A THEORY FOR SYSTEM ENGINEERING MANAGEMENT

L. D. Erasmus[1*] and G. Doeben-Henisch[2]

[1]Graduate School of Technology Management, University of Pretoria, South Africa
louwrence.erasmus@up.ac.za

[2]Department of Intelligent Systems, University of Applied Sciences Frankfurt am Main, Germany
doeben@fb2.fh-frankfurt.de

ABSTRACT

A theory for describing the system engineering management is presented in this paper. This theory is a further expansion of the theory for the systems engineering process presented in the tradition of the structuralist programme in the philosophy of science. An important aspect in the formulation of this theory is the inclusion of people as actors in the processes and interfaces as a major part in specifying requirements. Development of the theory will lead to the implementation in simulation software to study the dynamic characteristics of and interaction of people with/in the system-engineering management process as well as systematically validating the theory through empirical studies. Further, the theory enables the development of system-engineering management training-simulators to support life long learning of systems engineers.

1. INTRODUCTION

Systems engineering is performed as part of an acquisition process and has two disciplines (Defense Acquisition University [2]): "*the technical knowledge domain in which systems engineering operates, and system engineering management*." Thus, the subject of systems engineering is treated as part of engineering as well as management sciences and also noted in (INCOSE [11]). Currently, apart from the attempt of (Erasmus & Doeben-Henisch [10]), the authors are not aware of formal theories based on the dual character of systems engineering, i.e. a strong mathematical foundation combined with management theory.

The development of a formal theory for the technical knowledge domain in which systems engineering operates is outside the scope of this paper. A first attempt to formulate a formal theory for system engineering management is treated in this paper.

In system engineering management, three major activities are performed (Defense Acquisition University [2]):

- Development phasing controls the system design process and defines baselines for coordinating the subsystems, disciplines and specialties design efforts.
- Systems engineering process defines a structure for solving design problems as well as tracking the flow of requirements through the design effort.
- Life cycle integration involves customers and direct stakeholders in the design process and ensures viability of the developed system for a sustainable life cycle.

The interaction rather than the intersection between the three activities described above must be considered.

Current description of system engineering management, (ISO 15288 [4], ISO 26702 [3], Defense Acquisition University [2], Blanchard [5], INCOSE [11]), are restricted to *primary objects* used and produced in the various activities (i.e. baselines, plans, schedules, integrated teaming, etc.) but the main actors of this process, i.e. the stakeholders, specialty engineers and discipline experts, etc. and the underlying structures are not addressed explicitly. A

[*]Corresponding author.

formalism must take the actors also into account and there relationship with the underlying structure and these are treated in the following subsections.

2. CONCEPT OF SYSTEM ENGINEERING MANAGEMENT

A more elaborated depiction for systems engineering management is necessary to show the interaction between different activities. Such a visualisation is shown in figure 1 based on various descriptions of system engineering management (Blanchard [5], INCOSE [11], ISO 26702 [3], Defense Acquisition University [2]) and experience of the authors. This visualisation is a further development of a depiction of the systems hierarchy level and the systems engineering process (De Waal & Buys [7]), with the following difference: what (De Waal et al. [7]) has called the 'systems engineering process', is named 'system engineering' in figure 1 and the systems engineering process depicted in (De Waal et al. [7]) is simplified to the requirements and design loops in figure 1.

A description of figure 1 follows.

2.1 Life cycle integration

The client initiates the process with a required operational capability (ROC). The client ($\mathcal{C}$) is part of the stakeholders ($\mathcal{S}$), thus $\mathcal{C} \in \mathcal{S}$. The ROC ($\mathcal{D}_{ROC}$) is actually part of the stakeholder documents of the operation and support stage requirements ($\mathcal{D}_{OS}$), thus $\mathcal{D}_{ROC} \subset \mathcal{D}_{OS}$ [1].

A system life-cycle definition is established by the system engineering manager ($\mathcal{E}_{SEM} \in \mathcal{E}$) together with the various life-cycle function managers ($\mathcal{E}_{LCM} \in \mathcal{E}$). Each life cycle function has its own life-cycle function documents, $\mathcal{D}_{LCF} \subset \mathcal{D}$, describing the stakeholder requirements for the applicable life cycle function. The life cycle integration for a system is coordinated with a plan to make the facilitation of this activity possible as described in (Defense Acquisition University [2]) and (ISO 26702 [3]).

