Managing Concurrency in Systems Engineering

Dieter Scheithauer
CASSIDIAN
Rechliner Str., 85077 Manching, Germany
dieter.scheithauer@cassidian.com

Abstract. The engineering of complex products and services cannot be performed without iterations. Products and services with a multi-level system architecture and a reasonable number of system elements usually show development activities on individual system elements performed in parallel. The development of each system element is widely managed independently. This paper is concerned with managing concurrency resulting from both, the iterative nature of systems engineering per se, and the multiple projects for developing all the system elements.
Lean systems engineering is promising improvements to the systems engineering process in general. However, lean systems engineering is still evolving. Particularly, the integration with all other system life cycle processes needs to be progressed without compromising the lean paradigm. In this paper, amendments to the current state of lean systems engineering are proposed in order to close some of the gaps.

Introduction

People or Process: What is more important? This was a title of a panel discussion at last year's INCOSE International Symposium (Zonnenshain, Sillitto and Kasser 2011). At first sight, it is surprising that this subject attracted the INCOSE community still. Otherwise, this panel would not have taken place at all. Thereby, common sense alone is sufficient to understand the mutual dependency. The systems engineering process alone does not lead to any results. It needs the imagination and creativity of people for both, the analysis of the situation, and the synthesis of a solution that may be innovative in itself, or that may not have been applied in the particular context before. The systems engineering process supports people to make progress by managing information and by controlling the common effort of team members with different professional backgrounds, with conscious and unconscious demands, and with specific skills.

Digging deeper into the subject, the analyses of success factors in systems engineering has led to various attributions to people and processes. Regarding processes the following views are influential. Traditionally, systems engineering emphasizes sequential life cycle phases, requirements based engineering and the V-model (Haskins 2010). Concurrent engineering recommended integrated product development teams and phase overlaps, specifically between engineering and production (Winner et al. 1988). Proponents of model-based approaches expect major improvements by relying on models (Friedenthal and Kobryn 2004). Lean systems engineering sets the focus on value and value streams (Oppenheim 2004 and 2011).

Similarly, a wide range of explanations exist regarding the contribution of people. Vincenti is convinced that the dedication of engineers to their job and their restless effort to solve any problem pave the way to success (Vincenti 1990). In contrast, Maier and Rechtin promote the

experienced engineer with a wide range of knowledge, and relying primarily on heuristics for making the essential architectural decisions (Maier and Rechtin 2009). Armstrong questions the appropriate ratio of divergent and convergent thinkers (Armstrong 2009).

Of course, all these analysis results are important. And, improving in each of these directions will contribute to increasing efficiency of the systems engineering effort. But is this sufficient? From a systems thinking perspective, it is not enough to cure individual symptoms. Generally, our view of the world is not just the sum of all isolated reactions to particular stimuli (Köhler 1947). Consequently, the mutual dependencies between process and people need to be considered for improved systems engineering performance. Optimized processes are rather useless, if people do not feel supported. And, the creativity of people needs to be guided and integrated by adequate processes for mastering systems engineering challenges in multi-disciplinary teams.

The principle ideas for managing concurrency propagated in this paper borrow from various scientific fields and approaches to systems engineering. They emerged over the past two decades from the engineering practice the author has been exposed to in various observer and acting roles.

When working as a technical consultant for the German military procurement agency it became a common observation in all programs and nearly all companies visited to see project teams struggling with the actual level of concurrency. People were overwhelmed by the amount of technical and time dependencies. They tried their best to firefight against recognized deviations. But this endeavor was likely to increase the prevailing overall confusion although some issues were solved. The state-of-the-art in process engineering provided only weak support for managing concurrency. Of course, it was the era of the waterfall model widely disregarding the iterative nature of systems engineering (US Department of Defense 1988).

When taking over responsibility for the development of the Charger Control System for the compound engine of the Grob STRATO 2C high altitude research aircraft (Tönskötter and Scheithauer 1995) the author was convinced of the importance to enable and to control concurrency of the development activities performed by the small project team. Otherwise, we would not succeed to deliver a certifiable product compatible with the program time schedule and within the limited budget. Paramount was the avoidance of inconsistencies creeping into the product documentation. We neither would have the time nor the resources to resolve inconsistencies resulting from performing the defined process without care. In conclusion, the project went well, not at least of the excellent performance of the project team in defining unconventional and innovative solutions. Flight testing of the demonstrator aircraft proved the feasibility to fly up to 24 km altitude with the intended mission aircraft.

When joining EADS in 1999, the development of the EF2000 Flight Control System was in crisis. Benchmarking with similar projects in other companies had concluded deficiencies in the process domain. The author was well aware of the situation as he had supported the German military procurement agency and the national airworthiness authorities since the development had started. It was the opportunity to apply the principles and methods developed before on a larger scale. Instead of a paper based implementation, the enterprise-wide product data management system was utilized to support engineering value stream mapping, status accounting and decision making. The recording of the dynamically performed process followed a static process model applying the process definition approach published in 2000 (Scheithauer and Schindler 2000). The static process model was also adopted for detailed

project planning and control. Eventually, the good news was to get the development project out of the news. Compared over the years, team culture changed dramatically. Reliable commitments are made to the program office with little deviation in the results. Today, the ideas and concepts progressively influence process standardization within CASSIDIAN and EADS (Autran and Scheithauer 2012).

