25th Annual INCOSE International Symposium (IS2015)
Seattle, WA, July 13 – 16, 2015

System Interface Engineering

Dieter Scheithauer
Breitensteinstr. 26, 83727 Schliersee, Germany
dieter.scheithauer@hitseng.eu

Abstract. The definition of the systems engineering value stream is based on modelling the information flow. Information flow models are beneficial for performing systems engineering management efficiently. The benefits of unidirectional information flow models may give the impression that systems engineering as its best is a deductive process with the stakeholder requirements leading to unique and unambiguous solutions. However, human communication and natural laws feature mutual dependencies that become evident in systems engineering especially when system interfaces and process interfaces are considered. This paper amends two previous papers on the systems engineering value stream by system interface engineering considerations. Starting with clarifying the scope of system interface engineering, the paper proposes a precise terminology and defines the additional system interface engineering activities integrated into the systems engineering value stream.

Introduction

For improved systems engineering performance, a process definition concentrating on the flow of information provides significant advantages. Just focusing on the resulting information itself may lead to inefficient systems engineering management. The not uncommon occurrence of schedule and cost overruns in conjunction with deficient product quality demonstrates the improvement potential for achieving a more successful systems engineering based on systems engineering value stream thinking. Queuing theory provides a theoretical foundation why flow management with multiple iterations is better suited for managing the development of innovative systems (Reinertsen 2009). Flow management enables tight control of all the iterations needed for incorporating system improvements well controlled and with minimized lead time.

Two previous papers are concerned with defining the systems engineering value stream. Scheithauer and Forsberg (Scheithauer and Forsberg 2013) interpret the V as the overall systems engineering value stream as unidirectional information flow in accordance with lean principles (Womack and Jones 2003; Oppenheim 2011). The overall systems engineering value stream describes the information flow across the systems and system elements within a system architecture. The hand-over of information within a system architecture is facilitated by configuration baselines defining each system and system element, see Figure 1. The quality of a configuration baseline is equivalent to the quality of the particular system or system element itself. Only information referenced in a released configuration baseline is allowed to be used for the development of other systems and system elements.

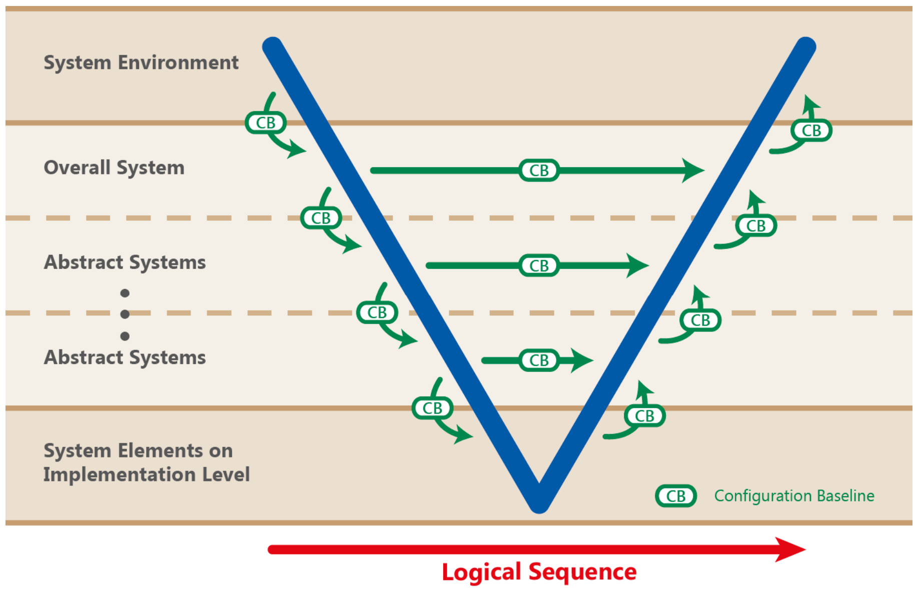


Figure 1: The Flow of Configuration Baselines in the V

The second published paper is concerned with the value streams for generating individual configuration baselines (Scheithauer 2014). These value streams are designated as work product generation sequences. Work products contain only information that is needed as input by downstream processes and activities. All other relevant information is subsumed as supporting data. In general, supporting data is important to understand the evolution of a system or system element. Released versions of work products are the smallest units of value from a configuration management perspective. Released versions of work products need not always to be complete regarding the final content expected at the end of development, but their quality must have been checked before release, and all omissions regarding incompleteness and inconsistencies must be stated within the work product. Supporting data should be stored and archived in the context of the released versions of work products they contributed to.

This paper is concerned with system interface engineering. System interface engineering considerations are essential to complete the high-level description of the value stream approach to systems engineering. Without considering system interface engineering, the other two papers may support an impression that systems engineering is a deductive exercise managed centrally at its best. For rather innovative systems for which many technologies must be invented in the course of development, this may be an adequate model since all information needs to be cascaded across the system architecture by requirements. But most systems today are not developed anymore in such a green-field approach. A more market-opportunistic behavior characterizes our sophisticated industrial cultures utilizing the already accumulated knowledge and experience of all the enterprises in the supply chain. An expectation that an enterprise just considers the allocated requirements and not all the knowledge and experience acquired in the past falls short of acknowledging our everyday experience and preferences. Usually, we prefer suppliers that have already run similar projects successfully with a reputation to be high-quality suppliers.

Utilizing Commercial-of-the-Shelf (COTS) in military and further government procurement, and INCOSE's intention for industrial outreach are demanding processes, methods and tools that integrate provisions for a more sophisticated system interface engineering aside pure hierarchical information flow.

