Agility Meets Systems Engineering: A Catalogue of Success Factors from Industry Practice

Ernst Stelzmann, Christian Kreiner, Gunther Spork,
Richard Messnarz, and Frank Koenig

Abstract. Agile software development methods are widely accepted and valued in software-dominated industries. In more complex setups like multidisciplinary system development the adoption of an agile development paradigm is much less straightforward. Bigger teams, longer development cycles, process and product standard compliance and products lacking flexibility make an agile behaviour more difficult to achieve. Focusing on the fundamental underlying problem of dealing with ever ongoing change, this paper presents an agile Systems Engineering approach as a potential solution. Therefore a generic Systems Engineering action model was upgraded respecting agile principles and adapted according to practical needs discovered in an empirical study. This study was conducted among the partners of the S2QI agile workgroup made up from experts of automotive, logistics and electronics industries. Additionally to an agile Systems Engineering action model, a list of 15 practical success factors that should be considered when using an agile Systems Engineering approach is one of the main outcomes of this survey. It was also found that an agile behaviour in Systems Engineering could be supported in many different areas within companies. These areas are listed and it is also shown how the agile action model and the agile success factors are related to them.

Keywords: Agility, Systems Engineering, experience.

1 Introduction

In IBM's Global CEO Study 2006 65% of all CEOs were expecting substantial change for their companies within the next 3 years. In 2008 this value increased to 83%. During the same period the number of CEOs saying their companies had already dealt with change successfully was only increasing from 57% to 61% [1].

So change is not only a factor that most companies have to deal with, it is also a challenge that a significant number of companies had not handled successfully yet. This paper examines companies that are developing systems with the help of Systems Engineering methods. The challenge this paper deals with is the dynamic market environment for system developers that is influenced by 4 interrelated factors (see Figure 1).

For developing software within this environment, some so called "Agile Methods" have been introduced, like Extreme Programming [2], SCRUM [3], Crystal [4] and Feature Driven Development [5]. The specific methods have following common characteristics [6]:

A. Riel et al. (Eds.): EuroSPI 2010, CCIS 99, pp. 245–256, 2010.

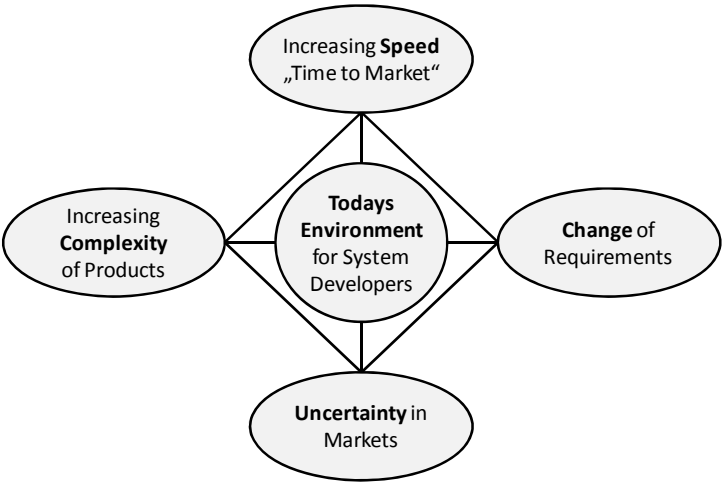

Fig. 1. Characterization of Market Environment

- Learning attitude
- Focus on value for the customer
- Short iterations delivering value
- Neutrality to change
- Continuous integration
- Test-driven
- Lean attitude
- Team ownership

More generic principles for agile SW development are stated by the well known Agile Manifesto (http://www.agilemanifesto.org).

Although Software is often a major component of systems, it is not the only one. Other components need to be treated differently, since they have to be produced physically. Sometimes the production takes a long time and thus it takes also a long time to implement changes (e.g. several months to prepare a casting mould). Development processes are also different for SW and different types of HW (mechanical, electronically, etc.). Hence it is also a necessary Systems Engineering task to synchronise the different development cycles. And usually system development projects are bigger than sheer SW development projects in terms of team size, complexity of product and development time. So it is questionable if these agile development methods for SW can also be applied for developing multidisciplinary systems. It is also questionable if principles, which result in an agile behaviour in SW development, will also support an agile behaviour in systems engineering. Therefore the idea of agility will be analyzed more in detail:

The most comprehensive and generic definition we found for agility is: “Agility is a persistent behaviour or ability of a sensitive entity that exhibits flexibility to accommodate expected or unexpected changes rapidly, follows the shortest time span,

uses economical, simple and quality instruments in a dynamic environment and applies updated prior knowledge and experience to learn from the internal and external environment.'' (Qumer and Henderson-Sellers [7])

An often used tool to visualise the concept of agility is John Boyd's OODA Loop [8]. It distinguishes 4 phases necessary for responding to change (new information) and calls them Observe, Orient, Decide and Act. It also points out how the later phases influence the earlier ones and shows feedback loops.