Kurt Levin, a German gestalt psychologist of the first half of the twentieth century, once stated that there is nothing more practical than a good theory. The paragraphs above provide some clues on the practice. The associated theory follows below. At first, the systems engineering challenge is explained in terms of knowledge growth and the resulting process needs. Next the meaning of value in systems engineering is examined, leading to the definition of any released version of a work product as the elementary unit of value in systems engineering. Afterwards, the engineering value stream is introduced as a work product generation sequence. Finally, the management of value streams is discussed primarily from a configuration management perspective. A project management oriented perspective may be elaborated in a future paper.

The Systems Engineering Challenge

Emergence is an essential characteristic of complex systems (Flood and Carson 2010). The systems engineering of complex systems aims for functionality only achievable by the overall product or service. However, most likely the overall system will show further, emergent behavior caused by side effects of design decisions. Occasionally, emergent functions will increase the success of a system. As a simple example, think about a word processing application with three distinct requirements for regular, bold and italic formatting. In this case, a capability of the system to overlay bold and italic formatting would be an emergent function due to the absence of a corresponding requirement. Obviously, it adds value to the product, and may increase competitiveness, if it is welcomed by users and if other vendors are unable to provide the same feature. Other emergent functions may be unintended, especially when they cause unwanted harm to people, society, or the environment.

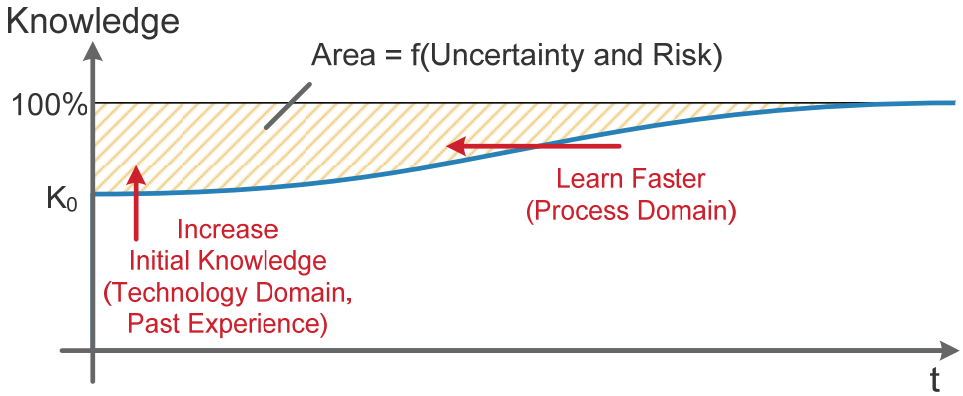

Figure 1. Knowledge Growth Curve.

Knowledge Growth Curve. At the beginning of the system life cycle, the uncertainty about emergent behavior is at its maximum. Over the whole system life cycle emergent behavior becomes recognized increasing the knowledge about the system. Figure 1 shows the knowledge growth curve over time. The s-shaped curve is an approximation justified due to many actors performing interdependent tasks. The y-axis is scaled by the hypothetical 100 percent line representing complete knowledge about the system that is not available a priori as

it is the final outcome of the system life cycle. Neglecting potential residual uncertainties at the end of the life cycle, the knowledge growth curve will converge with the 100 percent line. The then finite area between the knowledge growth curve and the 100 percent line is expressing uncertainty and risk over time. Minimizing this area will be equivalent to increasing the efficiency of the systems engineering effort.

Without utilizing complex mathematics, two characteristics are influential for determining the size of the area. The first is the initial knowledge. As higher the initial knowledge, as higher the efficiency will be. Initial knowledge may be improved by a number of measures. But we should be aware that reaching 100 percent initial knowledge is not achievable for complex systems. Otherwise, we would not call the system complex anymore, but somewhat trivial or complicated. And in the outmost consequence, concepts like the openness of the future and free will could not survive as the foundation of the human mind. The other way of minimization is the compression of the knowledge growth curve in the sense of faster learning. This is a pure process issue. Conclusively, enabling faster learning is the ultimate goal of any systems engineering process.

Life Cycle Phases. Structuring the system life cycle into system life cycle phases is one of the constitutional principles in systems engineering as an enabler for faster learning. At the beginning and at the end of each life cycle phase all the engineering activities performed on all the system elements are synchronized. The risk of major rework in an area more advanced due to principle decisions not made yet in others is reduced. This approach works best, if all lessons are learned in the most appropriate life cycle phase. But there is no guarantee for that. Thus, more fine granular life cycle models with an growing number of life cycle phases do not really increase efficiency. The probability to learn all lessons in the most appropriate life cycle phase is decreasing with a growing number of life cycle phases.

In most cases, it seems to be useful to divide the development of a product or service into three life cycle phases: a concept phase, a definition phase, and a development phase. In the concept phase, the environment of the overall system is analyzed to identify the operational requirements that are allocated to the overall system. Then, solution concepts are generated. Finally, the solution with the best score in terms of mission effectiveness and affordability is selected. For this purpose, the system needs not to be defined in detail from all viewpoints. Instead, effort is spent on the identification of risks of any kind that would hamper to bring the system into being. For identified risks, appropriate avoidance or mitigation measures are defined including demands for technology development. A concept phase may span over a long time period in which system concepts are reiterated in order to further mature the system concept.

In the definition phase, all logical system elements on all architectural levels are defined completely. In parallel the system integration concept should be derived. At the end of the definition phase, reliable estimates for system efficiency and total life cycle costs should be available.