System Interface Engineering Terminology

In order to evaluate the scope of system interface engineering, this section starts with the descriptions of two rather different system interface engineering scenarios. From these scenarios, a general definition of the term system interface is derived. Furthermore, a distinction between external and internal system interfaces is introduced with respect to system interfaces and process interfaces. Finally, the various views on interfaces as information flows, physical interfaces, and the implications of human communication are considered.

Scenario A. In a helicopter development program, the engineering team in charge of the overall helicopter makes a decision that a particular automatic flight control mode uses input data from a dedicated sensor assigned to the avionic system. The avionic system and the flight control system are developed by two separate engineering teams. In the course of their design activities, the avionic engineering team details the system interface by defining a resolution of the signal transmitted to the flight control system. Months later, the flight control engineering team admitted that they are unable to meet the accuracy requirements for the particular autopilot mode due to the coarse resolution of the received sensor signal.

The following evaluation of Scenario A focuses on answering two questions. First, was the avionic team right to unilaterally define the signal resolution? Second, what are the alternative ways of conduct for detailing system interfaces?

For discussing the first question, the roles of the three engineering teams have to be investigated. Each engineering team has to take responsibilities regarding the quality of their particular system (Scheithauer 2014-2). Each engineering team is committed towards stakeholders, the engineering team in charge of the upper level system, and the engineering teams responsible for the lower level systems. Regarding stakeholders including customers, each engineering team has its own particular share of responsibility in order to satisfy stakeholder needs. With respect to the engineering team in charge of the upper level system, an engineering team has to consider the demands and intentions expressed in the allocated requirements imposed on their system. The responsibility with respect to the teams engineering the lower level systems is fulfilled by clear and feasible demands expressed by the allocated requirements forwarded to the system elements. In addition, each engineering team has to take over self-responsibility by carefully applying the knowledge and experience of all the engineering disciplines involved in the engineering team, and by generating optimum value. The latter means to comprehensively define the functions and features of a system that are emergent on the particular architectural level without overly constraining the engineering teams in charge of lower level systems.

According to these criteria, the engineering team in charge of the overall helicopter has no obligation to define the signal resolution. The signal resolution has no impact on their design decision to assign the sensor to the avionic system. Only, if the technical feasibility of the interface would have been in question, they would have been obliged to dive deeper into the signal characteristics. In conclusion, the involvement of the engineering team in charge of the overall helicopter is not adding value. Indeed, this engineering team may lack the technical expertise to define detailed signal characteristics of the system interface at all.

This leaves the avionic engineering team and the flight control engineering team as the two candidates to detail the signal characteristics. As it is hard to define a clear precedence among them, it may be wishful to demand that both engineering teams need to be involved at once.

However, this would demand a level of synchronization not easily achievable in practice, and having a detrimental impact on the development flow.

Now turning to the second question: What are the alternative ways of conduct for detailing system interfaces? In Scenario A, the avionic engineering team was advanced in their design. If they would have approached the flight control engineering team immediately, three different responses would have been possible from the flight control engineering team. A clear yes or no would have been the two simple cases. More likely in this scenario, they would have been advised to ask again a few months later since the flight control engineering team had not entered the design of the particular autopilot mode yet. In case of such a response, the avionic engineering team has two alternatives to react: pausing their work on the topic, or continuing on the assumption that they get the approval for their design decision later. Both alternatives are valid. Process definitions demanding always pausing the work would allow the slowest progressing engineering team to determine the overall pace of development. This is rarely tolerated from a management perspective, and will be circumvented in practice. Similarly, always progressing without the consent of the other engineering team would be foolish as well. The decision for progressing relying on a later approval by the other engineering team should be based on the risk balancing the potential gain from continuing concurrent engineering and the costs associated with potential corrective actions. Even when this trade-off is performed by each engineering team on its own, the overall progress status of the program should be considered as well.

Scenario B. The engineering team designing a fighter aircraft makes a fundamental design decision for a single pilot instead of a crew of two in order to reduce aircraft mass and thereby maximizing agility. They understand that the pilot workload for basic flight control has to be minimized in order to free pilot capacity for performing other mission tasks. Consequently, they demand carefree maneuvering capabilities from the flight control system. The flight control engineering team interprets this allocated requirement. They have to understand the implications on their flight control system design tasks for developing a feasible solution. This includes a number of constraints including a minimum altitude above ground from that onwards the carefree maneuvering capability will be available, and pilot capabilities satisfying normal pilot skills. In the course of the flight control law definition they may find out that they could go for a better balance between the competitive requirements for agility and flight envelope protection, if the pilot capabilities would gradually exceed normal pilot skill levels. This has an impact on pilot training. A conflict may arise with the envisaged flight training infrastructure, if the additional routine training demanded cannot be performed using the training simulators installed in the fighter wings, but has to be performed on the few more sophisticated flight simulators installed at a central pilot training facility.

Scenario B has a lot in common with Scenario A. However, there are two major differences. First, this system interface has no direct impact on the system interfaces on any lower architectural level of the fighter aircraft. Second, there may be no higher level system as common ancestor of the fighter aircraft and the training facilities having an engineering capability for controlling system interfaces.

One may argue to consider the dependency between flight control law design and the training facilities not as a system interface at all, but as some other sort of dependability. Considering the ignorance of scenarios like Scenario B in the systems engineering text books, could support such a position. Only Weiss touches this issue marginally. He reports about missing awareness in engineering teams about all the neighboring systems they have interfaces with, when he talks about applying N^2 diagrams to the interaction among functional discipline organizations (Weiss 2013).

It is the position taken in this paper that Scenario B is a valid system interface scenario due to the following strong reasons. First, the Scenario B is common place on an economy scale, at least in a market economy. The systems engineering profession needs to consider it and needs to provide advice how to handle this kind of system interfaces as an important contribution for progressing in the direction of the goals of the INCOSE Vision 2025 (INCOSE 2014). Second, the proposed solution how to integrate system interface engineering into the systems engineering value stream is well suited to cope not only with conditions according to Scenario A, but also with Scenario B type system interface issues.