In a prior paper [9] we concluded that agility can be supported by enhancing any phase of this concept, but all phases must be considered to generate a comprehensive agile Systems Engineering approach. So it has to be considered:

- The ability to gather new information
- The ability to analyze this information
- The ability to decide for its relevance and how to react
- The ability to respond, which can also be called "flexibility"

And furthermore it has to be taken into account that changes could arise from outside of the development process (like changes of customer needs or laws that have an influence on the product) but could also become apparent after receiving feedback from previous development actions. And this does not have to be feedback on failures. High complexity within systems and short "time to market" often result in the fact, that not all requirements are known, when development starts. Also the fact that behaviour of complex systems can't be foreseen exactly, but emerges when all components of the system are integrated, makes it necessary to start development, without having all information in advance. So preliminary stated requirements may change or new requirements may show up when more information (feedback) becomes available during development. Since changes are easier (and cheaper) to handle in early development phases [10], the development process has to force early information gathering (feedback) about the behaviour of the system or possible failures within the state of development. And it should also provide the customer/user with all information on the system he needs to be able to give feedback about his real expectations about the system function. So these changes can be caused externally or internally and their moment of appearance is strongly influenced by the Systems Engineering process and interactions with customers and users.

So an agile Systems Engineering approach should be able to deal with both types of changes and should consider all phases of the OODA Loop. And according to the definition of agility it should do it fast.

The agile Systems Engineering approach presented in this paper consists of 3 parts that will be explained in the following 3 chapters:

- An agile Systems Engineering action model
- A list of supportive actions in different areas of companies
- A catalogue of success factors for applying agile methods

2 Agile Systems Engineering Action Model

There is no commonly accepted approach for an "Agile Systems Engineering" existing right now, but several companies try to apply agile SW development methods to

develop multidisciplinary systems. So do the companies, which are members of S2QI agile workgroup, an initiative to share experience on selected topics among its members. A survey in these companies (from automotive, logistics and electronics industries) revealed that using agile SW development methods may help in becoming more agile also for developing systems. But this is not a trivial undertaking. A lot of things have to be considered for generating a Systems Engineering approach that should be capable to show agile behaviour but should also comply to general Systems Engineering objectives like effectiveness and efficiency.

The first part of the approach in this paper is an action model for Systems Engineering. As a foundation the Systems Engineering Action Model designed at ETH Zurich [11] was used. It is a generic model that consists of principles and procedures for a good Systems Engineering practice. Other basic principles for the agile action model were taken from the Agile Manifesto and several agile SW development methods. Not all agile principles were applied, only those, which seemed to be appropriate for Systems Engineering. Furthermore ideas from analyzing the basic concept of agility with the help of the OODA Loop were implemented. Figure 2 shows a version of this Agile Systems Engineering Action Model that was already developed further after partners from S2QI agile workgroup had reviewed it. It consists of 4 main principles:

- Esteem Developers: This principle should assure that developers are treated in the way they deserve. Often processes are cared more about then developers. But developers are doing the work, they are the brainpower in each process and they ensure agile behavior. Therefore it is wise to consider all suggestions, stated in the figure, so that the developers can do their job in the best way.
- Incremental Development with close Customer/User Interaction: This principle proposes that it is not necessary to specify all requirements in detail at the beginning, because some (or many) of them will change during development. Therefore a “product vision” should be enough when the project starts. Then development proceeds by developing items that could be any physical part of the system, but also prototypes, simulation models, plans and so on. Items that push information gathering for eliciting requirements or for better understanding of the system should be developed first. A finished item should always be something that could be presented to the customer to receive feedback.
- Iterative Development of Increments: Each development item is developed in iterations of the displayed development cycle. It starts with a planning phase, where it is decided what to develop, who will develop and how long it may take. After that detailed objectives and test criteria are formulated. Then the designated developers start to develop their parts whereas “search for solutions” and “selection of solution” are technical planning steps and “realisation” is the phase in which a system increment is being produced, a SW feature is coded or a document is created. If the search for solution provided different variants, the next iteration can start by selecting another variant, after a negative test result. After a passed test, the developed item is presented to the customer.