The development phase is dedicated to implementation and system integration. System implementation may include the manufacturing of prototypes like test aircraft needed to perform system integration. Then, the development phase and the production phase may overlap as the assembly usually will be governed by manufacturing processes, and not by the systems engineering process. In extreme cases, production may end before system integration is complete like in case of a one-off border security protection system.

Figure 2 shows this system life cycle model corresponding to a V. The layers in the V represent the logical system levels and the implementation level at the bottom. Below the V the alignment with the life cycle phases is illustrated. Note that the overlap of the concept phase and the definition phase in Figure 2 does not exist on the time line.

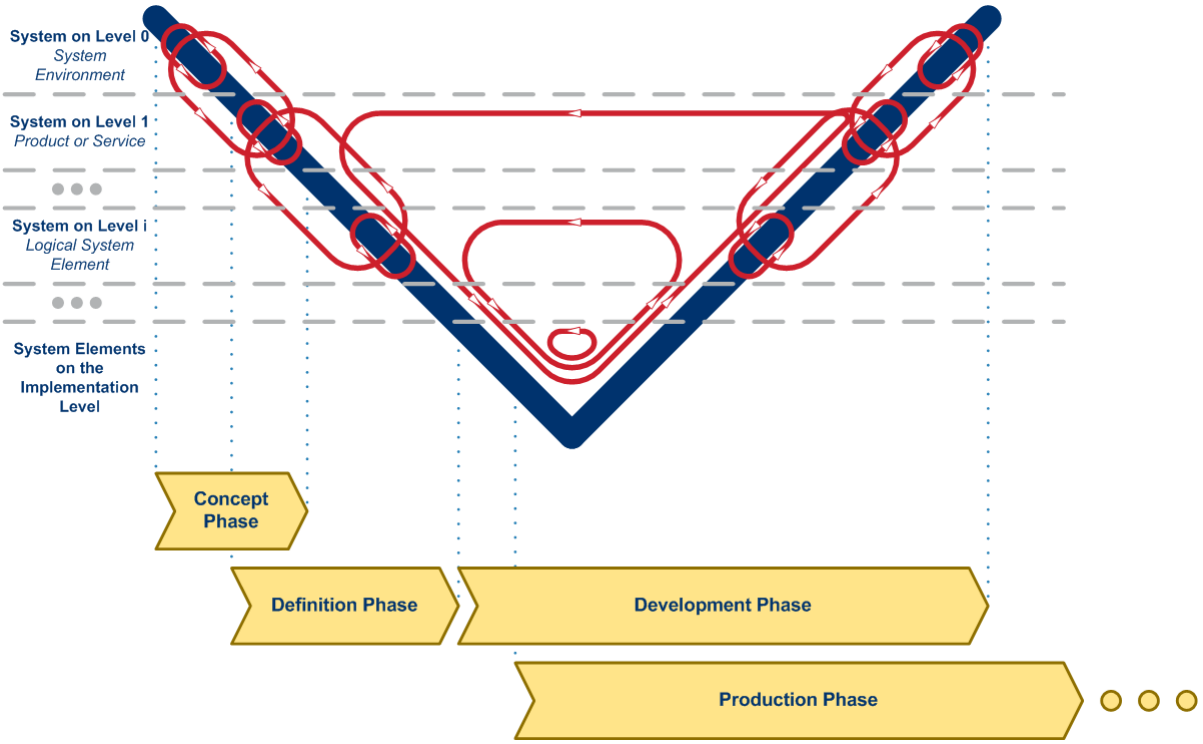

Figure 2. Engineering Life Cycle Phases and Feedback Loops.

Pre-Planned and Event Driven Iterations. Figure 2 also shows all the feedback loops in the V. For each individual system element internal feedback loops are needed on both sides of the V. Further feedback loops may affect lower level system elements. They may cross life cycle boundaries as well.

Controlling the feedback loops may be accomplished in a more pre-planned or a more event driven manner. Pre-planned iterations may be considered in project planning early. Additional budget and time contingencies could be kept low. Less remaining uncertainty results in a potential beneficial impact on profit or competitive pricing. The downside may be a loss in flexibility.

In contrast, event driven approaches need not to rely on assumptions about the future. But managing the incorporation of lessons learned in multi-level system architectures may become a real challenge. However, this challenge is close to the motivation for inventing the Toyota production system. Ohno saw the company faced with low production volumes of particular models and varying customer demands for particular models over time. The company's future competitiveness could be endangered, if import barriers in Japan would be lifted. It was the contemporary understanding of that time that the claim of high volume producers for higher efficiency due to an economy of scale was justified. Toyota reacted by inventing a mainly event-driven manufacturing system with the customers creating the initial events by their orders (Ohno 1988).

Development Philosophies. In systems engineering a combination of pre-planned and event-driven iterations may be most recommendable. The appropriate mix is mainly dependent from the nature of the systems engineering tasks to be performed. Three development philosophies may be used as prototypes for further considerations: waterfall, incremental, and evolutionary. These terms are used here in a somewhat stricter sense than in the original definitions (Larman and Basili 2003). Some overlapping features of the original descriptions are dropped in favor of three clearly distinguishable categories.

The waterfall development philosophy concentrates all required system capabilities to be developed in a single iteration. Additional repair cycles may be necessary. They need to be considered in terms of time and resource contingencies. The waterfall development philosophy is useful whenever the other two development philosophies do not fit. This is usually the case for system elements concerned with any kind of hardware development for which at least provisions for later improvements or extensions need to be considered up from the beginning. Otherwise, a major redesign in further iterations may drive costs and milestone delays.

