TECHNISCHE UNIVERSITÄT GRAZ
FAKULTÄT FÜR MASCHINENBAU UND WIRTSCHAFTSWISSENSCHAFTEN

Dissertation

# Agile Systems Engineering

## Eine Methodik zum besseren Umgang mit Veränderungen bei der Entwicklung komplexer Systeme

**Ernst Stefan Stelzmann**

**06.10.2011**

Begutachter:

Em. Univ.-Prof. Dipl.-Ing. Dr.sc.techn. Reinhard Haberfellner
Institut für Unternehmungsführung und Organisation
Technische Universität Graz

Univ.-Prof. Dipl.-Ing. Dr. techn. Hermann Kaindl
Institut für Computertechnik
Technische Universität Wien

Priv.-Doz. Dr.rer.pol. Peter Nagel
ETH Zürich

## Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Graz, am …….  ……..……..

(Ernst Stelzmann)

# Vorwort

An dieser Stelle soll kurz das Zustandekommen dieser Dissertation beschrieben werden. Das Interessante daran ist, dass vor Beginn der Arbeit sowohl der Erstbegutachter, als auch der Autor, unabhängig voneinander mit der Themenstellung in Berührung kamen. Prof. Haberfellner wurde im Rahmen einer Konferenz mit dem Schlagwort „Agile Systems Engineering“ konfrontiert. Nach einer Diskussion mit dem, ebenfalls bei dieser Konferenz anwesenden, MIT Professor Olivier de Weck beschlossen beide das Thema gemeinsam genauer zu beleuchten. Im Detail sollte dabei auf die unterschiedlichen Bedeutungen eingegangen werden, die das Schlagwort haben kann. Es kann nämlich einen agilen Systems Engineering Prozess bezeichnen aber auch die Entwicklung von agilen Systemen. Daraus entstand der Artikel „Agile SYSTEMS ENGINEERING vs. AGILE SYSTEMS Engineering“, der beim INCOSE Symposium 2005 in Rochester, NY, USA präsentiert wurde (Haberfellner, et al., 2005).

Der Autor der vorliegenden Arbeit wurde im Rahmen seiner Tätigkeit, bei einem Entwickler von Systemkomponenten in der Automobilindustrie, in praktischer Form mit dem Thema konfrontiert. Dort erlebte er den großen Einfluss, den Veränderungen auf die Entwicklungstätigkeit haben. Außerdem erkannte er die große Herausforderung, die häufige Veränderungen für die Standardisierung von Entwicklungsprozessen darstellen, welche in der Automobilindustrie ebenso ein aktuelles Thema ist. So wurde der Autor zwar nicht explizit mit dem Ausdruck „Agile Systems Engineering“ konfrontiert, aber mit einer Problemstellung im Systems Engineering für welche Agilität eine Lösungsmöglichkeit bieten soll.

Das gemeinsame große Interesse von Autor und Begutachter, diese Themenstellung wissenschaftlich aufzuarbeiten, sorgte nicht nur für eine schnelle Themenvereinbarung zu Beginn der Dissertation, sondern garantierte auch eine spannende Forschungsarbeit.

# Danksagung

Ich möchte mich bei allen Personen bedanken, die mich während meines Doktorstudiums in wissenschaftlichen aber auch in menschlichen Belangen unterstützt haben. Besonders bei Herrn Prof. Dr.sc.techn Reinhard Haberfellner, der mir die Möglichkeit gab, diese Dissertation zu verfassen, aber auch bei meinen Eltern und Großeltern, bei meinen Kollegen und bei meinen Interviewpartnern, speziell bei den Mitgliedern der S²QI Forschungsgruppe, die nun den Namen „Soqrates, Agiler Arbeitskreis“ trägt. Weiters gilt mein Dank meinem Zweitbetreuer Herrn Prof. Dr.techn. Hermann Kaindl und Herrn Priv.-Doz. Dr.rer.pol Peter Nagel, der meine Arbeit ebenso begutachtet hat.

# Zusammenfassung

Ausgangssituation für diese Dissertation ist eine Welt mit schnellen Veränderungen, großen Unsicherheiten und größer werdender Komplexität, in der dennoch erfolgreich Systeme entwickelt werden sollen. Unter dem Schlagwort „Agilität“ sind Prinzipien und Methoden bekannt geworden, die Lösungsmöglichkeiten für solch eine Umgebung bieten könnten.

Diese Dissertation erforscht das Prinzip der Agilität und überprüft seine Tauglichkeit zur Beherrschung der beschriebenen Umgebung im Systems Engineering. Der Begriff „Agilität“ wird dazu im Allgemeinen analysiert aber auch in seiner durch die Softwareentwicklung geprägten Bedeutung für ingenieurstechnische Disziplinen. Mit den Erkenntnissen aus der Literatur aber auch aus Interviews und Workshops mit Expertengruppen wird damit nach der qualitativen Forschungsmethode der „Grounded Theory“ das Konzept eines agilen Systems Engineerings gestaltet.

Für dieses Agile Systems Engineering wird ermittelt, in welchen Anwendungsfällen es Sinn macht, bzw. wann durch seine Verwendung ein Vorteil bei Systems Engineering Tätigkeiten erzielt werden kann. Es wird auch überprüft, welche Rahmenbedingungen ein agiles Vorgehen unmöglich oder schwer durchführbar machen. Für den Fall von Systems Engineering Projekten die nach Agilität verlangen, diese aber nicht oder nur schwer ermöglichen, wird nach Strategien gesucht, welche das Prinzip der Agilität zumindest ansatzweise ersetzen können, um auch hier ein Agile Systems Engineering durchführen zu können.

Zur praktischen Umsetzung von Agilität werden Prinzipien und Praktiken aufgezeigt, die bei Mitarbeitern, Prozessen, Produkten, in der Organisation und beim Kundenkontakt Agilität ermöglichen und fördern sollen. Schließlich wird anhand eines Fallbeispiels aufgezeigt, wie diese Prinzipien und Praktiken zusammenhängen und angewandt werden können. Nachdem es keine Standardlösung für ein möglichst agiles Systems Engineering gibt, wird dieses Fallbeispiel für unterschiedliche agile Vorgehensweisen aufgezeigt.

# Abstract

We live in a world of fast changes, big uncertainties and increasing complexity. Nevertheless we intend to develop successful systems. “Agility” is used as a buzzword for a collection of principles and methods that ought to provide solutions for this environment.

This dissertation investigates the principle of agility and its capability to master Systems Engineering in the described environment. The concept of agility is therefore analyzed in general but also within engineering disciplines, especially within software development where it is applied frequently. Research is done by literature research, interviews and workshops with a panel of experts. Following the Grounded Theory method an agile Systems Engineering approach is elaborated.

Areas of application for this Agile Systems Engineering are identified, as well as conditions that make agile approaches difficult or unfeasible. So it is shown in which context agile development is advantageous. In cases where agility is needed but unfeasible, strategies are proposed that can substitute the principle of agile development to some extent.

For practical realization of agility principles and practices are shown. They should enable and enhance agility of personnel, processes, products, organization and customer relations. With the help of a case study it is shown how these principles and practices are linked up and can be applied. Since no ideal agile Systems Engineering approach was found, this case study is described for different agile action models.

# Inhaltsübersicht

# Inhaltsverzeichnis

# 1 Einleitung

*„Das einzig Unveränderliche ist die Veränderung“*. Dieses Laotse zugeschriebene Zitat stammt bereits aus dem 6. Jahrhundert vor Christus. Betrachtet man den Fortschritt in Technik, Wissenschaft und Wirtschaft bis zum heutigen Tage, gilt es nun wohl mehr denn je. Man muss der Änderungsrate in diesen Gebieten sogar eine ständig steigende Tendenz zugestehen. Trotzdem werden sehr viele Dinge in diesen Bereichen als unveränderlich angesehen. Pläne werden darin in größter Detaillierung für eine weit entfernte Zukunft erstellt, um nach Ablauf der Zeit zu erkennen, dass die Welt für die geplant wurde, eine andere geworden ist. Z.B. werden entwickelte Produkte in anderen Stückzahlen oder mit anderer Funktionalität benötigt, als geplant oder die Entwicklung neuer Technologien hat sie vollkommen obsolet werden lassen. Neue Gesetze oder Umweltvorschriften erlauben plötzlich nicht mehr den Einsatz eines in jahrelanger Arbeit entwickelten Systems. Die Unternehmung, deren Strategie und Organisation mühsam entwickelt wurde, sieht sich plötzlich gänzlich anderen Herausforderungen gegenüber gestellt, als ursprünglich angenommen wurde.

Diese Dissertation beschäftigt sich mit der Disziplin „Systems Engineering“ und versucht sich dabei mit den Veränderungen, die in unserer Welt vorkommen können, auseinander zu setzen. Systems Engineering (SE) wird dabei als die Disziplin betrachtet, die sich mit ganzheitlicher System- oder Produktentwicklung beschäftigt, die in Projektform abgewickelt wird und gedanklich alle Lebenszyklusphasen des Produkts/Systems berücksichtigt. Es soll untersucht werden, wie die Betreiber dieser Disziplin auf Veränderungen reagieren, welche Lösungsprinzipien zu deren Bewältigung existieren und ob es einen allgemeingültige Systems Engineering Ansatz zur Beherrschung dieser Problemstellung gibt. Die Veränderungen, die betrachtet werden sollen, können verschiedene Ursachen und Auswirkungen haben. Im Groben lassen sich die Einflüsse, die zum vorherrschenden, stark dynamischen Marktumfeld führen, durch die in Abb. 1 dargestellten Faktoren beschreiben.

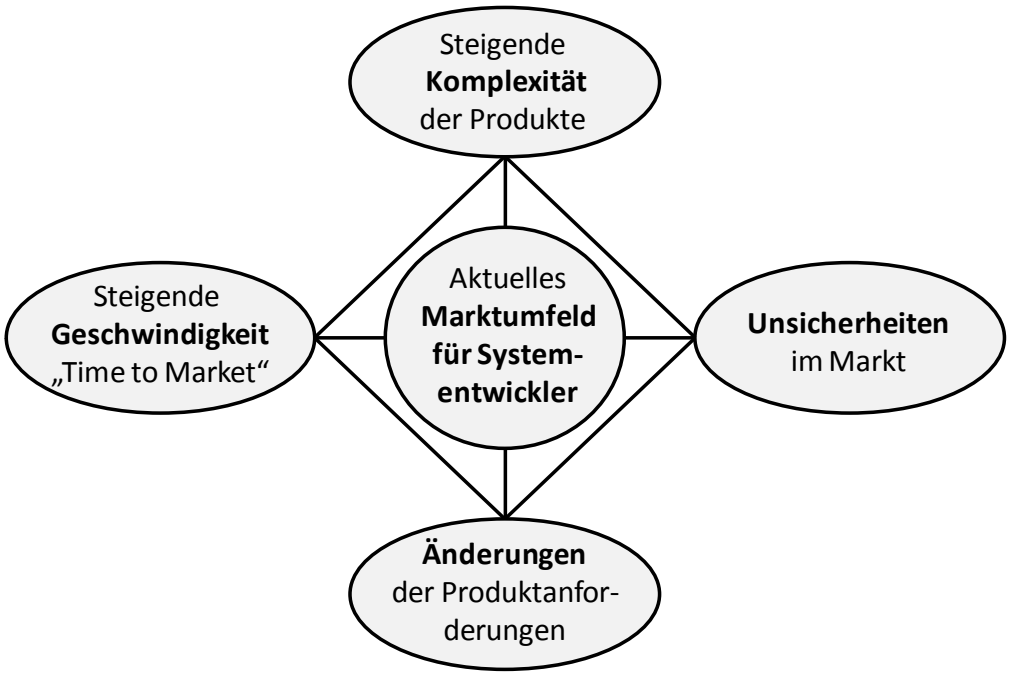


**Abb. 1: Einflüsse auf das Marktumfeld für Systementwickler**

In Systems Engineering naheliegenden Disziplinen, vor allem der Softwareentwicklung, sind in letzter Zeit Prinzipien und Methoden populär geworden, die unter dem Überbegriff „Agilität“ zusammengefasst werden können und die vorteilhaft zur besseren Beherrschung einer solchen Umwelt erscheinen. In dieser Arbeit soll nun untersucht werden, ob und wie Agilität auch im Systems Engineering Anwendung finden kann.

## 1.1 Problemstellung und Ziele

Die in Abb. 1 dargestellten vier Umgebungsfaktoren stehen alle in Zusammenhang und bilden gemeinsam die Ursache für die in dieser Arbeit betrachteten Veränderungen im Systems Engineering. Die Komplexität der Produkte ist die Ursache dafür, dass das Verhalten des fertigen Systems oft erst sehr spät in der Entwicklungsphase eruiert werden kann. Wichtige Informationen zum Systemverhalten kommen erst durch Tests am fertigen System ans Tageslicht. Entspricht das Verhalten nicht den Wünschen des Kunden oder Anwenders sind Änderungen durchzuführen. Die Komplexität des Systems kann, vor allem bei hohem Innovationsgrad, auch dazu führen, dass der Kunde beim Start der Entwicklung seine Anforderungen überhaupt noch nicht in allen Details kennt. Auch das kann dazu führen, dass Änderungen notwendig sind, sobald der Kunde durch einen erweiterten Informationsstand während der Entwicklung seine Anforderungen genauer spezifizieren kann.

Durch den Wunsch nach kurzen Entwicklungszeiten, um Produkte möglichst schnell auf den Markt zu bringen, entsteht auch eine mögliche Ursache für Änderungen. Nämlich dann, wenn aufgrund straffer Terminpläne wenig Zeit in die Formulierung oder Analyse der Anforderungen investiert wurde und erst in einer späteren Entwicklungsphase der Unterschied zwischen den angenommenen und den eigentlich gewünschten Anforderungen sichtbar wird. Bei hoher Komplexität der Produkte wird dieses Wirkprinzip noch verstärkt, da mehr Zeit zur Definierung und Analyse der Anforderungen notwendig ist.

Eine kürzere Entwicklungszeit hat bezüglich Veränderungen aber auch eine positive Wirkung. Es bleibt dem Markt nämlich schlichtweg weniger Zeit, um sich zu verändern, bevor die Entwicklung abgeschlossen ist. Ansonsten sind die Unsicherheiten im Markt eine wesentliche Quelle für Veränderungen. Neue Technologien, Gesetzesänderungen, wirtschaftliche Rahmenbedingungen und viele andere Dinge können dazu führen, dass sich die Wünsche des Marktes mehr oder weniger schnell und radikal verändern.

Anforderungen können vom Kunden aber auch außerhalb der eben beschriebenen Zusammenhänge geändert werden, z.B. weil sich seine Bedürfnisse geändert haben. Weitere Veränderungen können aus Fehlern resultieren oder weil sich Kunden oder Lieferanten ändern oder weil wichtige Mitarbeiter das Entwicklungsteam verlassen. Sie können auch zustande kommen, wenn z.B. Stückzahländerungen andere Fertigungstechnologien wirtschaftlicher werden lassen.

Die negative Auswirkung von Änderungen auf den Erfolg von Projekten ist unbestritten und empirisch nachgewiesen (Dvir, et al., 2004), (Maierhofer, et al., 2010 S. 66). Ebenso gelten Änderungen in späten Projektphasen als aufwändiger, als welche in frühen Projektphasen (Lock, 2003 S. 543), wie in Abb. 2 dargestellt ist. Aus diesen Gründen sind all die oben genannten Veränderungen eine Problemstellung für das Systems Engineering, welche in dieser Dissertation behandelt werden soll.

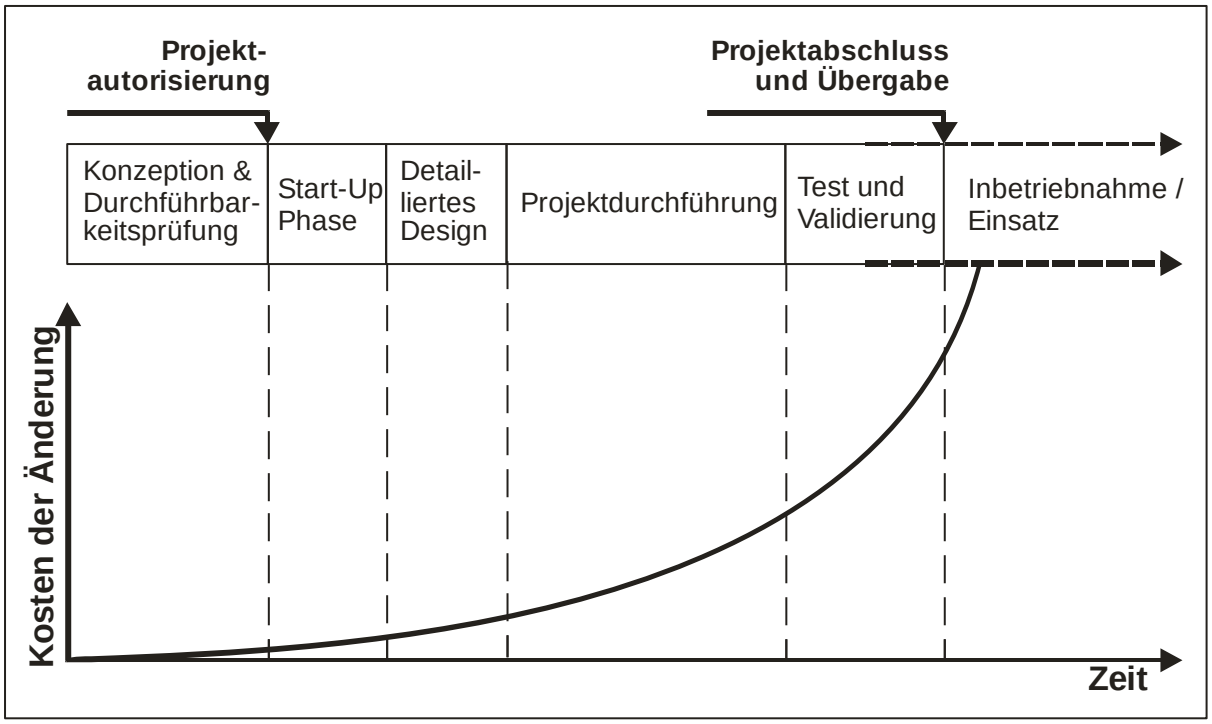


**Abb. 2: Änderungskostenkurve (Maierhofer, 2009 S. 12)**

In einer dem Systems Engineering naheliegenden Disziplin, der Softwareentwicklung, wurden zur Bewältigung dieser Problemstellung schon sogenannte „Agile Methoden" geschaffen. Mit diesen sollen Veränderungen beherrscht werden, egal wie stark und wie häufig diese auftreten oder wie vorhersehbar oder unvorhersehbar diese sind (Kettunen, et al., 2005). Ähnlich der später angeführten Definition von Systems Engineering, verlangen diese Methoden dabei auch nach Erfolgskriterien, wie sie für Entwicklungsmethoden üblich sind, wie Effektivität, Effizienz und Vorhersehbarkeit erfolgreicher Entwicklungsergebnisse (eine genauere Beschreibung erfolgt in Kapitel 2.2).

Ähnlich wie in der agilen Softwareentwicklung soll in dieser Arbeit für Systems Engineering eine Herangehensweise an die Problemstellung der Veränderungen gefunden werden, ohne die üblichen Erfolgskriterien außer Acht zu lassen.

Damit soll „Agile Systems Engineering", gleich wie das Ziel dieser Arbeit, die Antwort auf die Frage sein: „***Wie kann man in Umgebungen mit starken und häufigen, vorhersehbaren und unvorhersehbaren Veränderungen, in effizienter und effektiver Weise, plan- und vorhersehbar, erfolgreiche Systeme entwickeln und herstellen?"***

## 1.2 Forschungsprozess

Wie in Abb. 3 ersichtlich ist, wurde als erstes die Problemstellung analysiert. Dies geschah nicht nur durch Literaturrecherche, sondern auch in Zusammenarbeit mit einem später noch beschriebenen Expertenpanel (siehe Kapitel 3.2). In dieser Phase wurden nicht nur das in Kapitel 1.1 umrissene Problemfeld analysiert und die Grundlagen ausgearbeitet. Sondern es wurden auch die zahlreichen, unterschiedlichen Definitionen der Agilität und auch Unterschiede im Verständnis des Wesens der

Agilität ergründet und für den Gebrauch in dieser Arbeit zusammengeführt. In der anschließenden Phase wurden die Ziele der Arbeit und die Forschungsfragen festgelegt. Danach wurde durch Literaturrecherche und empirische Erhebungen nach Lösungskonzepten (Prinzipien und Praktiken) für die Problemstellung dieser Arbeit gesucht. Gefundene Prinzipien und Praktiken wurden danach hinsichtlich ihrer Wirkungsweise im Umgang mit Veränderungen analysiert. Es wurden auch Vorteile, Nachteile, potentielle Anwendungsgebiete und möglichen Hindernisse für die Anwendung untersucht. Die Erkenntnisse aus der Analyse der Einzellösungen wurden anschließend zu einem „Agile Systems Engineering“ Ansatz kombiniert. Der Austausch mit Experten aus Praxis und Forschung fand in fast allen Phasen dieser Arbeit statt, wie in Kapitel 3 näher ausgeführt wird. Dies sollte der Absicherung von Wissenschaftlichkeit aber auch praktischer Anwendbarkeit dienen.

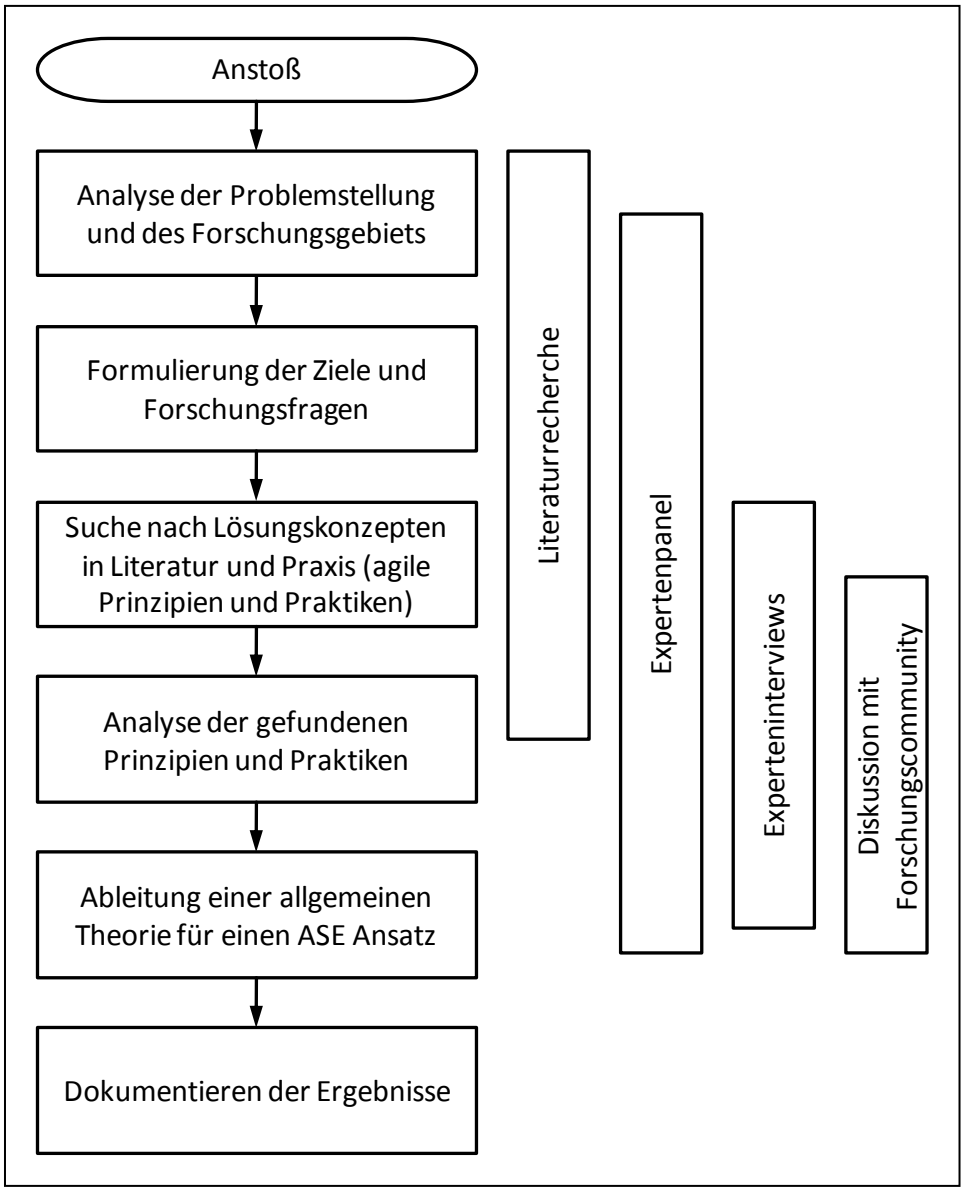


**Abb. 3: Forschungsprozess der Dissertation**

## 1.3 Forschungsfragen

Der Begriff „Agilität" soll hier im Sinne einer agilen Systementwicklung verstanden werden. Im Zusammenhang damit wurden die folgenden fünf Forschungsfragen formuliert:

1. **Was ist das Wesen der Agilität?**
   Diese Forschungsfrage soll sich mit der Definition der Agilität und ihrem Wirkprinzip beschäftigen, angefangen mit einer allgemeinen Sichtweise bis hin zu agiler Entwicklung und zu einer Sichtweise der Agilität, wie sie für Systems Engineering brauchbar ist.
2. **Welche Vorgehensweisen können die Agilität steigern?**
   Mit dieser Frage sollen Prinzipien, Praktiken und Methoden gefunden werden, welche bei praktischer Problemlösung im SE angewandt werden können, um die Agilität zu steigern.
3. **Wann besteht Bedarf an Agilität?**
   Diese Frage soll abklären, wann Agilität im Systems Engineering notwendig oder zumindest nützlich ist. Es sollen Umstände aufgezeigt werden, unter welchen Agilität Sinn macht.
4. **Was sind Voraussetzungen für Agilität?**
   Welche Gründe sprechen gegen Agilität? Wann ist Agilität nicht möglich oder nicht sinnvoll? Gibt es Umstände, die Agilität im Systems Engineering oder den Einsatz von agilen Methoden und Praktiken verhindern?
5. **Wie kann ein „Agile Systems Engineering" Ansatz aussehen?**
   Diese Frage soll zu einem umfassenden agilen Ansatz führen, der alle Facetten der Problemstellung und der Anwendung agiler Vorgehensweisen berücksichtigt und eine konkrete Hilfestellung bietet, um die Agilität des Systems Engineerings zu erhöhen.

## 1.4 Struktur der Arbeit

In Kapitel 2 werden die in der Literatur gefundenen Grundlagen für diese Arbeit gezeigt und damit die Forschungsumgebung dieser Dissertation skizziert. Zuerst wird die Disziplin des Systems Engineerings dargestellt und erklärt, in welchem Bereich darin diese Dissertation anzusiedeln ist. Danach werden die agilen Softwareentwicklungsmethoden vorgestellt, die als primäre Ideenlieferanten für agile Prinzipien und Praktiken gedient haben. Schließlich werden auch noch andere Fachgebiete erwähnt, in denen schon nach Agilität geforscht wurde. Dabei werden auch erste Vorarbeiten im Bereich des Agile Systems Engineerings vorgestellt.

In Kapitel 3 wird das Forschungsdesign detaillierter dargestellt und die gewählte Forschungsmethode erklärt. Es werden auch die für die empirische Studie gewählten Firmen und Gesprächspartner vorgestellt.

Kapitel 4 analysiert das Wesen der Agilität und beantwortet damit Forschungsfrage 1. Es wird dazu aufgezeigt, wie sie in anderen Disziplinen definiert und angewandt wird. Ergebnisse dieses Kapitel sind eine allgemeine Definition der Agilität und die Charakterisierung und Definition für das „Agile Systems Engineering" (ASE) dieser Arbeit. Außerdem wird untersucht, welches Funktionsprinzip im Allgemeinen unter „agiler Entwicklung" verstanden wird.

Kapitel 5 ist eine Zusammenstellung von einzelnen Prinzipien und Praktiken, welche zur Steigerung der Agilität im SE angewandt werden können. Dieses Kapitel ist die Antwort auf die zweite Forschungsfrage.

Die gefundenen Vorgehensweisen werden darin ausführlich auf ihren Nutzen hinsichtlich Agilität, Anwendungsbereiche, Vor- und Nachteile analysiert.

Kapitel 6 soll Anwendungsgebiete für ASE darstellen. Dazu soll aufgezeigt werden, wann es einer Steigerung der Agilität bedarf bzw. unter welchen Bedingungen man sich über Agilität Gedanken machen sollte. Dafür werden auch die verschiedenen Veränderungen analysiert, die Systems Engineering betreffen können. Weiters stellt Kapitel 6 dar, warum Agilität im Systems Engineering nicht immer bzw. nicht leicht umzusetzen ist und zeigt, unter welchen Umständen bestimmte Prinzipien und Praktiken im Systems Engineering nicht umgesetzt werden können. Damit beantwortet Kapitel 6 die Forschungsfragen 3 und 4. Außerdem werden Möglichkeiten diskutiert, wie Hindernisse für Agilität aus dem Weg geräumt werden können, wenn ein Bedarf besteht.

Kapitel 7 beantwortet die fünfte Forschungsfrage bzw. komplettiert sie durch eine Vorgehensmethodik, welche die Zusammenhänge des Agile Systems Engineering Ansatzes darstellt und Hinweise zu seiner Anwendung gibt. Dazu wird aufgezeigt in welchen Bereichen innerhalb von Unternehmungen Maßnahmen gesetzt werden können um die Agilität im Systems Engineering zu unterstützen und wie diese zusammenhängen. Es wird auch dargestellt, in welchem Kontext die in Kapitel 5 beschriebenen Maßnahmen eingesetzt werden sollen und es werden Vorgehensmodelle für ASE vorgestellt.

Kapitel 8 zeigt anhand eines Fallbeispiels wie Agile Systems Engineering in der Praxis umgesetzt werden kann. Um die unterschiedlichen Anwendungsfälle abzudecken wird das Fallbeispiel für die in Kapitel 7 beschriebenen unterschiedlichen Vorgehensmodelle beschrieben.

Kapitel 9 ist ein Resümee und ein Ausblick auf weitere Forschungstätigkeit.