- Flexible Design: This should be applied to the product itself, but also to other things like processes, organisation, production and supply chain. The Incremental- and Iterative Development principles are mostly important for agile reactions on changes that come apparent due to new information emerging during development. The Flexible Design principle is imperative for all changes, since there has to be flexibility somewhere (mainly in the product itself) for implementing changes. Some guidelines for making systems more flexible are presented later.

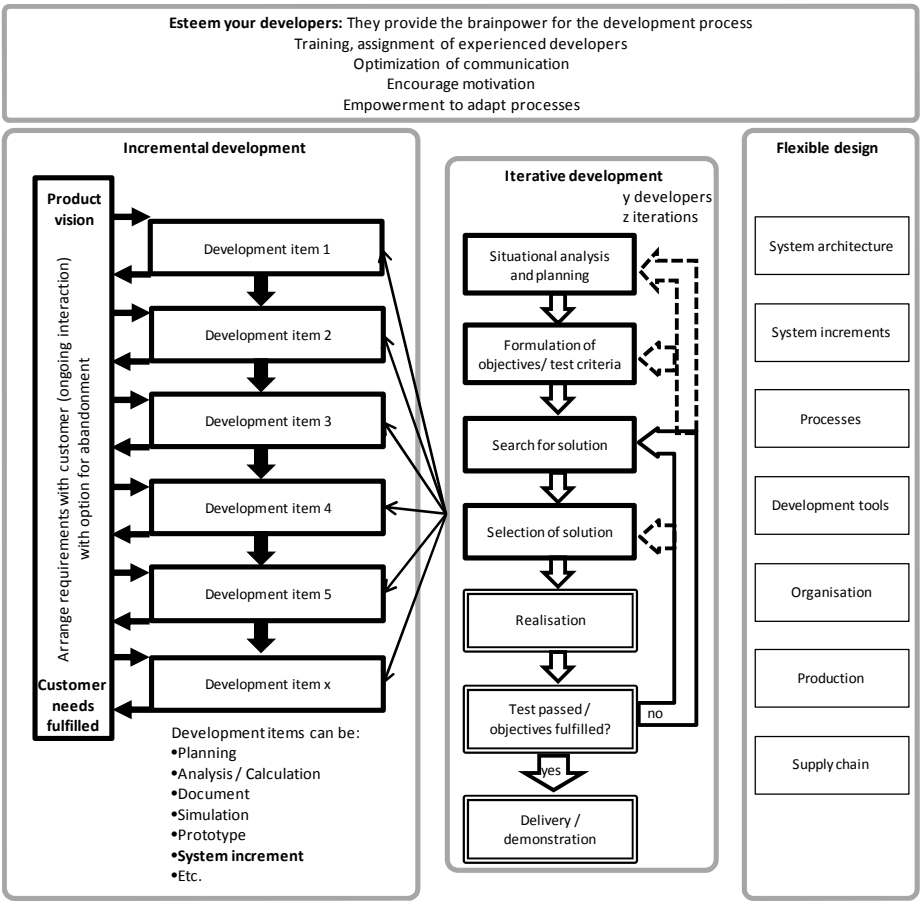

Fig. 2. Agile Systems Engineering Action Model

The agile Systems Engineering action model may be used to adapt existing development processes for more agility as well as to create new processes showing agile behaviour without losing necessary characteristics for good Systems Engineering practice. Some parts of it like handling of developers and customers and flexible design are partly process topics but are reaching beyond. Since neither Systems

Engineering nor Agility should be limited to process management a comprehensive look on system developing companies was taken to find more starting points for supporting agile behaviour in Systems Engineering.

3 Supportive Actions in Different Areas of Companies

The second part of the agile approach in this paper describes supportive actions for an agile behaviour within Systems Engineering. At first a classification into categories should be found to better handle the large number of possible options. It was chosen to use areas of system developing companies as shown in figure 3.