2.2 System engineering

A plan for the system engineering effort must be established, also called a system engineering management plan (SEMP) (ISO 26702 [3], INCOSE [11]). This effort consists of a combination between the systems-engineering process activity (not to be confused with system engineering), depicted as the requirements loop followed by the design loop in figure 1 and the development phasing activity.

2.2.1 Development phasing

Blanchard [5] describes three development phases namely conceptual design, preliminary design and detail design & development phases. Generally, these three development phases correspond to the description of the development stage as defined in the informative part of ISO 15288 [4] Annex B.

ISO 26702 [3] defines the development stages as system definition, subsystem definition (consisting of preliminary design, detail design, and FAIT (fabrication, assembly, integration, and test FAIT)). From the descriptions in ISO 26702 [3] and ISO 15288 [4] of life cycle stages, it can be concluded that the FAIT stage described in ISO 26702 can be split with some activities performed during development and the other during the production stages as described in ISO 15288. The activities of the FAIT stage performed during the development stage is included in the detail design & development phase described by Blanchard [5].

[1]The notation and definition for documents $\mathcal{D}$, experts $\mathcal{E}$, stakeholders $\mathcal{S}$ and support $\mathcal{A}$ in (Erasmus et al. [10]) is used here.

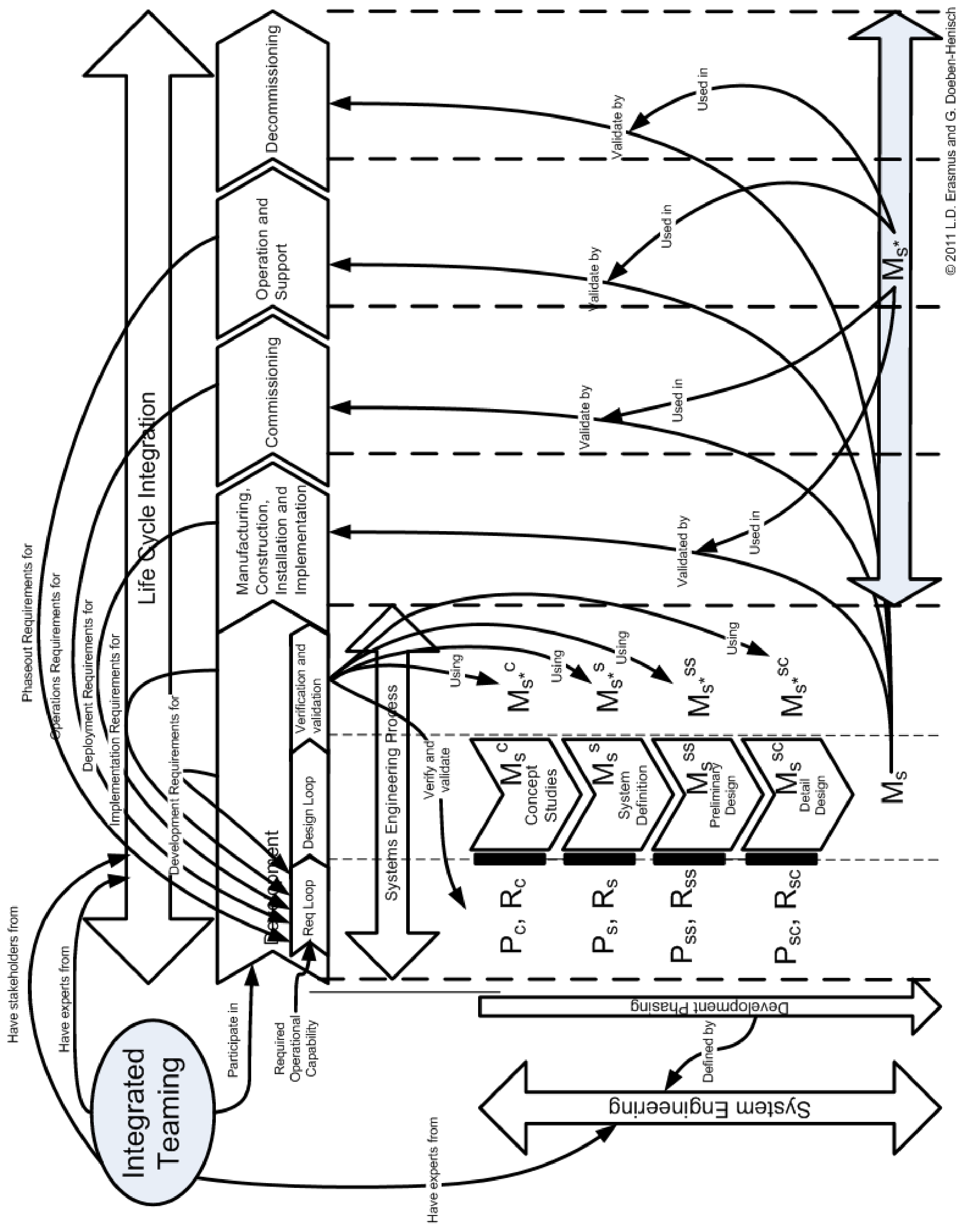

Figure 1: Illustration of a system-engineering management base theory

In ISO 15288 [4] Annex B, a distinction is made between the concept and development stages. Based on this, the description of Defense Acquisition University [2] on development phasing in terms of the progression through distinct levels or stages is tailored to *system definition, preliminary design and detail design stages*. These three stages are used in this paper and is comparable with the three development phases described by Blanchard [5].

For the rest of this paper, the term 'stage' is used in accordance with ISO 26702 [3] and ISO 15288 [4], rather than the more archaic term 'phase'. Further, for the purposes of this paper, the concept study stage is also included as part of the development stage.

2.2.2 Systems engineering process

There is a very close interaction between the systems engineering process and development phasing to define the different system hierarchical levels (hl) during development. From the descriptions in ISO 26702 [3], the different system hierarchy levels are defined progressively through the system engineering effort by applying the systems engineering process on a current system hierarchical level to define the items in the next level that comprise the current level. This action has the theoretical potential for defining unlimited system hierarchy levels. Practically, the number of levels are limited for the systems hierarchy level definition in systems engineering practice.