## 1.5 Begriffsverwendung

Die wichtigsten Begriffe dieser Arbeit, wie „Agilität“, „Systems Engineering“ und „Agile Software Development“ werden in den folgenden Kapiteln ausführlich erforscht und definiert. Hier sollen Begriffe definiert werden, die im Sprachgebrauch der ingenieurwissenschaftlichen Disziplinen oft unterschiedlich gebraucht werden oder unscharf definiert sind. Im weiteren Verlauf der Arbeiten gelten Aussagen, die für diese Begriffe getätigt werden, nur bezüglich der hier gemachten Definitionen. Diese orientieren sich nicht unbedingt an der gebräuchlichsten Verwendung, sondern enthalten oft Beschränkungen, die gezielt für diese Arbeit gemacht wurden.

**System:** Es wurde versucht einen Ansatz zu schaffen, der für Systeme im Allgemeinen Gültigkeit hat. Im Rahmen der empirischen Erhebungen standen aber keine Gesprächspartner für nicht technische Systeme zur Verfügung, weshalb aus Gründen der wissenschaftlichen Beweisbarkeit hier eine Einschränkung vorgenommen werden muss. Der Begriff „System“ wird in dieser Arbeit synonym mit dem Begriff „Produkt“ verwendet und bezeichnet das für einen Kunden zu entwickelnde technische Objekt. Dieses besteht, im Sinne eines Systems, immer aus mehreren, interagierenden Elementen, also z.B. aus mechanischer Hardware, elektronischer Hardware und Software. Ein Produkt, welches nur aus Software besteht, soll explizit ausgeschlossen werden. Die agile Entwicklung von Software alleine, wurde nämlich unter dem Begriff „Agile Software Development“ bereits ausführlich erforscht (siehe Kapitel 2.2). Nicht technische Systeme werden ebenso ausgeschlossen, da die Recherchen nur für technische durchgeführt wurden. Es ist aber davon auszugehen, dass wesentliche Teile dieser Arbeit auch in der Entwicklung von

nicht technischen Systemen Anwendung finden können. Deshalb wurde auch darauf geachtet die Begrifflichkeiten innerhalb des Ansatzes so allgemeingültig wie möglich zu halten.

**Hardware (HW):** Dieser Begriff wird nicht nur im Sinne von Computer-Hardware, sondern für alle materiellen Komponenten eines Systems verwendet.

**Systementwickler:** Mit diesem Begriff wird in dieser Dissertation eine Firma benannt, die für Kunden Systeme entwickelt. Dabei kann es sich auch um Subsysteme handeln, die dann vom Kunden in ein größeres System integriert werden. Außerdem kann die Firma ihrerseits wieder Komponenten von Lieferanten beziehen. Entscheidend ist, dass die Firma die Entwicklungsverantwortung für ein Objekt trägt, dass der oben gemachten Definition eines Systems entspricht.

**Systementwicklung:** Damit ist das konkrete und praktische Entwickeln eines Systems gemeint. Dieses ist nicht gleichzusetzen mit dem Begriff des „Systems Engineerings". Die Systementwicklung ist zwar ein Teil des SE, kann aber auch z.B. ohne die Anwendung von Lebenszyklus- oder Vorgehensmodellen stattfinden, die innerhalb der Disziplin des Systems Engineerings bereitgestellt werden.

**Methodik:** Die Grundlage dieser Arbeit ist der BWI-Hall Ansatz (Haberfellner, et al., 2002), der eine traditionelle Systems Engineering Methodik darstellt (siehe Kapitel 2.1.5). Das Ergebnis dieser Arbeit soll ein Ansatz sein, mit dem im SE die Agilität erhöht werden kann. Dafür wird aber nicht nach einer einzelnen Vorgehensweise (z.B. ein konkretes Prozessmodell für die agile Entwicklung eines bestimmten Systems) gesucht, sondern nach einer umfassenden Herangehensweise an die Problemstellung. Der Ansatz soll auch dabei behilflich sein, den Bedarf an Agilität oder die Voraussetzungen für die Anwendung agiler Methoden oder Praktiken zu analysieren. Er soll dann mögliche Methoden, Prinzipien und Praktiken aufzeigen und die Auswahl erleichtern. Und er soll Hinweise zu deren Umsetzung liefern. Damit erweitert der Lösungsansatz dieser Arbeit den BWI-Hall Ansatz um Agilität und soll deshalb ebenso als Methodik verstanden werden. Welche Inhalte eine SE Methodik haben soll, wird in Kapitel 2.1.4 näher erläutert. Innerhalb einer „Methodik", welche als höchste Hierarchiestufe von Vorgehensweisen verstanden wird, können folgende weitere Vorgehensweisen umgesetzt werden:

**Methode:** Auf der zweiten Hierarchiestufe wird der Begriff „Methode" verwendet. Darunter werden bereits konkrete Vorgehensweisen verstanden, wie z.B. die agilen Softwareentwicklungsmethoden Scrum oder Extreme Programming (siehe Kapitel 2.2). Die gedanklich übergeordnete Methodik, kann die Verwendung unterschiedlicher Methoden empfehlen. Eine Methode kann wiederum aus einer beliebigen Anzahl von Prinzipien und Praktiken bestehen, welche in dieser Arbeit gedanklich auf die dritte Hierarchiestufe gesetzt werden.

**Prinzip:** Wird von agilen Prinzipien gesprochen, so sind Ideen, Grundsätze oder theoretische Vorgehensweise gemeint, welche die Agilität steigern sollen. Die Anwendung bzw. konkrete Umsetzung eines Prinzips stellt eine Praktik dar.

**Praktik:** Eine agile Praktik ist somit eine konkrete, praktische Vorgehensweise, durch deren Umsetzung die Agilität erhöht werden soll. In Kapitel 5 wird versucht alle Vorgehensweisen als Praktiken darzustellen. Ist dies nicht in einer allgemeinen Form möglich, sondern erst im Kontext eines speziellen Projekts oder einer Unternehmung, so wird lediglich das zugrundeliegende Prinzip dargestellt. Wird im Weiteren von „Praktiken für den ASE Ansatz" gesprochen, so sind nicht nur die so benannten Praktiken, sondern auch die Prinzipien gemeint, eben in einer konkreten und praktischen Ausprägungsform.

# 2 Grundlagen

Dieses Kapitel soll die inhaltlichen Grundlagen dieser Arbeit darstellen. Als erstes wird demnach das Systems Engineering betrachtet. Dieser Begriff, von dem es in der Literatur mehrere teilweise verschiedene Auffassungen gibt, wird hier als Disziplin verstanden, in der alle Herangehensweisen einen Platz haben. Damit wird auch versucht, Ordnung in das Begriffschaos zu bringen und einen Ansatz zu finden, mit dem alle Autoren übereinstimmen können. Speziell wird dann die traditionelle BWI-Hall Methodik (Haberfellner, et al., 2002) dargestellt, auf deren Basis dann der ASE Ansatz dieser Arbeit erstellt werden soll.

Des Weiteren werden in diesem Kapitel alle Disziplinen aufgezeigt, die sich schon mit Agilität beschäftigt haben. Dabei sticht eindeutig die Softwareentwicklung hervor, in der die Diskussion hinsichtlich Agilität große Bedeutung hat (Boehm, et al., 2006). Auch eine Übertragung agiler Prinzipien der Softwareentwicklung in den Bereich des Systems Engineerings wurde bereits von einigen Autoren angedacht (Wilson, et al., 2003), (Dove, 2005), (Turner, 2007), (Kaindl, et al., 2010).

## 2.1 Systems Engineering

Die Definition des International Council on Systems Engineering lautet: (INCOSE, 2007 S. 1.5):

*„Systems Engineering ist ein interdisziplinärer Ansatz und ein Mittel zur Realisierung erfolgreicher Systeme. Seine Schwerpunkte sind die Definierung von Kundenanforderungen und Funktionalitäten in einer frühen Phase der Entwicklung, die Dokumentierung der Anforderungen, dann die Design Synthese und schließlich die System Validierung. Das Problem wird dabei immer in seiner Gesamtheit betrachtet. Systems Engineering berücksichtigt sowohl die geschäftlichen als auch die technischen Anforderungen aller Kunden mit dem Ziel ein qualitativ hochwertiges Produkt zur Verfügung zu stellen, das alle Anforderungen erfüllt.“*

Andrew Sage führt in seinem Standardwerk „Introduction to Systems Engineering“ mehrere Definitionen auf und schlägt vor, SE als Managementtechnik zu betrachten, welche Planung, Entscheidung und Ressourcenbelegung unterstützen soll (Sage, et al., 2000 S. 8).

Auch Blanchard und Fabrycky legen sich nicht auf eine Definition fest, erkennen aber 4 gemeinsame Prinzipien aller SE Ansätze: Ein Top-Down Vorgehen, eine Betrachtung aller System-Lebenszyklus-Phasen, eine Definition der Anforderungen zu Entwicklungsbeginn und Interdisziplinarität (Blanchard, et al., 2006 S. 18f).

Im deutschsprachigen Standardwerk „Systems Engineering: Methodik und Praxis“ wird SE als ein Denkmodell und eine Vorgehensmethodik zur Lösung komplexer Probleme definiert (Haberfellner, et al., 2002).

Die Anwendung von Methoden aus anderen ingenieurwissenschaftlichen Disziplinen zur praktischen Entwicklung von Systemen, ohne Berücksichtigung von übergeordnete Prinzipien und Vorgehensweisen, kann nicht dem SE zugerechnet werden, wenn dieses als konkrete Vorgehensmethodik betrachtet wird. Betrachtet man es hingegen als Disziplin, können darin alle einzelnen Methoden und Vorgehensweisen ihren Platz finden. Sowohl die theoretische Erforschung, als auch die praktischen Umsetzung können

dann als Teil dieser Disziplin gesehen werden. Einen ersten Schritt zu einer strukturierten Betrachtungsweise liefert das Systems Engineering Handbook in der Version 2a (INCOSE, 2004 S. 21). Darin werden 4 Ebenen (Descriptive Levels) zur Darstellung der Tätigkeiten im Systems Engineering verwendet:

1. Lebenszyklus Phasen (z.B. Konzept Definition, Entwicklung, Produktion)
2. Programm Aktivitäten (z.B. Auftragsanalyse, Vorentwicklung, Detailentwicklung)
3. SE Prozess (z.B. Anforderungsanalyse, Architekturdefinition, Systemdesign)
4. Ingenieurfachbereiche (z.B. Software, Maschinenbau, Ergonomie)

In der Version 3.1 des SE Handbooks (INCOSE, 2007) wurde diese Art der Strukturierung der SE Inhalte aber wieder verworfen. Angelehnt an die Norm ISO/IEC 15288 (IEEE, 2005) findet hier eine Strukturierung der SE Inhalte nach System-Lebenszyklen statt, für welche jeweils Prozesse definiert sind. Weiters werden auch einzelne Methoden für die Verwendung innerhalb einzelner Lebenszyklen oder Prozesse vorgestellt. Die Anlehnung an diese Norm hat zwar eine bessere Struktur und Verständlichkeit gebracht, aber es gibt, wie im SE Handbook auch dargestellt wird, für unterschiedliche Produkte unterschiedliche Lebenszyklen (INCOSE, 2007 S. 3.5). Auch ist das Prozessmodell auf Basis des V-Modells nicht als allgemeingültig anzusehen. Deshalb kann auch das SE Handbook keine allumfassende Definition und Strukturierung des SE liefern.

In dieser Arbeit soll Systems Engineering allgemeingültiger als Disziplin betrachtet werden, die sich mit der Entwicklung von komplexen Systemen und der Lösung von komplexen Problemstellungen beschäftigt. Betrachtungsfelder sind sämtliche Lebenszyklus-Phasen der zu entwickelnden Systeme und die darin durchzuführenden Tätigkeiten. Dazu stehen Prozessmodelle und andere Methoden zur Verfügung, die entweder speziell für die Systementwicklung generiert wurden oder auch aus sämtlichen Fachgebieten stammen können, die einen Bezug zum zu entwickelnden System haben. Praktische Anwendungsgebiete der Systems Engineering Disziplin sind kaum begrenzt, wurden bisher aber vor allem in der Luft-, Raumfahrt-, Verteidigungs- und Transportindustrie und der Telekommunikation gefunden (INCOSE, 2004 S. 9ff).

Um die Disziplin des SE zu strukturieren kann man System-Lebenszyklus-Modelle und Entwicklungsprozess-Modelle hervorheben, da sie die wesentlichen Vorgehensweisen, jeweils für eine spezielle Art von Systemen darstellen. Ebenso kann man einzelne Methoden und Tools aufzählen und kategorisieren, die entweder speziell für die Systementwicklung oder auch für allgemeine Ingenieursdisziplinen entworfen wurden und die nun bei der Entwicklung eines Systems verwendet werden können. Zusätzlich zu diesen Dingen gibt es aber auch übergeordnete Denkmodelle und Vorgehensweisen im Systems Engineering.

Diese übergeordneten, generellen Prinzipien, die zu einem erfolgreichen Arbeiten innerhalb der Systems Engineering Disziplin führen, können unter dem Ausdruck „SE Methodik“ zusammengefasst werden. Die Einführung und Anerkennung der Kategorie „übergeordnete Methodik“ wurde deshalb gewählt, weil sie eine Klammer darstellt, welche die unterschiedlichen Vorstellungen über das Wesen des Systems Engineerings zusammenführt. Es ist auch die SE Methodik, mit der sich diese Arbeit im Weiteren beschäftigen wird. Der ASE Ansatz als Endergebnis soll eine SE Methodik sein, die alle Aspekte der Agilität berücksichtigt.

Eine grafische Darstellung mit allen Konstrukten, die in der SE Disziplin unterschieden werden sollen, befindet sich in Abb. 4. Im Weiteren sollen diese Konstrukte kurz beschrieben und mit Beispielen hinterlegt werden.

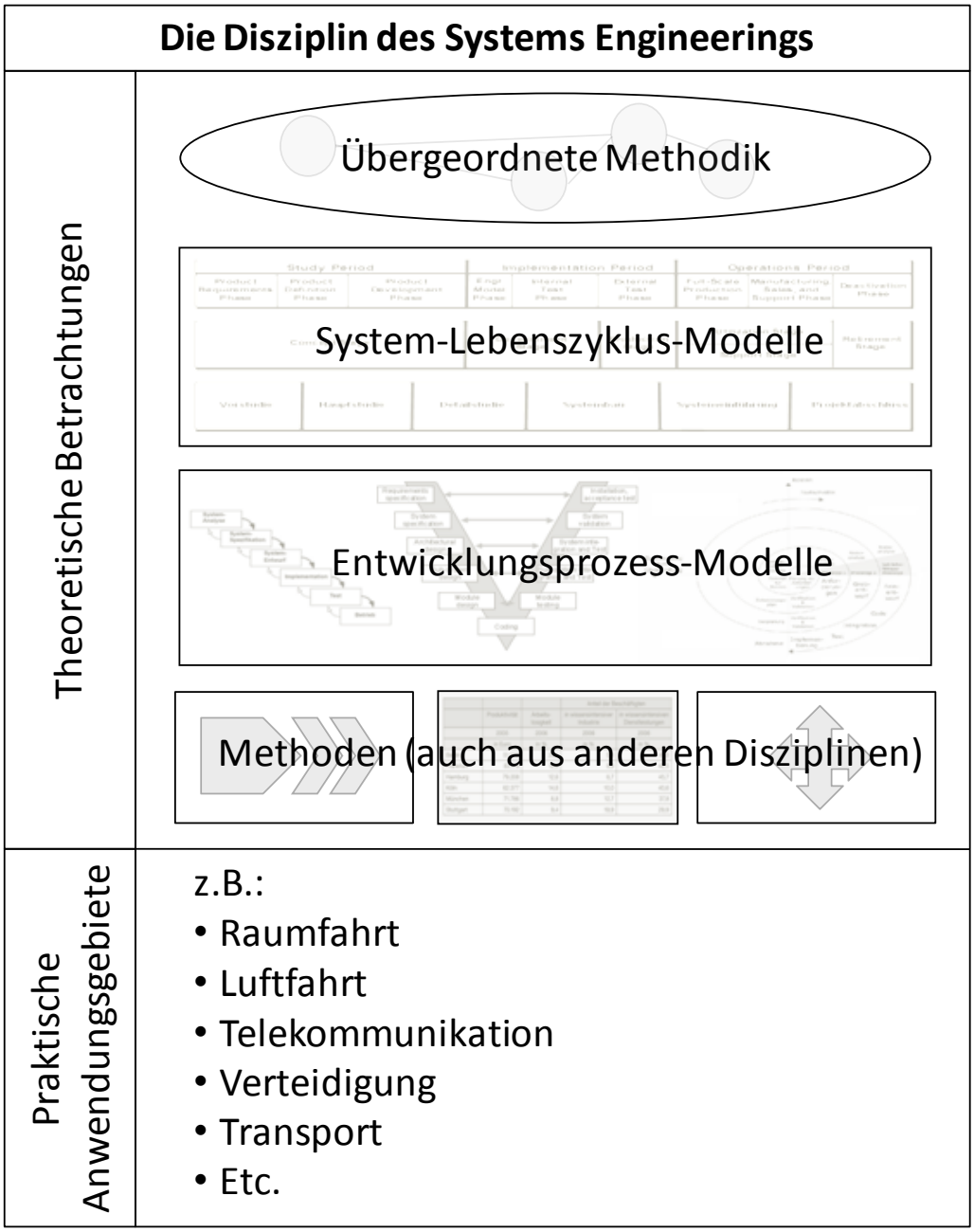


**Abb. 4: Systems Engineering als Disziplin**

### 2.1.1 System-Lebenszyklus-Modelle

Ein System-Lebenszyklus-Modell ist ein Konzept, welches alle Phasen, von der ersten Idee bis zu Stilllegung oder Recycling, aufzeigen soll. Es werden auch die gewöhnlichen Inhalte der Phasen dargestellt. Solche Modelle werden im Projektmanagement der Entwicklung für jeweils eine bestimmte Art/Klasse von Systemen genutzt.

Beispiele sind in Abb. 5 zu sehen, wobei das Lebenszyklusmodell der ISO/IEC 15288, seit der Version 3.1, auch im SE Handbook des INCOSE verwendet wird.

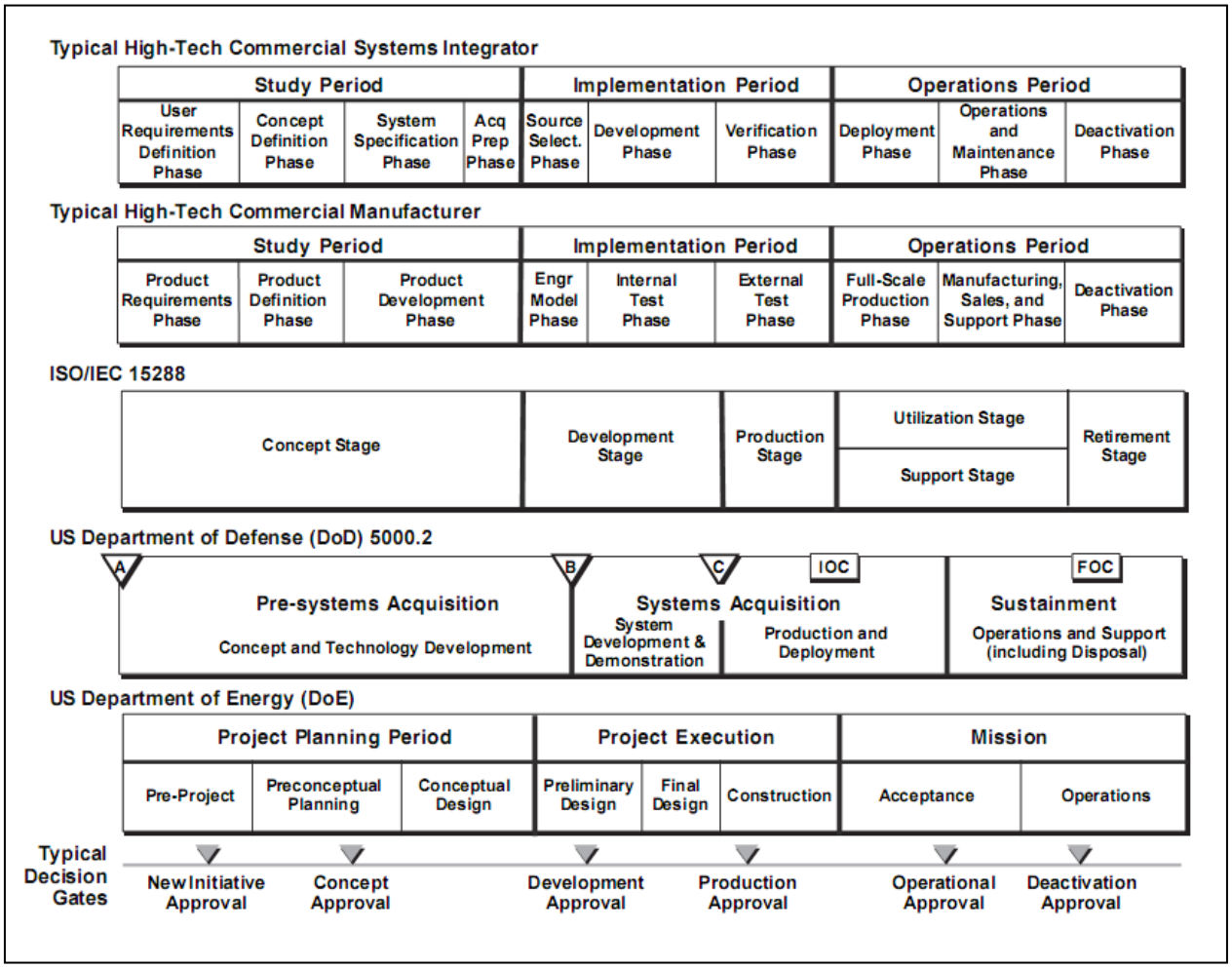


**Abb. 5: Vergleich verschiedener System-Lebenszyklusmodelle (INCOSE, 2007 S. 3.5)**

### 2.1.2 Entwicklungsprozess-Modelle

Diese empfehlen die Vorgehensschritte jeweils für die Entwicklung einer bestimmten Art oder Klasse von Systemen. Normalerweise werden darin auch die einzelnen Prozessschritte detailliert beschrieben, z.B. hinsichtlich Input, Output, durchzuführender Aktivitäten, Voraussetzungen und Kontrollelemente (INCOSE, 2007 S. 2.4ff).

Beispiele sind die Prozessmodelle der ISO/IEC 15288 (siehe Abb. 6), des Mil-Std-499, der IEEE 1220 oder auch agile Prozessmodelle wie Scrum (siehe Kapitel 2.2.1).

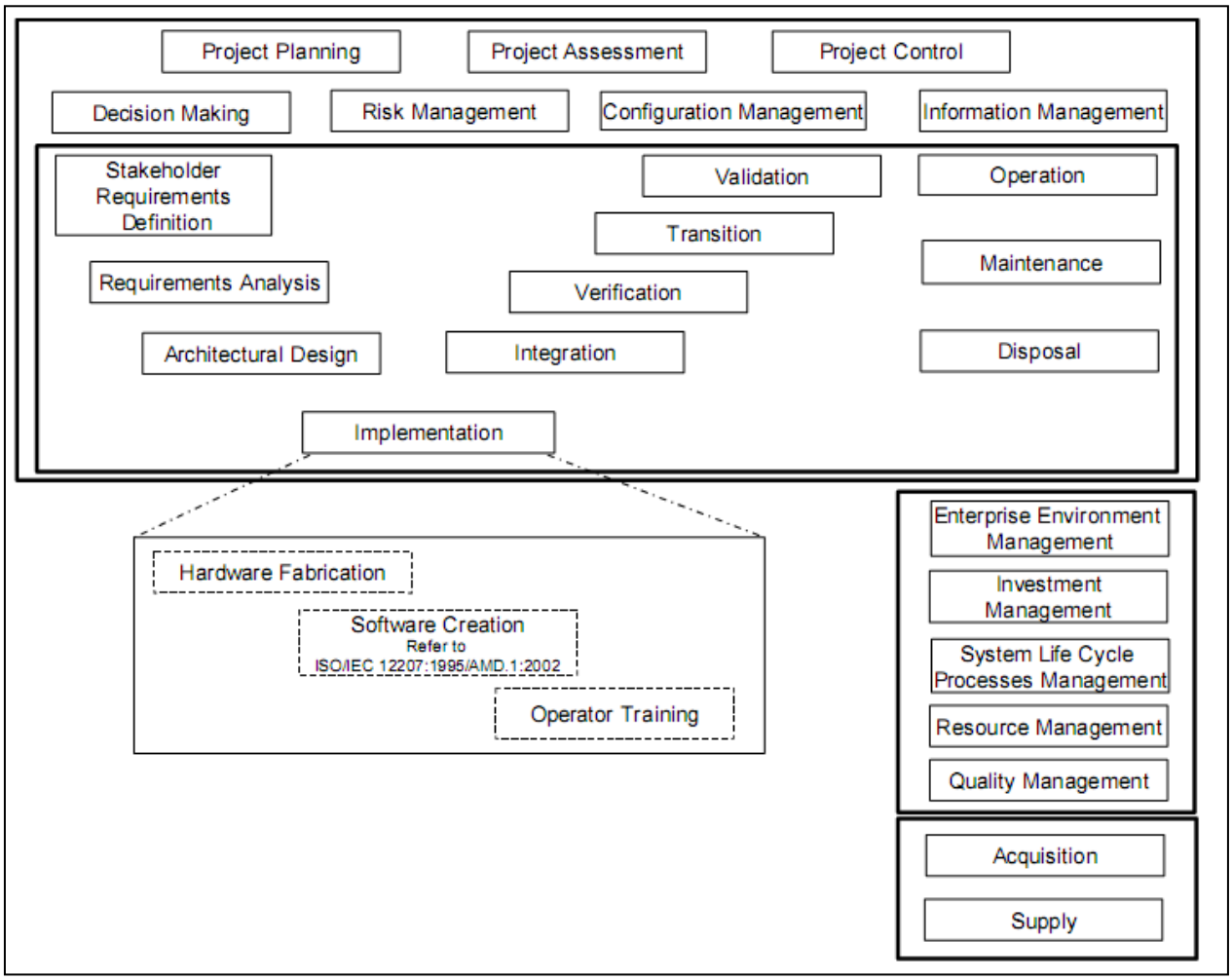


**Abb. 6: Beispiel für ein Prozessmodell zur Systementwicklung (IEEE, 2005)**

### 2.1.3 Methoden

Es gibt keine Methoden, die ausschließlich nur im Systems-Engineering verwendet werden können. Es gibt aber einige Methoden, wie z.B. die Design Structure Matrix[1] oder die Systems Modelling Language[2] die sehr eng in Zusammenhang mit Systems Engineering Tätigkeiten stehen. Eine Liste von gebräuchlichen Methoden ist in Haberfellner et al. (2002 S. 426ff) zu finden. Grundsätzlich kann und soll aber jede Methode verwendet werden, die zur erfolgreichen Entwicklung des Systems dient.

[1] Erklärung unter http://www.dsmweb.org
[2] Erklärung unter http://www.omgsysml.org

### 2.1.4 Übergeordnete Methodik

Eine übergeordnete Methodik soll ein ganzheitliches Konzept für erfolgreiches Systems Engineering sein. Sie soll sich mit allen Facetten des SE beschäftigen und einen Leitfaden für die erfolgreiche Systementwicklung geben. Die Verwendung der eben beschriebenen Lebenszyklus- oder Prozessmodelle, kann dabei ebenso ein Teil dieser Methodik sein, wie die Anwendung der folgenden Prinzipien:

- Das Systemdenken
- Top-Down Prinzip
- Prinzip der Einteilung in Phasen und Entwicklung eines generischen Phasenmodells
- Prinzip der konsequenten Betrachtung mehrerer Varianten
- Prinzip einer speziellen Vorgehensweise (Prozess) zur Lösung von einzelnen Problemen und Entwicklung eines generischen Problemlösungsprozesses
- Das Prinzip des strukturierten Arbeitens
- Betrachtung aller Lebenszyklen eines Systems in der Entwicklungsphase
- Prozessbetrachtung der Entwicklungsphase
- Prinzip der iterativen Entwicklung
- Prinzip des ständigen Einholens von Kundenfeedback
- Generelle Konstruktionsprinzipien für Systeme
- …

Als Methodik wird nicht nur die Summe dieser Prinzipien verstanden, sondern sie soll diese auch in den richtigen Kontext setzen und ihre Anwendung erklären.

Der BWI-Hall Ansatz (Haberfellner, et al., 2002) bietet ein solches gesamthaftes Konzept einer übergeordneten Methodik. Dieser entstand dadurch, dass der ETH-Professor Alfred Büchel im Jahr 1969 die Ideen von Arthur D. Hall aufgriff, die dieser in seinem Werk „A Methodoloy for Systems Engineering“ (Hall, 1962) veröffentlicht hatte. Eine Arbeitsgruppe am BWI Institut der ETH Zürich erweiterte diesen Ansatz und veröffentlichte ihn im Buch „Systems Engineering, Methodik und Praxis“, welches bereits in der 11. Auflage erschienen ist (Haberfellner, et al., 2002). Im nächsten Kapitel wird dieser Ansatz beschrieben, der auch als Grundlage dieser Arbeit verwendet und hinsichtlich Agilität weiterentwickelt werden soll.

### 2.1.5 SE Methodik nach BWI-Hall (Haberfellner, et al., 2002)

Im Zentrum dieses Ansatzes steht, wie in Abb. 7 dargestellt, der Problemlösungsprozess, dessen inhaltliche Komponente die Systemgestaltung und dessen organisatorische Komponente das Projektmanagement darstellt. Beide bedienen sich dazu dienlicher Techniken (Methoden und Praktiken). Als geistiger Überbau wird die SE Philosophie verstanden. Diese besteht einerseits aus dem Systemdenken, das es ermöglichen soll Systeme (komplexe Erscheinungen) besser zu verstehen und gestalten zu können, durch die Darstellung von Wirkungszusammenhängen und die Strukturierung und Abgrenzung von Ausgangssituation und Lösung. Andererseits besteht sie aus dem Vorgehensmodell, welches in Abb. 8 dargestellt wird.