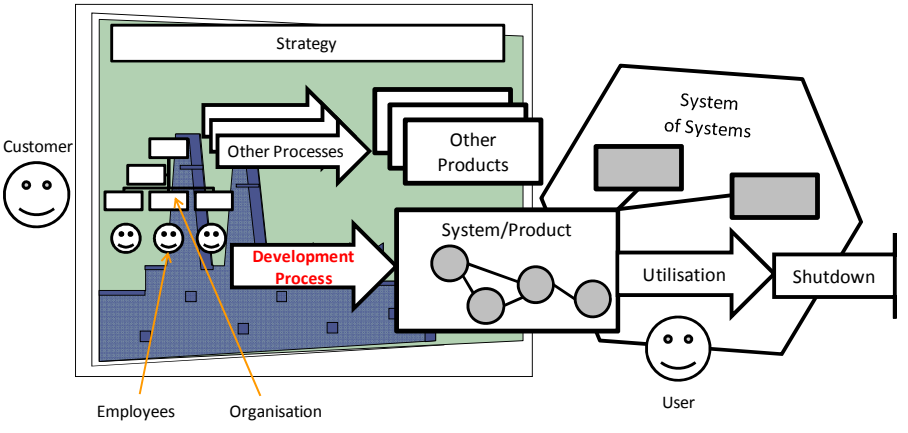

Fig. 3. Relevant Areas for Agility within System Developing Companies

- Interaction with customer and user

This is already covered by agile SW development methods [2], [3], [4]. Customer and user should be involved in system development during the whole project. They are required to define their real needs and prioritize them. If their needs change throughout the project or can be stated more precisely, this new information should be considered immediately. This is largely a process topic, but also communication and psychological issues have to be treated, therefore it was chosen to be a category.

- Organisation

The organisational form of the company and of the project is important for communication, which is seen as one of the major points for agile behaviour [2], [3], [4]. Also the composition of teams and the collaboration between different teams is important. Team members who leave the company can also be seen as changes, which have to be dealt with within this area.

- Employees / culture

Employees are the main drivers for agility because they naturally have the ability to gather new information, decide for their relevance and take action to change something.

So employees are able to fulfil all phases of the OODA-Loop. Therefore employees should be in centre of all agile considerations. Corporate culture and organization should be optimized to not hinder employees in being agile. It is also considered that high-skilled and experienced employees have more potential for being agile [2], [3], [4].

- Development process

Also the development process should not restrain employees in being agile. It should rather support agility by including tasks for observing new information and information processing. The management of requirements is essential for agility and also a fast providing of helpful prototypes or parts of the system. When changes appear the process should provide routines to adapt the product. An agile process is also adaptable for itself. Agile SW development methods are mainly focusing on process topics [2], [3], [4], [5] and seem to be good starting points for agile Systems Engineering processes.

- Product (System)

The system is the central demand of the customer and it is obvious that the system has to be changed when it doesn't fulfil the requirements. So the system has to provide the flexibility that is needed for reacting on change. To enhance flexibility of product architecture several concepts have been published. Fricke and Schulz [12] list following principles:

Ideality/simplicity (minimize number of interfaces, secondary functions …)

Independence (minimize impact of changing design parameters on other design parameters)

Modularity (minimize coupling among modules and maximize cohesion within modules)

Integrability (compatibility and interoperability applying generic, open or common interfaces)

Autonomy (objects are available to provide basic functionality independent from embedding system)

Scalability (units independent from scale or self-similar/fractals)

Non-hierarchical integration (units linked across system with no respect to type of modularity)

Decentralization (control of information, resources, attributes and properties distributed within system)

- Strategy

This topic addresses super ordinate decisions for the entire company. For example flexibility (limited to a special domain) for the company could be achieved with a platform strategy for similar products. When considering the company as a system itself, principles from 3.5, could be used to improve agile behaviour to form an "Agile Enterprise" [13].

Gathering of outside information could be done more efficiently on company level than in separated systems engineering teams. Changes happening there are not within the area of system developers' daily work. Therefore it makes no sense to make Systems Engineering processes responsive to all changes that might happen. Otherwise they would become very inefficient. An agile behaviour on enterprise (strategy) level might be a solution for this type of changes. Information gathering could be done on enterprise level and also strategies for reacting on change are better applicable by using all resources of an enterprise.

- System utilisation / market

On the one hand systems should be adaptable after launch (e.g. software-updates) and on the other hand it is sometimes possible to use systems for other purposes than originally planned. Therefore even the utilisation phase of a system should be taken into account to provide options for agility.

4 Agile Systems Engineering Success Factors

The third part of the agile Systems Engineering approach is a catalogue of success factors that have to be considered when applying "Agile Systems Engineering". They are not exclusively connected to the agile Systems Engineering action model, but should also be considered when using agile SW development methods for developing systems. The following factors were identified by studying real-world practices in companies working together in the S2QI agile workgroup. A factor was listed, if it was critical for successfully applying agile methods in at least two different companies. There are also several methods, principles and ideas named by S2QI partners presented as possible solutions.