The United States of America's (USA) Department of Defense (DOD) defines item levels (or system hierarchy levels) in MIL-HDBK-505 [1] as the following: component, sub-assembly, assembly, unit, groups, set, subsystem, and system.

In ISO 26702 [3], the system hierarchy levels are called the hierarchy of elements within a system and are defined as follows: either as parts, subcomponents, components; or as parts, subcomponents, sub-assemblies, components; then further as components, assemblies, subsystem, product and system.

In the South African context of systems engineering, the lowest system hierarchy level, or level 1, is the processing and materials level, level 2 is the components level, level 3 is the subsystem level, level 4 is the product level, level 5 is the product system level and level 6 the user system level (Erasmus [9], De Waal et al. [7]).

These different levels discussed above only concentrates on a system. De Waal et al. [7] discuss further levels (level 7 to level 10) used for system of systems and systems of systems in the South African context. The systems hierarchy levels are comparable with Boulding's hierarchy (Boulding [6] in De Waal et al. [7]).

Through the progressive application of the systems engineering process during each development stage, the problems $\mathcal{P}_{hl}$ (requirements) for the next lower systems hierarchical levels are established. This forms a graph or breakdown structure for the system engineering cycle. The system engineering cycle for a branch stops when the design ($\mathcal{M}_s^{hl}$) can be realised with an instance $\mathcal{M}_{s*}^{hl}$. The configuration of the relationships between the different levels of designs ($\mathcal{M}_s^{hl}$ for all levels hl) is called the configuration baseline (Defense Acquisition University [2]). The expected configuration baseline after each development stage is defined beforehand. The end of a development stage has been reach once the defined configuration baseline for that stage has been achieved.

An important object that is developed and used in the requirements loop is the identification and definition of interfaces as part of the requirements $\mathcal{R}_{hl}$ on each systems hierarchical level. Through the interface, the two interfaced elements' expected parameters and behaviours relative to each other are specified. This is discussed in more detail during the development of the formal theory later in this paper.

2.2.3 Development and production models

Models are used in systems engineering to represent information and "... *to verify down stream information against upstream information*" (Sparrius [12]). Once the models are in a tangible form, they can be used in test and evaluation for the purpose of qualification. Models are used in the widest sense of the word here and includes documents, software, scale and full physical instances, or a combination of the above (Sparrius [12]).