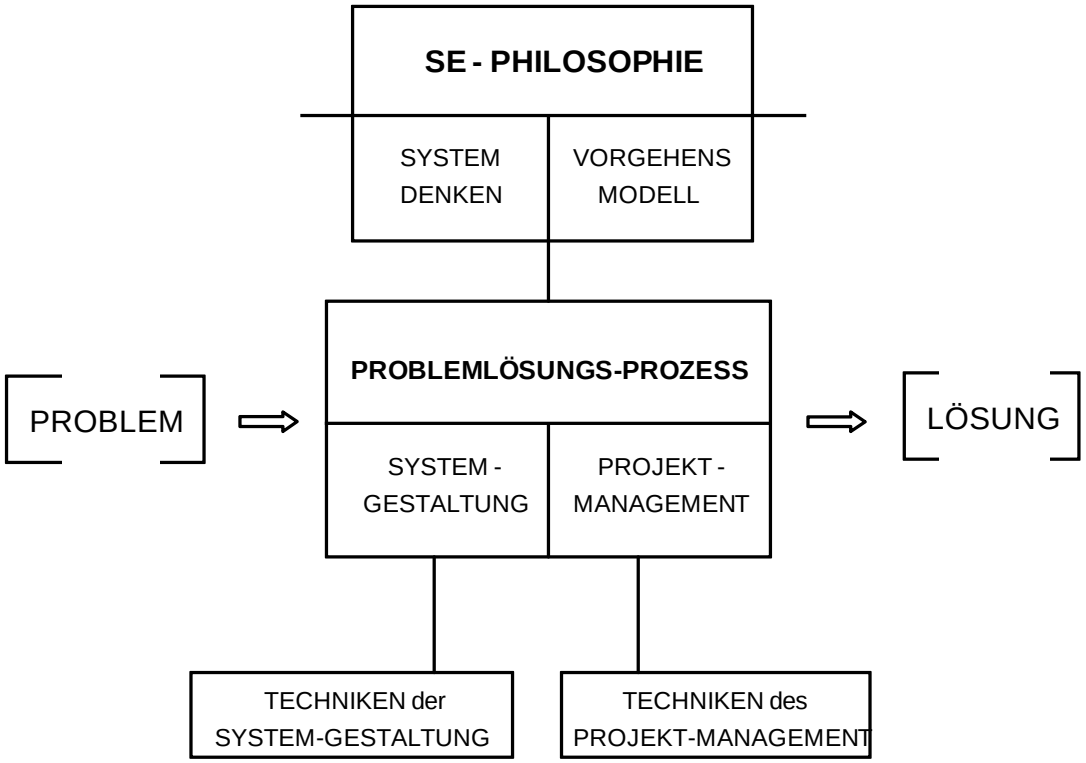


**Abb. 7: Komponenten des SE (Haberfellner, et al., 2002 S. XIX)**

Das Vorgehensmodell ist ein modular aufgebautes Konzept aus 4 Komponenten die miteinander verbunden werden können:

**Vom Groben zum Detail (Top-Down):** Bei diesem Prinzip werden Detailentscheidungen hinten angestellt. Das Betrachtungsfeld soll zuerst möglichst weit gefasst werden und dann schrittweise eingeengt. Die Lösung soll zuerst auf höherer Ebene als Prinzip oder Konzept erarbeitet werden, bevor die detaillierte Ausgestaltung beginnt.

**Variantenbildung:** Nachdem es unwahrscheinlich ist, dass immer die erste erdachte Lösung auch die beste ist, soll immer Überblick über mögliche Lösungsvarianten geschaffen werden, bevor man sich für eine entscheidet.

**Projektphasen:** Diese Idee soll dem Prinzip „Vom Groben zum Detail“ einen zeitlichen Rahmen geben und eine stufenweise Variantenausscheidung in den einzelnen dafür vorgesehenen Phasen ermöglichen. Das zeitliche Raster soll einen besseren Überblick über das Projekt schaffen und Zeitpunkte für Entscheidungen (auch Abbruchsmöglichkeiten) bieten. Wie in Abb. 8 dargestellt, sieht das BWI-Hall Modell dafür eine sequentielle Folge von Entwicklungsphasen und danach Realisierungsphasen vor.

**Problemlösungszyklus (PLZ):** Der PLZ stellt eine sich wiederholende Logik dar, die speziell in den Entwicklungsphasen aber auch zur Realisierung angewandt werden kann. Er beginnt mit einer Situationsanalyse und der Zielformulierung. Die Lösungssynthese stellt den kreativen Schritt der Lösungsfindung dar, die Lösungsanalyse wird, als kritischer Schritt, danach ausgeführt. Mit Hilfe der Bewertung soll die beste Lösungsvariante bestimmt werden, um danach die Entscheidung treffen zu können.

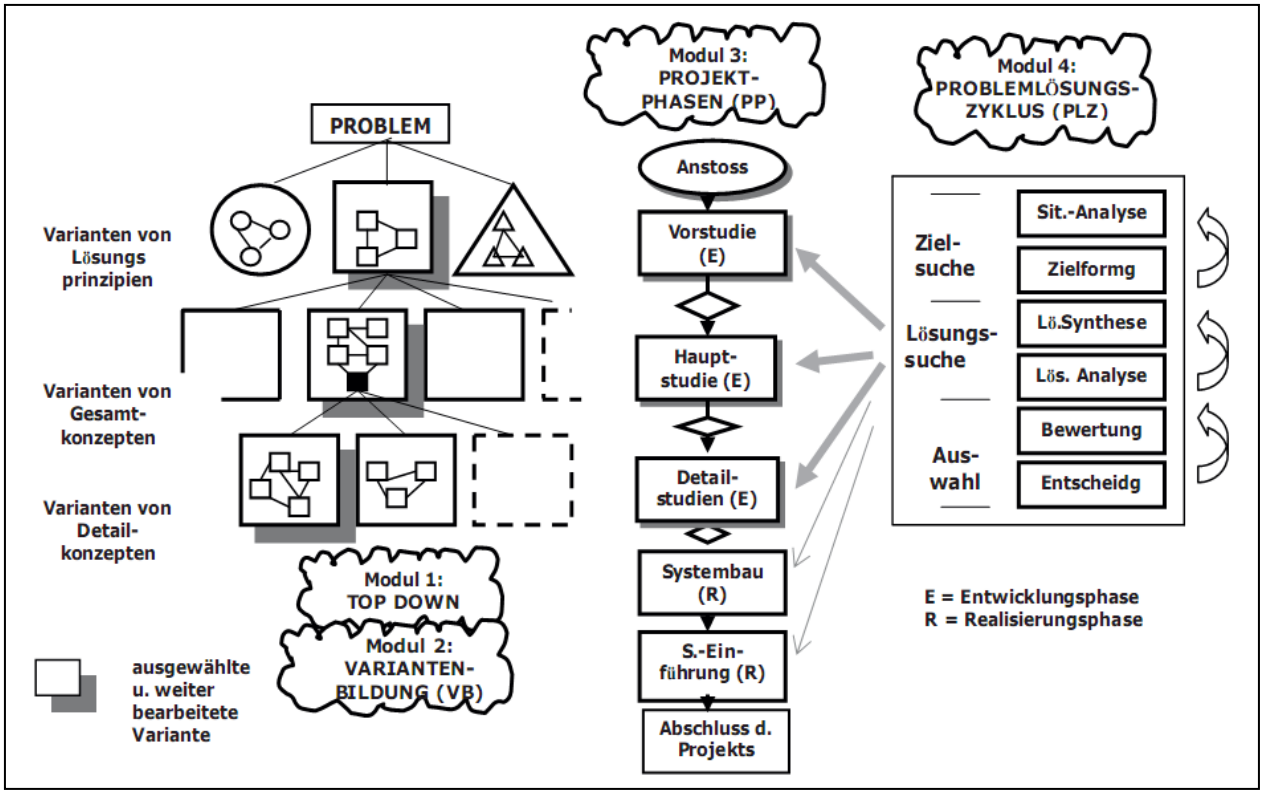


**Abb. 8: Das SE Vorgehensmodell (Haberfellner, et al., 2002 S. 29ff)**

Das Lösungskonzept der vorliegenden Arbeit soll, auf derselben Abstraktionsebene wie die BWI-Hall Methodik, sich um die in der Einleitung beschriebene Veränderungsproblematik kümmern. Es kann damit auch als „Agilitäts-Add-on“ zur BWI-Hall Methodik gesehen werden und soll dazu Ideen, welche zu einem besseren Umgang mit Veränderungen führen können, aufspüren, analysieren und gegebenenfalls einbringen. Die meisten der Ideen dazu wurden in der Softwareentwicklung gefunden.

## 2.2 Agile Software Development

Seit vielen Jahren werden in der Softwareentwicklung bereits Methoden propagiert, die eine schnelle Entwicklung mit früh verfügbarem Kundennutzen, besseren Umgang im Veränderungen und höhere Zufriedenheit von Kunden und Entwicklern garantieren sollen. Mit der Veröffentlichung des in Abb. 9 dargestellten „Manifesto for Agile Software Development“ (Beck, et al., 2001) wurde ein Dach für diese Methoden und die dahinterstehenden Werte geschaffen. Der Begriff der agilen Softwareentwicklung und die agilen Methoden gelten seither endgültig als etabliert.

We are uncovering better ways of developing software by doing it and helping others do it.
Through this work we have come to value:

**Individuals and interactions** over processes and tools
**Working software** over comprehensive documentation
**Customer collaboration** over contract negotiation
**Responding to change** over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

| | | |
|---|---|---|
| Kent Beck | James Grenning | Robert C. Martin |
| Mike Beedle | Jim Highsmith | Steve Mellor |
| Arie van Bennekum | Andrew Hunt | Ken Schwaber |
| Alistair Cockburn | Ron Jeffries | Jeff Sutherland |
| Ward Cunningham | Jon Kern | Dave Thomas |
| Martin Fowler | Brian Marick | |



**Abb. 9: Manifesto for Agile Software Development (Beck, et al., 2001)**

Das Manifest für agile SW Entwicklung stellt Individuen und deren Interaktionen über Prozesse und Werkzeuge (Übersetzung wie bei allen weiteren Prinzipien durch den Verfasser). Damit ist nicht gemeint, dass Prozesse, Dokumentation und Pläne unwichtig sind. Aber die Fähigkeit der Mitarbeiter zu geistig produktiver Arbeit und zur Kommunikation wird als wichtiger erachtet und durch agile Methoden stärker gefördert.

Das nächste Prinzip stellt laufende Software über ausführliche Dokumentation. Bei agilen Methoden soll so früh wie möglich eine lauffähige Software hergestellt werden. U.U. kann der Kunde diese bereits vor Vollendung des Projekts nutzen. Noch wichtiger ist aber die Möglichkeit die Software schon während der Entwicklung auszuprobieren und Feedback dazu zu geben. Auf diese Weise können Zielvorstellungen präzisiert werden oder ev. falsche Ziele erkannt und geändert werden.

Eine gute Zusammenarbeit mit dem Kunden wird als wichtiger erachtet als langwierige Vertragsverhandlungen. Die gängige Praxis von Festpreisverträgen hat das Problem, dass oft vor Projektbeginn der Entwicklungsaufwand nicht genau abgeschätzt werden kann. Außerdem können sich Anforderungen an das Produkt während der Projektlaufzeit ändern. Trotz genauer Einhaltung von

Verträgen kann es dazu kommen, dass weder der Kunde, noch der Entwickler mit dem Ergebnis zufrieden sind. Agile Methoden versuchen durch gute Zusammenarbeit das Projekt zu einem für alle Seiten zufriedenstellenden Ergebnis zu führen.

Das vierte Prinzip will, dass auf Veränderungen angemessen reagiert wird, statt strikt dem Plan zu folgen. Wenn während der Entwicklung Änderungen auftreten, soll das Team flexibel darauf reagieren und Pläne angemessen ändern. Das Auftreten von Veränderungen soll dabei nicht als Fehler oder Zusatzaufwand gesehen werden, sondern als normaler Bestandteil jeder Entwicklung akzeptiert werden.

Neben den 4 Grundprinzipien enthält das agile Manifest auch die folgenden 12 detaillierteren Prinzipien (Beck, et al., 2001):

- Die Zufriedenstellung des Auftraggebers durch frühe und durchgängige Auslieferung nützlicher Software hat oberste Priorität.
- Änderungen, auch spät in der Entwicklung, werden akzeptiert. Um dem Auftraggeber einen Wettbewerbsvorteil zu verschaffen sollen Anforderungsänderungen zu seinem Nutzen verarbeitet werden.
- Funktionsfähige Software soll wöchentlich oder monatlich produziert werden, wobei die Zeiträume so kurz als mögliche gewählt werden sollen.
- Das Management muss mit dem Entwicklungsteam über die gesamte Projektlaufzeit auf täglicher Basis zusammenarbeiten.
- Im Zentrum der Projekte stehen motivierte Individuen. Es muss den Projektmitarbeitern die benötigte Arbeitsumgebung und Unterstützung bereitgestellt werden, damit sie ihre Arbeit leisten können und es muss auch Vertrauen bestehen, dass sie ihre Aufgaben erfüllen.
- Die direkte, zwischenmenschliche Kommunikation bietet den effizientesten und effektivsten Weg Informationen auszutauschen.
- Funktionierende Software bildet das grundlegende Maß für den Projektfortschritt.
- Agile Methoden fördern eine nachhaltige Entwicklung. Die Projektsponsoren, Entwickler und Anwender sollen fähig sein, ein konstantes Arbeitstempo auf unbestimmte Zeit aufrechtzuerhalten.
- Ein durchgängiger Fokus auf technische Exzellenz und gutes Design verstärken die Agilität.
- Der Arbeitsumfang der nicht gemacht werden muss, soll durch Einfachhalten maximiert werden.
- Projektteams die sich selbst organisieren dürfen, kreieren die besten Architekturen, Anforderungen und Designs.
- Es soll in regelmäßigen Abständen eine Reflexion des Projektteams stattfinden, um Verbesserungsmöglichkeiten zu finden und die Vorgehensweise entsprechend zu optimieren.

Nachfolgend werden einige Methoden der agilen Softwareentwicklung kurz dargestellt. Im Wesentlichen bestehen diese aus Sammlungen von Prinzipien und Praktiken, die miteinander verknüpft, Anleitungen für Vorgehensweisen bei der Softwareentwicklung geben. Die einzelnen Praktiken werden dabei nur kurz erwähnt und nicht genau analysiert. In Kapitel 5 folgt diese genaue Analyse aller Praktiken hinsichtlich ihrer Anwendbarkeit im Systems Engineering.

### 2.2.1 Scrum (Schwaber, 2004)

Scrum ist eine Management-Methode für die Softwareentwicklung auf inkrementeller und iterativer Basis. Es werden keine spezifischen Mittel vorgeschrieben. Das Projektteam ist selbst dafür verantwortlich, die für den Erfolg notwendigen Mittel zu verwenden.

In Scrum Projekten gibt es drei verschiedene Rollen:

Der **Product Owner** legt das Ziel fest und stellt das Budget zur Verfügung. Er priorisiert die Anforderungen (Product-Backlog-Elemente) und stellt den Vertreter des Kunden dar.

Der **Scrum Master** ist eine Art Projektleiter. Er hat zwar die Verantwortung für den Projektablauf, aber keine Weisungsbefugnis, da das Team die Entscheidungen selbst treffen soll. Er ist unterstützend tätig, um dem Team effektives Arbeiten zu ermöglichen und den Scrum Prozess aufrecht zu erhalten.

Das **Team** trifft eigenständig Entscheidungen zur Erreichung der gesetzten Ziele. Es entscheidet selbst über das Arbeitspensum innerhalb einer Iteration (Sprint). Auch die Organisation und Arbeitszuteilung wird vom Team selbst bestimmt.

Die Vorgehensweise von Scrum bzw. der Scrum Prozess ist nur sehr grob definiert (siehe Abb. 10). Ein Projekt beginnt mit der Definition und Priorisierung der Anforderungen im sogenannten „Product Backlog“. Dieser wird aus der vom Kunden beschriebenen Produktvision abgeleitet. Der Product Backlog stellt damit die Summe der noch zu leistenden Arbeit dar. Er kann jederzeit von den Stakeholdern verändert werden, wobei der Product Owner die Verantwortung für das Endergebnis trägt. Am Anfang jedes Sprints, wird aus dem Product Backlog ein Sprint Backlog erstellt, der dann die Anforderungen für ein Entwicklungselement enthält. Die Dauer für einen Sprint soll etwa 2 Wochen bis 30 Tage betragen. Während eines Sprints finden im 24 Stunden Rhythmus kurze Treffen des gesamten Teams statt, in denen die zuletzt erledigte Arbeit, die für den aktuellen Tag geplante Arbeit und mögliche Probleme besprochen werden.

Am Ende eines Sprints werden die Ergebnisse präsentiert, die normalerweise lauffähige Module der Software darstellen. Nach der Ergebnispräsentation findet noch die sogenannte „Retrospektive“ statt, in welcher der letzte Sprint analysiert und besprochen wird um daraus zu lernen.

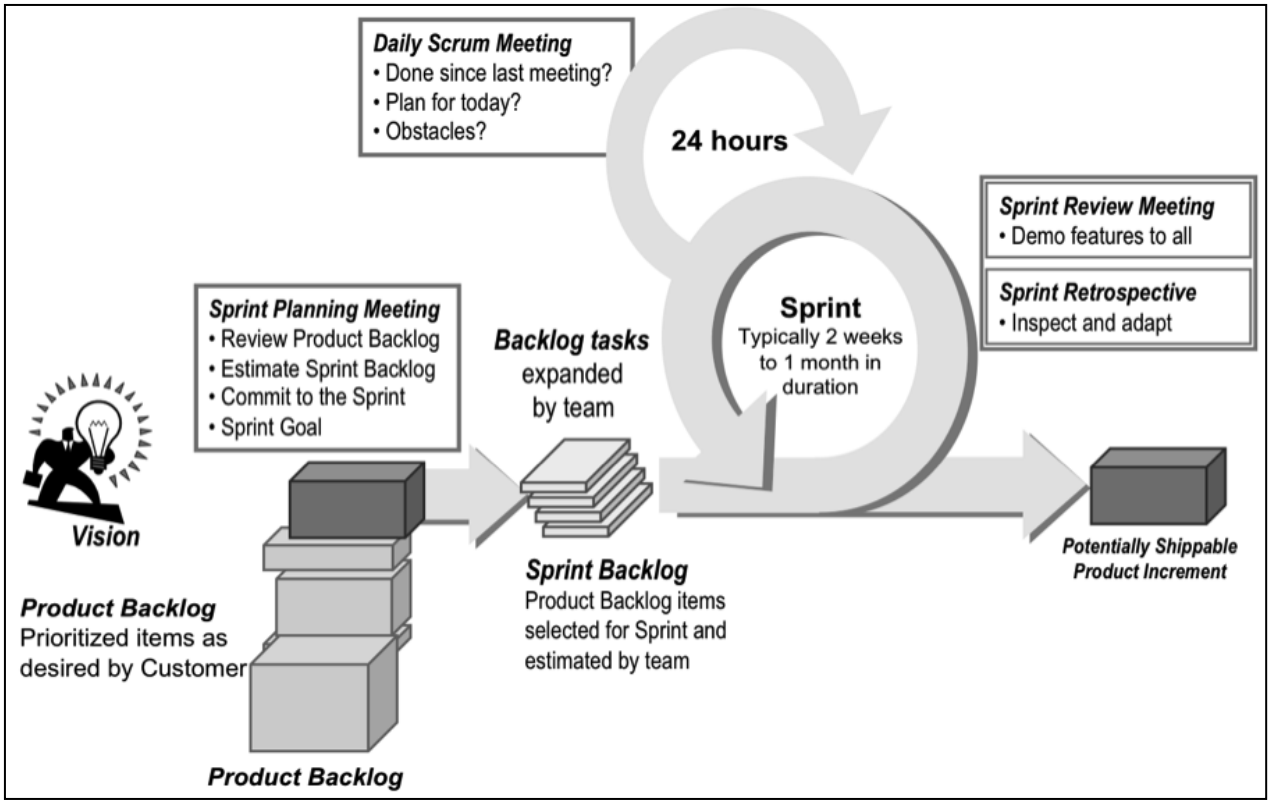


**Abb. 10: Scrum Vorgehensweise (Schwaber, 2004)**

### 2.2.2 Extreme Programming (Beck, et al., 2005)

Bei Extreme Programming (XP) wird die Entwicklung durch fortlaufende Iteration durchgeführt und es gibt zahlreiche, genau definierte Praktiken dazu. Es wird davon ausgegangen, dass Anforderungen und Aufwand zu Beginn noch unklar sind. Neue Funktionen werden laufend entwickelt, integriert und getestet. Einzelne Iterationen dauern ca. 1 – 4 Wochen. Größere in sich geschlossene Versionen, die aus mehreren Iterationen bestehen und für den Kunden bereitgestellt werden, nennt man Releases[3].

Anforderungen werden in XP als sogenannte User-Stories beschrieben, die Funktionsanforderungen an die Software aus der Sicht des Anwenders in Form einer sehr kurzen Geschichte beschreiben. Bei der Erstellung dieser User-Stories ist das ganze Team beteiligt. Die einzelnen User-Storys werden priorisiert und ihr Aufwand wird für die Planung der Releases abgeschätzt. Für die technische Umsetzung werden die User-Stories beim Start jeder Iteration in Tasks zerlegt. Während der Iterationen gibt es wie bei Scrum täglich kurze Meetings.

Extreme Programming betont besonders die personenbezogenen Aspekte der Entwicklung, im Sinne von fünf grundlegenden **Werten:**

- Kommunikation: Es soll ein einfacher und durchgängiger Informationsaustausch zwischen allen Stakeholdern möglich sein.

[3] Erklärung unter http://de.wikipedia.org/wiki/Entwicklungsstadium_%28Software%29

- Einfachheit: Es soll immer die einfachste Lösung gewählt werden, die alle Anforderungen erfüllt. Unnötige Prozesse und komplizierte Code-Konstrukte sollen aus der Entwicklung verbannt werden. Komplexität in der entwickelten Software darf nur entstehen, wenn sie sich durch die entstehenden Vorteile rechtfertigen lässt.
- Feedback: Nicht nur innerhalb des Projektteams, sondern zu allen Projektbeteiligten, speziell zum Kunden, soll Feedback gefördert werden.
- Mut: Es wird schon als Mut betrachtet, wenn man sich von den traditionellen Prozessmodellen abwendet und für die Anwendung von XP entscheidet. Auch danach soll man Selbstvertrauen haben, um riskante und ungewohnte Tätigkeiten durchzuführen. Außerdem ist hiermit auch das positive Reagieren auf Anforderungsänderungen durch den Kunden gemeint.
- Respekt: Jeder Projektmitarbeiter soll seine Meinung frei äußern dürfen und darf erwarten, von allen anderen mit Respekt behandelt zu werden. Konstruktive Kritik soll die Software verbessern, ohne dass zwischenmenschliche Probleme entstehen und dieses Ziel verhindern. Das Arbeitsklima soll Vertrauen und die Übernahme von Verantwortung fördern.

Neben diesen grundlegenden Werten gibt es folgende **Prinzipien**. Genauer erklärt und auf Anwendbarkeit im Systems Engineering überprüft werden diese in Kapitel 5, da sie teilweise auch in anderen agilen Methoden vorkommen und nicht mehrfach erklärt werden sollen:

- Menschlichkeit
- Wirtschaftlichkeit
- Beiderseitiger Nutzen
- Selbstähnlichkeit: Es soll versucht werden, vorhandene Lösungen wiederzuverwenden, auch wenn man sie skalieren oder anpassen muss.
- Ständige Verbesserung
- Vielfältigkeit: Unterschiedlichkeiten im Team können sich positiv ergänzen.
- Reflexion: Es soll nicht nur über das Ergebnis der Arbeit, sondern auch über die Arbeit selbst reflektiert werden.
- Fließen: Software soll nicht in großen Meilensteinen ausgeliefert werden, sondern es soll einen ständigen Fluss von kleinen, nützlichen Verbesserungen geben.
- Gelegenheit: Probleme sollen als Gelegenheiten für Veränderungen wahrgenommen werden.
- Redundanz: Redundanzen im Problemlösungsprozess können oft grobe Fehlschläge verhindern. Vor allem kritische Problemstellungen sollen auf unterschiedliche Arten behandelt werden. Z.B. sind im XP mehrere, redundante Praktiken zur Fehlersuche vorgesehen.
- Versagen: Wenn man die richtige Lösung nicht weiß, soll man den Mut haben und Fehlschläge in Kauf nehmen. Auch wenn man durch fehlgeschlagene Versuche noch keine Lösung erreicht hat, so kann man doch daraus lernen.
- Qualität: Qualität ist essentiell und soll nicht Kosten- oder Zeitdruck geopfert werden.
- Kleine Schritte: Das Entwickeln in kleinen Schritten vermindert das Risiko und ermöglicht erst viele der anderen Praktiken, wie z.B. das häufige Einholen von Feedback.
- Akzeptierte Verantwortung: Verantwortung soll den Teammitgliedern gegeben werden, aber diese müssen sie auch akzeptieren.

Schließlich gibt es in XP auch noch konkrete Praktiken. Im Folgenden werden die **Hauptpraktiken** erwähnt, die später in Kapitel 5 auf ihre Anwendbarkeit im SE überprüft werden:

- Zusammen Sitzen: Hierbei ist die örtliche Nähe gemeint, die bessere Kommunikation ermöglichen soll.
- Vollständiges Team: Im Team sollen alle Fähigkeiten vorhanden sein, die für die erfolgreiche Bewältigung des Projekts notwendig sind.
- Informativer Arbeitsplatz: Ein anderes Teammitglied soll bei Betrachtung des Arbeitsplatzes in 15 Sekunden feststellen können, woran gearbeitet wird und wie gut der Fortschritt ist.
- Energievolle Arbeit: Man soll nur solange arbeiten, wie man seine höchste Konzentration aufrecht erhalten kann. Gerade bei Wissensarbeitern ist nicht zu erwarten, dass eine Erhöhung der Arbeitszeit über normale Verhältnisse hinaus, auch den intellektuellen Output steigern kann.
- Pair Programming: Das Programmieren selbst soll immer in Zwei-Personen-Teams durchgeführt werden.
- User-Stories: Diese sollen zur Definierung der Anforderungen verwendet werden, wie in der Beschreibung von XP erwähnt wurde.
- Wöchentlicher Zyklus: Am Beginn jeder Woche soll die Arbeit konkret geplant werden.
- Quartalsweiser Zyklus: Die Ziele auf höherer Ebene sollen zu Quartalsbeginn geplant werden.
- Pufferzeit: Nicht im herkömmlichen Sinne, sondern im Sinne einer mit unwichtigen Tasks verplanten Zeit.
- 10 Minuten Build: Eine neue Software-Funktion soll innerhalb von 10 Minuten getestet werden können.
- Kontinuierliche Integration: Einzelne neue Funktionen und Änderungen sollen innerhalb weniger Stunden ins System integriert werden.
- Test-First Programmierung: Bevor man eine neue Funktion entwickelt, soll man den zugehörigen Test entwickeln.
- Inkrementelles Design: Das Systemdesign muss nicht zu Beginn fix definiert werden, sondern soll während der Entwicklung ständig angepasst werden dürfen.

### 2.2.3 Crystal (Cockburn, 2005)

Bei Crystal handelt es sich nicht um eine einzelne Methode, sondern um eine Familie von Methoden. Je nach Größe und Kritikalität des Projekts wird eine mehr oder weniger stark strukturierte Variante der Crystal Familie empfohlen (siehe Abb. 11).

Für die einzelnen Varianten werden jeweils konkrete Praktiken vorgeschlagen, diese sind aber nicht verpflichtend. Die Gemeinsamkeit der unterschiedlichen Varianten besteht in den folgenden sieben **Eigenschaften**:

- Regelmäßige Lieferungen: Es soll häufig (z.B. monatlich) getesteter Code den Anwendern vorgelegt werden um den Status des Projekts darzustellen, Feedback zu erhalten und Anforderungen abzugleichen.

- Reflektierte Verbesserungen: Regelmäßige Teamsitzungen abhalten um Verbesserungsvorschläge zu generieren. Es sollen dabei aktuelle Erfolge und Probleme, welche das Produkt aber auch das ganze Projekt betreffen, diskutieren werden.
- Osmotische Kommunikation: Durch räumliche Nähe soll jedes Teammitglied über die gesamte Kommunikation im Team im Bilde sein.
- Persönliche Sicherheit: Jeder soll seine Meinung und auch Kritik äußern können, ohne dafür gescholten zu werden.
- Schwerpunkte bilden: Ziele mit der höchsten Priorität sollen bearbeitet werden können, ohne dass die Entwickler mit häufigen Ablenkungen zurechtkommen müssen.
- Einfache Kontaktaufnahme mit Endanwendern: Es soll schnelles Feedback über die Qualität der fertig gestellten Ergebnisse erhalten werden und die Anforderungen sollen auf aktuellem Stand gehalten werden.
- Technische Umgebung mit automatisierten Tests, Konfigurationsmanagement und regelmäßiger Integration: Automatisierte Tests sorgen bei iterativer Entwicklung mit vielen kleinen Schritten für einen großen Zeitgewinn. Mit Konfigurationsmanagement können einzelne Module separat entwickelt werden. Regelmäßig Integration in kurzen Abständen soll dafür sorgen, dass Fehler schneller entdeckt werden.