System Interface – A Definition. ISO 15288 (ISO 2008) defines a system as a combination of interacting elements organized to achieve one or more stated purposes. A system element is a member of a set of elements that constitutes a system. No definitions for the terms interface or system interface are provided by ISO. Considering that the definition of the term system relies on the mutual dependency between system elements and interactions, a definition of the term system interface compatible to the two ISO definitions for system and system element could read: A system interface is an interaction between system elements. The ISO definitions follow a theme that interfaces are internal to a system. In order to include system interfaces according to Scenario B, we need at the outmost to emphasize the existence of a system architecture with the whole world as the top level node. However, this top level system is not subjected to ordinary engineering, and a single, unique system architecture may not be deducible from this top level node at all. Nevertheless, sometimes such a supportive assumption may be helpful as pure theoretical construct.

The understanding of system interface engineering in most text books on systems engineering is aligned with the scope implicitly defined by ISO 15288. From the text books reviewed (Buede 2009; Hybertson 2009; Parnell et al. 2011; Kossiakoff et al. 2011; Wasson 2006; Weiss 2013) nearly all use the term interface when considering system interfaces. Only Wasson more precisely talks about system interfaces. It is the proposal of and the convention used within this paper to talk about system interfaces when talking about the interfaces of the engineered system. The information flows within and between engineering teams are subsumed as process interfaces. For being precise, the term interface itself is not used due to its ambiguity unless both, system interfaces and process interfaces, are considered together.

The proposed definition of the term system interface is defined according to the system architecture terminology as illustrated in Figure 2. System interfaces of a system comprise the interaction of a system with its neighboring systems. A neighboring system is a system in another branch of the system architecture. In other words, neighboring systems have a common higher level system ancestor, at least theoretically. This hierarchy may be rather virtual as there may be no structures in the economy controlling the information flow from the level of the common ancestor. Only planned economies try to implement such a hierarchical model. Market-oriented economies are better illustrated as networks of interoperating enterprises. However, from the viewpoint of a particular system the assumption of such a virtual system architecture may be an appropriate approximations. Important is that all neighboring systems featuring system interfaces actually are considered completely.

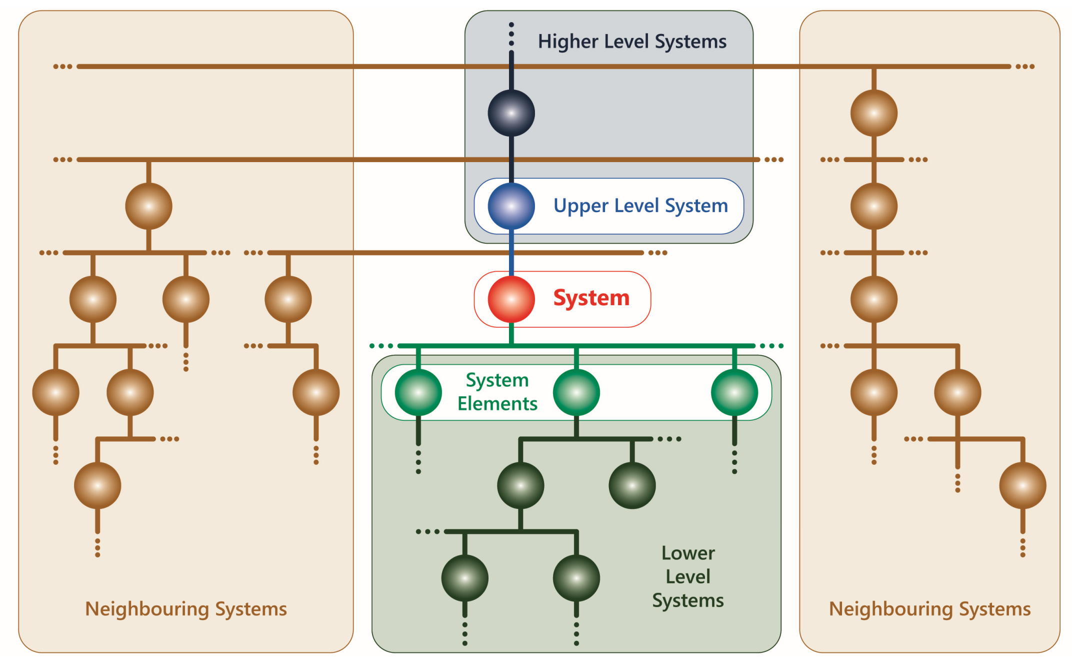


Figure 2: System Architecture Terminology

The definition of the term system interface makes no assumptions about the architectural levels the systems sharing a common system interface are placed on. This may depend on the scope of the emergent functions and features of both systems. The detailing of system interfaces may even affect neighboring systems on several architectural levels in a branch of the system architecture, depending on the value generation depths of each system in the particular branch. Consequently, the engineering team in charge of a particular system may interact with the engineering teams of the neighboring systems on several architectural levels.

External and Internal System Interfaces. A distinction between external and internal system interfaces is commonplace and can be found in most of the reviewed text books. Internal system interfaces are the system interfaces between the system elements of a system and external system interfaces are the system interfaces of the system with neighboring systems. Considering the process interfaces in addition, a slightly different distinction between internal and external interfaces is beneficial. In the architecture definition an engineering team in charge of a system defines the system elements together with the system interfaces among those system elements. For good reasons, these system interfaces are designated as internal system interfaces. The utmost responsibility for the feasibility of implementing these system interfaces within the affected lower level systems on all architectural levels successfully resides with the particular engineering team.