The incremental development philosophy builds on iterations with particular system capabilities added with each iteration. This approach provides opportunities to incorporate lessons learned in previous iterations in the course of later iterations. The incremental development philosophy is favorable for higher level logical system elements in large scale development programs, and for engineering value streams with a high number of processing tasks in a chain.

The evolutionary development philosophy is the most flexible to accommodate a high number of event driven iterations. With each iteration new capabilities may be added and existing capabilities may be improved following an evolutionary path. The evolutionary development philosophy excels when the duration of a complete iteration is rather short and when the effort on the implementation level is low. In all other cases, it may create an unforeseeable risk to match commitments in terms of quality, time and costs.

Value in Systems Engineering

Existing Definitions of Value in Systems Engineering. A sound understanding of value in systems engineering is an essential prerequisite for modeling engineering value streams. Main stream systems engineering would define value as validated system requirements being implemented correctly. However, the term value is not so much emphasized in the systems engineering literature. For lean thinking and lean systems engineering, value becomes the paramount focus of any process improvement initiative. Surprisingly, the definitions and the conceptual ideas found in the literature on value in systems engineering are not fully consistent. Even more severe, none of these definitions and conceptual ideas are convincing without further amendments.

In their first lean principle, Womack and Jones state that value can only be defined by the ultimate customer (Womack and Jones 1996 and 2003). Similarly, Oppenheim defines value as what the customer says it is, considers important, and is willing to pay for (Oppenheim 2011). These are far reaching abstractions for a customer selecting a model, the engine type, the color, and some optional features when ordering a car as in the Toyota case. For business to consumer contracts, some imbalance is principally tolerable granting the consumer rights on expected inherent characteristics that do not need to be specified explicitly. But that means in turn that some value is not directly defined by the customer

The simplicity of the first lean principle is further challenged by the inherent differences between manufacturing and systems engineering. A customer orders a car from an indeed large, but nevertheless limited number of known offers. In contrast, future systems engineering solutions and opportunities are not foreseeable by potential future customers. It is the task of the systems engineers to anticipate future customer needs for delivering the right products and services at the right point in time to the market. Thus, systems engineers define potential value for the customers in advance.

In the pioneering paper on lean enablers for systems engineering, value in systems engineering is defined in terms of customer satisfaction implying a flawless product over its life cycle (Oppenheim, Murman and Secor 2010). Why this should also imply minimum cost and shortest possible schedule, is not really compelling. In the same context, the current mapping of lean enablers for systems engineering to system life cycle processes according to ISO 15288-2008 (ISO 2008) contained in Oppenheim's book (Oppenheim 2011) is not fully convincing. Especially, no specific traces to configuration management exist, leaving the role of configuration identification according to ISO 10007 (ISO 1993) for defining value in systems engineering weakly addressed only.

In his definition of product development value stream mapping, McManus associates value with the output of the process, and efficiency with the process itself (McManus 2005). Furthermore, the output consists of information. Here ends the brief overview summarizing the published definitions about value in systems engineering so far.

Ohno's Autonomation Principle. Ohno defines the two guiding principles just-in-time and autonomation for reducing waste (Ohno 1988). For a reasonable definition of value in systems engineering, the autonomation principle deserves attention. Ohno defines autonomation as a machine having the capability built in to stop automatically in case of irregularities. By this, forwarding of defective parts to downstream processes is prevented.

The autonomation principle can be translated straight forward to systems engineering: Information needs to be released before being forwarded to downstream processes. Release means to make an explicit decision on forwarding information to downstream processes after the quality has been assessed. In systems engineering, it is not unusual that information is released with omissions. Omissions summarize the differences of the current content with respect to the expected content. They may be caused from capabilities not yet implemented intentionally. Or, they may represent new lessons learned in course of generating and releasing the information. All omissions need to be stored together with the information since they contain information on quality. Omissions inform downstream processes of any restrictions to be considered for further processing the received information.

Work Products and Supporting Data. The term work product is introduced at this point. A work product comprises the information handed over to downstream processes including the known omissions at the point of release. Other information like change control records, trade-off studies, review comments and responses, and important communication records are called supporting data. Supporting data should also be retained in the context of the version of the work product they contributed to. The work product contains the currently valid information. To understand why the work product has evolved as it is, the information contained in the supporting data provides the necessary hints.

Because released work products contain the information needed by downstream processes, they represent value in systems engineering. Work products establish points of reference for subsequent work. Hence, the importance of configuration identification was emphasized above. In reverse, information that is not released does not represent value. If unreleased information is used by downstream processes, most likely, inconsistencies will creep in, resulting in low quality configuration baselines, demanding more repair cycles, and eventually create confusion about current achievements and the remaining tasks until delivery.

Although work products may be also formatted as human readable documents, their purpose goes beyond a document as understood in the still prevailing document centered culture. They serve as the basic building blocks of engineering value streams. Scoping of a work product's content is a matter of the engineering value stream definition as further detailed in the next section.

Defining Engineering Value Streams

Differences Between Manufacturing Value Streams and Engineering Value Streams. Value streams in engineering have a lot in common with value streams in manufacturing. But they also show remarkable differences. The differences need to be considered carefully to avoid inappropriate analogies and conclusions. But first, the similarities are summarized.