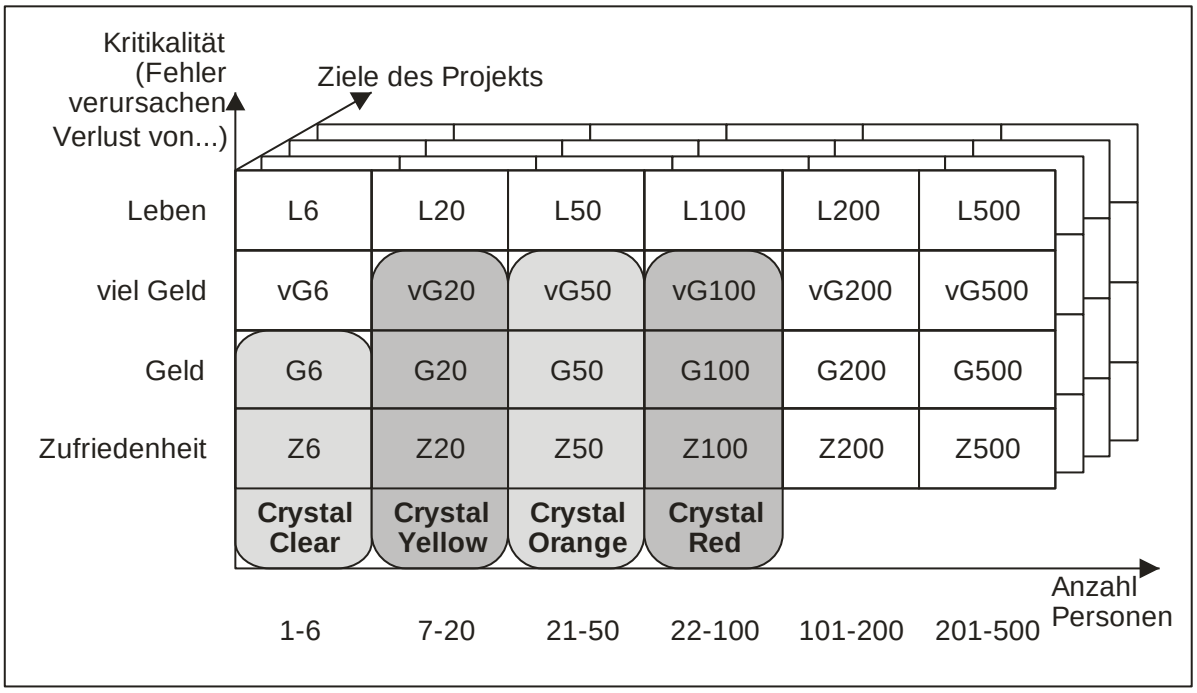


**Abb. 11: Die Crystal Methodenfamilie (Hruschka, et al., 2009 S. 55)**

### 2.2.4 Feature Driven Development (Palmer, et al., 2001)

Feature Driven Development wurde ursprünglich von Jeff De Luca als eine Sammlung von Praktiken, Methoden und Rollen definiert. Das entscheidende Merkmal dieser Methode sind die Features (zu entwickelnde Funktionen die einen Mehrwert für den Kunden darstellen), anhand derer die Organisation durchgeführt wird. Das Prozessmodell besteht aus 5 Prozessen:

1. Entwicklung des Gesamtmodells: Hier wird der Umfang des zu entwickelnden Systems festgelegt und es werden Modelle für die einzelnen Bereiche des Systems erstellt.
2. Erstellung der Feature-Liste: Die festgelegten Systembereiche werden in Features zerlegt, wobei ein Feature nicht länger als 2 Wochen zu seiner Realisierung benötigen darf.
3. Planung je Feature: Hier wird die Reihenfolge der zu entwickelnden Features geplant, unter Berücksichtigung der Abhängigkeiten zwischen diesen, ihrer Komplexität und der Auslastung der Teams von Programmierern.
4. Entwurf je Feature: Die einzelnen Features werden hier vorbereitet, indem ihr Design grob entworfen wird.
5. Konstruktion je Feature: Die Features werden programmiert und getestet.

Die ersten 3 Prozesse werden am Anfang des Projekts in wenigen Tagen durchgeführt. Prozess 4 und 5 werden zyklisch durchgeführt bis alle Features, die jeweils nur 2 Wochen Entwicklungszeit benötigen dürfen, fertig entwickelt sind.

### 2.2.5 Lean Software Development (Poppendieck, et al., 2009)

Ideen, die aus dem Produktionssystem von Toyota stammen und als Lean Manufacturing bekannt wurden (Womack, et al., 1991), wurden hier für die Softwareentwicklung adaptiert. Ziel ist es auch hier, die Kundenzufriedenheit zu erhöhen, bei gleichzeitiger Einsparung von Kosten, was durch Vermeidung von unnötigen Prozessen und unnötigen Funktion und Einsparung von Ressourcen erreicht werden soll.

Der Kern von Lean Software Development besteht aus 7 Prinzipien:

- Vermeidung von Unnötigem: Vermieden werden sollen unvollständig geleistete Arbeit, unnötige Prozesse, Zusatzfunktionen, Wartezeiten, unnötige Bewegungen sowohl von Personen als auch von Dokumenten und Mängeln.
- Qualität von Anfang an einbauen: Spätere Tests sollen nicht dazu verwendet werden die Qualität abzusichern. Der Code soll von Anfang an qualitativ hochwertig geschrieben werden. Die häufig durchgeführten Tests werden nicht als Qualitätssicherung, sondern als Teil der Entwicklung gesehen.
- Schaffe Wissen: Die Softwareentwicklung soll als Prozess gesehen werden, der Wissen schafft. Zu Projektbeginn kann deshalb nicht das ganze Wissen über Anforderungen und Aufwände vorhanden sein. Dieses muss erst während der Entwicklung generiert werden. Dazu ist eine Entwicklung in kleinen Schritten, auf welche Feedback eingeholt wird, vorteilhaft. Das Lernen soll aber auch generell gefördert werden, nicht nur innerhalb eines Projekts, sondern über Projekte hinweg.

- Entscheidungsmöglichkeiten offen halten: Entscheidungen, welche das Endprodukt unwiderruflich beeinflussen, sollen so spät wie möglich getroffen werden bzw. es sollen zumindest Optionen offen gelassen werden, die es erlauben, auf Änderungen zu reagieren.
- Schnelle Lieferung: Einerseits bedeutet dies, dass der Kundennutzen schneller geliefert werden soll. Andererseits wird dadurch auch die Chance geringer, dass sich Kundenanforderungen in der Zwischenzeit ändern können.
- Respekt für die Mitarbeiter: Im Zentrum jeder Entwicklung stehen die Mitarbeiter, da sie das dazu nötige Wissen tragen und die Prozesse umsetzen. Ein respektvoller Umgang mit Mitarbeitern, die Möglichkeit zur Selbstorganisation und die Förderung der Motivation sollen zu besserem Projekterfolg führen.
- Optimiere das Ganze: Fehler sollen nicht einer einzelnen Person oder einem einzelnen Prozess zugeordnet werden. Es soll immer das ganze System bzw. der ganze Entwicklungsablauf betrachtet werden, der zu diesem Fehler geführt hat.

### 2.2.6 Zusammenfassende Charakteristik der agilen Softwareentwicklungsmethoden

Auch wenn in einigen Methoden recht ungewöhnliche Praktiken propagiert werden, bei denen auf den ersten Blick der Zusammenhang zur Agilität nicht feststellbar ist, so gibt es doch auch starke Ähnlichkeiten zwischen den einzelnen Methoden.

Eine agile Methode erfüllt nach Yourdon (1999) die folgenden Prinzipien:

- Iterative Entwicklung in mehreren Zyklen
- Inkrementelle Auslieferung
- Selbstorganisation: Das Team bestimmt den besten Weg zum Abschluss des Projekts selbst.
- Emergenz: Prozesse, Prinzipien und Arbeitsstrukturen werden nicht bei Projektstart festgelegt, sondern werden je nach Bedarf während des Projekts festgelegt oder geändert.

Für Richard Turner sind es 8 wesentliche Charakteristika, welche eine agile Methode ausmachen (Turner, 2007):

- Ständiges Lernen als Geisteshaltung
- Fokus auf Kundennutzen
- Kurze Iterationen, die Nutzen generieren
- Neutralität gegenüber Veränderungen
- Laufende Integration neu entwickelter Module
- Test-driven: Zuerst die Testfälle schreiben, dann erst mit dem Programmieren beginnen.
- Lean: Vermeidung von Unnötigem
- Team-ownership: Jeder im Team hat Verantwortung für Planung, Prozesse und Qualität.

Damit sind die Methoden der agilen Softwareentwicklung für diese Arbeit ausreichend genau umrissen. Welche Prinzipien und Praktiken daraus im Systems Engineering Anwendung finden können, wird in Kapitel 5 analysiert.

## 2.3 Agilität in anderen Bereichen

Beginnt man für die Disziplin des Systems Engineerings nach Agilität zu suchen, so kann die agile Software-Entwicklung als erste Adresse dafür gesehen werden. Bei einigen Ideen besteht durchaus die Möglichkeit dieser direkt zu übernehmen. Aus diesem Grund und weil sich agile Softwareentwickler schon intensiv mit der Übernahme von allgemeinen agilen Ideen in ihre Domäne beschäftigt haben, wird ein Großteil der gesammelten Methoden in Kapitel 5 aus diesem Bereich stammen. Nachdem sich diese Dissertation aber nicht auf die Übernahme von Methoden und Praktiken der agilen SW-Entwicklung beschränkt, sollen auch andere Gebiete auf brauchbare Ideen durchsucht werden.

Schon vor Jahrtausenden wurde der Wert der Agilität in der Kriegsführung erkannt. In Sunzis (alternative Transkription: Sun Tzu) Werk „Die Kunst des Krieges“ wurde schon ca. 500 Jahre v. Chr. die Agilität als wesentlicher Erfolgsfaktor bei kriegerischen Auseinandersetzungen erachtet. Bei zahlreichen berühmten Schlachten von der Antike bis zur Neuzeit war die Agilität der entscheidende Faktor über Sieg und Niederlage (Boyd, 1986). Diese Erkenntnisse haben wesentlichen Einfluss auf heutige Militärdoktrinen z.B. auf die, der US Marines (US-Marine-Corps, 1997).

Dabei beschränkt sich die Betrachtung der Agilität nicht nur direkt auf Kampfhandlungen, bei denen ein Vorteil durch größere Schnelligkeit und Wendigkeit von Kampfeinheiten und schnelleres Erkennen von und Reagieren auf neue Kampfsituationen hergestellt werden soll. Auch die Vorbereitung der Schlacht (schnelleres Verlagern von Truppen) und die Entwicklung neuer Kriegstechnologien stehen im Betrachtungsfeld der Agilität (Boyd, 1986).

Außer für militärische Zwecke, wurde die Agilität hauptsächlich im technoökonomischen Bereich betrachtet. In der Fachliteratur wurden neben der agilen Softwareentwicklung die folgenden Themengebiete gefunden, die sich explizit mit Agilität beschäftigen. Diese sollen an dieser Stelle nur kurz umrissen werden. Für ASE interessante Ideen und Methoden aus den einzelnen Fachgebieten werden dann gesammelt in Kapitel 5 vorgestellt.

Unter dem Begriff „**Agile Manufacturing“** (Agile Fertigung) hat sich eine wissenschaftliche Diskussion entwickelt, die sich damit beschäftigt, wie in der Fertigung Agilität erreicht werden soll (Gunasekaran, 1998). Es soll hier für Produktionssysteme herausgefunden werden, wie diese schnell und kosteneffizient auf neue Produkte umgestellt werden können, um auf unvorhergesehene Änderungen des Marktes bzw. der Auftragslage reagieren zu können. Weiters soll die Agilität auch die kosteneffiziente Fertigung von maßgefertigten Kundenlösungen ermöglichen („Mass-Customizing“). Auch mit der Ausnutzung neuer Informationstechnologie und der Zusammenarbeit von Firmen (z.B. schnelle Verlagerung von Produktionskapazitäten) beschäftigt sich dieses Themengebiet (Maskell, 2006).

Als aktuelle wissenschaftliche Themenstellung kann z.B. die Entwicklung eines Entscheidungsmodells genannt werden, welches für die Automobilindustrie eine Hilfestellung bei der Investitionsentscheidung für Produktionssysteme bietet und dabei die verfügbare Agilität berücksichtigt (Elkins, et al., 2004).

Sehr ähnliche Themenstellungen werden auch unter dem Begriff „**Agile Supply-Chain Management**“ diskutiert (Dove, 1996), (Mason-Jones, et al., 1999).

Der Begriff „**Agile Business Process Management**“ (Agiles Geschäftsprozessmanagement) ist (noch) nicht sehr etabliert. Nachdem die Fragestellung, wie man Geschäftsprozesse agil gestalten kann, aber in der

einleitend beschriebenen Marktumgebung sehr relevant ist, ist eine Häufung wissenschaftlicher Artikel zu diesem Thema zu erwarten (Hinkelmann, et al., 2006), (Schatten, et al., 2007).

Bereits stärker etabliert hat sich der Begriff „**Agile Enterprise**“ (Agile Unternehmung). Unter diesem Begriff beschäftigen sich zahlreiche Autoren mit der Fragestellung, wie Unternehmungen in sehr dynamischen Umfeldern zu gestalten sind. Unter dem Dach dieses Begriffs könnte man die anderen agilen Fachgebiete vereinen. Insbesondere geht es hier aber um die Betrachtung der Unternehmung als Ganzes und um Lösungen auf strategischer oder unternehmungsorganisatorischer Ebene (Dove, 2001), (Pal, et al., 2005), (Zobel, 2005).

Auch „**Agile Systems Engineering**“ hat sich als Begriff in der Fachwelt schon etabliert. Eine übergreifende agile Systems Engineering Methodik wurde bisher in der Literatur aber noch nicht vorgestellt. Man findet Werke, die zur Entwirrung des Begriffschaos rund um Agilität und Systems Engineering beitragen (Haberfellner, et al., 2005), (Madni, 2008), (Dove, et al., 2008) und häufig den Vorschlag agile Softwareentwicklungsmethoden auch im SE anzuwenden (Boehm, 2006), (Turner, 2007), (Boehm, 2008), (Haberfellner, et al., 2008). Es wird auch auf Herausforderungen eingegangen, welche für den Einsatz von agilen Methoden bei der Entwicklung von Hardware bestehen (Kaindl, et al., 2010), (Stelzmann, et al., 2010). Außerdem werden einzelne Methoden vorgeschlagen um die Agilität im SE zu verbessern (Wilson, et al., 2003), (Dove, 2005), (Gilb, 2005), (Haberfellner, et al., 2005), (Malotaux, 2006), (Dove, 2006), (Gilb, 2006), (Madni, 2008).

Die Artikel zur Definition werden gesammelt in Kapitel 4 ausgewertet, einzelne Methoden und Prinzipien aus den erwähnten Werken werden in Kapitel 5 beschrieben.

Zuvor soll aber die Vorgehensweise bei der empirischen Studie erklärt werden, da in allen weiteren Kapiteln die in der Praxis gefundenen Erkenntnisse immer gleich den in der Literatur gefundenen Aussagen gegenübergestellt werden sollen.

# 3 Forschungsdesign

Der Ablauf der Forschungsarbeit wurde im Groben schon in Kapitel 1.2 erklärt. Hier sollen nun die Forschungsmethode der „Grounded Theory“ (Glaser, et al., 2006) und die Durchführung der empirischen Erhebung dargestellt werden. Wie in Abb. 3 skizziert, wurde diese in einer relativ frühen Phase begonnen. Dieser Weg war deshalb notwendig, weil es vorher weder ein allgemeingültiges Verständnis der Agilität gab, noch eine allgemeine Theorie zu einer Agile Systems Engineering Methodik. Um einen brauchbaren Ansatz für die betrachteten Industrien liefern zu können, musste dazu von Anfang an auch empirisch abgestimmt werden, was die eigentliche Problemstellung ist, für welche die Industrie eine Lösung durch Agilität erwartet. Deshalb wurden auch schon für die Analyse der Problemstellung und für die Zieldefinierung empirische Daten erhoben. Somit beinhalten fast alle Kapitel auch Erkenntnisse welche durch empirische Erhebungen zustande gekommen sind.

## 3.1 Forschungsmethode

Der wesentliche Inhalt der Dissertation ist eine Konzeptgestaltung bzw. die Erstellung einer neuen Theorie. Deshalb erschien von Anfang an eine qualitative Herangehensweise geeigneter zu sein als eine quantitative. Die wesentlich größere Offenheit und Flexibilität des qualitativen Ansatzes gestattet es, Ideen für eine agile Vorgehensweise im SE in vielen verschiedenen Bereichen zu finden, die dann zu einer neuen Theorie miteinander kombiniert werden können. Ein Messen eines Zusammenhangs um diesen möglichst genau beschreibbar und statistisch vorhersagbar zu machen, was der Sinn von quantitativen Methoden ist, hat keine Bewandtnis für das Ziel dieser Arbeit. Vielmehr sollen hier Zusammenhänge tiefgründig verstanden werden, um daraus schrittweise eine neue Theorie zu erstellen. Aus diesen Gründen fiel die Entscheidung auf das qualitative Methodenspektrum.

In der qualitativen Forschung gibt es, aufgrund der Heterogenität der bearbeiteten Fragestellungen, kein definiertes Repertoire an Methoden für die Datenerhebung und Analyse. Vielmehr ist es in der qualitativen Forschung wichtig, eine geeignete Herangehensweise an die Problemstellung zu finden (Lamnek, 1993 S. V).

Weitere Merkmale der qualitativen Forschung sind nach Flick, et al. (2000 S. 24):

- Orientierung am Alltagsgeschehen
- Erhebung der Daten im Kontext
- Reflexivität des Forschers
- Verstehen als Erkenntnisprinzip
- Offenheit
- Konstruktion der Wirklichkeit als Grundlage
- Entdeckung und Bildung von Theorien als Ziel

Wie bereits erwähnt, gibt es einen vielfältigen und nicht eingeschränkten Katalog an Methoden für die qualitative Forschung. Bekannte und häufig verwendete Methoden sind nach Ebster, et al. (2008 S. 140):

- qualitative Interviews
    - narratives Interview
    - fokussiertes bzw. problemzentriertes Interview
    - Leitfadeninterview
- Gruppendiskussion
- teilnehmende Beobachtung
- qualitative Inhaltsanalyse

Als Erhebungsmethode wurde für diese Dissertation einerseits das Leitfadeninterview gewählt, mit dem eine Reihe von Experten befragt wurde. Andererseits ergab sich auch die Möglichkeit, innerhalb eines Expertenpanels an Gruppendiskussionen teilzunehmen, welches sich zum Erfahrungsaustausch zwischen Industrieunternehmen gebildet hatte. Außerdem konnte der Autor bei einigen Konferenzen die Thematik mit anderen Mitgliedern der Forschungscommunity diskutieren. Zusätzlich sollten auch die Erkenntnisse, die der Autor durch seine eigene Berufserfahrung bei Magna Powertrain in der Systementwicklung von Fahrzeugkomponenten erworben hatte, genutzt werden. Die „Grounded Theory Method“ ist eine qualitative Vorgehensweise der Forschung, welche die Kombination all dieser Datenquellen erlaubt (Glaser, et al., 2006 S. 161ff, 252f). Damit lässt sich die Abb. 12 zeichnen, welche die Quellen darstellt, die für den Erkenntnisgewinn zur Generierung der Agile Systems Engineering Methodik verwendet wurden.

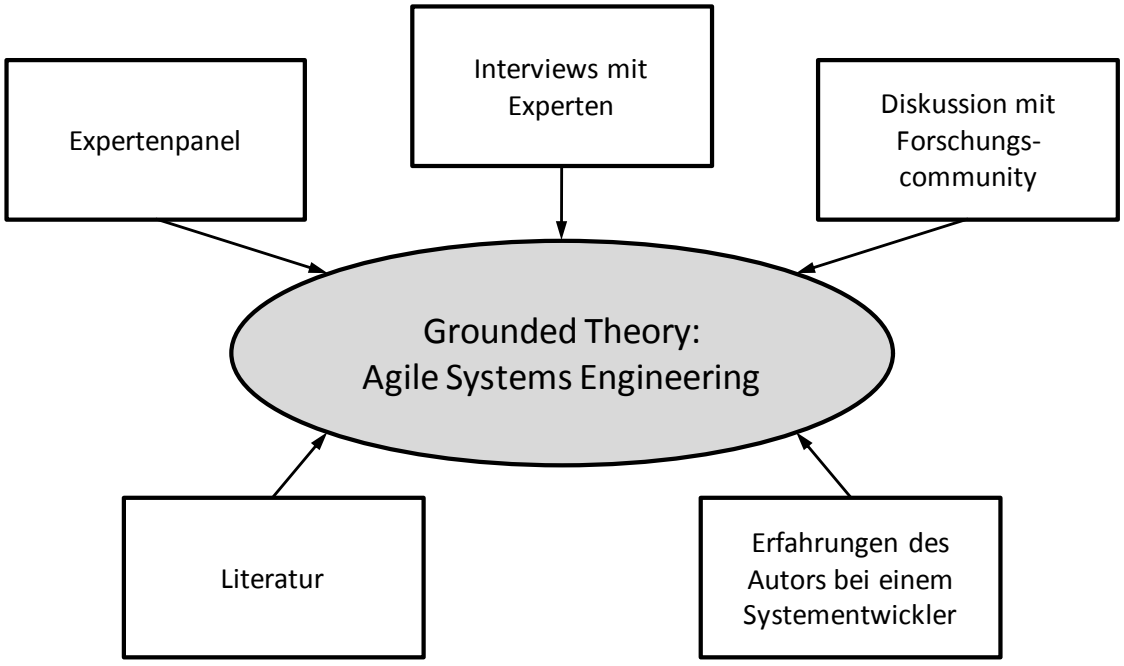


**Abb. 12: Verwendete Datenquellen für die Grounded Theory**

Die Grounded Theory Method sieht nun eine wechselweise Datenerhebung und –analyse zur Entwicklung einer praktisch anwendbaren und nachvollziehbaren Theorie vor (Glaser, et al., 2006). Der Forschungsprozess wird dabei so lange fortgesetzt, bis eine theoretische Sättigung eintritt (Glaser, et al., 2006 S. 61). Diese ist dann erreicht, wenn es scheint, dass neue Daten die Theorie nicht mehr weiterentwickeln können.

Nachdem die Vorgehensweise vielen Regeln unterliegt, wird eine rigorose Anwendung der Grounded Theory Method von Paris und Hürzeler - speziell für Dissertationen - als nicht sinnvoll angesehen. Zu sehr würde dabei der Prozess in den Vordergrund rücken und die Interpretation und die tiefergehende Reflexion verlorengehen (Paris, et al., 2008 S. 4). Sie plädieren deshalb für einen pragmatischen Einsatz, vor allem bei der Datenanalyse (Paris, et al., 2008 S. 4).

Das Forschungsdesign der vorliegenden Dissertation folgt dieser Empfehlung und der Aussage von Glaser und Strauss, die Ursprungsquelle aller Theoriebildung sei die Erkenntnis des Beobachters selbst (Glaser, et al., 2006 S. 251). Auf diesem Weg werden in den weiteren Kapiteln die aus den verschiedenen Datenquellen gewonnenen Erkenntnisse zu einer allgemeinen Theorie zusammengeführt. Vorher sollen aber noch die Details zu den Interviews und zu den Gruppendiskussionen mit dem angesprochenen Expertenpanel erklärt werden.

## 3.2 Expertenpanel

Kurz nach Beginn der Arbeit formierte sich eine Gruppe mit Vertretern aus der Industrie zum Erfahrungsaustausch bezüglich der Anwendung agiler Methoden. Diese Gruppe stellte einen Teil der „Styrian Systems and Software Quality Initiative“ ($S^3QI$) dar, die später auch internationale Teilnehmer aufnahm, weshalb das „Styrian“ aus dem Namen gestrichen wurde ($S^2QI$). Der Autor der vorliegenden Arbeit wurde eingeladen, in dieser Arbeitsgruppe die wissenschaftliche Erfassung und Ausarbeitung der durch die Teilnehmer aus der Industrie eingebrachten Erfahrungen durchzuführen.
Das anfänglich ungenau definierte Themengebiet wurde dabei bald auf einen Erfahrungsaustausch und ein Finden von Best Practices für die Anwendung von agilen Methoden im Systems Engineering eingeengt. Dieses Thema passte nun perfekt zu der Aufgabenstellung der Dissertation, weshalb zahlreiche Ideen und umfangreiche Erfahrungen als Inputs für diese Arbeit genutzt werden konnten.

Die Teilnehmer dieser Gruppendiskussionen und die Firmen, welche diese vertreten haben, werden im Folgenden aufgeführt:

- DI Dr. Richard Messnarz: International Software Consulting Network (ISCN)
- DI Gunther Spork: Magna Powertrain
- DI Gerhard Lichtenecker: Magna Steyr Fahrzeugtechnik
- Peter Müller: Flextronics International
- DI Dr. Christian Kreiner: Salomon Automation
- DI Dieter Cramer: Siemens AG München
- DI Ralf Bayrleithner: Siemens AG Österreich
- DI Ralph Miarka: Siemens AG Österreich
- DI Dr. Johannes Pusterhofer: Siemens IT Solutions and Services
- Frank König: ZF Friedrichshafen

Die Gruppendiskussionen fanden halbjährlich statt. Es sei angemerkt, dass nicht alle Personen durchgängig bei allen Diskussionen anwesend waren.

Die Funktion aller angeführten Personen kann mit methodischer Unterstützung in der System- und Softwareentwicklung beschrieben werden, in Führungs-, Mitarbeiter- und Beraterfunktion.

## 3.3 Interviews

Um nicht ausschließlich von dieser Expertengruppe abhängig zu sein und um ein breiteres Feld von Firmen aber auch von anderen Funktionen innerhalb derselben Firmen abzudecken, wurden zusätzlich 12 Interviews geführt. Außerdem empfiehlt auch die Grounded Theory die Befragung unterschiedlicher Gruppen (Glaser, et al., 2006 S. 55). Die Auswahl der Interviewpartner wurde weder zufällig noch im Vorhinein durchgeführt. Es wurde je nach Forschungsfortschritt jeweils ein nächster Interviewpartner gewählt, der einen weiteren Erkenntnisgewinn erwarten ließ, was dem Prinzip des „Theoretical Samplings“ entspricht (Glaser, et al., 2006 S. 45ff). Die Interviewpartner waren in chronologischer Reihenfolge:

1. Dr. Friedrich Santner: Anton Paar
2. DI Dr. Christian Kreiner: Salomon Automation (Aufgrund seiner weitgehenden Erfahrungen wurde mit diesem Mitglied der S²QI Gruppe auch ein separates Interview durchgeführt.)
3. Mag. Gernot Grömer: Österreichisches Weltraumforum
4. DI Matthäus Decker: Siemens Transportation
5. DI Gunther Zarnhofer: Magna Powertrain
6. DI Dr. Mario Ivanisin: Magna Weltraumtechnik
7. DI Dr. Bernd Wiermeier: Magna Steyr Fahrzeugtechnik
8. DI Günther Jelesic: Andritz AG
9. DI Thomas Linortner: Magna Powertrain
10. Ewald Buch: Kendrion Binder Magnete
11. DI Martin Bazant: Bombardier Transportation Switzerland
12. Johannes Knafl: Andritz AG

Die Interviewpartner wurden nach ihren Erfahrungen mit Entwicklungsprojekten von technischen Systemen befragt. Um möglichst viele Ideen zur Themenstellung zu erfassen, wurden die Interviews offen gestaltet. Es gab auch keine vordefinierten Fragestellungen, die allen Interviewpartner gestellt wurden. Außerdem wurden die Fragen mit Fortschreiten der Theoriebildung weiterentwickelt. Trotzdem wurde als Hilfestellung ein Leitfaden erstellt, um dem Gespräch eine Struktur zu geben und um die vollständige Abdeckung aller Teilgebiete der Thematik sicherzustellen. Der Leitfaden enthielt die nachstehenden Themengebiete, welche im Folgenden jeweils mit beispielhaften Fragestellungen beschrieben werden sollen.

***Interviewleitfaden:***

- Allgemeines
  - Vorstellung des Thema
  - Vorstellung der Personen und ihrer Funktionen
- Entwicklungsprojekte
  - Welche Art von Projekten und Produkten? Wer war Auftraggeber?
  - Prozesse, Organisation, Projektgröße (Teamgröße, Dauer, Komplexität)?
- Veränderungen
  - Welche traten auf und was waren die Ursachen?
  - Wie wurde mit den Veränderungen umgegangen? Proaktives oder reaktives Vorgehen?
  - Welche Auswirkungen hatten die Veränderungen und auch die Reaktionen darauf auf die Entwicklung?
  - Wie wurde der Projekterfolg von den Veränderungen beeinflusst?
  - Wo besteht Bedarf nach Lösungen und was soll damit erreicht werden?
- Agilität
  - Welche Vorgehensweisen haben sich im Umgang mit Veränderungen als erfolgversprechend erwiesen, welche weniger?
  - Unter welchen Voraussetzungen können diese Vorgehensweisen eingesetzt werden?
  - Wie kann die Wahrnehmung und Analyse der Veränderungen verbessert werden?
  - Wo kann Flexibilität zur Verfügung gestellt werden?
  - Welche Faktoren haben generell darauf Einfluss, ob trotz zahlreicher Veränderungen, erfolgreich entwickelt werden kann.
- Generischer Ansatz einer agilen SE Methodik
  - Vorstellung des aktuellen Forschungsstandes
  - Wie weit bildet dieser Ansatz vorhandene Vorgehensweisen ab?
  - Wo kann dieser Ansatz vorhandene Vorgehensweisen unterstützen bzw. verbessern?
  - Was kann am Ansatz verbessert werden?
  - Wann kann dieser Ansatz angewandt werden, wann nicht?

# 4 Das Wesen der Agilität

Hier soll sowohl die Verwendung des Begriffs „Agilität“ im allgemeinen Sprachgebrauch, als auch in verschiedenen Fachgebieten betrachtet werden, um schließlich zu einer genauen Charakterisierung des Agile Systems Engineerings zu gelangen. Außerdem soll dargestellt werden, welche Konzepte oder Funktionsprinzipien normalerweise unter einer agilen Entwicklung verstanden werden. Dazu wird der Stand der Literatur abgebildet. Es wird aber auch auf die Verwendung des Begriffs „Agilität“ in der Industrie (erhoben durch das Expertenpanel) eingegangen.

## 4.1 Agilität im allgemeinen Sprachgebrauch

Im Duden wird das Wort „agil“ mit flink, wendig, beweglich beschrieben. Das Fremdwörterlexikon von Wahrig führt dieselben Eigenschaftswörter an und beschreibt „Agilität“ mit agilem Wesen, Flinkheit, Gewandtheit und Beweglichkeit. Die Herkunft des Wortes liegt im Lateinischen wo „agilis“ so viel bedeutet wie beweglich, flink oder flott.

Langenscheidts Englisch-Deutsch übersetzt „Agility“ mit „Flinkheit, Behändigkeit, Aufgewecktheit“ und „agile“ mit „flink und behände (Verstand)“. Interessant ist hierbei, dass nicht nur auf die Beweglichkeit von materiellen Dingen, sondern explizit auch auf den Verstand hingewiesen wird.