For the engineering teams in charge of the system elements, the internal system interfaces of the engineering team in charge of their upper level system are already external system interfaces. If one of these engineering teams cannot define a feasible technical solution for an external system interface, they need to escalate the issue upwards the system architecture. The escalation may need to continue until the engineering team for which this system interface is an internal system interface becomes involved. Considering that each architectural definition does not only define internal system interfaces, but also allocates external system interfaces to particular system elements, the escalation cascade may affect several architectural levels.

Corrective actions may then be investigated for any of the affected higher level systems for containment in order to avoid the escalation further upwards the system architecture.

Information Flows, Physical Interfaces, and Human Communication. Systems engineering text books propose a distinction between logical and physical interfaces. Physical interfaces may be easily conceived as mounts, electronic circuitry, electrical wiring and so forth. The definition of what a logical interface might be remains always a bit weak. Buede follows in his text book a rather information centric approach (Buede 2009). Incidentally, he generates a lot of evidence that logical interfaces are coincident with modelling an interface as information flow. Due to these hints, a more concise distinction between logical interfaces, e.g. information flows, and physical interfaces may be proposed.

Information flows are unidirectional. Information flows from one source to one or multiple sinks. In systems engineering, information models provide multiple benefits. In the scope of the subject of this paper, they provide abstracted views supportive to define system interfaces as well as process interfaces. However, the implementation of information flows finally relies on some sort of physical implementation. All physical implementations have to deal with natural laws. Physical laws always define mutual dependencies as for example Newton's laws and Kirchhoff's laws. Nevertheless, engineers have been creative to make electrical energy flow into one direction, and to generate controlled movement in any intended direction with the technologies and products they invented. This is achieved not by cheating nature, but by exploiting large differences in magnitude. Physical stress can be measured because the monitored structure is much stiffer than the strain gauge. Operational amplifiers nearly eliminate the impedance of the output load on the input by very high amplification rates.

Standardization of physical interfaces and adjacent network layers reduces the risk when information flows are converted into a physical implementation. The reliance on interface standards increases the confidence of engineering teams in their assessment of the feasibility that their design may be implemented successfully by lower level system. However, standardization always codifies the knowledge and experience of the past. Innovative systems and system designs may reveal the limitations of certain interface standards, and of the combined application of several interface standards in the particular context.

When considering process interfaces, the physical interfaces may only impose a process risk under certain conditions like in collaborative engineering distributed over the globe. A higher risk to process interfaces is imposed by the intricacies of human communication (Schulz von Thun 2010). In a green-field approach, systems engineering may be modelled as a deductive process claiming that lower level requirements can be fully deduced from higher level requirements. Then, it is reasonable for highly innovative systems based on new technology to define a completely new terminology and associated semantics that have to be used by all engineering teams across the system architecture. In more market-opportunistic approaches, existing terminology and semantics of multiple engineering disciplines clash on each other. No authority converging all the different discipline specific languages will be available as their will be nobody with a deep understanding of all the disciplines involved. To further illustrate the difference between information exchange and human communication consider the following everyday life example:

Two persons are sitting in a passenger car. At once, the passenger says to the driver "The traffic lights are red". The information content of this message is quite clear: The traffic lights are red. However, this is only one part of the story. The first question to ask is what motivated the passenger to talk about the traffic lights in this moment. It may be assumed, that the passenger noted in addition to the red traffic lights that the driver did not start to decelerate the car as expected usually. Therefore, the passenger may have talked about the red traffic lights by courtesy to appeal to the driver to decelerate the car.

There is a further relationship aspect concerned with how the driver receives this message. Let us assume that driver and passenger are a married couple. If the wife is the passenger, her husband may just react on the reminder – and/or due to own observation - by decelerating the car. In the other case, a debate may start in addition about not trusting the driver's skills and the execution of male dominance.

We can learn a few things from this example. First, human communication features content and relationship aspects (Watzlawick, Beavin and Jackson 1969). Second, every piece of information exchanged gets its meaning from the social context.

System Interface Engineering in a System-of-Systems Context

So far, system interface engineering has been considered with respect to the basic system architecture. In this section, further issues are addressed that are unique at the boundaries between enterprises. The interpretation of the V as overall systems engineering value stream already considers organizational boundaries. The system environment above the overall system subsumes all the higher level systems and the neighboring systems of the overall system. Within the enterprise, the overall system and all the abstract systems are engineered. The system elements on the implementation level at the bottom of the V comprise all the lower level systems procured from suppliers.

This section features two topics. First, the role of Interface Control Documents is discussed. Second, a product taxonomy is introduced considering the complete range of procured systems from Commercial-Of-The-Shelve products to systems developed on demand.

Interface Control Documents (ICDs). ICDs are an important asset in the systems engineering inventory. They are the means of choice for interface control (Parnell, Driscoll and Henderson 2011). They are of critical importance for contracting system elements on the implementation level. They are aiming for a detailed and complete description of all the system interfaces of a system element on the implementation level. Usually, ICDs are compiled according to pre-defined templates. Checking that all sections and data fields are filled out and contain the appropriate data, is the main technique to assess the completeness of the system interface definition. According to Parnell et al., ICDs frequently contain mind-numbing details. Of course, compiling a high-quality ICD is in most cases a tedious task. The effort is mainly justified by the gain from high- quality content of supplier contracts. Incompleteness of contract specifications inevitably lead to later contract changes. Changing contracts may consume a lot of time as not only the technical content, but also commercial issues have to be agreed anew. In large programs, renegotiating supplier contracts quite often provides an opportunity for the supplier to maximize the own profit.