In terms of this paper, models $\mathcal{M}^{l}_{s*}$ are used to verify the implementation of requirements ($\mathcal{R}_l$) in the design ($\mathcal{M}_l$). The requirements ($\mathcal{R}_l$) are also validated in the context of $\mathcal{P}_l$ and $\mathcal{M}^{l-1}_{s*}$ using $\mathcal{M}_l$ and $\mathcal{M}^{l}_{s*}$. Figure 1 shows the emergences and there use of the various models. The various development stages and their corresponding models as per MIL-HDBK-505 [1] are as follows:

- the concept studies stage ($\mathcal{M}^{c}_{s*}$) -- the exploratory development model (XDM),
- the system definition stage ($\mathcal{M}^{s}_{s*}$) -- the advanced development model (ADM),
- the preliminary design stage ($\mathcal{M}^{ss}_{s*}$) -- the engineering development model (EDM) or service test model,
- the detail design stage ($\mathcal{M}^{sc}_{s*}$) -- the pre-production model (PPM) or prototype,

The actual system required for operation is produced as the production model (PM) ($\mathcal{M}^{p}_{s*}$) (MIL-HDBK-505 [1]) is not used in the development stage, but for qualification of the rest of the system life cycle stages. Qualification in this context has the meaning to validate the processes in the respective life cycle stages with the applicable configuration baselines.

It might be stated that the production model is no longer a model in the sense of representing the systems engineering documentation, but this is not true. The production model is a representation of the product data pack. The product data pack is a product of system engineering consisting of a subset of the systems engineering documentation. Thus it can be concluded that the production model is a representation of a subset of the systems engineering documentation, just like any other model in the development stage.

2.2.4 Integrated teaming

Integrated development requires that all the life cycle quality factors (ISO 26702 [3]) and other life cycle needs are concurrently taken into account during the development of a system (Defense Acquisition University [2], INCOSE [11]). This is also called integrated product and product development (IPPD) (Defense Acquisition University [2], INCOSE [11], ISO 26702 [3]). The concurrent consideration of life cycle needs can be greatly improved through the use of interdisciplinary teams, also called integrated product teams (IPT) (Defense Acquisition University [2], INCOSE [11], ISO 26702 [3]) or a Team-of-Six in the South African context of systems engineering management.

With integrated teaming, technical management (system engineering), the discipline experts, life-cycle function management or business area management stakeholders are forming an IPT team (Defense Acquisition University [2]) under the leadership of the system engineer responsible for executing the systems engineering process for a specific system element. The IPT has the objective to produce a design solution solution that satisfies originating requirements and to communicate that design solution to all stakeholders (Defense Acquisition University [2]).

3. INTRODUCING A FORMAL THEORY

Systems engineering is generally described in colloquial terms as a 'cradle to grave', 'sperm to worm', or 'lust to dust' approach to system development.

It could be said that the life of a system starts with some *ideas* in the head of a stakeholder (the 'lust' part of system development). In order to bring these ideas 'to life' and make them work, it is necessary to *transform* or *convert* these ideas into a form to enabling *communication* between different brains. The *coordination* between different stakeholder and expert brains using various descriptions e.g. text descriptions, diagrams, formulas or other artifacts, is necessary for ideas to become *working ideas*. In the following paragraphs, a formalization is proposed for some important parts of systems engineering management.

3.1 Process description with functions

An adapted view of system engineering management is introduced in a previous section and summarised in the figure 1. The explanations so far in (ISO 15288 [4], ISO 26702 [3], Defense Acquisition University [2], Blanchard [5], INCOSE [11]) have a *semi formal* character, because the words and expressions of an everyday language, like English, enriched with diagrams have been used to communicate the model of system engineering management. For a formal theory, the description with a formal language need to be extended to achieve a formal theory of the system engineering management.

The starting point is the *life cycle process* within which possible stages can be identified and defined.

One possibility to formalise a process is to use the concept of a *mapping* or a *function*, often used in the format of a computer *software program*. In writing out a function, several options exits, amongst others:

$$[o_1, \cdots, o_k] = sembase(i_1, \cdots, i_n) \tag{1}$$

$$sembase : I_1 \times \cdots \times I_n \longmapsto O_1 \times \cdots \times O_k \tag{2}$$

$$sembase(i_1, \cdots, i_n) = [o_1, \cdots, o_k] \tag{3}$$

A frequent used way in writing out a function is (1), expressed in the form $[y] = f(x)$ to declare a computer software function using only input and output values. In (1) the function $sembase$ computes the output values $o_1, \cdots, o_k$ given the input values $i_1, \cdots, i_n$. The common mathematical way in writing out a function is to describe it as a mapping as in (2), expressed as $f : X \longmapsto Y$ mapping from definition sets into value sets. In (2) the mapping function $sembase$ maps the elements of the sets $I_1, \cdots, I_n$ into the elements of the sets $O_1, \cdots, O_k$. Another way is (3), expressed as $f(x) = y$, where the function $sembase$ is applied to the arguments $i_1, \cdots, i_n$ and computes the value $o_1, \cdots, o_k$.