Sehr häufig wird das Wort „Agilität“ im Zusammenhang mit Personen höheren Alters verwendet, die sich durch gute geistige und körperliche Leistungsfähigkeit auszeichnen. Für jüngere Personen wird dieses Wort kaum verwendet, auch wenn sie besonders fit sind. Daraus kann geschlossen werden, dass Agilität bei Jungen als selbstverständlich angesehen wird. Vermutlich weil man mit Agilität nicht den absoluten Wert der Leistungsfähigkeit herausstreichen will, sondern die Fähigkeit, sich ständig ändernden Lebensbedingungen anzupassen und aktiv und beweglich zu bleiben und diese Fähigkeiten häufig mit dem Alter schwinden. Die Werbung fragt dann nach Müdigkeit, Unwohlsein, Verwirrtheit und Vergesslichkeit und will diese Dinge vornehmlich mit Präparaten beseitigen um einem alternden Menschen wieder Agilität zu geben.

Im Sport wird dieses Wort verwendet, um die Leistungsfähigkeit eines Sportlers herauszustreichen und auch hier ist nicht der Absolutwert der Leistungserbringung gemeint, sondern eine Fähigkeit zur schnellen Reaktion auf die augenblicklichen Anforderungen der Sportart. Z.B. wird ein Fußballer als besonders agil bezeichnet, wenn er sich blitzschnell auf die sich ständig verändernde Spielsituation einstellen kann, z.B. bei einem Wechsel von einer Angriffs- zu einer Verteidigungssituation.

Agilität bedeutet umgangssprachlich also so viel wie körperliche oder geistige Wendigkeit oder Flinkheit und sie ist eine Fähigkeit oder Eigenschaft. Im Folgenden soll versucht werden, die Fähigkeit der Agilität genauer zu analysieren:

- **Reaktion**: Bei Agilität ist immer eine neue Information, neue Situation oder Veränderung vorhanden, auf die man reagiert. Beim Sportler ist es der Spielverlauf, bei einer alternden Person sind es Anforderungen des täglichen Lebens auf die reagiert werden soll. Auch das Wort Anpassungsfähigkeit kann in diesem Zusammenhang genannt werden. Die Agilität kann aber auch durch Maßnahmen gesteigert werden, die im Vorhinein (proaktiv) gesetzt werden.

- **Wahrnehmungsfähigkeit**: Die Wahrnehmung der neuen Information oder Veränderung ist natürlich eine Voraussetzung für die Fähigkeit zur Reaktion. Eine Verbesserung der Wahrnehmungsfähigkeit kann also als Maßnahme zur Steigerung von Agilität gesehen werden.
- **Flexibilität**: Flexibilität kann als das Potential an Möglichkeiten zur Reaktion gesehen werden. Dieses kann körperlich und/oder geistig sein. Auch die Wörter Veränderbarkeit oder Anpassbarkeit können hier verwendet werden.
- **Leistungsfähigkeit**: Die Leistungsfähigkeit ist ein wesentliches Potential für die Flexibilität oder Anpassbarkeit. Z.B. ist eine schnelle Reaktionsfähigkeit einem Fußballer nur von Nutzen, wenn er auch genügend Kraft besitzt, um mit einer geeigneten Aktion reagieren zu können.
- **Zielgerichtetheit**: Die Reaktion, die in einer agilen Handlung durchgeführt wird, dient nicht dem Selbstzweck, sondern der Erreichung eines höheren Ziels. Z.B. kann ein Fußballer nur dann als agil bezeichnet werden, wenn die beobachtete Aktion den versuchten Spielgewinn erkennen lässt.
- **Willen**: Eine Reaktion ist auch nur dann agil, wenn man sich willentlich dafür entschieden hat. Die Entscheidung darf dabei bewusst oder z.B. bei einem Sportler auch unbewusst, also durch Training automatisiert, getroffen werden. Wichtig für die Erfüllung einer agilen Handlung ist, dass die Reaktion nicht zufällig zustande gekommen ist.
- **Geschwindigkeit**: Die Geschwindigkeit der Reaktion stellt eine gute Möglichkeit dar, um den Grad der Agilität zu bestimmen. Absolutwerte für Agilität sind zwar kaum bestimmbar, es gibt auch keine Grenze zwischen Agilität und Nicht-Agilität. Man kann aber eine Reaktion, die zu spät für die Erreichung des Ziels durchgeführt wurde, durch einen Mangel an Agilität erklären.
- **Weitere Eigenschaften**: Mit den obigen Begriffen sollte die Fähigkeit der Agilität im Allgemeinen ausreichend charakterisiert sein. Sie erhebt aber keinen Anspruch auf Vollständigkeit. Mögliche andere Charakteristika sind: effektiv, effizient, zweckmäßig, graziös, gewinnbringend, …

## 4.2 Definition des Agile Systems Engineerings

Um eine passende Definition für ASE zu finden, soll nun die Verwendung des Begriffs „Agilität“ in ingenieurwissenschaftlichen Disziplinen untersucht werden.

Die gefundenen Definitionen der Agilität in den untersuchten Fachgebieten sind ähnlich, aber nicht identisch. Das hängt damit zusammen, dass die Ziele, die durch mehr Agilität erreicht werden sollen, auch nicht identisch sind. Außerdem wird nicht von allen Autoren großer Wert darauf gelegt, dass alle Eigenschaften, die sich hinter dem Wort „Agilität“ verbergen, erfüllt werden. Das Wort Agilität hat sich zu einem Übergriff oder Schlagwort für eine ganze Reihe von Lösungsprinzipien verschiedener Disziplinen entwickeln, die nur entfernt mit eigentlicher Agilität zu tun haben. Es werden also unterschiedliche Problemstellungen unter dem Begriff der Agilität „gelöst“, die nicht (genau) der Problemstellung dieser Dissertation entsprechen. Der Vergleich von Zielen und Definitionen ist zumindest nötig um festzustellen, ob die Übernahme von agilen Konzepten aus anderen Disziplinen in das Systems Engineering sinnvoll ist.

Es werden nun Betrachtungsweisen aufgezählt und diskutiert, die von Autoren der in Kapitel 2.2 und 2.3 beschriebenen Domänen der Agilität stammen:

Rick Dove betrachtet die Unternehmung als Ganzes (Strategie, Organisation, …). Agilität liegt für ihn im effektiven Verwalten und Anwenden von Wissen, um der Unternehmung genug Potential zu geben, damit sie in einem sich ständig verändernden und unvorhersehbaren Geschäftsumfeld erfolgreich sein kann (Dove, 2001 S. 9ff). Neben dem „Verwalten und Anwenden von Wissen“, das er als Wissensmanagement bezeichnet, sieht Dove die Fähigkeit zur Veränderung als zweites großes Standbein der Agilität.

Weiters beschreibt Dove die Agilität als, über Anpassungsfähigkeit und Flexibilität hinausgehende Fähigkeit einer Unternehmung sich schnell, kostengünstig, stabil und ohne Einschränkungen verändern zu können (Dove, 2005 S. 11). Die Steigerung der Agilität bedeutet also im Falle von unvorhersehbaren Ereignissen, dass die Kosten und die Zeit für die Reaktion reduziert werden und die Zielgenauigkeit und der Möglichkeitsbereich für die Reaktion vergrößert werden (Dove, 2005 S. 2).

Auf den Punkt gebracht ist Agilität die effektive Reaktion auf eine Chance oder ein Problem, wobei mit „effektiv“ zeitgerecht (um Nutzen zu liefern), finanziell tragbar, berechenbar und umfassend (befriedigt alle Anforderungen) gemeint ist (Dove, 2006 S. 4). Um das unter unsicheren Bedingungen erreichen zu können, benötigt man zumindest Situationskenntnis, Entscheidungsfähigkeit und Änderungsfähigkeit (Dove, et al., 2008).

Die Tätigkeiten im Rahmen der Agilität sind dabei nicht reaktiv auf den Zeitpunkt nach Erkennen der Chance/Gefahr beschränkt, sondern können und sollen auch proaktiv sein. Die Unternehmung soll und muss also schon im Vorhinein darauf ausgelegt werden, im Bedarfsfall agil reagieren zu können. Das bedeutet aber nicht, dass hierbei genau definierte Reaktionen für definierte Anlassfälle entwickelt werden sollen. Die Anlassfälle, die durch die Agilität bedient werden sollen, wurden ja als unvorhersehbar definiert. Es geht hier nur darum, der Organisation proaktiv das Potential zu geben mit Unvorhersehbarem in einem möglichst breitem Spektrum umgehen zu können (Dove, 2001 S. 11). Statt den Begriffen „proaktiv“ und „reaktiv“ werden von Haberfellner für Systeme die Begriffe „antizipierend“ (das System versucht Veränderungen zu prognostizieren und Anpassungen vorwegzunehmen) und „reagierend“ (das System entwickelt erst nach der Veränderung eine Reaktionsstrategie) verwendet (Haberfellner, 1973 S. 31f).

Bei der Betrachtung von Fertigungssystemen wird der Fertigung Agilität zugeschrieben, wenn sie der Unternehmung das Potential gibt, in einem Geschäftsfeld ständiger und unerwarteter Veränderungen erfolgreich zu sein. Dabei soll auf die sich verändernden Märkte durch schnelles und effektives Reagieren und die Einbindung des Kunden in den Entwicklungsprozess geantwortet werden (Gunasekaran, 1998), (Elkins, et al., 2004), (Mason-Jones, et al., 1999).

Im Geschäftsprozessmanagement wurden ebenfalls Versuche unternommen um Prozesse agil zu gestalten, die aber trotzdem definierten Prozessregeln entsprechen. Die Ziele des Agilen Prozessmanagement sind (Hinkelmann, et al., 2006):

- Die Strenge der Geschäftsprozessstruktur aufzuweichen, um mit außergewöhnlichen und unvorhersehbaren Situationen, mit stark unterschiedlichen In- und Outputs, mit komplexen Aufgaben, mit innovativen Prozessen und neuen Erfahrungen besser umgehen zu können.

- Kopfarbeitsprozesse besser zu formalisieren zur Unterstützung der Entscheidungsfindung, um den Regeln zu entsprechen, eine Konsistenz in der Entscheidungsfindung zu gewährleisten und um die Arbeit transparent und nachvollziehbar zu machen.

Schatten und Schiefer sehen folgende Punkte als Kernprinzipien des agilen Geschäftsprozessmanagements (Schatten, et al., 2007):

- Den Mitarbeitern mehr Entscheidungsbefugnis geben und Flexibilität und Reaktionsfähigkeit höher gewichten als Effizienz und Vorhersagbarkeit.
- Schnelles und intelligentes Reagieren auf Ereignisse im Geschäftsumfeld.
- Höchste Priorität auf das Erkennen von Kundenanforderungen und das Darauf-Reagieren legen, um ständig Kundennutzen zu liefern.

Fricke und Schulz betrachten die Veränderlichkeit von technischen Systemen und unterscheiden dabei folgende Systemeigenschaften, welche in Abb. 13 dargestellt werden (Fricke, et al., 2005 S. 347):

- Robustheit: Kennzeichnet ein System, das unabhängig von Veränderungen in seiner Umwelt, seine vorgesehene Funktion erfüllt.
- Flexibilität: Kennzeichnet ein System, das leicht verändert werden kann.
- Agilität: Kennzeichnet ein System, das schnell verändert werden kann.
- Anpassungsfähigkeit: Kennzeichnet ein System, das sich an verändernde Umweltbedingungen selbstständig anpassen kann.

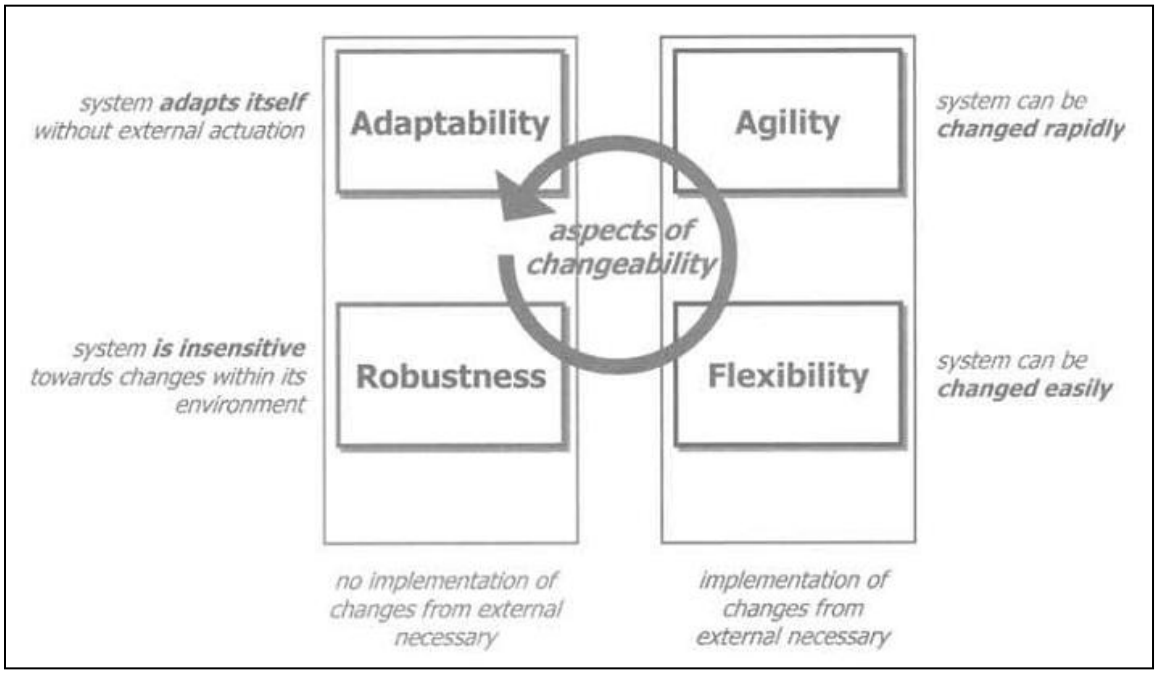


**Abb. 13: Die vier Aspekte der Veränderlichkeit (Fricke, et al., 2005 S. 347)**

In der Definition von Fricke und Schulz finden sich nun Differenzen zur bisherigen Definition von Agilität. Die Unterscheidung von Agilität und Flexibilität durch die Geschwindigkeit erscheint zwar angebracht, sie ist aber nur ein Kriterium von vielen. Die Unterscheidung der Anpassungsfähigkeit von der Agilität durch

den dort nicht mehr benötigten Eingriff von außen, hält aber weder der Analyse der Verwendung des Wortes „Agilität“ im allgemeinen Sprachgebrauch stand, noch ist sie mit Definitionen aus anderen Fachgebieten kompatibel. Wo immer man einer Person (die auch als System gesehen werden kann) Agilität zuschreibt, so ist diese von sich aus zu einer Veränderung/Reaktion fähig. Ist eine Unternehmung agil, so sind es Teile von ihr (z.B. Mitarbeiter der Unternehmung als Subsystem des Systems „Unternehmung“) die für eine Veränderung sorgen. Trotzdem ist die Unterscheidung, ob ein System von außen oder innen verändert wird, für die Betrachtung der Agilität sehr wichtig. Nur soll im Weiteren eben die innere selbsttätige Veränderungsfähigkeit (Anpassungsfähigkeit) als Charakteristik der Agilität gesehen werden. Die Möglichkeit ein System von außen anpassen zu können, soll als „Anpassbarkeit“ bezeichnet werden (Haberfellner, 1973 S. 30f).

Haberfellner und de Weck haben die ständig ansteigende Geschwindigkeit in der Systementwicklung im Auge und vor allem die Unsicherheit der zukünftigen Benutzeranforderungen. Um dem Rechnung zu tragen, betrachten sie die Anwendung von Agilität in Systemen, wie auch für Systementwicklungs-Prozesse. Sie sehen folgende Dinge als zwingend für ein System an, welches agil sein soll (Haberfellner, et al., 2005 S. 12):

- Flexible Elemente innerhalb des Systems, die es ihm erlauben, einfach und schnell verändert werden zu können.
- Sensoren, die Eigenschaften der Umwelt wahrnehmen können, um Entscheidungsträger zu informieren, wenn Veränderungen angebracht sind.
- Einen Entscheidungsmechanismus, der die Situation analysieren und Veränderungen in Gang setzen kann.

Damit kann ein agiles System schnell und ohne größere Umstellungskosten von einem Systemzustand zu einem anderen wechseln, ohne dass die Komplexität erhöht wird. Betrachtet man nicht das System selbst, sondern den Prozess der Systementwicklung dann wird dieser als agil bezeichnet, wenn er:

- flink, gewandt und schnell ist;
- anpassungsfähig und reaktionsfähig auf neue, manchmal unerwartete Informationen, die während der Systementwicklung verfügbar werden, ist und
- Anforderungen und Lösungen nicht so früh wie möglich festlegen will (Haberfellner, et al., 2005 S. 1).

Azad Madni sieht in Produktentwicklungsprozessen Unsicherheiten aufgrund von sich weiterentwickelnden Kundenanforderungen, neuen Gesetzen und sich verändernden Umgebungen und will unter diesen Umständen mit agilen Entwicklungsprozessen ständig die neuesten Technologien ausnutzen und gleichzeitig die Zuverlässigkeit der Produkte sicherstellen. Agile Entwicklungsprozesse werden charakterisiert durch (Madni, 2008 S. 1):

- Flexibilität: Die Fähigkeit auf erwartete Veränderungen zu reagieren.
- Bewusstsein: Fähigkeit den Zusammenhang des Geschehens zu erkennen.
- Effizienz: optimale Ressourcen-Verwendung und kurze Testzyklen.
- Anpassungsfähigkeit: Fähigkeit schnell und kosteneffizient auf unerwartete Veränderungen reagieren zu können.

Charakteristiken für ein agiles Produkt sind (Madni, 2008 S. 1):

- Erweiterbarkeit: Möglichkeit neue Funktionalität hinzu zu fügen.
- Modularität: Möglichkeit Module auszutauschen, ohne die Integrität des Systems zu gefährden.
- Wiederherstellbarkeit: Möglichkeit im Falle eines Fehlers Module auszutauschen und wieder zusammenzufügen.

Mit agilen Prozessen soll Flexibilität und Geschwindigkeit in die Entwicklung gebracht werden und der Raum für möglichst viele Lösungsvarianten möglichst lange offen gehalten werden. Ein agiles Produkt/System hingegen ist anzustreben, wenn es nicht möglich ist die Anforderungen der Zukunft ausreichend genau vorherzusagen (Madni, 2008 S. 2).

Die wohl bekannteste Definition für die Anwendung von Agilität in einem Fachgebiet lieferte das Manifest für Agile Software Entwicklung im Jahr 2001 (Beck, et al., 2001). Der genaue Wortlaut und die verantwortlichen Autoren wurden bereits in Abb. 9 im Kapitel 2.2 wiedergegeben.

Dieses agile Manifest wurde als Reaktion auf herkömmliche Softwareentwicklungsprozesse formuliert, die oft als schwerfällig, langwierig, unzweckmäßig und weder kunden- noch entwicklerfreundlich galten. Mithilfe von verschiedenen agilen Softwareentwicklungsmethoden soll Software entwickelt werden, die den Kundenanforderungen besser entspricht und die Entwickler nicht mit der Einhaltung von unnötigen Prozesserfordernissen belastet. Als kleinster gemeinsamer Nenner für die Definition und das Ziel der agilen Softwareentwicklung wurde eben dieses agile Manifest erstellt. Die Autoren der einzelnen agilen Methoden und ihre Anwender weichen in ihren Sichtweisen über die Definition der Agilität aber trotzdem voneinander ab. Von manchen Autoren werden diese Methoden auch nicht als vollständige Prozessmodelle akzeptiert, sondern eher als Sammlungen von Techniken und Aktivitäten gesehen (Fritzsche, et al., 2007 S. 1). Bei den einzelnen Techniken muss dann separat wieder überprüft werden, inwiefern diese Agilität repräsentieren oder fördern. Das soll die Überlegungen zu Definition und Ziel der Agilität aber nicht weiter beeinflussen und wird daher in Kapitel 5 durchgeführt, wo die Praktiken und Prinzipien, die Agilität ins Systems Engineering bringen können, analysiert werden. Hier sollen aber noch einige Vertreter der agile Softwareentwicklungsmethoden mit ihren Definitionen der Agilität zu Wort kommen um Möglichkeiten, die das Prinzip Agilität bietet, darzustellen und dann daraus eine endgültige Charakterisierung für „Agile Systems Engineering“ zu entwerfen.

Einer der erste Autoren, der von Agilität in der Softwareentwicklung gesprochen hat, war Mikio Aoyama. Sein 1998 vorgestellter „Agile Software Process” soll die Entwicklungszeit von Software durch die Integration von leichtgewichtigen, modularen, inkrementellen und iterativen Prozessen verkürzen (Aoyama, 1998 S. 3). Der Prozess ist aber auch ausgerichtet auf schnelle und flexible Anpassung bei Veränderungen des Prozesses, des Produkts und der Umgebung (Aoyama, 1998 S. 4).

Für Barry Boehm sind agile Methoden charakterisiert durch leichtgewichtige (mit wenig bürokratischem Aufwand verbundene) Prozesse mit kurzen iterativen Zyklen. Anwender werden aktiv involviert. Das Wissen wird im Team verbreitet ohne es zu dokumentieren und das Team organisiert sich selbst. Prozesse, Prinzipien und Arbeitsstrukturen sind nicht vorherbestimmt, sondern entstehen oder ändern sich während des Projekts. Weitere wichtige Prinzipien der Agilität sind für Boehm die positive Grundeinstellung der Entwickler gegenüber Veränderungen, einfaches Design und experimentelles Entwickeln (Boehm, et al., 2006 S. 17ff).

Qumer und Henderson-Sellers finden bei der Betrachtung von Softwareentwicklungsmethoden die folgende Definition von Agilität (Qumer, et al., 2006 S. 505), (Qumer, et al., 2007 S. 2): „Agilität ist die anhaltende Eigenschaft oder Fähigkeit eines empfindsamen Wesens, das Flexibilität zeigt, um erwarteten oder unerwarteten Veränderungen unmittelbar Rechnung zu tragen. Es macht dies schnellstmöglich mit ökonomischen, einfachen und qualitativ hochwertigen Mitteln in einem dynamischen Umfeld, wendet dabei vorhandenes Wissen und Erfahrung an und lernt vom internen und externen Umfeld.“ Eine Softwareentwicklungsmethode wird als agil betrachtet wenn sie folgenden Charakteristika entspricht (Qumer, et al., 2007 S. 2):

- Personen- und Kommunikationsbezogen
- Flexibel: Fähig sich jederzeit erwarteten oder unerwarteten Veränderungen anzupassen.
- Schnell: Unterstützt schnelles und iteratives Entwickeln des Produktes.
- Lean: Reduziert Entwicklungszeit und Kosten und verbessert Qualität.
- Reaktionsfähig: Reagiert angemessen auf erwartete oder unerwartete Veränderungen.
- Lernfähig: Unterstützt Verbesserungen während und nach der Produktentwicklung.

Richard Turner schlägt vor, die Prinzipien der agilen Softwareentwicklung im Systems Engineering anzuwenden. Das Systems Engineering soll dadurch dieselben Vorteile erfahren, die auch für die Softwareentwicklung versprochen werden. Diese sind kürzere Lieferzeiten, bessere Termintreue und höhere Kundenzufriedenheit. Außerdem sollen dadurch kurzfristige Veränderungen, höhere Komplexität und plötzlich neu auftauchende Anforderungen besser beherrscht werden (Turner, 2007).

Ein anderer Vorschlag für ein agiles Systems Engineering kommt von Wilson und Mooz. Sie definieren es als schnelles Benutzer- und Stakeholder- Anforderungsmanagement, welches eine Konzeptauswahl, eine Architekturentwicklung, Systemintegration, Verifikation und Validation enthält und in einer Entwicklungsumgebung stattfindet, die durch schnelle Anpassung an Veränderungen, ein nicht-hierarchisches Baseline-Management und die Abwesenheit von wertloser Bürokratie gekennzeichnet ist (Wilson, et al., 2003 S. 793).

Das S²QI Expertenpanel versucht die Agilität im SE hauptsächlich durch die Anwendung von agilen Softwareentwicklungsmethoden zu steigern und versucht Lösungen für die diesbezüglichen Herausforderungen zu finden (Stelzmann, et al., 2010). Die Ziele die erreicht werden sollen, sind primär eine Erhöhung der Entwicklungsgeschwindigkeit und eine bessere Kommunikation zwischen Entwicklern und Endkunden. Aber auch andere, wie eine Bewältigung der Variantenvielfalt oder von ausufernder Dokumentation. Naturgemäß sind Vertreter der Industrie mehr an der Lösung ihrer Probleme, als an einer exakten Definition der Agilität interessiert. So waren diesen neben einer ausreichenden Flexibilität vor allem Eigenschaften wichtig, die man mit Effizienz und Effektivität zusammenfassen kann.

Wie zu sehen war, gibt es durchaus unterschiedliche Vorstellungen von „Agilität“ und „Agile Systems Engineering“. Befasst man sich mit dieser Thematik ist deshalb immer die Begriffsdefinition des betrachteten Autors zu berücksichtigen. In dieser Arbeit wurde „Agile Systems Engineering“ als Lösungsansatz für die Problemstellung der Dissertation definiert. Will man in Umgebungen mit starken und häufigen, vorhersehbaren und unvorhersehbaren Veränderungen, in effizienter und effektiver Weise, plan- und vorhersehbar, erfolgreiche Systeme herstellen, so können die in Abb. 14 aufgezeigten

Charakteristika als Anforderungen für das SE gesehen werden. Man kann diese Charakteristika auch als Teilfunktionen der Agilität im Systems Engineering verstehen.

| Charakteristika eines agilen Systems Engineerings | Unterschiede zur traditionellen SE Methodik |
|---|---|
| **Situationskenntnis:** Durch Agile SE wird laufend ein schnelles Erkennen von neuen, relevanten Informationen und Veränderungen forciert. Durch frühes Berücksichtigen können spätere Änderungen damit ev. auch vermieden werden. | Die Situation wird nicht nur zu Beginn sondern laufend betrachtet. Es ist zulässig und leicht möglich Situationsänderungen einzubeziehen. |
| **Kontinuierliche Verbesserung:** Der Entwicklungsprozess wird als laufende Verbesserung des Systems gesehen. Ständiges Lernen wird als unumgänglich betrachtet und forciert. Neu erhaltene Informationen werden umgehend zur Verbesserung von Prozessen und Produkten genutzt. | Bisher eher nur in Lessons Learned Workshops nach Abschluss der Projekte umgesetzt. Bei ASE wird davon ausgegangen, dass Entwicklung ein ständiges Erlernen von neuem Wissen über das System ist und dass dieses neue Wissen sofort in Prozesse, Organisation und Produkte einfließen soll. |
| **Bewältigung von Veränderungen:** Agile SE Prozesse sind ausgelegt für einen effektiven und effizienten Umgang mit Veränderungen. | Veränderungen werden nicht als unangenehme Ereignisse oder gar als Fehler gesehen oder so behandelt, sondern als normaler Bestandteil jeder Entwicklung. Man ist auf laufende Änderungen vorbereitet. |
| **Flexibilität:** Sollten neue Informationen es erforderlich machen, können Prozesse und Organisation leicht und schnell angepasst werden. Flexibilität in den Produkten wird forciert, um Anpassbarkeit zu gewährleisten, falls diese während der Entwicklung oder in der Nutzungsphase notwendig wird. | Bisher wurde im Vorhinein entschieden, welches Prozessmodell und welche Organisation verwendet werden sollen. Flexibilität in Produkten wurde nur dann betrachtet, wenn dies explizit im Vorhinein gefordert wurde. |
| **Effizienz:** Agile SE ist um kurze Entwicklungszeiten (schnellstmögliche Bereitstellung von Kundennutzen) und Kosteneffizienz bemüht. Im Prozess sollen keine unnützen Tätigkeiten vorhanden sein und im Produkt keine unnützen Funktionen (Lean). | Das Argument wird bei ASE bewusst darauf gelegt, aktiv betrieben und laufend überprüft und verbessert. |
| **Effektivität:** Agile SE ist zweckmäßiges, zielgerichtetes und am Kundennutzen orientiertes Arbeiten (Qualität). | Nicht nur die Erfüllung vertraglich vereinbarter Anforderungen, sondern der tatsächliche Kundennutzen wird verstärkt betrachtet und seine Erfüllung laufend überprüft. |

**Abb. 14: Charakteristika des Agile Systems Engineerings**

Damit ist ASE für die vorliegende Dissertation ausreichend beschrieben und es ist festgelegt, welche Funktionalität der zu erstellenden ASE Ansatz erfüllen soll. Andere Charakteristika, wie Kommunikation, weitere mitarbeiterbezogene Prinzipien oder die iterative Entwicklung wurden bewusst nicht explizit erwähnt. Diese Punkte können nämlich als konkrete Maßnahmen zur Umsetzung gesehen werden. Die in

Abb. 14 formulierte Charakteristik soll so lösungsneutral wie möglich „Zieleigenschaften“ für Agile Systems Engineering aufzählen, um danach erst nach brauchbaren Lösungen dafür zu suchen.

Ohne genaue Charakterisierung und mit nur wenigen Worten kann man Agile Systems Engineering auch als „Generierung von Werten durch Beherrschung von Veränderungen“ bezeichnen (McMahon, 2006 S. 30). Diese „Generierung von Werten“ ist auch das, was für den Kunden zählt. Deshalb kann ein agiles Systems Engineering auch nicht durch eine Reihe von Praktiken und Prinzipien oder die Anwendung einer Methode definiert werden. Mit „Beherrschung von Veränderungen“ soll zum Ausdruck gebracht werden, dass ein agiles Systems Engineering Veränderungen als unvermeidbar ansieht und deshalb versuchen muss, die Kosten dieser Veränderungen so niedrig wie möglich zu halten, bzw. den Projekterfolg trotz der Veränderungen sicherzustellen.

## 4.3 Allgemeines Funktionsprinzip agiler Entwicklung

Agilität ist also eine Fähigkeit und ASE ist ein Systems Engineering, welches seine Anwender darin unterstützt, in der Entwicklung agil zu sein. Nachdem die abstrakten Definitionen gefunden wurden, soll nun noch gezeigt werden, welches konkrete Prinzip gewöhnlich mit dem Begriff der „Agilität“ bezeichnet wird. Dieser Begriff wurde speziell in der Softwareentwicklung geprägt und es muss besonders betont werden, dass das damit bezeichnete Prinzip nur die **so genannte** „agile Entwicklung“ darstellt. In der Softwareentwicklung wird „Agilität“ nämlich als Namen für Methoden verwendet, welche mehr oder weniger mit der eigentlich Bedeutung des Wortes „Agilität“ zu tun haben können. Das im Folgenden untersuchte Prinzip ist also nicht vorbehaltlos als Lösungsansatz für die Problemstellung dieser Arbeit zu sehen, da darin die Agilität nicht als die Anwendung eines bestimmten Prinzips oder einer bestimmten Methode sondern als Fähigkeit gesehen wird, mit dem Ziel diese zu steigen. Um Möglichkeiten zur Steigerung der Agilität im Allgemeinen zu finden, soll die „agile“ Softwareentwicklung aber natürlich genau untersucht werden. Die in Kapitel 2.2 dargestellten agilen Softwareentwicklungsmethoden haben auch erkennen lassen, dass durchaus eine konkrete gemeinsame Vorstellung von „agiler Entwicklung“ existiert. Es soll nun mithilfe von Modellen und Konzepten diese „agile Entwicklung“ analysiert werden und versucht werden, ein gemeinsames zugrundeliegendes Funktionsprinzip zu identifizieren.