- SF1: Agile project setup (alias project launch meeting)

During project launch it is necessary to set or adapt several organisational and process properties, because standardised values are not always appropriate. Right-sized project setup and management structures balance project management, quality, and development efficiency.

Possible solution: Dependent on the product, albeit starting from the organization's standard, it is possible to scale the approval plan (honouring a minimal set of quality gates), communication/reporting processes, team configuration and number of development cycles (mostly hardware). In an agile mindset, this process could occupy the project's first agile iteration (SCRUM: sprint).

- SF2: Change response strategies

Late changes do occur. Systematically responding resp. preparing to respond to changes can effectively mitigate their – typically negative – impact (extra costs and time). Flexibility exposed in this way is value on its own.

- SF3: Direct customer communication

It is necessary to organise communication between customer and development to be as direct as possible, because direct communication is efficient, has high bandwidth and the least losses.

Possible solution: Form pairs of customers and function owners on several levels to facilitate direct and parallel communication paths and leverage common (sub-) domain knowledge and vocabulary.

- SF4: Customer/market oriented vs. mandatory requirements priorities

Prioritisation of requirements to balance market orientation and technically mandatory requirements (e.g. safety or legal) is needed.

Possible solution: Drive Software development cycles by requirements derived from a prioritised feature/requirements backlog. Mandatory/major feature items are flagged, and cannot be postponed.

- SF5: Software and hardware development coordination and collaboration

Sub-disciplines within product development often have different cycle times, as well as different vocabulary. Coordination/Collaboration is essential to deliver an integrated product successfully.

Possible solution: Synchronise development iterations at several (integration) milestones. Use hardware emulation to enable early and instant test feedback to software developers (software cycles are normally much shorter than hardware cycles, esp. with agile software development)

- SF6&7: Flexible product (line) architecture and systematic reuse

Flexibility of product architecture is inevitable. Systematic reuse was seen as success factor at the beginning of S2QI's research. It is now seen as an option to ensure efficient, albeit flexible product development within a certain domain focus. Therefore SF7 was included in SF6 as possible solution.

Possible solution: Analyse your domain to define your core assets, the scope of your domain and your goals. Develop your components reusable. Take care of their interfaces, parameterisation and internal flexibility. A common architecture is essential. Have a named architect (role). In case of software biased products, this goal oriented, systematic reuse approach is called "Software product line".

- SF8: Effectively linking requirements and tests

Test cases must be updated, in order to be able to continuously check your product's features. Documentation effort should be kept low. Making requirements executable makes them unambiguous (necessary for execution) and reusable as automated test cases. Exercising executable requirements in verification phase and later as test case against the system also lets requirements errors, inconsistencies, etc. show up earlier.

Possible solutions: Make requirements executable so that they can serve as automated test cases. Use Test Driven Development. Red-Red-Green: have the test case executable and failing (red) before unit development. Develop functionality (frequent feedback: still red) until tests are "green". Think of unit tests as "casting moulds" for software units. Automated unit tests and module tests. "Test" the test cases (requirements) in return. Automated acceptance tests (FDD, ATDD). Think of a "function probe".

- SF9: Know your agile method

E.g. when using agile software development methods like SCRUM, it is necessary to clearly distinguish between product backlog (project/product features) and sprint

backlog (single iteration features). When adopting agile development methods in conservative environment, multiple time-boxed iterations appear in addition to the global project development cycle. Clearly distinguishing between these is obviously necessary. It is reported however, that this separation is not done quite often.

Possible solution: Introduce all participants to your agile method. Product backlog (mostly) contains the current priority-sorted list of customer features. Sprint backlogs contain features to be realised during the current agile iteration (typically 1 month). Sprint backlogs are planned by the development team by taking features from the product backlog. A systematic, fair method for prioritisation of the product backlog is very helpful.

- SF10: Team work, team thinking, team taking responsibility

Working agile involves a lot of informal communication and tacit knowledge. The better the team spirit, the better their performance.

Possible solution: Do common planning (esp. sprint planning) but also common detailing of product backlog items within team. Develop and maintain the team's common goal orientation. Force "collective" commitment to reach the (self-planned!) sprint goals. Allow collective ownership of the realized substance. Do pair working (on demand).

- SF11: Synchronise sprint cycles with general organisation control cycles

Especially in mixed environments, where agile teams work within classical structures, sprint end delivery/reporting/retrospective events should be synchronised with company standard control structures.