The above notations neither explicitly express the *actors* involved in performing the *real process* function, (e.g. a system, computer software or human), nor explicitly describe a process on *how* such a process work. The more detailed information about the way how a function is *processed* is usually given in the 'body' of a function declaration of a computer program or in the 'body' of a function definition. The other information about the processing actors is usually assumed as a *context knowledge*, i.e. it is necessary to know whether a certain function has to be processed by a human or a machine.

The following notation is used in the formalisation of a base theory for system engineering management (SEMBASE):

$$sembase|_{\mathcal{S},\mathcal{E},\mathcal{A}} : I_1 \times \cdots \times I_n \longmapsto O_1 \times \cdots \times O_k \tag{4}$$

where the *actors* of this process are the *stakeholders* ($\mathcal{S}$), the *experts* ($\mathcal{E}$) and applicable facilities and support systems named as *SEM enabling system* ($\mathcal{A}$). The processing's detail description of the different actors is demanding and will be described in a future paper (see below). Without any loss for not having any detail, the expected output from the actors in such a process can be described as well as the input which are needed to yield such output. Thus, it is defined that

$$sembase|_{\mathcal{S},\mathcal{E},\mathcal{A}} : 2^{\mathcal{D}} \longmapsto \mathcal{M}_{\mathcal{S}*} \tag{5}$$

where the actors $\mathcal{S}, \mathcal{E}, \mathcal{A}$ use *documents* $(\mathcal{D})$ as starting point and then organise a process which has as (minimally) outcome a *working system* $\mathcal{M}_{\mathcal{S}*}$.

SEMBASE is understood as a "... *complex system of processes that will normally possesses concurrent, iterative and time dependent characteristics*" (ISO 15288 [4]) and can formally be expressed as a graph of mappings:

$$\begin{aligned} F(g) &\iff g = \langle P_{LC}, Intf \rangle && (6)\\ P_{LC} &= \{f_1, f_2, \cdots, f_n\} && (7)\\ Intf &\subseteq P_{LC} \times P_{LC}^n, n \in \{0, 1, 2, ...\} && (8) \end{aligned}$$

whose vertices P_{LC} are *life cycle processes* of SEMBASE and the edges are *interfaces* $Intf$ between the life cycle processes. The concept of an interface is formalised later in this paper.

For the purposes of this paper, the graph of mappings for SEMBASE is simplified to only a *sequence* of different *subprocesses* that can be formally expressed as a concatenation of mappings [2]:

$$f = f_1 \circ f_2 \circ \cdots \circ f_n \qquad (9)$$

where f_1 is a first mapping, f_2 a second mapping and f_n a last mapping and the output of the first mapping is the input to the second mapping and this is continued until the output of the $n - 1$ mapping is the input for the n mapping.

For SEMBASE, the following partial mappings are assumed

$$\begin{aligned} sembase = \; & development \circ implementation \circ commissioning \\ & \circ operations \circ decommissioning \qquad (10) \end{aligned}$$

where $development$ is the partial mapping representing the graph of mappings for the development stage, $implementation$ is for the manufacturing, construction, installation and implementation stage, $commissioning$ is for the commissioning stage, $operations$ is for operation and support stage, and $decommissioning$ is for the decommissioning stage. These partial mappings can be defined as follows (assuming minimal input and output parameters):

$$\begin{aligned} development &: 2^{\mathcal{D}} \longmapsto \mathcal{M}_{\mathcal{S}} && (11)\\ implementation &: \mathcal{M}_{\mathcal{S}} \longmapsto \mathcal{M}_{\mathcal{S}*} && (12)\\ commissioning &: \mathcal{M}_{\mathcal{S}*} \longmapsto \mathcal{M}_{\mathcal{S}*} && (13)\\ operations &: \mathcal{M}_{\mathcal{S}*} \longmapsto \mathcal{M}_{\mathcal{S}*} && (14)\\ decommissioning &: \mathcal{M}_{\mathcal{S}*} \longmapsto \emptyset && (15) \end{aligned}$$

The input and output values of the partial mappings for commissioning, operations, and decommissioning are probably too simple with regard to the real processes. This should be refined in the future.