At first sight, it surprises why the compilation of ICDs has a reputation for being so tedious. Viewing ICDs from a value stream perspective provides an explanation. The compilation of ICDs is at odds with any kind of value stream approach, even with the weak value stream interpretation of common systems engineering text books. Nevertheless, ICDs have their merits and need to be considered by a strict systems engineering value stream approach as well. From above, it should have become clear that system interfaces, after being introduced as internal interfaces, may be refined as necessary further downwards the system architecture adding more details to the system interface definition contained in system requirements and allocated requirements. If the requirement management process is implemented accordingly, ICDs may be derived from the requirements repository as status reports, continuously providing the means to monitor the completeness status of the system interface definitions. Whether then these status reports are configured as separate work products, or are just managed as an integral part of the allocated requirements imposed on the appropriate system elements on the implementation level, may follow organizational preferences.

Product Taxonomy. The product taxonomy described in this section has been proposed before (Scheithauer 2015). Figure 3 contains an overview showing three distinct product categories. Category I products are COTS products. The interfaces of COTS products are fixed. It is the responsibility of the organization utilizing a COTS product to design all provisions that the COTS product is successfully integrated into the upper level system. Category III products constitute the opposite approach. The upper level system is widely free to define a custom interface for the system element. To a great extent, the demonstration that the system element can be integrated into the upper level system may be delegated to the supplier. Category II products comprise all COTS products that allow the selection of particular options. Three sub-categories are proposed. When all the options are taken as offered and the combination of options is valid, products fall into Category IIa. They have to be integrated into the upper level system like Category I products. COTS products of Categories IIb and IIc demand adoptions of the manufacturing process and additional system development activities respectively. The procurement of Category IIb and IIc products may bear some risks, especially when the procuring organization still assumes to procure a COTS product, but some problems have been encountered having wider implications on system design and manufacturing.

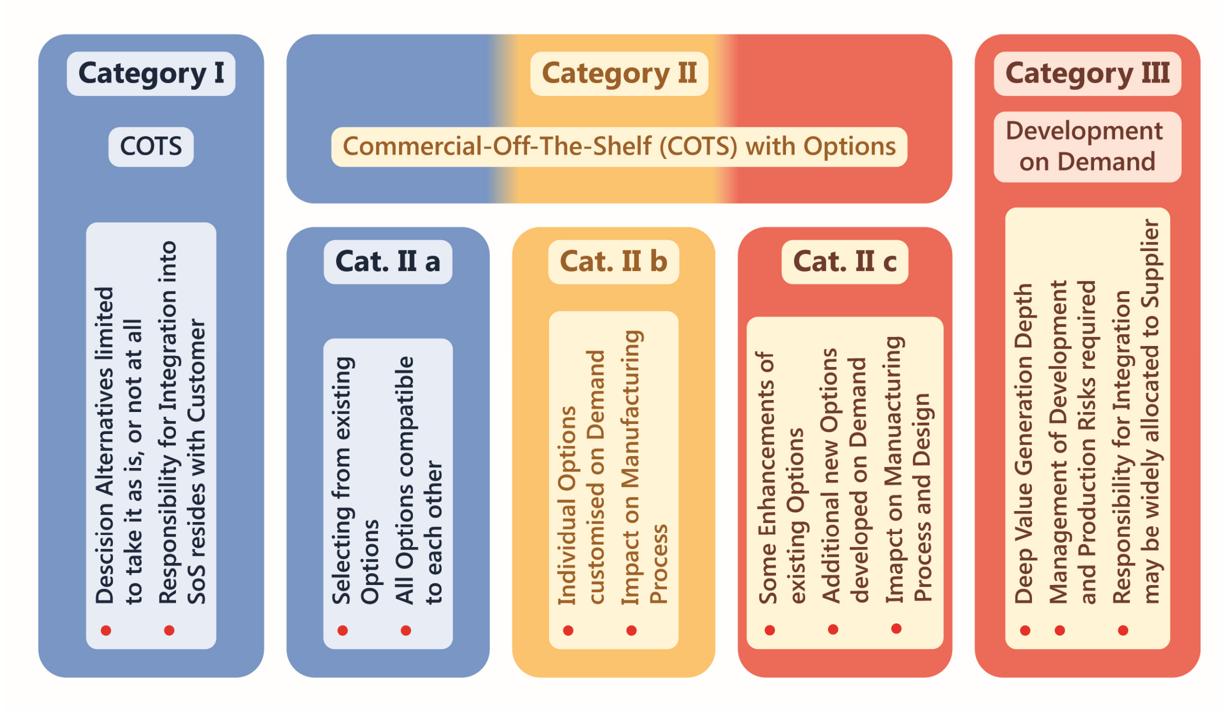

Category I
COTS
Descision Alternatives limited to take it as is, or not at all
Responsibility for Integration into SoS resides with Customer
Category II
Commercial-Off-The-Shelf (COTS) with Options
Cat. II a
Selecting from existing Options
All Options compatible to each other
Cat. II b
Individual Options customised on Demand
Impact on Manufacturing Process
Cat. II c
Some Enhancements of existing Options
Additional new Options developed on Demand
Imapct on Manuacturing Process and Design
Category III
Development on Demand
Deep Value Generation Depth
Management of Development and Production Risks required
Responsibility for Integration may be widely allocated to Supplier

Figure 3: Product Taxonomy

Integrating System Interface Engineering into the Systems Engineering Value Stream