### 4.3.1 Der OODA Loop

Um darzustellen was Agilität konkret bedeutet, wird häufig der OODA Loop verwendet (Adolph, 2006), (Boehm, et al., 2006). Der Observe-Orient-Decide-Act (OODA) Loop wurde vom Kampfpiloten und Militärstrategen John Boyd ursprünglich entworfen um darzustellen, wie bei einem Luftkampf schnelle Entscheidungen als Reaktion auf verändernde Bedingungen getroffen werden (Boyd, 1976). Die Erkenntnisse Boyds haben dazu geführt, dass die Entwicklung von Kampflugzeugen auf die Erreichung einer höheren Agilität ausgelegt wurde. Der ursprüngliche OODA-Loop ist in Abb. 15 zu sehen. Die vier Phasen des OODA Loops sind:

1. **Observe** (Wahrnehmen): In der ersten Phase werden Informationen aus der Außenwelt wahrgenommen. Dies können unabhängig Ereignisse oder auch Reaktionen auf Aktionen sein, die der Beobachter zuvor durchgeführt hat.
2. **Orient** (Zurechtfinden/Analysieren/Einordnen): Die wahrgenommene Situation wird hierauf einer Analyse unterzogen, interpretiert und es werden Handlungsalternativen erarbeitet. Dabei

werden nicht nur die neuen Informationen analysiert. Auch vorherige Erfahrungen, die eigene Herkunft und kulturelle Traditionen haben einen Einfluss darauf, welche Handlungsalternativen entwickelt werden.

3. **Decide** (Entscheiden): Im nächsten Schritt wird entschieden, welche Handlungsalternative verfolgt werden soll, bzw. ob überhaupt eine Aktion stattfinden soll.
4. **Act** (Agieren): Die Entscheidung wird in die Tat umgesetzt.

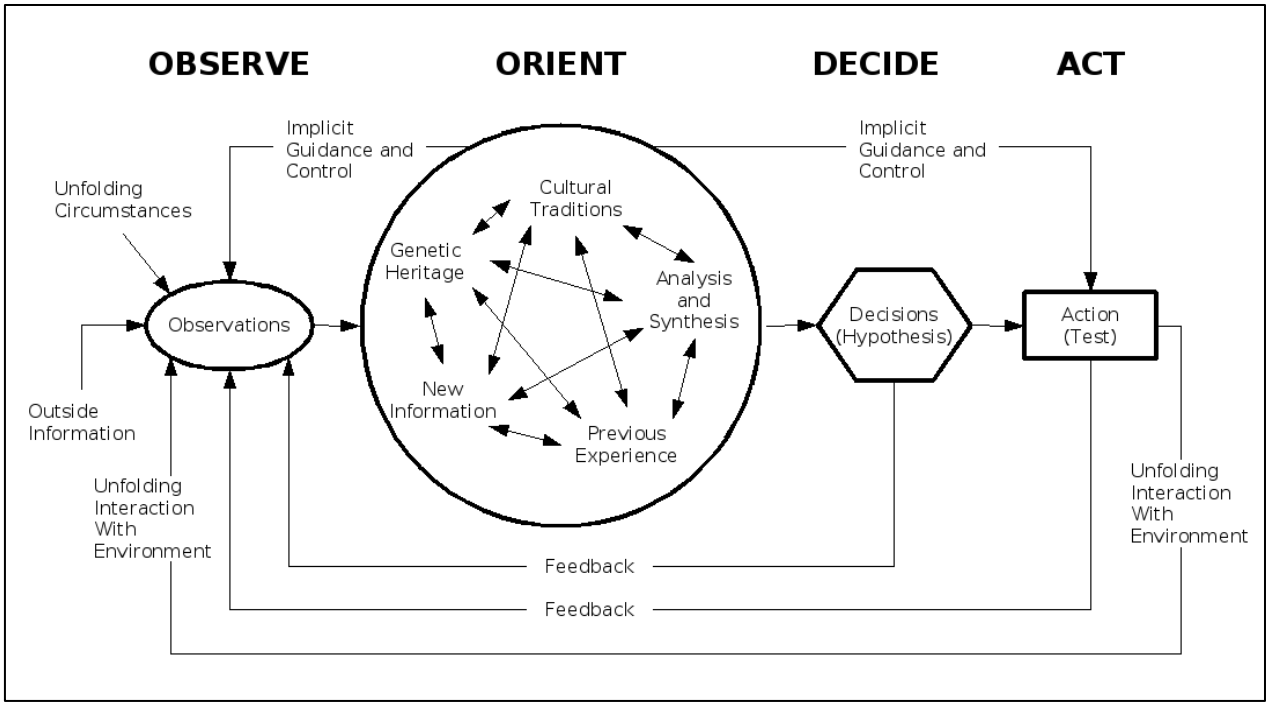


**Abb. 15: Der OODA Loop (Boyd, 1996)**

Der OODA Loop kann damit nicht darstellen, was Agilität ist. Er kann anhand der vier Phasen aber analysieren warum ein Pilot agiler handeln kann, als ein anderer. Um die Agilität zu steigern, können dann gezielte Maßnahmen gesetzt werden. Z.B. kann ein neues Flugzeug für einen besseren Überblick des Piloten gestaltet werden, wenn die Observe-Phase zu lange dauerte. Dazu betrachtet Boyd nicht nur die Phasen selbst, sondern auch die komplexen Wechselwirkungen zwischen ihnen. Boyd weist besonders auf den großen Einfluss der Orient-Phase auf alle anderen hin, aber auch darauf, wie das Feedback wiederum auf die Orient-Phase zurückwirkt. Dazu tätigt er folgende Aussagen (Boyd, 1996):

- Ohne unsere Herkunft, kulturellen Traditionen und vorherigen Erfahrungen mit Veränderungen (neuen Situationen, unbekannten Phänomenen) können wir kein implizites Repertoire an psychologischen Fähigkeiten aufbauen.
- Ohne Analyse und Synthese von verschiedenen Bereichen oder von unterschiedlichen unabhängigen Informationsquellen können wir keine neuen Repertoires für den Umgang mit unbekannten Phänomenen und unvorhergesehenen Veränderungen entwickeln.
- Ohne den vielseitigen, impliziten Prozess von Vorhersage, Einfühlung, Zuordnung und Verwerfung (mit Querbeziehungen über all diese Bereiche) können wir keine Analyse und Synthese durchführen.

- Ohne OODA Loops können wir weder ausreichend viele Informationen für die oben genannten Prozesse sammeln, noch im Zusammenhang mit diesen Prozessen entscheiden, noch Aktionen umsetzen.
- Ohne OODA Loops, die obigen Aussagen und die Fähigkeit, den OODA Loop schneller durchzuführen als Konkurrenten, kann man nicht verstehen, gestalten oder sich an verändernde Realitäten anpassen, die unsicher, sich ständig verändernd und unvorhersehbar sind.

Aus diesen Aussagen kann man erkennen, wie komplex die Zusammenhänge und Einflussfaktoren im Umgang mit Veränderungen sind. Sie stehen auch im Einklang mit den Forderungen der agilen Softwareentwicklungsmethoden nach einer Ausrichtung der Methoden auf personenbezogene Aspekte. Mehr noch weisen sie aber auf die ständig ablaufenden Zyklen von Beobachtungen, Analysen und Handlungen hin und auf die Wichtigkeit des erhaltenen Feedbacks.

Außerdem erlaubt der OODA Loop auch die Festlegung eines gewünschten Mindestwerts an Agilität. Bei einem Kampfpilot wäre dieser, dass sein eigener OODA-Loop schneller abläuft, als der des Gegners. Ein Entwicklungsprojekt könnte demnach als agil bezeichnet werden, wenn der OODA Loop im Projekt (Neuorientieren und Handeln der Entwickler) schneller abläuft, als Veränderungen im Umfeld passieren (Adolph, 2006 S. 4).

Für die Lebenszyklus- oder Entwicklungsphasen von Systemen hat Boehm das in Abb. 16 gezeigte Modell des OODA Loops abgewandelt. Darin wird dargestellt, mit welchen Methoden ein agiles Entwicklungsteam die ersten 3 Phasen durchführt, während ein Systeminkrement schon realisiert und zur Generierung von Feedback genutzt wird (Boehm, et al., 2006 S. 7). Der OODA Loop kann also dazu genutzt werden, um Entwicklungstätigkeiten hinsichtlich Agilität zu analysieren und zu optimieren. Die vom OODA Loop beschriebene Agilität lässt einen wesentlichen Zusammenhang mit einer zyklischen Vorgehensweise und einer Verwendung von Feedback erkennen.

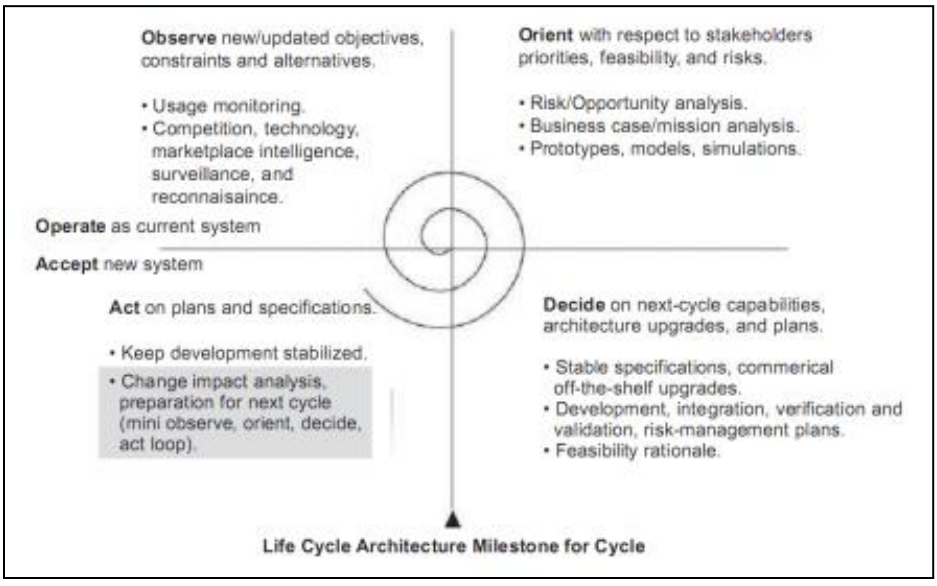


**Abb. 16: OODA Loop für Entwicklungsphasen (Boehm, et al., 2006 S. 6)**

### 4.3.2 Das Spiralmodell

Ein Modell, welches von vielen Autoren im Bereich der Agilität genannt wird (Wilson, et al., 2003 S. 797), (Haberfellner, et al., 2005 S. 5), ist das von Boehm in den 80er Jahren entwickelte Spiralmodell (Boehm, 1988). Dieses Modell eines zyklischen Entwicklungsprozesseses zeigt für Software, in welchen Phasen einzelne Inkremente (Prototypen) entwickelt werden (siehe Abb. 17).

Auch wenn dieses Modell ursprünglich zur schrittweisen Reduktion des Risikos von Entwicklungsprojekten entworfen wurde (Boehm, 1988), ist nachvollziehbar, wieso es auch die Agilität von Entwicklungsprozessen erhöhen kann. Wie beim OODA Loop findet darin ein laufender Zyklus aus Analysieren, Umsetzen, Testen und Einholen von Feedback zur weiteren Richtungsbestimmung statt.

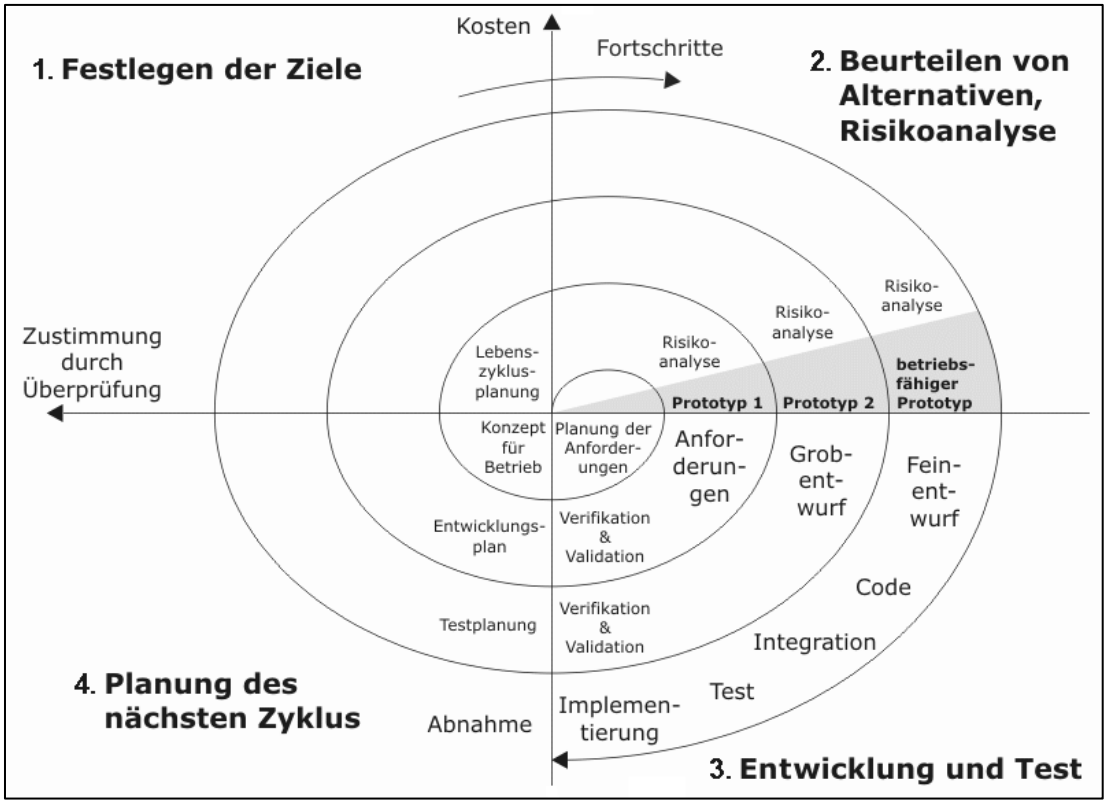


**Abb. 17: Spiralmodell (Boehm, 1988) (Grafik: de.wikipedia.org)**

### 4.3.3 Iterative and Incremental Development

Nachdem in praktisch allen agilen Softwareentwicklungsmethoden (2.2) eine zyklische Entwicklung in kleinen Schritten vorgesehen ist, zeichnet sich diese als kleinster gemeinsamer Nenner des Verständnisses der agilen Entwicklung ab. In der Literatur zur Softwareentwicklung findet man auch die Bezeichnung „Iterative and Incremental Development (IID)“ (Larman, et al., 2003). Mit „Iterativer Entwicklung“ wird die Wiederholung des Prozesses bezeichnet, unter „Inkrementeller Entwicklung“ versteht man die Erweiterung des Systems in kleinen Portionen (Kaindl, 2005 S. 3). Eine genauere Analyse dieser Vorgehensweisen in der Entwicklung findet in Kapitel 5.4.3 statt.

In der Literatur wird ein iteratives oder inkrementelles Entwickeln einerseits als Teil oder Charakteristik der agilen Methoden gesehen (Turner, 2007 S. 12), andererseits wird IID aber auch als Überbegriff

verwendet, unter dem die agilen Methoden, das Spiralmodell und andere zusammengefasst werden (Larman, et al., 2003 S. 51, 54). In der vorliegenden Dissertation wird nicht weiter auf diese Eingliederungen eingegangen. Es wurde damit aber gezeigt, dass es ein weit verbreitetes Verständnis für die praktische Durchführung agiler Entwicklung gibt und dieses im Zusammenhang mit einer zyklischen Vorgehensweise und Einholung von Feedback zu Verbesserungszwecken steht. Die meisten dieser Modelle sind durch die Softwareentwicklung geprägt und unterscheiden sich leicht. Deshalb soll für diese Arbeit noch ein allgemeines Modell agiler Entwicklung gestaltet werden, das mehr oder weniger eine Zusammenfassung all dieser Modelle darstellen und alle Zusammenhänge und Details erklären soll.

### 4.3.4 Der Agile Entwicklungszyklus (AEZ)

Der kleinste gemeinsame Nenner der unterschiedlichen Auffassungen einer agilen Entwicklung ist also eine zyklische Entwicklung in kleinen Schritten mit Feedbackschleifen. Diese muss aber einigen Regeln folgen, damit sie ihre Funktion sinngemäß erfüllen kann. Deshalb wurde für die vorliegende Arbeit der „Agile Entwicklungszyklus“ entworfen, der die zyklische Entwicklung mit allen Zusammenhängen beschreiben soll, welche Agilität fördern. Er zeigt das grundlegende Wirkprinzip auf, das hinter den eben beschriebenen Modellen und Methoden der agilen Softwareentwicklung steckt und für einen besseren Umgang mit Veränderungen sorgen soll. Er sammelt auch die vielfältigen Regeln aus den Methoden der agilen SW Entwicklung zusammen und führt sie auf übergeordnete Zusammenhänge und Prinzipien zurück. Wie in Abb. 18 gezeigt, besteht die Grundstruktur des AEZ aus 4 Phasen, die eine Anlehnung an den OODA Loop erkennen lassen. Literaturhinweise werden im Folgenden immer beispielhaft für Methoden angegeben, aus denen die Prinzipien extrahiert wurden.

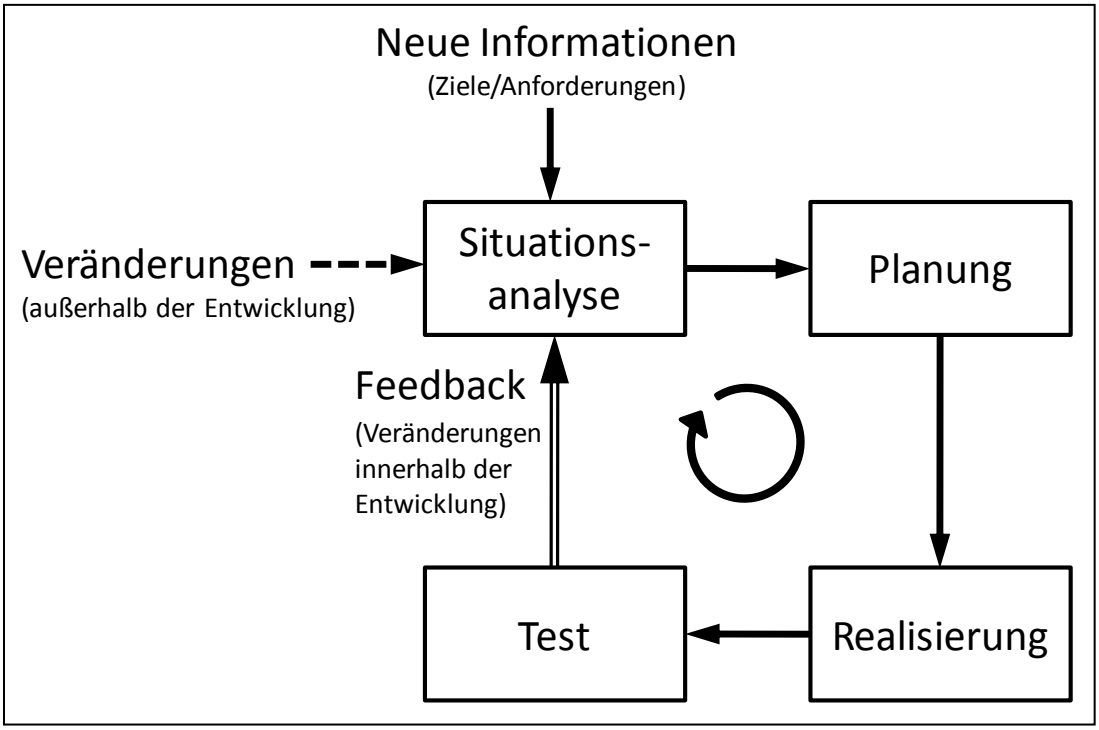


**Abb. 18: Der agile Entwicklungszyklus (AEZ)**

Die 4 Phasen des Entwicklungszyklus sind folgendermaßen zu interpretieren:

1. **Situationsanalyse:** Der agile Entwicklungszyklus beinhaltet eine Phase, in der neue Informationen und Veränderungen von außerhalb und innerhalb der Entwicklung (Feedback) wahrgenommen werden. Als relevant erkannte Informationen werden in die Entwicklung einbezogen (Schwaber, 2004 S. 6), (Boyd, 1996).
2. **Planung:** Jeder Entwicklungsschritt wird zumindest im Detail separat geplant (Schwaber, 2004 S. 6), (Cockburn, 2005 S. 163), (Poppendieck, et al., 2009 S. 186). Unter „Planung“ wird hier nicht nur die organisatorische, sondern auch die technische Planung bzw. der Entwurf des Systems verstanden.
3. **Realisierung:** Im Zyklus ist auch die praktische Umsetzung bzw. die Herstellung (eines Teils) des Systems enthalten (Schwaber, 2004 S. 6), (Poppendieck, et al., 2009 S. 186ff), (Boyd, 1996).
4. **Test:** Jedes Ergebnis eines Entwicklungsschritts wird getestet bzw. vorgeführt, um darauf Feedback zu erhalten (Beck, et al., 2005 S. 97ff), (Schwaber, 2004 S. 6), (Poppendieck, et al., 2009 S. 188f).

Die Hauptfunktion, welche der AEZ umsetzen soll, kann mit „**kontinuierliche Verbesserung**“ bezeichnet werden. Der AEZ geht davon aus, dass man eine Systementwicklung nicht im Vorhinein vollständig planen kann. Während der Entwicklung findet ein laufendes Generieren von neuen Informationen und Lernen statt. Der AEZ sieht damit die Entwicklung als ständig laufenden Prozess von Verbesserungen, wie sie auch in den agilen Softwareentwicklungsmethoden beschrieben wurde (Beck, et al., 2005 S. 28, 51), (Poppendieck, et al., 2009 S. 29ff).

Das Vorgehen in den vier beschriebenen Phasen soll dazu nach den im Folgenden genannten **Prinzipien** umgesetzt werden. Es werden dabei jeweils Querverweise zu den genaueren Beschreibungen dieser Prinzipien in Kapitel 5 genannt. Dort werden diese auch auf die Möglichkeit hin geprüft, durch separate Anwendung im Systems Engineering die Agilität steigern zu können.

- **Entwicklung in kleinen Schritten (5.4.3):** Der Entwicklungsumfang wird in kleine Schritte (Teilumfänge) aufgeteilt. Die Aufteilung soll dabei so erfolgen, dass die bereits entwickelten Teile zur Informationsgewinnung für den weiteren Entwicklungsablauf genutzt werden können (Beck, et al., 2005 S. 33, 62), (Poppendieck, et al., 2009 S. 183ff), (Gilb, 2005 S. 187).
- **Flexibles Anforderungsmanagement (5.1.4):** Die Produktanforderungen müssen nicht zu Beginn der Entwicklung präzise und definitiv formuliert vorliegen. Es wird davon ausgegangen, dass Anforderungsänderungen unvermeidlich sind und deshalb ein flexibles Handhaben der Anforderungen vorgesehen (Beck, et al., 2005 S. 44f), (Schwaber, 2004 S. 7ff)**.**
- **Zyklische Entwicklung (5.4.3):** Für jeden Entwicklungsschritt wird ein vollständiger Entwicklungszyklus durchgeführt (Schwaber, 2004 S. 5f) (Cockburn, 2005 S. 163), d.h. es wird analysiert, geplant, umgesetzt und getestet. Eine nicht zyklische Entwicklung wäre es, wenn man z.B. zuerst die Analyse für alle Entwicklungsschritte macht, danach die Planung für alle, usw. Sollte ein Zyklus während eines Durchlaufs aufgrund einer neuen Erkenntnis plötzlich sinnlos erscheinen, kann er aber natürlich abgebrochen werden.
- **Priorisierung der Entwicklungsschritte (5.1.5):** Die Bestimmung des Inhalts der Entwicklungsschritte erfolgt nicht willkürlich, sondern wird nach Priorität geplant durchgeführt,

z.B. sollen Entwicklungsinhalte, die einen großen Erkenntnisgewinn oder Nutzen erwarten lassen, zuerst umgesetzt werden (Schwaber, 2004 S. 8), (Gilb, 2005 S. 2ff).

- **Einbindung des Kunden (5.1.3):** Um sicherzustellen, dass die Entwicklungsergebnisse den Kundenwünschen entsprechen, sollen die Tests immer unter Beteiligung des Kunden durchgeführt werden (Beck, et al., 2005 S. 61).
- **Nutzbare Zwischenergebnisse (5.4.3):** Der Kunde soll in die Lage versetzt werden, den Entwicklungsstand der Zwischenergebnisse in seiner vorgesehenen Anwendungen hinsichtlich der Nutzbarkeit prüfen zu können. Beim „Ausprobieren" des Produkts oder eines Zwischenergebnisses kann der Kunde oder spätere Endanwender nämlich am besten beurteilen ob es den Anforderungen entspricht. Deshalb soll jeder realisierte Entwicklungsschritt auch in einer nutzbaren Form gestaltet werden (Schwaber, 2004 S. 12f).
- **Feedback (5.1.3):** Dieses wird als wesentlicher Input für den weiteren Verlauf der Entwicklung gesehen (Beck, et al., 2005 S. 18), (Poppendieck, et al., 2009 S. 177ff), (Gilb, 2005 S. 3) , (Boyd, 1996).
- **Expliziter Prozess (5.4.3)**: Der AEZ ist ein festgelegter, geplanter Prozess und wird keinesfalls chaotisch oder improvisiert durchgeführt (Schwaber, 2004 S. 37).
- **Periodizität (5.4.3):** Die Zyklen sollen in gleichbleibenden Zeitdauern durchgeführt werden, wobei diese möglichst kurz sein sollen. In der Softwareentwicklung wird von 2-4 Wochen gesprochen (Beck, et al., 2005 S. 46), (Schwaber, 2004 S. 7f).
- **Timeboxing (5.4.8):** Wird die vorgegebene Zykluszeit überschritten, ohne dass die geplanten Aufgaben erledigt sind, wird der Zyklus dennoch beendet und die bisher erzielten Ergebnisse werden präsentiert (Schwaber, 2004 S. 136), (Oesterreich, 2006).

Kriterien für den AEZ sind außerdem, wie bei der Periodizität schon angedeutet, die **Größe der Schritte** bzw. die **Zeitdauer der Zyklen**, die mit „klein" und mit dem Bsp. von 2-4 Wochen als „kurz" betrachtet werden können. Eine genaue Quantifizierung hängt von zahlreichen Faktoren ab (im Wesentlichen von der Umsetzbarkeit der anderen Prinzipien) und wird in 5.4.3 diskutiert.

Mit der Durchführung der 4 Phasen nach den genannten Prinzipien schlägt der AEZ auf prinzipieller Ebene eine agile Vorgehensweise für die Entwicklung vor. Man kann ihn auch als Methode (hinsichtlich der Begriffsbestimmung in Kapitel 1.5) zur Gestaltung von Entwicklungsprozessen, welche Agilität fördern sollen, bezeichnen. Für die Anwendung in einem Agile Systems Engineering scheint der AEZ auch durch die Umsetzung oder Förderung der Teilfunktionen der Agilität vielversprechend zu sein, die im Folgenden aufgezeigt werden. Eine genauere Begründung der Vorteile erfolgt in Kapitel 5 bei der detaillierten Erklärung der einzelnen Prinzipien und Praktiken.

- Situationskenntnis: Durch das zyklische Auftreten der Situationsanalyse forciert der AEZ ein wiederholtes Aktualisieren der Situationskenntnis. Die tatsächlichen Kundenanforderungen (siehe 5.1.1) werden mithilfe des Feedbacks besser verstanden. Aber auch das Überprüfen der Umgebung hinsichtlich neuer Informationen oder Veränderungen wird forciert. Durch das frühe Realisieren und Testen von Zwischenergebnissen, kann auch früh auf das spätere Systemverhalten geschlossen werden.
- Kontinuierliche Verbesserung: Der AEZ ist eine praktische Umsetzung dieses Prinzips. Durch ihn soll das Entwicklungsteam laufend lernen und das System laufend verbessert werden. Sogar bei

einem Abbruch des Projekts sollte, durch das Abwickeln eines oder mehrerer Entwicklungsschritte, ein Lernen in der Unternehmung oder zumindest im Entwicklungsteam möglich gewesen sein.

- Bewältigung von Veränderungen: Wie in Abb. 18 gezeigt, bietet der AEZ in jedem Zyklus die Möglichkeit zur Einbringung von Änderungen. Damit erleichtert er den Umgang mit unklaren oder veränderlichen Anforderungen. Die Umsetzbarkeit dieser Änderungen zum jeweiligen Zeitpunkt der Einbringung wird aber natürlich auch noch von vielen weiteren Faktoren beeinflusst.
- Flexibilität: Diese Teilfunktion des ASE wird vom AEZ nicht wesentlich gesteigert. In gewisser Weise wird der Prozess auch flexibler gestaltet, da nach jedem Zyklus Änderungen durchgeführt werden können. Der AEZ sorgt aber weniger für Flexibilität, als er davon ausgeht, dass eine hohe Flexibilität im Produkt vorhanden ist.
- Effizienz: Durch das frühe und häufige Testen werden Fehler schneller erkannt, was ihre Kosten verringern sollte. Dadurch, dass die Detailplanung nicht im Voraus, sondern für jeden Zyklus separat durchgeführt wird, kann auch die Effizienz der Planung verbessert werden. Ebenso kann von Zyklus zu Zyklus Optimierungspotential in der Prozessgestaltung zur Effizienzsteigerung aufgezeigt werden. Außerdem kann mit den häufigen Deadlines (jedes Zyklusende) positiver Druck auf die Entwickler ausgeübt werden, was die Effizienz ebenso steigern sollte. Details dazu werden in 5.4.3 erklärt.
- Effektivität: Nachdem das Feedback des Kunden für die weitere Entwicklung genutzt wird und diese dadurch in die richtige Richtung gelenkt wird, sollte das Entwicklungsergebnis am Schluss genau den Kundenforderungen entsprechen. Auch die Abbruchsmöglichkeit, sobald der vorgeführte Entwicklungsstand den Kundenwünschen entspricht, kann als Maßnahme zur Steigerung der Effektivität gesehen werden. Ebenso besteht der Vorteil einer Risikoreduktion: Durch das frühe und häufige Überprüfen des Entwicklungsstandes, beschränkt sich das Risiko für den Fall eines Abbruchs auf die bereits umgesetzten Zyklen.