An iteration in systems engineering is equivalent to the assembly of a single product item. In both cases, the value stream comprises all activities to be performed from the start until the end. Backflow in an assembly line is as detrimental to efficiency as circular dependencies between work products in an engineering value stream. In production, raw materials and components received from suppliers are processed and assembled by machines and humans to manufacture products. In engineering, a value stream comprises typically the development of a single system element. As far as logical system elements are concerned, allocated requirements from above drive the design, and completed integration of the system elements on the next lower level provide the prerequisite for system integration. The results are configured in released work products and are assigned to system configuration baselines.

In manufacturing, each car produced generates revenues on itself. To minimize the networking capital, the total cycle time to assemble a single car should be as short as possible. In systems engineering, the duration of a single iteration has typically a negligible impact on the networking capital since many iterations are required before a product or service generates revenues. Instead, the engineering value streams of all system elements need to be well coordinated in order to minimize overall development time.

In a perfect lean manufacturing value stream, all iterations are independent from each other. But in systems engineering later iterations are dependent on the results of earlier iterations, and they have an impact on iterations performed even later.

The solution space to manufacture valid products that satisfy customers is known a priori. Deviations can be assessed quantitatively relying on absolute numbers (Deming 1986). This is not possible in systems engineering. Omissions may be either caused by disregarding the initial knowledge and lessons learned earlier, or they represent new lessons learned. The first kind of omissions should be avoided. The second kind of omissions is beneficial to be learned as early as possible. No omissions in early iterations would be suspicious to not really having tackled essential issues yet. A lot of newly raised omissions at the end of the development indicate a

lack in quality. Milestone delays and budget overruns have to be expected. Not absolute numbers, but omission regression over time provides key performance indication.

Process Definition Model. Main characteristics of the process definition model are briefly summarized here. A more elaborate description has been presented twelve years ago (Scheithauer and Schindler 2000). Engineering value streams are built by defining the process tasks representing the generation and check activities for releasing a work product, and by defining the release sequence of work products, e.g. the work product generation sequence. The initial paper was published when we started to utilize the process definition model. Lessons learned since 2000 have led to a more sharpened terminology and more focused definitions. And, the proposed basic workflow model has never been used in practice. When starting the transition to value stream thinking, a huge demand for workflow support was raised. After team members had understood their work product generation sequence they argued that it is not needed. It would have more a potential to hamper them than increasing efficiency. Thus, only mini workflows are used in the product data management system for status changes and corresponding housekeeping actions.

The Project Team. If company culture relies on assigning personal responsibilities to particular documents in order to deliver the final version with a given budget at fixed date, the transition to value stream thinking will be the main challenge. When the buzzword was process orientation in the 1990s, the ultimate goal was already customer satisfaction (ISO 1994), but process orientation was taken mainly as the bottle to fill in the document centered tradition. Process orientation now means usually to have a process for changing a particular document. Of course, reliable information on the status of each work product and consistent content of a particular work product are a benefit. However, a consistent system configuration baseline is expected at the end of the development without setting the focus on maintaining consistency between all the work products of a work product generation sequence continually. Thus, many omissions may be revealed rather late close to the intended end of development requiring firefighting and heroism for timely resolution.

The fate of process orientation should be taken as a warning for lean systems engineering. Document centered thinking is still an issue today. Figure 7 of the article on lean enablers for systems engineering (Oppenheim, Murman and Secor 2010) shows some survey results. The criterion of interest is: “Train to constantly recognize who is the internal customer for every person and every task, and stay connected to him/her to avoid rework”. On a scale ranging from zero to five, systems engineering experts rank this task high with an average value above four. Practitioners rank it between two and three. Apparently, practitioners anticipate systems engineering less as a team effort. Clearly defined sub-tasks allocated to individual engineers connected via controlled interfaces are expected to be sufficient. With this attitude personal motivation is the key demand on engineers as individuals. See Achtziger and Gollwitzer for modeling motivation and action (Achtziger and Gollwitzer 2010).

For mastering the engineering value stream, group dynamics within the engineering team can not be ignored. The model of group dynamics defined by Tuckman (Stahl 2007) with the phases forming, storming, norming, and performing is helpful to explain the role of the work product generation sequence for an engineering team. It is a constitutional necessity in the team building process to generate the work product generation sequence together. This means that the generation of the work product generation sequence is aligned with the storming phase so that a sustainable agreement and commitment of the whole project team to common goals and

behavioral standards is achieved in the norming phase. Be aware that the norming will suffer, if the storming is suppressed. Statements like “it should be clear to everybody how it overall works”, or “we will do it as we always did it before” indicate a tendency to avoid storming. If the norming is not successful, performing will be disturbed by falling back to the storming phase (Stahl 2007). Most likely this will occur in a rather unpleasant moment. Stress levels will be high, and tension between team members may lead to team disintegration.

The role of people was already described in the initial book on lean (Womack, Jones and Roos 1990). Although the second book of Womack and Jones contains more evidence for the important role of people in their case studies, this is not considered by their five lean principles (Womack and Jones 1996). It is the merit of Morgan and Liker to widen the focus on the role of people again (Morgan and Liker 2006). That Oppenheim, Murman and Secor have taken up the idea of a sixth lean principle titled “Respect People” is a beginning (Oppenheim, Murman and Secor 2010). But with time advancing this principle may be spawned to explicitly express its implications.