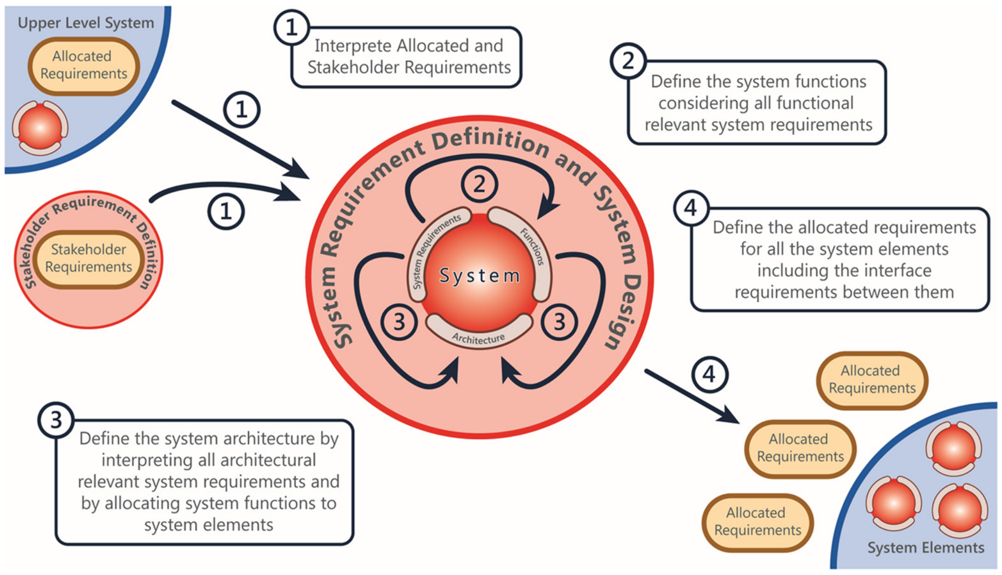
This section contains the process definition amendments for system interface engineering. The process definitions are aligned with the coarse process breakdown laid down in the paper about the V-model views (Scheithauer and Forsberg 2013). We start with the four development sub-processes: (1) Stakeholder Requirement Definition, (2) System Requirement Definition and System Design, (3) System Integration Preparation, and (4) System Integration. Afterwards, the assurance sub-processes validation and verification are addressed. Finally, specific provisions for systems engineering management are considered. This includes the requirement management of system interface requirements, and change control activities on system interfaces.

Stakeholder Requirement Definition. When eliciting and defining stakeholder requirements, the attention concentrates on the individual stakeholders and stakeholder groups to gain as much insight into their particular needs as possible. Those needs may include demands on system interfaces in which only a single stakeholder or stakeholder group is interested. However, many stakeholder statements may only be understood, if dependencies between several stakeholders or stakeholder groups are considered in combination. Overlapping, additive or even competing needs for system interfaces may exist among various stakeholders or stakeholder groups. Extra care has to be taken when innovative system solutions are envisaged. In this case, traditional roles and dependencies between various stakeholders or stakeholder groups may become obsolete, or new dependencies may result. In this case, it is recommended to clarify the boundaries of each stakeholder or stakeholder group regarding their interest on the system carefully.

System Requirement Definition and System Design. System requirement definition and system design result in the generation of three complementary system views in terms of system requirements, functional definition, and architectural definition. Together, these three views have to provide a consistent and complete description of the system's design. The whole sub-process may be further decomposed into four steps as shown in Figure 4.

In the first step, the allocated requirements and further stakeholder requirements are analyzed. It is important for the engineering team to gain complete awareness of all external system interfaces. For this reason, all external system interface requirements have to be traced upwards the system architecture until the architectural level introducing the corresponding system interfaces is reached. For system interfaces external to the overall system, this means to go upwards through all architectural levels to the overall system. When the overall system level is reached, the system life cycle description including the concept of operations has to be evaluated for identifying neighboring systems for which no common ancestor can be found in the mapped system architecture.

In many cases, system interface requirements may find their origin in internal system interfaces established somewhere upwards in the system architecture. For all requirements found by this traceability exercise, the recorded rationale needs to be analyzed in order to understand the implications on the own system or system element. Furthermore, from all requirements found, the downward traces in other branches of the system architecture should be followed. Thus, the engineering team gets full awareness which other systems may have a stake in their external system interfaces, and which refinements of the system interface requirements have occurred in other branches. The first step results in a first set of system requirements. Compared to the allocated requirements imposed on the system and further stakeholder requirements elicited, the system requirements may be refined in order to be precise in the language of the engineering disciplines represented in the engineering team in charge of the development of the particular system.



Upper Level System
Allocated Requirements
Stakeholder Requirement Definition
Stakeholder Requirements
1 Interprete Allocated and Stakeholder Requirements
2 Define the system functions considering all functional relevant system requirements
System Requirement Definition and System Design
System
System Requirements
Functions
Architecture
4 Define the allocated requirements for all the system elements including the interface requirements between them
3 Define the system architecture by interpreting all architectural relevant system requirements and by allocating system functions to system elements
Allocated Requirements
Allocated Requirements
Allocated Requirements
System Elements

Figure 4. System Requirement Definition and System Design

The second step starts with identifying functions and sub-functions based on the system requirements. It continues with further elaboration of the functions and sub-functions resulting in functional models of the system. The freedom in designing functions is constrained by the external system interfaces imposed on the system. To some extent, the functional interfaces may be refined for own purposes. As far as external system interfaces are concerned, compatibility with the corresponding interface design in other branches needs to be maintained. For this purpose, harmonization of system interface requirements on a peer-to-peer basis may be required. In the section below on systems engineering management, it is proposed how this may be accomplished. When generating functional models the capability to integrate them with functional models needs to be observed in addition.

The architectural definition establishes the internal system interfaces between the system elements. Furthermore, inherited external system interfaces have to be allocated to the system elements designated for satisfying those external system interfaces. As far as further system interface requirements are established or existing system interface requirements are refined, harmonization with the other systems or system elements involved in those system interfaces has to be achieved.