Natürlich bietet der AEZ nicht nur Vorteile, sondern hat auch Nachteile bzw. unterliegt einigen wesentlichen Risiken. Diese werden separat für die einzelnen Prinzipien und Praktiken in Kapitel 5 analysiert. In Kapitel 6 wird gezeigt, wieso sich bereits in der SW Entwicklung die agilen Methoden (und damit der AEZ) nur unter bestimmten Voraussetzungen eignen und es wird auch nach zusätzlichen Voraussetzungen für eine allgemeine Anwendung im Systems Engineering gesucht. In Kapitel 6.4 werden Abhilfemaßnahmen erklärt, die den AEZ zumindest teilweise umsetzbar machen sollen, wenn bestimmte Voraussetzungen nicht gegeben sind. In Kapitel 7.5.2 wird dann aus dem AEZ und unter Zuhilfenahme der Maßnahmen aus 6.4 ein für SE anwendbares Vorgehensmodell entworfen.

## 4.4 Zusammenfassung

Das Fazit dieses Kapitels beantwortet die erste Forschungsfrage nach dem Wesen der Agilität:

**Agilität** wurde als eine Fähigkeit identifiziert, welche einen effektiven und effizienten Umgang mit Veränderungen erlaubt (genauere Charakterisierung unter 4.1).

**Agile Systems Engineering** ist eine SE Methodik, welche Agilität berücksichtigt und fördert. Um dies zu gewährleisten, muss sie folgende Charakteristika erfüllen (Details in Abb. 14):

- Situationskenntnis
- Kontinuierliche Verbesserung
- Bewältigung von Veränderungen
- Flexibilität
- Effizienz
- Effektivität

Die ASE Methodik dieser Arbeit nutzt die traditionelle BWI-Hall Methodik als Grundlage und erweitert diese hinsichtlich Agilität.

Sucht man nach einer konkreten Vorgehensweise welche mit „agiler Entwicklung“ bezeichnet wird, so findet man zumeist eine zyklische Entwicklung in kleinen Schritten mit Feedback-Loops. Für diese Vorgehensweise, welche einen hohen Grad an Agilität in der Entwicklung gewährleisten soll, wurden alle gefundenen Hinweise analysiert um ein zugrundeliegendes Funktionsprinzip zu identifizieren. Dieses wurde im **Agilen Entwicklungszyklus (AEZ)** abgebildet (siehe Kapitel 4.3.4). Der AEZ ist aber nicht gleichzusetzen mit agiler Entwicklung, er stellt lediglich eine Methode dar, mit welcher unter Umständen die Agilität in der Entwicklung erhöht werden kann.

In einer konkreten Anwendung kann der AEZ auch als eine Methode innerhalb einer ASE Methodik verwendet werden. Damit ist er auch bereits ein Teil der Antwort auf Forschungsfrage 2 nach Vorgehensweisen für Agilität im SE. Diese werden nun weiter im nächsten Kapitel behandelt.

# 5 Prinzipien und Praktiken für Agile Systems Engineering

Der im vorigen Kapitel vorgestellte AEZ, stellt eine Sammlung von Prinzipien und Praktiken dar, mit denen agile Entwicklungsprozesse gestaltet werden können. Andere, häufig im Zusammenhang mit Agilität erwähnte Prinzipien, stehen ebenso in Beziehung zum Entwicklungsprozess, können aber auch in anderen Bereichen Anwendung finden. Nachdem Systems Engineering nicht auf den Prozess beschränkt ist, sondern alle Lebenszyklusphasen des herzustellenden Systems betrachten soll, macht es auch Sinn, in all diesen Phasen nach Möglichkeiten für Agilität zu suchen. Innerhalb der Unternehmung kann über die Entwicklungsabteilung hinaus in anderen Bereichen nach Anwendungsmöglichkeiten für agile Maßnahmen gesucht werden.

Der Übersichtlichkeit wegen, werden die gefundenen Praktiken hinsichtlich ihres primären Anwendungsgebietes aufgeteilt. Zur Klassifizierung wurde Abb. 19 entworfen, in der aufgezeigt wird, wo Agilität innerhalb der Unternehmung gefördert werden kann, um agiles Verhalten im Systems Engineering zu unterstützen.

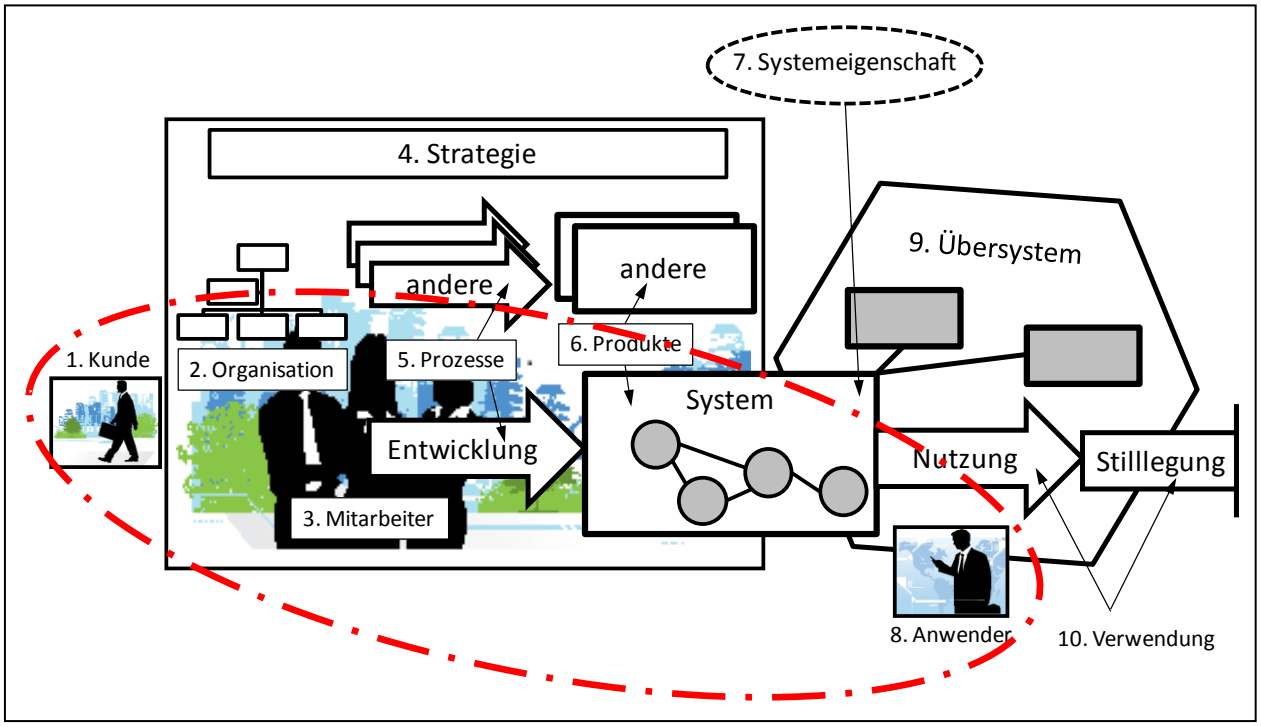


**Abb. 19: Bereiche für Anwendung von Agile Systems Engineering Maßnahmen**

Die Agilität (oder auch nur Flexibilität oder eine andere Teilfunktion des ASE) kann in diesen Eingriffsbereichen auf ganz unterschiedliche Weise gefördert werden. Die in Abb. 19 dargestellten Elemente können folgendermaßen interpretiert werden, wobei aufgezeigt werden soll, warum diese im Rahmen der Agilitäts-Thematik betrachtet werden sollen:

1. **Kunde**: Dieser stellt die Anforderungen, deren Erfüllung das Ziel jeden Systems Engineerings ist. Er ist auch oft verantwortlich für Veränderungen, deren Beherrschung das Ziel des ASE ist. In der Kommunikation zum Kunden liegt häufig die Ursache für ungenaue oder falsch definiert

Anforderungen. Der Kunde kann auch Feedback geben und bestimmt oft wesentliche Prozessmerkmale, wie z.B. die Dokumentation.

2. **Organisation**: Diese hat einen Einfluss auf Kommunikation und Wissensaustausch und damit auf das Lernen und die kontinuierliche Verbesserung. Sie kann selbst auch verändert werden, wenn dies vorteilhaft erscheint. Es stellen sich Fragen nach Organisationsformen, die man schnell ändern kann, aber auch nach solchen, die mit Veränderungen gut umgehen können.
3. **Mitarbeiter**: Die Mitarbeiter sind wesentlicher Faktor für jede Entwicklungstätigkeit, da sie das für die Entwicklung notwendige Wissen haben, verwalten und die Entwicklungsprozesse durchführen. Praktisch alle ASE Charakteristika stehen im Zusammenhang mit den Mitarbeitern. Negative Einflüsse auf die Mitarbeiter können den Erfolg des Systems Engineering Projekts gefährden und auch Veränderungen können solche negativen Einflüsse darstellen. Die Mitarbeiter selbst können aber auch Ursache für Veränderungen sein (z.B. Ausscheiden von Mitarbeitern). In beinahe allen Aspekten der Agilität müssen die Mitarbeiter also berücksichtigt werden. In diesem Bereich sollen auch Aspekte der Unternehmungskultur betrachtet werden, die auch einen Einfluss auf den Umgang mit Veränderungen haben können.
4. **Strategie**: Ein wesentlicher Kernpunkt einer Strategie ist die Festlegung der Produkt-Markt-Kombination. Diese legt fest welche Produkte mit welchen Eigenschaften auf welchen Märkten angeboten werden sollen. Dadurch wird auch das Umfeld bestimmt, welches die Ursachen für viele Veränderungen beinhaltet. Ebenso haben Entscheidungen hinsichtlich der Produkte Einfluss auf die Flexibilität, die bei der Entwicklung möglich ist. Unzählige weitere Betrachtungsweisen sind für die Strategie ebenso möglich, werden hier aber nicht weiter verfolgt. Es wird hierzu auf die Literatur zum Thema „Agile Enterprise“ verwiesen (siehe Kapitel 2.3).
5. **Prozesse**: Der Entwicklungsprozess bestimmt wesentlich den Informationsfluss in der Entwicklung und hat starken Einfluss auf die Möglichkeit der Mitarbeiter, agil zu handeln. Er bestimmt, wie mit Veränderungen umgegangen wird und hat auch Einfluss darauf, wie schnell Fehler oder entwicklungsinterne Veränderungen ans Tageslicht kommen. Er selbst kann auch geändert werden, genauso wie andere Prozesse. Diese können Agilität auch anders beeinflussen. Z.B. können Marketingprozesse Veränderungen im Markt früher aufzeigen. Produktionsprozesse erlauben es, wenn sie leicht anpassbar sind, noch Veränderungen am Produkt nach Produktionsstart einzubringen usw. Die vorliegende Dissertation beschränkt sich auf den Prozess der Entwicklung.
6. **System/andere Produkte**: Das System ist zugleich Ursache und Wirkungsbereich von Veränderungen. Es bietet mehr oder weniger Flexibilität. In Zusammenhang mit anderen Produkten können auch neue Möglichkeiten, für Flexibilität oder die Einschränkung dieser entstehen, z.B. bei Wiederverwendung von Modulen oder Plattformstrategien. Das Produkt bietet aber auch die Möglichkeit während der Verwendungsphase die Agilität in der Entwicklung zu unterstützen. Nämlich dann, wenn es Sensoren besitzt um Sachverhalte aufzuzeichnen, die für die Entwicklung weiterer Produkte relevant sind und diese an die Entwickler senden kann. Als Beispiel sei hier die Überwachung der Auslastung einer Anlage und die Rückmeldung der Daten an die Entwicklungsabteilung genannt (Aschbacher, et al., 2010).
7. **Systemeigenschaft**: Das System kann auch für sich selbst agil sein. Dazu ist es notwendig, dass es selbst alle Charakteristika der Agilität aufweist (siehe Kapitel 4.1) oder man könnte auch sagen,

dass es selbstständig den OODA-Loop umsetzen können muss. Solche Systeme werden in dieser Dissertation nicht betrachtet.

8. **Anwender**: Die zukünftigen Benutzer des Systems können am besten beurteilen, wie es zu funktionieren hat. Deshalb sind sie wichtige Ansprechpersonen für den Erhalt von Feedback.
9. **Übersystem**: Die Agilität eines Übersystems kann durch ein neu hinzugefügtes System erhöht werden, das zusätzlich Agilität einbringt (z.B. die Agilität einer Armee durch den Einsatz von Hubschraubern).
10. **Verwendung**: Ein System kann mehrere Anwendungsmöglichkeiten haben, zwischen denen es flexibel umschalten kann. Auch eine neue Nutzungsmöglichkeit nach der Stilllegung wäre ein möglicher Bereich agiler Betrachtungsweisen. Beides wird in dieser Dissertation aber nicht mehr weiter ausgeführt.

Der Anwendungsbereich für den ASE Ansatzes dieser Dissertation wird in Abb. 19 durch die strichpunktierte Abgrenzung dargestellt. Die darin aufgezeigten Elemente wurden zu 5 Haupteingriffsbereichen zusammengefasst, die für die Einteilung der agilen Maßnahmen in den folgenden Unterkapiteln dienen sollen:

5.1 Kunde (und Anwender)

5.2 Organisation

5.3 Mitarbeiter

5.4 Entwicklungsprozess

5.5 System

Die im Folgenden beschriebenen Prinzipien und Praktiken wurden in der Literatur und bei den empirischen Erhebungen gefunden. Die meisten davon finden bereits in der Softwareentwicklung Anwendung. Die Auflistung erhebt keinen Anspruch darauf, eine vollständige Zusammenstellung aller Maßnahmen zu sein, die im ASE angewandt werden können. Auch andere Praktiken, die der Erreichung der Ziele des ASE dienen, können nachträglich in den Ansatz integriert werden.

Die Darstellung der Prinzipien und Praktiken erfolgt jeweils in 3 Punkten. Zuerst erfolgt eine Erklärung mit Hinweisen aus der Literatur. Danach werden empirisch gesammelte Daten mit Anwendungsbeispielen aus der Systementwicklung und Meinungen der befragten Personen wiedergegeben. Schlussendlich wird eine Bewertung des Autors hinsichtlich einer Verwendbarkeit für ASE durchgeführt. Es sei hier noch einmal darauf hingewiesen, dass die Anwendung immer für Systeme, bestehend aus mehreren Komponenten und nicht für Software alleine betrachtet wird. Wenn in den Beschreibungen Praxisbeispiele erwähnt werden, ist immer die Entwicklung eines Produkts, bestehend aus Hardware und Software gemeint. Wird in einem Unterkapitel von „dem Prinzip“ oder „der Praktik“ ohne nähere Bezeichnung gesprochen, so ist immer die/das in der Überschrift des Unterkapitels stehende Praktik/Prinzip gemeint.

## 5.1 Maßnahmen in der Beziehung zum Kunden

Hier geht es um den Kontakt und die Zusammenarbeit mit Kunden und Anwendern. In diesem Bereich ist es vor allem die Gewinnung von Informationen zu den Anforderungen, die für ein agiles Verhalten essentiell ist. Der Kunde als Auftraggeber und der spätere Anwender des Systems stellen die Anforderungen. Diese sind oft zu Beginn eines Projektes noch nicht genau bekannt oder werden ungenau definiert. Auch können sie sich während der Entwicklungszeit verändern. Es gilt hier, möglichst genau herauszufinden, was Kunde und Anwender wirklich wollen, um diese zufriedenzustellen und das Projekt damit erfolgreich abzuschließen.

### 5.1.1 Fokus auf tatsächlichen Kundennutzen

**Erklärung:** Im Unterschied zum traditionellen SE, in dem es hauptsächlich um die Erfüllung der niedergeschriebenen Anforderungen geht, wird hier mehr Wert auf den tatsächlichen Kundennutzen gelegt. Hatte man im SE bisher den Kunden als durch die Anforderungen repräsentiert angesehen, so soll im Rahmen einer agilen Entwicklung besser ergründet werden, was er tatsächlich will (Turner, 2007 S. 12). Einerseits können die anfänglich definierten Anforderungen fehlerhaft oder ungenau niedergeschrieben sein oder dem Kunden waren seine exakten Wünsche zu diesem Zeitpunkt noch nicht bewusst (Gilb, 2005 S. 4), (Schwaber, 2004 S. 4). Andererseits liegt es in der Natur der Sache, dass die Beschreibungen in Anforderungskatalogen die tatsächlichen Anforderungen (Kundennutzen) nur repräsentieren, aber nicht mit ihnen ident sind (Kaindl, et al., 2008). Es kann also ein Unterschied bestehen zwischen dem, was der Kunde niederschreibt und dem, was er meint. Berücksichtigt man diesen Unterschied nicht, wird es schwer, den tatsächlichen Kundennutzen zu verstehen und können häufige Änderungen dahingehend die Folge sein (Kaindl, et al., 2010 S. 310).

Die Fokussierung auf den tatsächlichen Kundennutzen, dient also nicht nur der Erhöhung der Kundenzufriedenheit und damit der Effektivität, sondern auch einer Erkennung und Vermeidung zukünftiger Änderungen.

Bei den agilen Softwareentwicklungsmethoden wird die Entwicklung, durch die bereits erwähnten und im AEZ dargestellten Feedbackzyklen, automatisch dem tatsächlichen Kundenwunsch angepasst. Ken Schwaber spricht dabei auch von einer empirischen Prozesskontrolle (Schwaber, 2004 S. 2).

Tom Gilb empfiehlt, am Beginn des Projekts einen strukturierten Prozess zur Identifizierung aller Stakeholder durchzuführen, die den Projekterfolg oder –misserfolg beeinflussen können. Die weitere Entwicklung soll sich dann am Nutzen aller Stakeholder orientieren (Gilb, 2006 S. 2). In der S²QI Gruppe wurde speziell darauf hingewiesen, dass der auftragsgebende Kunde nicht mit dem Anwender übereinstimmen muss. Z.B bekommt ein Komponentenentwickler in der Fahrzeugindustrie, die Anforderungen vom Fahrzeughersteller gestellt. Der Nutzen entsteht aber erst bei der Fahrtätigkeit des Endanwenders. Auch in agilen Methoden wie z.B. Crystal wird auf die Wichtigkeit des Kontakts mit dem Endanwender hingewiesen (Cockburn, 2005 S. 67).

Eine Schwierigkeit sieht Gilb auch darin, den Vorgang steuer- und kontrollierbar zu machen. Dazu muss der tatsächliche Kundennutzen messbar gemacht und ein Zielwert definiert werden, der dann durch einen messbaren Qualitätslevel des Produkts erreicht werden soll. Als Schwierigkeit ist auch die bei der Festsetzung des Zielwerts im Vorhinein vorhandene Erwartung des Kunden über die Art der Lösung zu

sehen. Dadurch wird der Lösungsbereich oft eingeschränkt, obwohl dies nicht notwendig gewesen wäre. Hier muss versucht werden, lediglich den tatsächlichen Nutzen genau zu definieren und es dem Entwickler zu überlassen, wie er erreicht werden soll (Gilb, 2006 S. 5f).

**Empirische Erhebungen:** In der empirischen Studie konnte kein Systementwickler gefunden werden, der explizit den tatsächlichen Kundennutzen über die definierten Anforderungen stellt. Einige Beobachtungen lassen aber zumindest eine implizite Anwendung dieses Prinzips erkennen. Z.B. wird bei der Anforderungsanalyse bzw. bei der Überführung des Lastenhefts in ein Pflichtenheft versucht, den tatsächlichen Nutzen zu erkennen. Das hängt aber zumeist von den handelnden Mitarbeitern ab (gute/erfahrene Mitarbeiter können die tatsächlichen Anforderungen besser abschätzen) und geschieht vermehrt dann, wenn niedergeschriebene Anforderungen schwer zu erreichen sind. Ein Interviewpartner erwähnte auch, dass er bei Änderungswünschen des Kunden den tatsächlichen Nutzen verstärkt diskutiert, um eventuell die Änderung abzuwenden oder ihren Umfang abzuschwächen. Beide Fälle deuten aber darauf hin, dass hier vor allem zur Vereinfachung der Entwicklungstätigkeit agiert wird.

Mehrere Interviewpartner wiesen darauf hin, wie wichtig gut beschriebene Anforderungen für die Entwicklung von komplexen Produkten sind. Der Kunde soll auch die Verantwortung dafür tragen, dass die Formulierungen mit seinen Wünschen genau übereinstimmen. Kann er davon ausgehen, dass auch bei schlecht definierten Anforderungen nach seinen tatsächlichen Wünschen entwickelt wird, besteht laut den Interviewpartnern die Gefahr, dass er sich weniger Mühe macht und schlecht definierte Anforderungen die Entwicklung schwieriger machen.

**Bewertung:** Bei diesem Prinzip scheint es also, dass es nicht einfach ins SE übernommen werden kann, unabhängig von einer umfangreicheren, agilen Vorgehensweise. Speziell wenn komplexe Produkte wasserfallartig[4] oder mit wenigen Iterationen zu Fixpreisen entwickelt werden und deshalb versucht wird Änderungen zu vermeiden, ist es schwierig zu verfolgen. Es setzt auch voraus, dass Kunde und Entwickler sich dessen bewusst sind, dass die Festlegung von Anforderungen oft eine komplexe und nicht bürokratisch zu administrierende Angelegenheit ist. Hinsichtlich der Erreichung des tatsächlichen Kundennutzens zeigt sich aber, dass eine agile gegenüber einer traditionellen Vorgehensweise einen Vorteil hat. Dieser ist aber nur kontextabhängig (geeignete Vertragsbestimmungen, Preisfestsetzung, Kunde spielt mit) erreichbar. Ist der richtige Kontext nicht vorhanden, kann man höchstens bei der Anforderungsanalyse versuchen, den tatsächlichen Kundennutzen besser abzuschätzen, um spätere Änderungen zu vermeiden oder darauf vorbereitet zu sein.

### 5.1.2 Vertrauensvolle Zusammenarbeit statt übertriebene Vertragsverhandlungen

**Erklärung:** Dieses Prinzip wurde aus dem Manifest für agile Softwareentwicklung abgeleitet (Beck, et al., 2001). Dabei wird versucht, den Kunden zu einer gemeinschaftlichen Zusammenarbeit zu bewegen, um die Ziele zu erreichen (Cockburn, et al., 2001 S. 132). Im Gegensatz dazu wird bei traditioneller Entwicklung das gewünschte Ergebnis durch einen Vertrag im Vorhinein möglichst genau definiert, was bei häufig vorkommenden Änderungen die Entwicklung durch ständige Nachverhandlungen der Vertragsdetails behindern kann.

[4] Erklärung des Begriffs unter http://de.wikipedia.org/wiki/Wasserfallmodell

**Empirische Erhebungen:** In der empirischen Studie wurden einige Beobachtungen zu diesem Prinzip gemacht. Durch gute Zusammenarbeit wollten sich die Systementwickler Folgeaufträge sichern und waren deshalb bei Änderungswünschen sehr entgegenkommend. Eine Schwierigkeit war hierbei aber die Preisfestsetzung. Der Preis ist oft ein wesentliches Kriterium für den Kunden bei der Vergabe eines Entwicklungsprojektes und sollte deshalb bei Vertragsabschluss bekannt sein. Dazu muss auch das Produkt im Vertrag genau definiert sein. In der Praxis der Systementwickler wurden ausschließlich solche Fixpreisprojekte gefunden. Deshalb wurde auch von zahlreichen Fällen mit Änderungen berichtet, wo Kostenfragen die gute Zusammenarbeit belastet hatten. Beim Nachverrechnen von Anforderungsänderungen kam es immer wieder zu Diskussionen, ob es sich wirklich um eine Änderung handelt oder die Beschreibung der Anforderung nur anders (falsch) ausgelegt wurde. Aufgrund dieser Probleme wurde von den meisten Interviewpartnern doch ein Vertrag mit detaillierten Anforderungen empfohlen, auf den im Streitfall zurückgegriffen werden kann. Es gilt hier auch, den Kunden nicht zu einer nachlässigen Formulierung der Anforderungen zu verleiten, wie in Kapitel 5.1.1 bereits erwähnt wurde.

**Bewertung:** Eine Aufweichung oder gar ein Verzicht auf klare Verträge ist im Systems Engineering also problematisch. Ein direkter positiver Einfluss auf die Agilität ist auch nicht erkennbar. Trotzdem sollte dieses Prinzip auch hinsichtlich der Agilität berücksichtigt werden, da es das Arbeitsklima mit dem Kunden fördert und damit auch eine bessere Kommunikation (Feedback) erwarten lässt. Außerdem ist zu erwarten, dass ein Kunde, dessen Änderungswünsche wohlwollend behandelt werden, seinerseits auch zu Zugeständnissen bei Änderungswünschen des Entwicklers bereit ist.

### 5.1.3 Kunde in Entwicklung einbinden (Feedback)

**Erklärung:** Dieses Prinzip ist wesentlich für ein agiles Entwickeln und wird von vielen agilen Entwicklungsmethoden propagiert (Beck, et al., 2005 S. 18), (Poppendieck, et al., 2009 S. 177ff), (Gilb, 2005 S. 3). Bei herkömmlicher Entwicklung wird zu Beginn des Projektes ein detaillierter Anforderungskatalog erstellt. Dieser wird als mehr oder weniger definitiv betrachtet, was eine Kontaktaufnahme mit dem Kunden zur weiteren Abstimmung nicht mehr notwendig macht bzw. eher seltene Abstimmungszeitpunkte zulässt. Bei agiler Entwicklung wird aber nicht erwartet, dass die tatsächlichen Anforderungen konstant bleiben und mit einem am Anfang generierten Anforderungskatalog endgültig dargestellt werden können. Deshalb holt man sich vom Kunden oder Anwender laufend neuen Input zu den Anforderungen, nachdem dieser die Möglichkeit hatte, unter Zuhilfenahme aktueller Entwicklungsergebnisse, seine Anforderungen zu präzisieren oder zu aktualisieren (Turkington, 2007 S. 9).

Durch Begutachtung eines bereits entwickelten Teils des Gesamtsystems durch den Kunden oder auch Anwender können Missverständnisse aufgedeckt werden oder es kann ein neues Verständnis entstehen und resultierend daraus können die Anforderungen vom Kunden verfeinert, erweitert oder ausgetauscht werden (Dove, et al., 2008 S. 5).

Extreme Programming (Beck, et al., 2005 S. 61) schlägt eine Anwesenheit des Kunden bei den wöchentlichen Planungssitzungen vor, um zu überprüfen ob das Geplante mit den Anforderungen übereinstimmt (Kettunen, et al., 2005 S. 593). Als eigene Praktik empfiehlt Extreme Programming einen zukünftigen Anwender des Systems ständig verfügbar zu halten, damit dieser dem Entwicklungsteam

Fragen beantworten kann (Paulk, 2001 S. 3). Der Anwender kann dabei nicht nur nach dem Fertigwerden eines entwickelten Teils um Feedback gefragt werden, sondern jederzeit wenn er verfügbar ist. Dies kann den Feedback-Zyklus oft auf wenige Minuten reduzieren und soll helfen, Fehlentwicklungen schnell zu erkennen oder möglichst viele Ideen durch den Kunden bewerten zu lassen (Cockburn, 2002 S. 11). Die Einbindung des Endnutzers reduziert damit auch die Zeit zwischen Entscheidungen und dem Sichtbarwerden der Konsequenzen der Entscheidungen (Cockburn, et al., 2001 S. 131).

In einer Fallstudie zur agilen Softwareentwicklung für ein großes Luftfahrt-Projekt spricht Bedoll eine vom Kunden gestellte Person an, welche die Hälfte ihrer Zeit mit den Entwicklern und die andere Hälfte mit den Anwendern verbracht hat. Diese Einbeziehung des Kunden und der Anwender in die Entwicklung wird als wesentlicher Erfolgsfaktor für das Projekt genannt. Einerseits konnte durch das ständige Feedback des Kunden/Anwenders zielgerichteter entwickelt werden. Andererseits wurde durch die direkte Kommunikation der Prozess- und Dokumentationsaufwand verringert. Und ebenso gab es durch den besseren Informationsfluss keinen Vertrauensverlust, wenn Zieltermine für bestimmte Funktionalitäten nicht eingehalten werden konnten. Bedoll meint, dass der Kontakt auf keinen Fall für längere Zeit unterbrochen werden darf. Es besteht sonst die Gefahr, dass sich die Entwickler zu lange in eine falsche Richtung bewegen, wenn der Kunde nicht die Übereinstimmung seiner Wünsche mit den Entwicklungsergebnissen sicherstellt (Bedoll, 2003 S. 30ff).

Wie schon erwähnt, ist die direkte Kommunikation mit dem Kunden ein Erfolgskriterium für dessen Einbindung in die Entwicklung und für brauchbares Feedback. Bei agiler Entwicklung soll die Kommunikation zwischen Entwicklerteam und Kunden laut Karlström auch schon möglichst früh stattfinden. Vor allem funktionale Probleme sollen möglichst früh identifiziert und gelöst werden (Karlström, et al., 2006 S. 216f).

**Empirische Erhebungen:** In den Interviews wurde vereinzelt von sogenannten „Resident Engineers“ (Kundenvertreter beim Entwickler) berichtet. Diese wurden aber nur bei sehr großen Projekten (Gesamtfahrzeug) oder bei kritisch gewordenen Projekten eingesetzt und hatten dann eher den Status von Überwachern.