The Definition Steps. The recommended method to define the work product generation sequence includes the following consecutive steps. In the first step, the nature of information and the dependencies in between should be identified. Standards, books and practices within the organization are valuable sources to be visited. The second step comprises the scoping of work products. The content of particular work products should be as limited as necessary to ensure that no team member is running idle waiting for released work products as input for own work. On the other hand, it makes no sense to increase the number of work products further. Only administrative effort would be increased without any benefit for the engineering team. Furthermore, work products containing design information should integrate all specialty engineering considerations in accordance with the concurrent engineering preference for integrated product development teams (Winner et al. 1988). Otherwise, circular dependencies between work products would be established increasing the administrative overhead needed for their resolution.

The third step is to define the release sequence of the work products. Drawing the complete work product generation sequence on one chart, representing process tasks by blocks and work products by connecting lines, seems to be the best method. If the drawn work product generation sequence shows circular dependencies, the team needs to revert to step two. The last step in the generation process is to group adjacent process tasks to indicate that activities in a single iteration on these process tasks may be performed in parallel, but without compromising the release sequence.

Managing Engineering Value Streams

Two types of management challenges exist: Generating value in each particular engineering value stream, and coordinating the overall development in the V. Overarching is the principle to perform management reviews for approving planned investments, and for checking goal achievement after the investment is completed.

Design Decisions and Action Driving Decisions. A common saying is that managing means to make decisions. In a paper titled “Sustainable Decision Making in Systems Engineering” the author introduced two categories of decisions (Scheithauer 2011). The first category comprises design decisions. These decisions have their share in increasing the knowledge about the

system under development. Design decisions should be prepared and best made by people having the appropriate competence. Thus, design decisions are rarely made by the whole team. For each design decision, the contributions of individual team members will vary according to the nature of the subject.

The author does not know an already coined term for the second category. For the time being, these decisions may be called action driving decisions. Action driving decisions have to ensure that the right design decisions are made at the right point in time by the right people. In other words, the purpose of action driving decisions is to ensure that overall learning takes place as fast as possible. Action driving decisions do not require detailed discipline specific knowledge, but need the buy-in of the whole engineering team. Therefore, action driving decisions should be made by the whole team, respectively by representatives from all engineering disciplines involved in the particular engineering value stream. When talking about managing concurrency, we talk about making the right action driving decisions.

Concurrency in Engineering Value Streams. Iterations are induced to the work product generation sequence by a two step process as shown in Figure 3. In the first step, each possible new demand and each feedback record from previous iterations is evaluated for the need for change. This leads to a prioritized list of what is called the product backlog in Scrum (Pichler 2010). Taking items from the product backlog, work packages are defined and integrated in the time schedule and resource allocation plan. In Scrum a work package is called a sprint backlog. However, the term sprint is discarded here because it stands for a time boxed approach. Sprints in Scrum are performed strictly sequentially one after another. In work product generation sequences with a high number of work products in a chain, and engineers with specific knowledge needed to perform particular process tasks, this approach would be not efficient. There will be time overlaps of several iterations performed in parallel.

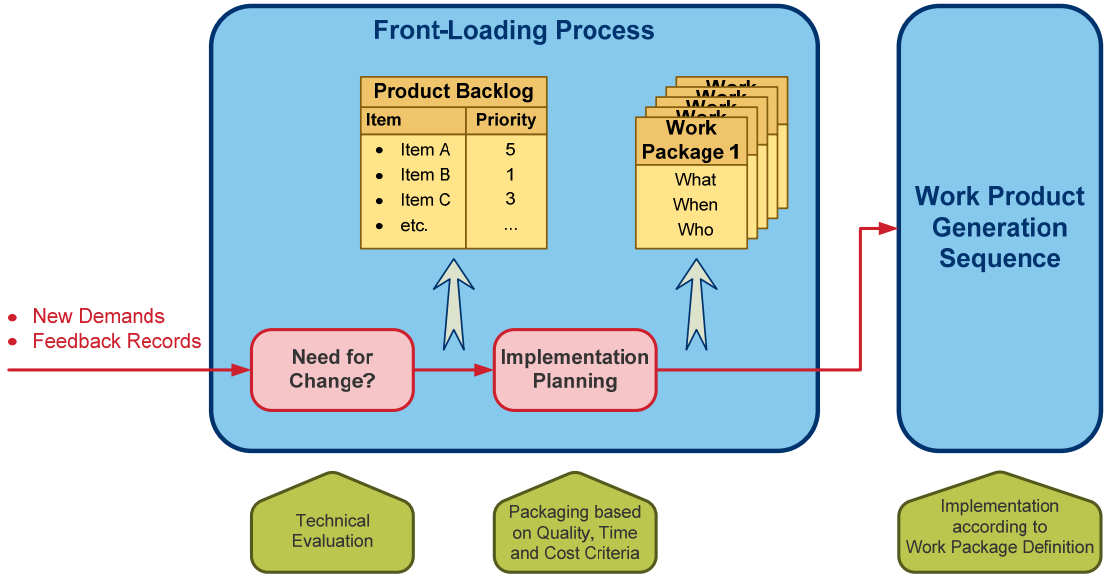

Figure 3. Work Product Generation Sequence Front Loading Process.