In the last step, the requirement sets containing the allocated requirements for the system elements need to be established. Not uncommonly, system interface requirements have to be refined in order to describe the demand on the system interface from the viewpoint of the particular system element. When the system element is designated as a system element on the implementation level, e.g. is procured from a supplier, it is important at this step to ensure that system interfaces have been defined completely. For this purpose, the compilation of Interface

Control Documents (ICDs) are an established method. The information content of the ICDs should be a summary of all system interface requirements including all refinements. Completeness of an ICD may be evaluated by checking, if all information requested by the ICD template is available. If this is not the case, the system design is not complete. Due to the implications of immature ICDs, it is then recommended to update the system requirements and the system design first before the allocated requirements are forwarded to the supplier.

System Integration Preparation. System integration preparation comprises the following steps: (1) definition of assurance requirements, (2) definition of the system integration concept, (3) the establishment of system integration environments, and (4) system integration preparation. System interface specific considerations apply to all these activities.

Assurance requirements need to consider the demonstration of compliance of external system interfaces. By demanding comprehensive testing of external system interfaces, risks for the integration of higher level systems are reduced. The benefits are an earlier detection of any flaws in the implementation of external system interfaces and the containment of changes on lower architectural levels reducing the costs for incorporating the lessons learned. Even more important are assurance requirements demanding the execution of internal system interfaces between the system elements. The goal is primarily to verify the functionality and features that are emergent on the particular system level. Again, correction costs stay comparably low in case of eliminating design flaws before they become evident on higher architectural levels.

By default, a system integration concept should consider all system integration activities across the whole system architecture. Overarching integration policies have to be defined. A system integration concept may determine on which level specific kinds of tests, for example timing tests, should be performed primarily. These measures guarantee a balanced and efficient system integration approach. The system integration concept needs to provide guidance about an integration strategy of models that may follow a path for being first used for system design validation and then virtual product validation. Models may also be employed to simulate external system interfaces in the context of particular system integration environments. In all these cases, the models have to provide valid representations of external and internal system interfaces.

Assurance requirements and the system integration concept are the basis for defining and establishing system integration environments. At this step, the system interface specific considerations included in the assurance requirements and the system integration concept have to be obeyed.

For efficiency reasons, system integration planning may follow some sort of incremental approach. System elements may not be implemented completely or may still have some omissions limiting the expected characteristics of system interfaces when particular integration activities may be commenced. It is important to trace all the omissions of system interface requirements in order to avoid any damages of the system integration environment or the equipment to be integrated due to deficient system interface implementations.

System Integration. Successful system integration is rather sensitive for the quality of the implemented system interfaces. System integration shall ensure that all the system interfaces between the system elements to be integrated and between system elements and the system integration environment function as expected. A defensive and progressive system integration sequence is recommended in order to avoid any damages to the equipment. It should be ensured that system interfaces operate as expected before verification is commenced.

Validation. The overall systems engineering value stream defines several validation steps to ensure progressively that the engineering results satisfy stakeholder needs, and that unintended functions are absent. For all validation steps, the system interface specific considerations correspond to the system interface engineering activities defined for the development sub-processes. First, stakeholder requirements are validated. The consistency of the demands from all stakeholders or stakeholder groups regarding system interfaces is checked. Second, when an engineering team receives the allocated requirements imposed on the system they are responsible for, they need to check that all external system interfaces are defined appropriately. This includes upwards traceability checks as well as downward traceability checks in other branches of the system architecture for shared system interfaces. Third, in system design validation, the presence of all external system interfaces in functional models needs to be ensured. Furthermore, the modelling of internal system interfaces needs to be evaluated regarding appropriateness. Fourth, validation of the virtually integrated systems should also consider the correct implementation of external and internal system interfaces without being jeopardized by the system design of system elements on lower architectural levels. Fifth, when it comes to operational validation the external system interfaces of the overall system are examined. In especially, it shall be assured that all stakeholders and stakeholder groups can perform their tasks on the system without conflicts or gaps. Sixth, feedback from real operation of the system should be evaluated regarding conflicts between different stakeholders or stakeholder groups, and any missing or deficient system interfaces should be identified for improving the operation of updated or future systems.

Verification. When verification is performed as planned, all system interfaces should be verified as comprehensively as demanded by the assurance requirements. When the verification coverage analysis reveals omissions and deviations, it is important to evaluate the impact of all omissions and deviations on external system interfaces. Before the results of this analysis is not available, no concessions should be granted.

Systems Engineering Management Considerations. Two scenarios demand specific provisions in systems engineering management. One scenario is concerned with refining system interfaces throughout the system architecture. The other scenario comprises the exceptional case that internal system interfaces cannot be implemented on lower architectural levels as envisaged originally.

In a value stream based approach, the demand for the added value by each engineering team is a paramount criterion to assess efficiency. The added value by an engineering team in charge of a particular system somewhere in the system architecture may be summarized by the following three objectives. First, the engineering team has to propose a technical solution that satisfies the allocated requirements and the stakeholder needs behind. Second, the engineering team develops the system functions and system features that are emergent on the particular architectural level. Third, the engineering team ensures that the system elements can be implemented.

For fulfilling these objectives, it is not needed to define the internal system interfaces exhaustively. Trying to do so, may instead impact overall efficiency adversely for two reasons. Engineering teams on lower architectural levels may have the better knowledge to cope with the intricacies of implementing the system interface, but their solution space is constraint by detailed system interface requirements that serve no real purpose for those who have introduced those constraints on the internal system interface. The other inefficiency results when a too detailed system interface requirement has to be changed. Several architectural levels may be involved in changing the requirement. The value added by the several engineering teams is low, and the change process is rather time consuming. Furthermore, during the time needed for processing the change, the engineering team demanding the change may proceed based on the assumption that the change will finally flow down to them as demanded. Sometimes, a big surprise may result when the requirement change is different compared to what has been demanded for some good reasons claimed by another engineering team. For these reasons, this paper proposes, to refine system interface requirements on a peer-to-peer basis when needed, and to propagate only problems reporting the inability to implement a system interface as required upwards to the engineering team that has defined the internal interface in the first place. In case of external system interfaces of the overall system, the engineering team in charge of the overall system is the final recipient of the problem report, of course.