Possible solution: Synchronise (some) sprint cycle ends with non-agile control structure due dates.

- SF12&13: Team and inter-team organisation

Teams only work well, when they are not too big and not too small [14]. Working agile involves a lot of informal communication and tacit knowledge that has to be exchanged across teams' borders. Therefore team size and interfaces between teams are critical. Possible solutions should consider large projects (e.g. SCRUM of SCRUMs), hardware/software teams, geographically separated teams etc.

- SF14: Ensure minimal requirements documentation

While keeping requirements documentation to minimum, sufficient information has to be available for serious sprint planning and task effort estimation.

Possible solution: Have a minimal requirements documentation pattern (e.g. description, additional documents, test cases, acceptance criteria). Possibly it can be supported in an RM tool.

- SF15: Generic requirements

Not really critical but by formulating generic requirements usefully work could become more efficient. E.g. by using parameters for generic requirements they could become reusable.

5 Conclusion

While agile software development is already widely accepted and adopted, there is no commonly accepted approach for agile Systems Engineering. Some companies were already trying to apply agile methods from SW development to the development of complete systems. So did the industry partners of S2QI agile workgroup. Since Systems Engineering is usually a complex endeavour there was no silver bullet found, to provide an agile solution for all issues. Systems show complexity for themselves with interdependencies between components but also Systems Engineering processes have a lot of interdependencies to processes and other things in different areas of system developing companies. Therefore a lot of elements have to be considered when a more agile behaviour within Systems Engineering should be achieved.

To provide a guideline for Systems Engineering with agile behaviour an agile Systems Engineering action model was presented that was combined from knowledge about Systems Engineering and different agile concepts and methods and reviewed by S2QI partners. This action model has an emphasis on process management but also addresses other topics like personnel and flexibility in products. In this sense system developing companies were analysed for possible areas where agility could be supported. Categorized into these areas starting points for agility were presented.

Finally a catalogue of success factors for applying agile Systems Engineering methods was elaborated. This list is the result from a survey conducted among partner companies of S2QI agile workgroup and is not directly connected to the presented agile action model. The catalogue of success factors should also be considered when applying only separate agile principles or an agile SW development method to system development. This catalogue is intended to be a starting point for discussion, extension, abstraction and other strategies to foster deeper understanding as well as widening practical applicability. To provide a comprehensive and agile approach that is capable of solving all issues within Systems Engineering more research is needed.

References

[1] Palmisano, S.J.: IBM global CEO study 2008 – the enterprise of the future (2008), http://www-935.ibm.com/services/us/gbs/bus/html/ceostudy2008.html

[2] Beck, K.: Extreme Programming Explained: Embrace Change. Addison-Wesley Professional, Reading (October 1999)

[3] Schwaber, K.: Agile Project Management with SCRUM. Microsoft Press, Redmond (2004)

[4] Cockburn, A.: Crystal Clear: A Human-Powered Methodology for Small Teams. Addison-Wesley Professional, Reading (October 2004)

[5] Palmer, S., Felsing, J.: A practical guide to Feature Driven Development. Prentice Hall PTR, Englewood Cliffs (2002)

[6] Turner, R.: Towards agile systems engineering processes (2007), http://www.stsc.hill.af.mil/CrossTalk/2007/04/0704Turner.html

[7] Qumer, A., Henderson-Sellers, B.: Crystallization of agility back to basics. In: ICSOFT 2006, vol. (2), pp. 121–126 (2006)

[8] Wikipedia: OODA Loop, http://en.wikipedia.org/wiki/OODA_Loop (26.5.2010)
[9] Stelzmann, E., Spork, G., Kreiner, C., Messnarz, R., Koenig, F.: Practical Use of Agile Methods within Systems Engineering. In: International Spice Days, Stuttgart, June 21-23 (2010)
[10] Lock, G.: Project Management. Gower Publishing Limited, Hampshire (2007)
[11] Haberfellner, R., et al.: Systems Engineering: Methodik und Praxis. In: Daenzer, W.F., Huber, F. (eds.) 11th edn. (2002)
[12] Fricke, E., Schulz, A.P.: Design for changeability (dfc): Principles to enable changes in systems throughout their entire lifecycle. Systems Engineering Journal 8(4) (2005)
[13] Dove, R.: Response Ability: The Language, Structure, and Culture of the Agile Enterprise. Wiley, Chichester (2001)
[14] Brooks, F.P.: The Mythical Man-Month: Essays on Software Engineering, 2nd edn. Addison-Wesley Professional, Reading (1995)