The restriction for constant takt times propagated by Oppenheim for less demanding systems engineering challenges (Oppenheim 2004 and 2011) is also abandoned. At first, it is nearly impossible to select a unique takt time appropriate for all the feedback loops indicated in Figure 2. At second, there is no value generated and no networking capital minimized by setting constraints on each work package to take always the same effort, and to spread the effort

evenly over all process tasks. The paramount goal is to minimize waiting. In engineering value streams various degrees of freedom may be exploited for achieving this: varying amount of effort for each iteration, fast track and short track iterations, pausing of particular iterations. Of course, it is not easy to master all these degrees of freedom. Planning tools need to be amended, and humans have to build up appropriate competence for fully utilizing the capabilities of the process. But this is less complicated than mastering firefighting because sound information on status and dependencies is available continually.

The implementation of the outlined process for front-loading the work product generation sequence is mainly challenged by a phenomenon that may be called the wisdom of the coffee corner. People talk about issues they suspect or they are aware of in informal discussions, but this information does not reach the front-loading process. For explanation, the group dynamics model introduced above and social psychology need to be consulted again:

The team is in the performing phase. In a good team, the whole team is committed to make progress. Nobody wants to stay aside. In a bad one, individuals prefer to hide their concerns. In both cases, team members fear to be sanctioned by the group, if they disturb performing by raising issues. Of course, they need some relieve. Modern office layouts placing the meeting space for informal discussions in the center of the office are not really suited to solve the issue.

What is required, is to give people dedicated opportunities to reflect the work of the team and the results achieved, and to use some strict rituals to report issues with the assertion that reporting the issue has no adverse impact on the person reporting. The first measure may be implemented by short meetings each morning to talk about observations and needs (called sprint meetings in Scrum) and by dedicated workshops off-site the working area to sort out major issues. The long established formalisms for issue and problem reporting provide the appropriate ritual for generating feedback. But to accept no query without a proposal for solution, will keep people quiet due to heightening the barriers. This would be more a part of the problem than a contribution to a suitable solution.

Concurrency in the Overall Development. Managing all projects in the overall V is concerned with the coordination of all the engineering value streams to deliver the product as demanded in terms of quality, time and costs. These demands are set in a top-down process following marketing needs for product or service capabilities, competitive pricing, and time to market.

In contrast, the project managers of the individual engineering value streams generate their plans in a bottom-up fashion as described above. Projects have to balance their progress, time schedule and budget utilization with the commitments in terms of quality, time and costs they made to the program management. It is the purpose of the program management to react on deviations against these commitments, and to synchronize the overall progress.

As for the management of engineering value streams, program management needs to minimize waiting on a program scale. An incremental development philosophy and applying sound systems engineering and project management practices as defined in the literature (Forsberg, Mooz and Cotterman 2005) are the most suitable approach. Overall program management controls the hand-over of information between the project teams. Program management is in charge to perform the management reviews at the entry and exit point of each life cycle phase. In regularly performed maturity status reviews, achievements, major omissions and the overall risk status are assessed.

Efficiency of the program management benefits from a higher quality of the system development results delivered by the engineering value streams and improved adherence to the top-down plans. They will rarely have to interfere with the project management of any system element in a firefighting mode. Instead they control the program by defining increment content and associated milestone and resource plans.

Conclusion

Integrating state-of-the-art knowledge from various scientific disciplines and approaches to systems engineering as described within this paper paves the way for performing systems engineering management for complex systems successfully. Among all the detailed considerations for substantiating the proposed approach three traits are most essential and are therefore summarized below.

At first, iterations are the rule in systems engineering, not the exceptions. A process model based on the mapping of logically sequences to the time line linearly, and ignoring the need for and the benefits of feedback cycles as nuisance signs of principally avoidable imperfection is a rather coarse view on the systems engineering practice. It is better to optimize the iteration cycles for being executed concurrently and faster.

At second, value stream thinking means a real paradigm shift away from document centered and otherwise process oriented traditions. High quality of the systems engineering effort is represented by high quality system configuration baselines as a whole. This is hardly achievable by controlling content and progress of individual work products on a per work product basis alone. For real-time systems engineering management, the evolution of system configuration baselines need to be continuously controlled in the first place.

At third, systems engineering teams consist of groups of human beings. General knowledge on group dynamics apply. The heterogeneity of multi-disciplinary systems engineering teams even increases the social-psychological challenge. Most important are commitment and will to succeed in achieving project goals. Otherwise, when things become tough, a lot of explanations will show up why goals could not be achieved. Sometimes people start early to test their later reasoning why project failure was almost inevitable from the beginning. Then, people seconded to a project team have to achieve a common agreement about what to do together. In other words, they have to define and commit themselves to their engineering value stream. With less priority, it has to be sorted out which methods and tools are applied for the systems engineering activities.

Finally, the author is tempted to amend the statement from Kurt Levin cited in the introduction with the following sentence: And, there is nothing worse than taking an inadequate theory for guidance.

References

Achtziger, A., and P. M. Gollwitzer 2010. „*Motivation und Volition im Handlungsverlauf*.“ In *Motivation und Handeln*, 4th Edition, edited by J. Heckhausen, and H. Heckhausen, 309-335. Berlin (GE), Heidelberg (GE): Springer-Verlag.

Armstrong, J. R. 2009. “Divergent Thinking in Systems Engineering Practice: Is There a Shortfall?” Paper presented at the 19th INCOSE International Symposium, Singapore, 20-23 July.

Autran, F., and D. Scheithauer 2012. „Introducing Systems Engineering Views in Product Lifecycle Management.“ Paper presented at the 22nd INCOSE International Symposium, Rome (IT): 9-12 July.

Deming, W. E. 1986. *Out of the Crisis*. 1st MIT Press Edition (2000). Cambridge, MA (US): MIT Press.