The peer-to-peer refinement of system interfaces has a further potential to slow-down development progress. When an engineering team is approaching another engineering team for refining a system interface in a specific way, the response may be either an agreement to the change or a clear refusal. A third possible response is the appeal to come back later since the engineering team has not progressed to provide a definitive response and to make a commitment. Due to this third outcome, system interface requirement documents that have to be signed for release by both parties are not the best solution. Such interface requirement documents are elements in two different work product generation sequences. When work product generation sequences are used as intended to ensure always consistency of configuration baselines, the slowest progressing team would determine the overall pace of development. Not sharing the same work product in several work product generation sequences is the better advice. Refined system interface requirements belong to the system requirements and should be part of the work product containing the complete system requirement set. If consent from the other engineering team has not been granted yet, an assumption should be raised with the expectation to resolve the assumption due to a later approval by the other engineering team. The team that is further advanced is now in the position to decide to what extent they progress the development further taken the risk of assumptions into account.

Conclusions

Together with two previous papers (Scheithauer and Forsberg 2013; Scheithauer 2014), this paper provides a comprehensive description of the systems engineering value stream. It is the specific contribution of this paper to consider the information flow of the systems engineering process that is not following the system architecture upwards and downwards, but has to take place between neighboring systems in separate branches of the system architecture.

References

Buede, D. M. 2009. *The Engineering Design of Systems: Models and Methods*. 2nd Edition. Hoboken, NJ (US): John Wiley and Sons.

Hybertson, D. W. 2009. *Model-Oriented Systems Engineering Science: A Unifying Framework for Traditional and Complex Systems*. Boca Raton, FL (US): CRC Press.

INCOSE (International Council on Systems Engineering). 2014. *A World in Motion – Systems Engineering Vision 2025*.

ISO and IEC (International Organisation for Standardisation and International Electrotechnical Commission). 2008. ISO/IEC 15288-2008. *Systems and Software Engineering – System Life Cycle Processes*.

Kossiakoff, A., W. N. Sweet, S. J. Seymour, and S. M. Biemer 2011. *Systems Engineering: Principles and Practices*. 2nd Edition. Hoboken, NJ (US): John Wiley and Sons.

Oppenheim, B. W. 2011. *Lean Systems Engineering - With Lean Enablers for Systems Engineering*. Hoboken, NJ (US): John Wiley and Sons.

Parnell, G. S., P. J. Driscoll, and D. L. Henderson. *Decision Making in Systems Engineering and Management*. 2nd Edition. Hoboken, NJ (US): John Wiley and Sons.

Reinertsen, D. G. 2009. *The Principles of Product Development Flow: Second Generation Lean Product Development*. Redondo Beach, CA (US): Celeritas Publishing.

Scheithauer, D. 2014. "Systems Engineering Value Stream Modelling." Paper presented at the EMEASEC 2014, Cape Town (ZA): 27-30 October.

Scheithauer, D. 2014-2. “Qualität im System Design.” In *Tag des Systems Engineering*, edited by M. Maurer and S.-O. Schulze. München (GE): Carl Hanser Verlag.

Scheithauer, D. 2015. “System Interfaces and System Interoperability in a System-of-Systems Context.” Paper presented at the NATO LS-SCI-276, Lisbon (PT): 30-31 January.

Scheithauer, D., and K. Forsberg 2013. "V-Model Views." Paper presented at the 23nd INCOSE International Symposium, Philadelphia PA (US): 24-27 June.

Schulz von Thun, F. 2010. *Miteinander reden 1: Störungen und Klärungen. Allgemeine Psychologie der Kommunikation*. 48. Auflage. Hamburg (DE): Rowohlt.

Wasson, C. S. 2006. *System Analysis, Design, and Development: Concepts, Principles, and Practices*. Hoboken, NJ (US): John Wiley and Sons.

Watzlawick, P., Beavin, J., and Jackson, D. 1969. *Menschliche Kommunikation – Formen, Störungen, Paradoxien*. Bern (CH): Hans Huber.

Weiss, S. I. 2013. *Product and Systems Development: A value Approach*. Hoboken, NJ (US): John Wiley and Sons.

Womack, J. P., and D. T. Jones 2003. *Lean Thinking: Banish Waste and Create Wealth in Your Corporation*. New York, NY (US): Free Press.

Biography

Dieter Scheithauer studied electrical engineering with special emphasis on automatic control at the Universität der Bundeswehr München. He received the degree of a Diplom-Ingenieur univ. in 1980 and a doctor's degree (Dr.-Ing.) in 1987. After twelve years, his service as Technical Officer in the German Air Force ended in 1988.

From 1988 to 1999 he has been employed by Industrieanlagenbetriebsgesellschaft GmbH (IABG) mainly delivering technical expertise to the German Ministry of Defence and other government organizations. Throughout this time he contributed in various roles to the flight control system development for major European military aircraft and helicopter programs. Furthermore, he acted as project manager and chief engineer for unconventional airborne and ground-based systems demanding high-integrity technical solutions.

In 1999 he has joined the company that is now the Airbus Group. During the initial years he was in charge of process improvement for the EF2000 flight control system development. Later he held a position as Senior Expert Systems Engineering Processes within the division Airbus Defence and Space. He was influential progressing systems engineering within the division and on group level. In 2014 he left the company.

In 2014 he started his own business as independent systems engineering trainer, coach and consultant.

He is a former president and an honorable member of GfSE e.V. – The German Chapter of INCOSE. He became an INCOSE CSEP in 2010, and an INCOSE ESEP in 2012.