020) MOD architecture framework. Ministry of Defense
- Moullec M-L, Bouissou M, Jankovic M, Bocquet J-C, Réquillard F, Maas O, Forgeot O (2013) Toward system architecture generation and performances assessment under uncertainty using Bayesian networks. *J Mech Des* 135:041002–041001
- Moullec M-L, Jankovic M, Eckert C (2015) The impact of criteria in system architecture selection: observation from industrial experiment. International conference on engineering design (ICED), July 27–30, Milan
- Nato (2018) NATO architecture framework, version 4
- Pacheco CL, Garcia IA, Calvo-Manzano JA, Arcilla M (2015) A proposed model for reuse of software requirements in requirements catalog. *J Softw* 27:1–21
- Parslov JF, Mortensen NH (2015) Interface definitions in literature: a reality check. *Concurr Eng* 23: 183–198
- Pimmler TU, Eppinger SD (1994) Integration analysis of product decompositions. In: ASME conference on design theory and methodology conference, Minneapolis
- Qasim L, Hein A, Jankovic M, Garnier J-L (2019) Towards a reconfiguration framework using data collected from the use phase. International conference on engineering design (ICED), August 5–8, Delft
- Qasim L et al (2021) A model-based method for system reconfiguration submitted to the journal of mechanical design. *J Mech Design*. To be Published
- Ross AM, Hastings DE, Warmkessel JM, Diller NP (2004) Multi-attribute tradespace exploration as front end for effective space system design. *J Spacecr Rocket* 41:20–28
- Sarkar S, Dong A, Henderson JA, Robinson P (2014) Spectral characterization of hierarchical modularity in product architectures. *J Mech Des* 136:011006
- Scherer H, Albers A, Bursac N (2017) Model based requirements engineering for the development of modular kits. *Proc CIRP* 60:145–150
- Shishko R, Aster R (1995) NASA systems engineering handbook. NASA Special Publication, 6105
- Simmons WL (2008) A framework for decision support in system architecting. PhD, Massachusetts Institute of Technology
- Simpson TW, Maier JR, Mistree F (2001) Product platform design: method and application. *Res Eng Des* 13:2–22
- Song W (2017) Requirement management for product-service systems: status review and future trends. *Comput Ind* 85:11–22
- Sousa-Poza A, Kovacic S, Keating C (2008) System of systems engineering: an emerging multi-discipline. *Int J Syst Syst Eng* 1:1–17
- Steward D (1962) On an approach to the analysis of the structure of large systems of equations. *SIAM Rev* 4:321–342
- Steward D (1981) The design structure system: a method for managing the design of complex systems. *IEEE Tran Eng Manage* 28:79–83
- Stump G, Simpson T, Yukish M, Bennett L (2002) Multidimensional visualization and its application to a design by shopping paradigm. 9th AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimization. 5622
- Sutherland TA (2009) Stakeholder value network analysis for space-based earth observations. Massachusetts Institute of Technology
- The_Open_Group (2020) The TOGAF Standard, Version 9.2
- Toval A, Nicolás J, Moros B, García F (2002) Requirements reuse for improving information systems security: a practitioner's approach. *Requir Eng* 6:205–219
- Tukker A, Tischner U (2006) New business for old Europe: product-service development, competitiveness and sustainability. Greenleaf Publications
- Vermaas PE, Eckert CM (2013) Special issue “Functional descriptions in engineering”. *Artif Intell Eng Des Anal Manuf* 27

- Winer E, Abdul-Jalil M, Bloebaum C (1998) Development of a geographic independent virtual design environment for large-scale design. 7th AIAA/USAF/NASA/ISSMO symposium on multidisciplinary analysis and optimization. 4744
- Wyatt D, Wynn D, Clarkson J (2008) Synthesis of product architectures using a DSM/DMM-based approach. 10th international design structure matrix conference, Stockholm
- Yassine AA, Braha D (2003) Complex concurrent engineering and the design structure matrix method. *Concurr Eng* 11:165–176
- Yassine A, Whitney D, Daleiden S, Lavine J (2003) Connectivity maps: modeling and analysing relationships in product development processes. *J Eng Des* 14:377–394
- Zachman J (2006) The zachman framework for enterprise architecture. Zachman Framework Associates Virginia
- Zhang Z (2007) Effective requirements development-a comparison of requirements elicitation techniques. In: Berki E, Nummenmaa J, Sunley I, Ross M, Staples G (eds) *Software quality management XV: software quality in the knowledge society*. British Computer Society, pp 225–240
- Ziv-Av A, Reich Y (2005) SOS – subjective objective system for generating optimal product concepts. *Des Stud* 26:509–533
- Zubrin R (2011) Case for mars. Simon and Schuster