Flood, R. L., and E. R. Carson 2010. *Dealing With Complexity: An Introduction to the Theory and Application of Systems Science*. 2nd Edition. New York, NY (US): Plenum Press.

Forsberg, K., H. Mooz, and H. Cotterman 2005. Visualizing Project Management: Models and Frameworks for Mastering Complex Systems. 3rd Edition. Hoboken, NJ (US): John Wiley and Sons.

Friedenthal, S. A., and C. Kobryn 2004. „Extending UML to Support a Systems Modeling Language.“ Paper presented at the 14th INCOSE International Symposium, Toulouse (FR): 20-24 June.

Haskins, C., ed. 2010. *Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities.* Version 3.2. Revised by M. Krueger, D. Walden, and R. D. Hamelin. San Diego, CA (US): INCOSE.

ISO (International Organisation for Standardisation). 1993. ISO 10007. *Quality Management Systems – Guidelines for Configuration Management*.

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

ISO (International Organisation for Standardisation). 1994. EN ISO 9001. *Quality Management Systems – Requirements*.

Köhler, W. 1947. *Gestalt Psychology: The Definitive Statement of the Gestalt Theory*. New York,NY (US): Liveright Publishing Corporation.

Larman, C., and V. R. Basili 2003. “Iterative and Incremental Development: A Brief History.” IEEE Computer (June 2003): 2-11.

Maier, M. W. and E. Rechtin 2009. *The Art of Systems Architecting*. 3rd Edition. Boca Raton, FL (US): CRC Press.

McManus, H. L. 2005. “Product Development Value Stream Mapping.” Release 1.0, Massachusetts Institute of Technology, Lean Advancement Initiative (Cambridge, MA, US).

Morgan, J. M.,and J. K. Liker 2006. *The Toyota Product Development System: Integrating People, Process,and Technology*. New York, NY (US): Productivity Press.

Ohno, T. 1988: *Toyota Production System: Beyond Large Scale Production*. Boca Raton, FL (US): CRC Press.

Oppenheim, B. W. 2004. “Lean Product Development Flow.” *Systems Engineering Journal* 7: 352-376.

Oppenheim, B. W., E. M. Murman, and D. A. Secor 2010. “Lean Enablers for Systems Engineering.” *Systems Engineering Journal* 13.

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

Pichler, R. 2010. *Agile Product Management with Scrum*. Boston, MA (US): Pearson Education.

Scheithauer, D., and Schindler, A. 2000. „A Standardisation Concept for Non-Standard Development Projects.“ Paper presented at the 2nd European Systems Engineering Conference, Munich (GE): 13-15 September.

Scheithauer, D. 2011. “Nachhaltige Entscheidungsfindung im Systems Engineering.” In *Tag des Systems Engineering*, edited by M. Maurer and S.-O. Schulze. München (GE): Carl Hanser Verlag.
Stahl,E. 2007. *Dynamik in Gruppen*. 2. Auflage. Weinheim (GE), Basel (CH): Beltz Verlag.
Tönskötter, H., and D. Scheithauer 1995. „The STRATO 2C Propulsion System: A New Compound Engine and Control Concept for High Altitude Flying.“ Paper presented at the AGARD Conference on Advanced Aero-Engine Concepts and Controls, Seattle, WA, 25-29 September.
US Department of Defense. 1988. Military Standard DoD-STD-2167A. *Defense System Software Development*.
Vincenti, W. G. 1990. *What Engineers Know and How They Know It: Analytical Studies from Aeronautical History*. Baltimore, MD (US), London (UK): The John Hopkins University Press.
Winner, R.I., J. P. Pennell, H. E. Bertrand, and M. M. G. Slusarczuk 1988. „The Role of Concurrent Engineering in Weapon System Acquisition.“ IDA Report R-338, Institute for Defense Analyses (Alexandria, VA, US).
Womack, J. P., D. T. Jones, and D. Roos 1990 and 2007. *The Machine That Changed the World*. New York, NY (US): Free Press.
Womack, J. P., and D. T. Jones 1996 and 2003. *Lean Thinking: Banish Waste and Create Wealth in Your Corporation*. New York, NY (US): Free Press.
Zonnenshain, A., Sillitto, H., and Kasser, J. 2011. „People or Process: Which is more important?“ Panel held at the 21st INCOSE International Symposium, Denver, CO (US), 20-23 June.

Biography

Dieter Scheithauer studied electrical engineering with special emphasis on automatic control at the Universität der Bundeswehr München resulting in the degree of a Diplom-Ingenieur univ. in 1980 and a doctor’s degree (Dr.-Ing.) in 1987. His service as Technical Officer in the German Air Force ended in 1988. From 1988 to 1999 he was employed by Industrieanlagenbetriebsgesellschaft GmbH (IABG). He worked in a branch mainly delivering technical expertise to the German Ministry of Defence and other government organizations. Throughout his professional career 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 for unconventional airborne and ground-based systems. In 1999 he joined the European Aeronautic Defence and Space Company. Since then he has worked mainly in the field of process engineering. He now holds a position as Senior Expert Systems Engineering Processes within CASSIDIAN. Convinced of the importance of systems engineering for the company, he contributed to achieve its high level of recognition by CASSIDIAN and EADS today.
He is a former president and an honorable member of GfSE – The German Chapter of INCOSE. He represents CASSIDIAN on the INCOSE Corporate Advisory Board. And, he is an INCOSE Expert Systems Engineering Professional.