3.2 First structure

A first outline of a theory for systems engineering management can then be defined as:

$$\begin{aligned} SEMBASE(x) &\iff x = \langle \mathcal{D}, \mathcal{M}_{\mathcal{S}}, \mathcal{M}_{\mathcal{S}*}, sembase \rangle |_{\mathcal{S},\mathcal{E},\mathcal{A}} && (16)\\ sembase &= development \circ implementation \circ commissioning \\ & \quad \circ operations \circ decommissioning && (17) \end{aligned}$$

The index $\mathcal{S}, \mathcal{E}, \mathcal{A}$ indicates that the mapping $sembase$ is realized by actors from the sets stakeholders, experts, and SEM enabling system, respectively.

[2]This is done to achieve a first linear formalisation of SEMBASE and does not hold true for the general case.

3.3 Development process

The formal theory is restricted to the main properties of the life cycle process. However, there are further refinements necessary to describe system engineering properly. The first necessary refinement is a subdivision of the development stage:

$$\begin{aligned} development &:= requirements \circ design & (18)\\ requirements &: 2^{\mathcal{D}} \longmapsto \mathcal{M}_{\mathcal{R}} & (19)\\ design &: \mathcal{M}_{\mathcal{R}} \longmapsto \mathcal{M}_{\mathcal{S}} & (20)\end{aligned}$$

This refines the first outline of the theory as follows:

$$\begin{aligned} SEMBASE(x) &\iff x = \langle \mathcal{D}, \mathcal{M}_{\mathcal{R}}, \mathcal{M}_{\mathcal{S}}, \mathcal{M}_{\mathcal{S}*}, sembase\rangle|_{\mathcal{S},\mathcal{E},\mathcal{A}} & (21)\\ sembase &= requirements \circ design \circ implementation\\ &\qquad \circ\, commissioning \circ operations \circ decommissioning & (22)\end{aligned}$$

The design models $\mathcal{M}_{\mathcal{S}}$ are designed based on the requirement models $\mathcal{M}_{\mathcal{R}}$ which in turn are derived through analysis of the problem P. The problem P is implicitly described in some documents from the set of possible documents $2^{\mathcal{D}}$. In the preceding sections, further distinctions have been made between *different stages* of the development process aligning a requirement model $mr_i \in \mathcal{M}_{\mathcal{R}}$ with a design model $md_j \in \mathcal{M}_{\mathcal{S}}$, eventually extended by a further alignment with an implemented model $mi_k \in \mathcal{M}_{\mathcal{S}*}$. This presupposes that one can distinguish between the different models of higher or lesser complexity.

3.4 Ordering by complexity

If the requirements models M_R or design models M_S should be ordered according to a *complexity criterion* ($<_{Compl}$), a criterion must be introduced to allow comparisons between two sets M, M' so that $M <_{Compl} M'$ means that the set M has a lower complexity than the set M' according to the criterion $<_{Compl}$.

3.4.1 Complexity ordering of design models

For design models such an ordering relation $<_{Compl}$ can be derived from the fact that every design model md_i can be refined by sub-elements (e.g. subsystems, components, etc.) $\{md_{i+1,1}, md_{i+1,2}, \cdots, md_{i+1,m}\}$. This generates a *design model graph* (DMG) as the set

$$\{md_0, (md_0, md_{1,1}), (md_0, md_{1,2}), \cdots, (md_0, md_{1,m})\} \qquad (23)$$

which can be generalised to

$$\{md_i, (md_i, md_{i+1,1}), (md_i, md_{i+1,2}), \cdots, (md_i, md_{i+1,m})\} \qquad (24)$$

Thus a *design model extension* is understood as that operation which takes a given element at the 'bottom' of a design model graph DGM -- called a 'leaf' in graph theory -- and *extends* this element by associating further elements as subcomponents to this selected element. In this way it is possible to define the *levels* $level(gdm_r)$ of a design model graph $dmg_r \in DMG$ as the highest element index which appears in this graph (counting the first element in dmg_r as level zero). A more precise measure of the complexity of a design model could be to take also into account the number of all elements in the design model graph.

3.4.2 *Complexity ordering of requirements*

The required complexity of the design models is driven by the requirements model $\mathcal{M}_{\mathcal{R}}$ which is connected to the design model through the intended system interface. A requirements model $\mathcal{M}_{\mathcal{R}}$ has been defined in (Doeben-Henisch, Bauer-Wersing, Erasmus, Shrader & Wagner [8]) as a model describing the *behaviour* how users (described as user interfaces $INTF_U$) in an environment (described as an environment interface $INTF_E$) attempt to solve a task $g_i \in \Gamma$ by using a technical system through the system interface $INTF_S$. The solution of a task g_i is understood as a sequence of actions $a_i \in ACT$ triggering the states of the interfaces. Deviating from the original definition in (Doeben-Henisch et al. [8]), generally, an *interface* is an *directed interface graph* DIG:

$$\begin{aligned} DIG(g) &\iff g = \langle Z, Tr \rangle && (25)\\ Z &\subseteq 2^{PROP \times VAL} && (26)\\ Tr &\subseteq Z \times Z^n, n \in \{0,1,2,...\} && (27) \end{aligned}$$

whose vertices Z are *states* of the interface and the edges are *transitions* Tr of these states triggered by action events. An *interface state* is a collection of properties $PROP$ which can show $n \geq 2$ different values VAL. Depending on the action of a user, the environment or 'built in' events an interface state can change into another state, which is represented in the interface graph as an edge representing a transition. Two interface states $z_i, z_j \in Z$ are called *different* if they differ with regard to at least one property-value pair $(p_i, v_j) \in PROP \times VAL$. This difference includes not only the case that a complete pair is present or absent but also the case, that a given pair (p_i, v_j) changes only its value from v_j to a $v_k \neq v_j$. Thus an interface graph $ig_i \in DIG$ can be understood as a directed graph whose vertices are interface states which are all different. A *path* in an interface graph (which can include loops) represents then follow-up states of the interface triggered by a series of action events. The *complexity of an interface graph* can then be measured by the length of a path for a partial measure or by the number of states for a global complexity measure[3].

3.5 Interface Reliability

Applying the definition of an interface graph DIG to user (U), environment (ENV), and intended system (SYS) a requirements model $\mathcal{M}_{\mathcal{R}}$ contains at least three interface types $INTF|_{U,ENV,SYS}$ which are correlated by triggering events. This means that the *output* of one interface can be the *input* for the other interfaces. The system interface $INTF_{SYS}$ description is describing the state of the interface completely. However, the *user interface* $INTF_U$ description describes only the interface of the user *as it is presupposed for the correct interaction with the system*. Whether the *real user* is indeed conforming to this expected states of a user interface is not guaranteed[4].

3.6 Interaction between Interfaces

For the interaction between the three above identified interfaces the following mappings are established:

$$uact \quad : \quad INTF_U \longmapsto INTF_{SYS} \times INTF_{ENV} \qquad (28)$$

[3]For human factors engineering, which is not at stake here, this complexity measure is of central importance. From this complexity measure a lot of important human factors parameters are directly depending like 'ease of learning' and 'error rate'. Associated with these parameters are many other parameters like 'time needed for training', 'cognitive load' etc. These parameters again have further implications like probability of errors, increasing costs for training and test, etc. that leads to an increasing risk.

[4]In the case of safety critical systems the conformance or non-conformance of the real user with the required user interface is a main source of possible failures. It is nearly impossible to guarantee this conformance. Conversely, one can construct a probability model to forecast the probability of failures depending from the complexity of the interfaces involved.

$$sysact : INTF_{SYS} \longmapsto INTF_U \times INTF_{ENV} \quad (29)$$
$$envact : INTF_{ENV} \longmapsto INTF_{SYS} \times INTF_U \quad (30)$$

Because every interface can be seen as a directed interface graph, these mappings have to be unified with the notation of an interface graphs. The assumed logic behind these mappings is that e.g. a state $z_{U.i}$ of a user interface $INTF_U$ can be the trigger for a transition in the system interface $INTF_{SYS}$. Thus a mapping

$$uact_i = (z_{U,i}, \langle (z_{SYS,j}, \langle z_{SYS,j+1,1}, z_{SYS,j+1,2} \rangle \rangle) \quad (31)$$

should be included in the transitions of the system interface as a *condition* making the transition dependent from the activation of the state in the user interface:

$$tr_{SYS,i} = (\langle z_{(U,i)}, z_{(SYS,j)} \rangle, \langle z_{(SYS,j+1,1)}, z_{(SYS,j+1,2)} \rangle \rangle \quad (32)$$

3.7 Interface as Actor

Furthermore, every interface graph can be interpreted as the *execution graph* of a finite *automaton* receiving messages from the other automata and responding with messages. Thus, one can interpret any interface as an actor mediating messages. The user interface $INTF_U$ represents the required behaviour of a user (which is not guaranteed), the environment interface $INTF_{ENV}$ represents the expected behaviour of an environment (which is based on estimates), and the system interface $INTF_{SYS}$ represents the required behaviour of the intended system, which is guaranteed if qualified.

3.8 Satisfying the requirements

Finally one has to state, that all actors of the requirements model M_R are guided by a finite set of *goals* Γ to be reached. The behaviour of the participating interfaces is *satisfying*, if it can be shown that for every goal $g_i \in \Gamma$ there exists at least one path in the user interface $INTF_U$ and in the corresponding system interface $INTF_{SYS}$ which leads from an initial state to a state satisfying the goal g_i. The proposed formalisation allows a decision on whether certain goals can be reached and additionally one can construct models to compute the probability of reaching a certain goal; additionally the probability of failures can be computed while trying to reach a certain goal.

4. CONCLUSION

In this paper the basic structure for a formal system engineering management theory, called system engineering management base theory, have been derived. Using standards, publications and well known practices elements and processes, which have to be regarded as essential elements, relations, and processes for such a *base theory* have been identified.

The system engineering management base theory described so far covers not the whole subject. Further refinement of the theory will be presented in an upcoming paper called 'system engineering management theory level 2' covering the complex issue of non-functional concepts, e.g. safety, reliability, sustainability, human factors and other life cycle quality factors. In terms of the philosophy of science these non-functional terms are *meta terms* defined on the basis of the system engineering management base theory.

There is another dimension to system engineering management, which has to be dealt with, namely, the psychological conditions of the acting humans as well as the necessary training and learning processes to enable humans to become well trained systems engineers. Learning in this context is assumed to happen all the time throughout the life of a systems engineer. This poses a challenge in assessing the quality of learning taking place during life long learning of systems engineers.

The formal theory will enable the development of mathematical models describing system engineering management. The formalisms together with mathematical descriptions can be used for the testing of existing theories, models and methods in system engineering management through simulation and validated with empirical measurements on system-engineering management practice, similar to the agenda in (Erasmus et al. [10]). The validated models can be used to study the dynamic characteristics of system engineering management.

Using the theoretical framework for system engineering management, systems engineers can be supported during learning and working activities. One important supporting tool for this will be simulator software for open learning spaces -- abbreviated S4OLS -- which is under development by an international open source project[5].

References

[1] 1998. *MIL-HDBK-505 Military Standard: Definitions of Item Levels, Item Exchangeability, Models, and Related Terms*. DEPARTMENT OF DEFENSE, Wosh;ngton, D. C., United States of America.

[2] 2001. *Systems Engineering Fundamentals*. Defense Acquisition University Press, Fort Belvoir V.A., USA.

[3] 2007a. *ISO 26702:2007/IEEE Std 1220-2005, IEEE Standard for Application and Management of the Systems Engineering Process*. International Organisation for Standards, Geneva, Switzerland.

[4] 2007b. *ISO/IEC 15288:2007 System Life Cycle Processes*. International Organisation for Standards, Geneva, Switzerland.

[5] Blanchard, B. S. 2008. *System Engineering Management*. Wiley, 4th edition.

[6] Boulding, K. E. 1956. General Systems Theory – The Skeleton of Science. *Management Science*, 2(3):197 -- 208.

[7] de Waal, J. and Buys, A. 2007. Interoperability and standardisation in the department of defence: An exploratory study. *South African Journal of Industrial Engineering*, 18(1):175--190.

[8] Doeben-Henisch, G., Bauer-Wersing, U., Erasmus, L., Schrader, U., and Wagner, W. 2008. Interdisciplinary Engineering of Intelligent Systems. Some Methodological Issues. In Loula, A. and Queiroz, J, editors, *Advances in modeling adaptive and cognitive systems*. Joint Conferences of SBIA 2008 (the 19th Brazilian Symposium on Artificial Intelligence); Salvador (Brazil).

[9] Erasmus, L. 2005. Application of IEEE Std 1220 and ISO 15288 in the Development of a Systems Engineering Master Schedule for the PBMR Project to satisfy nuclear regulatory requirements. In *INCOSE SA Conference Proceedings*.

[10] Erasmus, L. D. and Doeben-Henisch, G. 2011. A Theory of the System Engineering Process. In *10th AFRICON Conference: Sustainable Energy & Communications Development for Africa*, Livingston, Zambia.

[11] Haskins, C., editor 2011. *Systems Engineering Handbook - A guide for system life cycle processes and activities*. Number INCOSE-TP-2003-002-03.2.1. International Council on Systems Engineering, San Diego, CA, United States of America.

[12] Sparrius, A. 2010. *Course: Acquisition Management*. Ad Sparrius System Engineering and Management (Pty) Ltd., P O Box 74922, Lynnwood Ridge, 0040, South Africa.

[5]See for details the website www.os-pe.org.