Andere Fälle von häufigem Feedback, bei dem man von einer Einbindung des Kunden in die Entwicklung sprechen kann, konnten auch gefunden werden. Z.B. bei einem Komponentenentwickler in der Fahrzeugindustrie, dessen Kunde das Gesamtfahrzeug simultan entwickelte. Die Anpassung der Komponente an den aktuellen Entwicklungsstand des Gesamtfahrzeugs war natürlich von größtem Interesse für den Kunden, weshalb er diesbezüglich häufig in Kontakt mit dem Entwickler trat.

Bei anderen Entwicklern (z.B. Papiermaschinenhersteller) wurde aber auch von Kunden berichtet, die absolut kein Interesse an Zwischenergebnissen hatten und auch kein Feedback geben wollten. Dies wurde aber als verständlich betrachtet, weil bei dieser Art von System klare Anforderungen zu Beginn definiert werden können und ein zwischenzeitliches Feedback selten notwendig ist.

Im Allgemeinen wurde die direkte Kommunikation und Einbindung des Kunden in die Entwicklung aber von allen Interviewpartnern als wichtiges Erfolgskriterium betrachtet. Nur müsse der Projektleiter den Überblick dabei behalten und dürfe es nicht zulassen, dass auf unterer Ebene, ohne seine Zustimmung Zusagen gemacht werden, ohne zu überprüfen, welchen Einfluss diese auf das Gesamtsystem haben.

Im Expertenpanel wurde auch eine bessere Verknüpfung des Entwicklers mit dem Endanwender als ein Ziel genannt, das mithilfe von agilen Methoden erreicht werden soll. Durch das Feedback des Anwenders soll die Produktqualität verbessert werden. Es wurde aber auch angesprochen, dass die Einholung des Feedbacks natürlich Kosten verursacht und dass Kunden vor allem deshalb oft dazu nicht bereit sind.

**Bewertung:** Die zahlreichen Literaturhinweise lassen die Relevanz dieser Praktik hinsichtlich agiler Entwicklung und speziell hinsichtlich der Informationsgewinnung und Effektivität erkennen. Literatur und empirische Erhebungen lassen auch auf ein großes Erfolgspotential schließen, vor allem wenn diese Praktik als Teil des AEZ angewandt wird. Das einzige Ausschlusskriterium für diese Praktik ist ein Kunde, der nicht dazu gewillt ist (was aber nicht unbedingt negativ zu sehen ist, da je nach Kontext das Feedback auch nicht notwendig sein kann). Die Sinnhaftigkeit der Praktik ist natürlich umso größer, je mehr Zwischenergebnisse zur Verfügung stehen und je eher auf das Feedback des Kunden auch dementsprechend reagiert werden kann.

### 5.1.4 Flexibles Anforderungsmanagement

**Allgemeine Erklärung:** Im herkömmlichen Systems Engineering werden die Anforderungen meistens zu Beginn des Projektes definiert und dann vom Entwicklungsteam als absolut und unveränderbar angesehen. Auf eine Priorisierung wird oft verzichtet, es wird davon ausgegangen, dass alle Anforderungen zu erfüllen sind. Während der Entwicklung wird nicht laufend überprüft, ob die Anforderungen (noch) mit den tatsächlichen Wünschen des Kunden übereinstimmen (Turner, 2007 S. 12). Agile Prinzipien, die mögliche Änderungen der Anforderungen berücksichtigen sollen, wurden in den vorigen Kapiteln schon vorgestellt. Hier sollen nun konkrete Praktiken zum Umgang mit den Anforderungen vorgestellt werden. Empirische Beobachtungen und Bewertung werden am Ende des Kapitels für alle Praktiken gemeinsam dargestellt.

***(User-)Stories***

Diese Praktik stammt aus dem Extreme Programming. Im Gegensatz zu präzise formulierten Anforderungen, welche Systemspezifikationen darstellen, werden hier einfache, allgemein verständliche Geschichten über die gewünschten Systemfunktionen formuliert (Beck, et al., 2005 S. 44).

***Produktvision***

Eine Vision des Produkts soll zu Beginn des Projekts den Grund für das Projekt und das erwünschte Ergebnis beschreiben, ohne detaillierte Anforderungen wiedergeben zu müssen. In Scrum wird die Vision als Basis für den Product Backlog verwendet (Schwaber, 2004 S. 68).

***Product- und Sprint-Backlog***

Diese beiden Praktiken entstammen der agilen Softwareentwicklungsmethode Scrum (Schwaber, 2007 S. 124ff). Im Product-Backlog werden dabei die Anforderungen an das neue System, aufbauend auf einer groben Produktvision in einem änder- und erweiterbaren Anforderungskatalog niedergeschrieben und vom Kunden priorisiert. Für jeden einzelnen Entwicklungsschritt werden daraus die Anforderungen, die in diesem Schritt entwickelt werden sollen, entnommen und zu einem sogenannten Sprint-Backlog zusammengestellt. Die Auswahl der Anforderungen erfolgt gemeinsam mit dem Kunden. Während des Entwicklungsschrittes (Sprints), der gewöhnlich 4 Wochen dauert, sind keine Änderungen des Sprint-

Backlogs durch den Kunden vorgesehen (Seibert, 2007 S. 42). Dadurch entsteht für die Entwickler zumindest kurzzeitig eine stabile Anforderungsliste. Änderungen werden nur im Product-Backlog erfasst und können nur zu jedem Start eines neuen Entwicklungsschrittes zur Bearbeitung übergeben werden.

***Priorisierung***

Die Priorisierung der Anforderungen durch den Kunden und Entwicklung in Reihenfolge der höchsten Priorität gilt in vielen agilen Entwicklungsmethoden als Erfolgskriterium (Schwaber, 2004 S. 8), (Poppendieck, et al., 2009 S. 71), (Gilb, 2005 S. 295, 302). Einerseits weil damit bei Erfüllung der wichtigsten Anforderungen unter Umständen bereits eine lauffähige Software vorhanden ist, die dem Kunden bereits Nutzen liefert, bevor das gesamte Projekt abgeschlossen ist. Andererseits hat der Kunde damit die Möglichkeit, das Projekt zu beenden, sobald ihm die Software als tauglich erscheint, auch wenn noch Anforderungen mit niedriger Priorität noch nicht fertig entwickelt wurden. Dies kann auch bei kritischen Endterminen für das Projekt ein Vorteil sein. Sollte der Endtermin überzogen werden, ist sichergestellt, dass die Software zu diesem Zeitpunkt zumindest die wichtigsten Anforderungen erfüllt. Aufgrund der Produkteigenschaften von Software ergeben sich hierbei Möglichkeiten, die nicht ohne weiteres auf Systeme (Hardware + Software) übertragen werden können. Trotzdem bietet eine Priorisierung auch wesentliche Vorteile im Systems Engineering, die im Kapitel 5.1.5 gemeinsam mit weiteren Hinweisen zur Priorisierung beschrieben werden.

Die eben beschriebenen Praktiken sind natürlich nur Vorschläge, um ein agileres Verhalten zu ermöglichen. Sie bilden keineswegs ein vollständiges Anforderungsmanagement-System ab, das im Systems Engineering sehr komplex werden kann. Z.B. ist bei sicherheitskritischen Systemen der Umgang mit kritischen Anforderungen genau zu definieren. Neben vielen Anforderungen, die sich während der Entwicklung verändern können, gibt es eben auch Anforderungen, die auf keinen Fall geändert werden dürfen. Es ist speziell die Aufgabe des Systems Engineerings, diese vom Kunden genannten kritischen Anforderungen zu verwalten (McMahon, 2006).

In der Softwareentwicklung wurde erfolgreiches Anforderungsmanagement durch agile Entwicklungsmethoden unter Einbeziehung von Kunden und Anwendern schon durch zahlreiche Fallstudien dargestellt z.B. (Hansson, et al., 2006 S. 1308). Dort wurde eine Webseite eingerichtet und der jeweilige Entwicklungsstand war Kunden und Endnutzern ständig verfügbar. Dadurch konnten diese laufend Feedback geben und die Anforderungen weiterentwickeln und genauer spezifizieren. Alle damit zusammenhängenden Anfragen und Anregungen wurden in einer Datenbank gespeichert, ähnlich einem Product-Backlog. Die Inhalte der Datenbank wurden genutzt, um neue Ideen zu generieren, oft schon bevor neue Entwicklungsschritte gestartet wurden.

**Empirische Erhebungen:** In den empirischen Erhebungen zur gemeinsamen Entwicklung von Hardware und Software konnten solche Praktiken (mit Ausnahme der Priorisierung) nicht gefunden werden. Die Gründe dafür sind dieselben, wie in den Kapiteln 5.1.1 und 5.1.2 beschrieben. Es gab Fälle (z.B. Entwicklung einer Förderanlage) wo innerhalb der Systementwicklung die Softwareentwicklung alleine mit Scrum durchgeführt wurde. Auf Systemebene fand aber ein herkömmliches Anforderungsmanagement statt. In der S²QI Gruppe war auch die Synchronisation der verschiedenen Entwicklungsprozesse (und damit auch des Anforderungsmanagements) von Hardware und Software ein

wichtiges Thema. Bei Systemen wie Papiermaschinen oder Eisenbahn-Fahrgestellen konnten sich die meisten Interviewpartnern aber nicht vorstellen, mit diesen Praktiken zu arbeiten.

**Allgemeine Bewertung:** Es ist zu vermuten, dass es nicht an den Praktiken selbst liegt, sondern Gründe wie die Preisfestsetzung und die schwierige Änderbarkeit der Hardware dafür verantwortlich sind, dass die meisten Interviewpartner sie für ungeeignet in der Hardwareentwicklung halten. Zu erwähnen ist auch das Risiko des „Verlierens" von sicherheitskritischen Anforderungen in einem leicht veränderbaren Anforderungsmanagement. Wären diese Probleme gelöst, würden die agilen Softwareentwicklungsmethoden ein großes Portfolio an Praktiken zum agilen Änderungsmanagement bereithalten. Eine alleinige Anwendung eines agilen Anforderungsmanagements kann natürlich keine wesentliche Verbesserung der Agilität bringen, sondern nur den Umgang mit Veränderungen erleichtern und Flexibilität in das Anforderungsmanagement bringen. Das ist aber eine Voraussetzung für eine insgesamt agile Entwicklung.

### 5.1.5 Priorisierung

**Erklärung:** Die in Kapitel 5.1.4 bereits genannte Praktik der Priorisierung soll in diesem Kapitel weiter ausgeführt werden, wobei hauptsächlich Kriterien gesucht werden, mit deren Hilfe die Prioritäten von Anforderungen bestimmt werden sollen. Eine Prioritätensetzung bei den Anforderungen wurde schon im traditionellen SE empfohlen (Haberfellner, et al., 2002 S. 148). Dabei geht es hauptsächlich um die Wichtigkeit der Anforderungen. Hinsichtlich Agilität liegt die Bedeutung der Priorisierung vor allem aber auch darin, dass man die Reihenfolge der Entwicklungsschritte danach ausrichtet. Möglichkeiten dazu sollen im Folgenden erklärt werden.

Nachdem bei agiler Entwicklung davon ausgegangen wird, dass Veränderungen unvermeidbar sind, soll diese Annahme Berücksichtigung bei der Priorisierung finden. Auch wenn manche agilen Softwareentwicklungsmethoden darum bemüht sind, die Kosten für Veränderungen auch in späten Projektphasen möglichst gering zu halten, ist im Allgemeinen und vor allem bei komplexen Systemen davon auszugehen, dass Änderungen in späten Phasen viel höhere Aufwendungen verursachen. Deshalb soll versucht werden, diese Änderungen möglichst früh zu provozieren (Malotaux, 2006 S. 2). Natürlich hat man dabei keinen Einfluss auf Veränderungen, die durch externe Ereignisse notwendig werden. Vielmehr geht es um Änderungen, die schon erkennbar sein könnten, aber noch nicht erkannt wurden, weil dazu noch nicht genug Informationen aus der Entwicklung zur Verfügung stehen. Deshalb kann es ein Prinzip der Priorisierung sein, jene Entwicklungsschritte zuerst durchzuführen, von denen man sich einen großen Informationszuwachs erwartet.

In der „Evolutionary Project Management" Methode werden mehrere Prinzipien vorgeschlagen, nach denen priorisiert werden soll, ohne diese zu gewichten. Diese Prinzipien sagen, dass jene Dinge zuerst entwickelt werden sollen, die ein hohes Risiko beinhalten, einen hohen Nutzen stiften und die einen hohen Leistungsgewinn bezüglich eines bestimmten Parameters (Ziels) erreichen (Gilb, 2005 S. 2ff).

Eine Priorisierung zur Risikominierung ist dabei so zu verstehen, dass man zuerst versucht, kritische Teile zu entwickeln, die ein hohes Risiko beinhalten bzw. zu denen man keine Erfahrung hat. Dadurch kann man Wissen zu diesen Teilen gewinnen und kümmert sich erst um fortführende Dinge, wenn man sichergestellt hat, dass das Konzept realisierbar ist (Gilb, 2005 S. 2f).

Natürlich stellt der Kundennutzen ein wesentliches Kriterium für die Priorisierung dar, wie bereits in Kapitel 5.1.4 erwähnt. Deshalb definieren normalerweise Kunden oder Anwender die Prioritäten. Das kann aber auch vom Entwickler getan werden, z.B. bei Scrum durch den Product Owner (siehe 5.1.6) (Schwaber, 2004 S. 20).

Gilb meint, dass alle Stakeholder berücksichtigt werden sollen. Bei der Priorisierung soll man sich die Frage stellen, wer im Augenblick der interessanteste Stakeholder ist und welche Funktionen ihm den größten Nutzen stiften könnten. So kann man z.B. bei einer absehbaren Überziehung von Zeit oder Kosten recht früh einen wesentlichen Teil des Kundennutzens erreichen (Gilb, 2005 S. 3). Und im Gegensatz zu großen Entwicklungen, die erst mit Abschluss des Projekts den Nutzen liefern, hat man dadurch den Vorteil, schon früh einen Teil des Nutzens präsentieren zu können (Gilb, 2006 S. 6f).

Die Priorisierung nach Leistungszuwachs ist eine Verfeinerung des Nutzen-Prinzips. Hier geht es darum, Entwicklungen, die ein Leistungskriterium des Systems wesentlich verbessern, höher zu priorisieren als Entwicklungen, die dies nur marginal tun. Damit kann die Entwicklung möglicherweise auch dahingehend optimiert werden, dass man sie beendet, wenn ein Leistungskriterium bereits seinen Zielwert erreicht hat (Gilb, 2005 S. 6). In einem weiteren Artikel schlägt Tom Gilb die Priorisierung nach Kosten-Nutzen Effizienz vor (Gilb, 2008 S. 3f).

Eine Priorisierung kann auch im Änderungsfall stattfinden. Wenn mehrere miteinander konkurrierende Reaktionsmöglichkeiten zur Verfügung stehen oder wenn durch eine Veränderung nicht mehr alle Anforderungen erfüllt werden können, muss eine Auswahl getroffen werden. Hierfür sollen im Vorhinein Verfahren oder Prozesse entwickelt werden, die möglichst schnell zur besten Entscheidung führen (Dove, 2005 S. 5f). Bisher wurden die Änderungen meistens in der Reihenfolge des Entstehens abgearbeitet. Gilb schlägt aber vor, Änderungen nicht nach dem Zeitpunkt des Erkennens der Notwendigkeit abzuarbeiten, sondern nach dem generierten Wert bzw. nach dem Verhältnis von Nutzen zu Kosten. Es sollen also auch die Änderungen, die den höchsten Nutzen bringen, die höchste Priorität erhalten (Gilb, 2005 S. 10).

**Empirische Erhebungen:** In der empirischen Studie wurde nicht nach Richtlinien für die Prioritätenbildung gesucht. Es wurde nur gefragt, ob eine Priorisierung stattfindet. Dabei wurde einige Male das Sicherheitsrisiko als wesentliche Priorität genannt. Anforderungen, die damit markiert werden, müssen nachverfolgbar behandelt werden und dürfen nicht lösch- oder änderbar sein. Andere Prioritäten werden aber kaum verwendet und werden vom Kunden auch oft abgelehnt, weil dieser die Meinung vertritt, dass ohnehin alle Anforderungen erfüllt werden müssen. Im Expertenpanel war deshalb auch nur die Rede davon, wie man sicherheitskritische Anforderungen kennzeichnen muss, damit diese in einer agilen Entwicklung nachverfolgbar bleiben. Nachdem keine, den Kriterien der Agilität entsprechende, Entwicklung in kleinen Schritten gefunden wurde, konnte natürlich auch keine Prioritätenbildung zur Bestimmung der Reihenfolge der kleinen Entwicklungsschritte gefunden werden.

**Bewertung:** Die Argumente in der Literatur zur Agilität sprechen klar für eine Priorisierung, egal ob bei der anfänglichen Anforderungsaufnahme oder nach Änderungen. Nachteile oder Risiken einer Prioritätensetzung erscheinen dem Autor nicht nachvollziehbar. Deshalb ist diese Praktik in unveränderlichen Umgebungen als genauso vorteilhaft zu betrachten, wie in veränderlichen. Die wesentlichen Vorteile hinsichtlich Veränderungen können aber nur bei einer insgesamt agilen

Entwicklung realisiert werden, da die Bestimmung der Reihenfolge der Entwicklungsschritte nach Priorität natürlich nur bei einer Entwicklung in kleinen Schritten möglich ist.

### 5.1.6 Einsatz eines Product Owners

**Erklärung:** In der agilen Softwareentwicklungsmethode Scrum vertritt ein sogenannter „Product Owner" die Kundenanforderungen innerhalb des Entwicklungsteams (Schwaber, 2007 S. 121). Er verwaltet den Product Backlog, setzt Prioritäten fest und führt die Budgetierung durch. Es ist auch möglich, dass der Kunde selbst die Rolle des Product Owners übernimmt (Schwaber, 2004 S. 53ff).

Bei agilem Systems Engineering schlägt McMahon den Systemingenieur als besten Kandidaten für die Rolle des Product Owners vor. Dieser kann dann auch bei großen Projekten als Koordinationsstelle zwischen einzelnen Sub-Teams dienen (McMahon, 2006 S. 27).

**Bewertung:** Die Umsetzung dieser Idee wurde bei den Interviews nicht explizit gefunden. Es macht aber Sinn zu versuchen, die Sichtweise des Kunden im Entwicklerteam vertreten zu lassen, vor allem natürlich wenn der Kunde nicht permanent anwesend ist. Für den ASE Ansatz soll die Funktion eines Product Owners also darin bestehen, den Kunden zu repräsentieren, wenn dieser nicht für eine Einbindung in die Entwicklung zur Verfügung steht.

## 5.2 Maßnahmen in der Organisation

In diesem Kapitel werden organisatorische Prinzipien und Praktiken beschrieben. Hauptsächlich geht es dabei um Maßnahmen, die den Mitarbeitern agiles Verhalten ermöglichen sollen. Diese sind im Wesentlichen Maßnahmen zur Verbesserung des Informationsaustausches innerhalb der Unternehmung bzw. des Entwicklerteams. Aber auch die Flexibilität der Organisation selbst ist ein Thema, da manche Veränderungen auch die Anpassung der Organisation notwendig machen können.

### 5.2.1 Aufteilung großer Teams

**Erklärung:** Die Teamgröße ist ein wesentliches Kriterium für die Qualität der Kommunikation (siehe dazu 5.3.5). Es gibt verschiedene Kanäle, wie persönliches Gespräch, Telefonat oder Email um die Kommunikation möglichst effizient zu gestalten. Bei der Festlegung der Kommunikationsmittel ist aber immer auf die Teamgröße zu achten. Je kleiner ein Team ist, umso einfach funktioniert die Kommunikation. Auch in Fällen aus der Softwareentwicklung wurde bestätigt, dass Kommunikation am besten funktioniert, wenn das Teams klein ist, die Teammitglieder im selben Büro sitzen und direkt kommunizieren (Hansson, et al., 2006 S. 1299-1308).

Deshalb wird in der agilen SW Entwicklung eine kleine Teamgröße empfohlen, bei XP sind es z.B. 10 Entwickler (Kettunen, et al., 2005 S. 595) und bei Crystal Clear sind es 8 (Cockburn, 2005 S. 282). In einem Fallbeispiel zur Software-Entwicklung in der Luftfahrtindustrie ist auch davon die Rede, dass bei nur 3 Entwicklern und einem Halbzeit-Tester, die Dokumentation wesentlich reduziert werden konnte (Bedoll, 2003 S. 31).

Es kann also davon ausgegangen werden, dass die Agilität bei kleinen Teams höher ist. Wird ein größeres Team an Entwicklern benötigt, empfiehlt XP die Aufteilung des Projektumfangs (Beck, et al., 2005 S. 112). Dabei soll das Projekt so aufgeteilt werden, dass die einzelnen Teile unabhängig voneinander jeweils von

einem kleinen Team entwickelt werden können. Als größte Schwierigkeit wird die Integration der einzelnen Teile gesehen und vor allem die häufige Integration (Beck, et al., 2005 S. 112).

Ken Schwaber berichtet von zahlreichen größeren Projekten, die aufgeteilt wurden, damit Scrum verwendet werden konnte (Schwaber, 2004 S. 119). Dabei wurde jeweils zu Beginn des Projekts die Infrastruktur für die Kommunikation zwischen den Teams aufgebaut (was in Scrum Sprints abgehandelt wurde) (Schwaber, 2004 S. 122). Die Verknüpfung der Teams funktionierte im Wesentlichen durch ein tägliches kurzes Meeting, an dem je ein Teammitglied aller Teams teilnahm. Diese Meetings werden „Scrum of Scrums“ genannt (Schwaber, 2004 S. 44).

Auch Highsmith berichtet von vielen größeren Softwareentwicklungsprojekten, die agil durchgeführt wurden und bei denen mehrere kleine Teams verwendet wurden (Highsmith, 2010 S. 271). Er nennt technische Praktiken, speziell für SW Entwicklung, die dabei behilflich sind (Highsmith, 2010 S. 294ff).

**Empirische Erhebungen:** In den empirischen Erhebungen wurden natürlich Möglichkeiten zur Aufteilung bzw. Untergliederung von großen Entwicklungsteams gefunden, meistens nach funktionalen Gesichtspunkten oder nach Baugruppen. Die schnellste Frequenz der abgehaltenen Meetings war aber wöchentlich, weshalb hier nur bedingt ein Vergleich mit einem Scrum of Scrums gezogen werden kann. Das Expertenpanel bestätigte die Wichtigkeit einer effizienten Aufteilung von großen Teams. Allgemeingültige Regeln dafür konnten aber noch nicht gefunden werden.

**Bewertung:** Der Autor vermutet, dass je nach Größe, Art von Projekt und Produkt, Kultur und vielen weiteren Kriterien eine unterschiedliche Aufteilung und Verknüpfung der Teilteams sinnvoll ist. Dass dies bei großen Projekten durchgeführt werden muss, erscheint aber nicht nur aus Gründen der Agilität, sondern auch aus organisatorischen Gründen notwendig zu sein. Die Agilität ist hier nur ein weiterer Aspekt, der speziell nach direkter und schneller Kommunikationsmöglichkeit verlangt.

### 5.2.2 Örtliche Zusammenlegung des Teams

**Erklärung:** Aus denselben Gründen wie in 5.2.1 empfehlen die agilen Softwareentwicklungsmethoden auch die gemeinsame Nutzung eines Büroraums für das ganze Team (Beck, et al., 2005 S. 37), (Cockburn, 2005 S. 57). Die örtliche Nähe der Projektmitglieder reduziert die Kosten der Informationsweitergabe (Cockburn, et al., 2001 S. 131). Bei Extreme Programming wird sogar eine Sitzaufteilung gefordert, die Augenkontakt ermöglicht (Beck, et al., 2005 S. 6). Ein kleines Team, das in einem Büro sitzt, kann sich jederzeit Fragen stellen und jedes der Teammitglieder kann nebenbei zuhören, ohne die eigene Arbeit unterbrechen zu müssen. Der Informationsaustausch findet schneller statt (Cockburn, 2002 S. 10).

Ist das Projektteam nicht nur auf verschiedene Büroräume aufgeteilt, sondern sogar geografisch voneinander getrennt, können folgende Probleme entstehen und zu einem Misserfolg des Projektes führen (Wilson, et al., 2003 S. 799):

- Das Projektteam hat keine gemeinsame Vision.
- Die Teammitglieder und ihre Rollen können nicht genau identifiziert werden.
- Die Ziele des Teams sind nicht mit den individuellen Zielen vereinbar.
- Die Kommunikation ist nicht so klar und ausführlich wie erforderlich.

**Empirische Erhebungen:** In den Expertendiskussionen und Interviews hat sich gezeigt, dass örtliche Nähe sehr oft nicht möglich ist. Nicht nur weil große Teams verwendet werden, sondern auch weil die Entwicklung auf mehrere Regionen, Länder oder sogar Kontinente verteilt durchgeführt wird. Außerdem werden Entwickler auch sehr häufig gleichzeitig in mehreren Projekten mit unterschiedlicher Teamzusammensetzung eingesetzt.

**Bewertung:** Der Vorteil, den örtliche Nähe bietet, liegt auf der Hand. Die Voraussetzungen für diese Praktik sind aber in vielen Fällen einfach nicht gegeben.

### 5.2.3 Dynamische Teamauswahl nach benötigten Fähigkeiten

**Erklärung:** Laut einer Extreme Programming Praktik muss das Entwicklungsteam jene Personen enthalten, welche alle Fähigkeiten und Kontakte haben, die zur erfolgreichen Entwicklung benötigt werden (Beck, et al., 2005 S. 6). Diese Fähigkeiten können durchaus dynamisch sein. Auch während des Ablaufs von Projekten sollen Mitarbeiter ausgetauscht werden, wenn neue Fähigkeiten benötigt oder andere obsolet werden (Beck, et al., 2005 S. 39).

**Empirische Erhebungen:** In der Industrie wurden einige Möglichkeiten zur Umsetzung dieser Praktik gefunden. Ein Austausch von Teammitgliedern wird aber nur in seltenen Fällen durchgeführt. Meist wird dabei die ausgetauschte Person als Versager gesehen, die durch eine kompetentere Person ersetzt werden musste. Bei den untersuchten Systementwicklern, wird meistens eine Matrix-Projektorganisation verwendet. Dabei können die einzelnen Projektmitarbeiter immer das Wissen ihrer Linienabteilung konsultieren bzw. Unterstützung daraus erhalten. Außerdem wurden Fälle gefunden, in denen nicht vorhandene Kompetenzen durch Beiziehen von Beratern oder Fremdvergabe eines kleinen Entwicklungsumfangs erworben wurden.

**Bewertung:** In der Systementwicklung werden oft Fähigkeiten aus ganz unterschiedlichen Fachgebieten benötigt. Dementsprechend muss auch das Team gestaltet werden. Optimierungspotential hinsichtlich Agilität ist hier in zweierlei Hinsicht vorhanden. Einerseits sollen Veränderungen als normal angesehen werden und damit auch Veränderungen in der Teamzusammensetzung. Andererseits muss der Austausch von Teammitgliedern oder das Hinzuziehen neuer Entwickler oder Berater möglichst effizient geschehen.

### 5.2.4 Team Selbstorganisation

**Erklärung:** Dies bedeutet, dass Prozess- und Organisations-Verantwortung vom Management zu den Entwicklern verlagert werden soll. Einerseits gibt es keine streng festgelegte Hierarchie im Entwicklungsteam und andererseits darf das Team sich selbstständig jeweils so organisieren, wie es glaubt, die Anforderungen bestmöglich erfüllen zu können (Seibert, 2007 S. 44).

Diese Praktik fördert die Fähigkeit der Organisation, schnell auf Veränderungen reagieren zu können. Probleme werden auf der niedrigstmöglichen Organisationsebene gelöst (Turkington, 2007 S. 9). Das ist deshalb vorteilhaft, weil Bürokratie und hierarchische Organisation oft schnelle und genaue Reaktionen auf Veränderungen verhindern (Dove, et al., 2008 S. 5).

Die Selbstorganisation ist aber nicht als Führungslosigkeit zu sehen, sondern viel mehr als ein Führungsstil, der großen Wert auf folgende Punkte legt (Highsmith, 2010 S. 52):

- Die richtigen Personen einstellen.
- Produktvision und Projektgrenzen müssen verständlich gemacht werden.
- Zusammenarbeit wird angeregt und gefördert.
- Auf Übernahme von Verantwortung wird bestanden.
- Selbstdisziplin
- Führung und Lenkung von Personen ist wichtiger als Kontrolle.

Im agilen SW- Prozessmodell von Mikio Aoyama werden die Prozesse in solche, die vom Management vorgegeben sind (öffentliche) und solche, die der Entwickler selbst organisieren soll (private Prozesse) aufgeteilt. Die privaten Prozesse stellen dabei Teilprozesse der übergeordneten öffentlichen Prozesse dar und unterstehen der alleinigen Verantwortung der jeweiligen Entwickler. So werden die Entwickler in ihrem Bereich nicht durch die Formalitäten des öffentlichen Prozesses gebremst und können eine höhere Effizienz und Flexibilität erreichen (Aoyama, 1998 S. 6).

**Empirische Erhebungen:** In den Diskussionen mit den Vertretern aus der Industrie wiesen diese immer wieder auf die große Wichtigkeit eines selbstverantwortlichen Handelns hin. Organisation und Prozesse wurden aber in der Praxis immer vom Management bestimmt. Nicht nur weil in den meist großen Projekten sonst ein Chaos befürchtet wird. Sondern auch weil Abänderungen, sollten diese Verbesserungen bringen, auch für weiter Projekte genutzt werden sollen. Eine völlige Freigabe der Prozesse und Organisation wurde von keinem Gesprächspartner als machbar erachtet. Vor allem die Größe einzelner Projekte, die Tatsache, dass Mitarbeiter normalerweise in mehreren Projekten gleichzeitig eingesetzt werden und dass auch während des Ablaufs von Projekten Mitarbeiterfluktuation besteht, machen eine Organisation „von oben“ erforderlich. Diese Punkte und außerdem der Wunsch vieler Kunden nach zertifizierten Prozessen, erlauben ihre freie Gestaltung oft nicht. Trotzdem wird aber versucht, Organisation und Prozesse an die Projekteigenschaften anzupassen. Dies geschieht aber in einem vorgegebenen Rahmen z.B. beim Projekt Kick-Off durch Auswahl eines für die Projektgröße passenden Prozesses (Stelzmann, et al., 2010 S. 6).

**Bewertung:** Komplexität und Größe vieler Systementwicklungsprojekte scheinen eine umfangreiche Selbstorganisation zu verhindern. Die Idee von Mikio Aoyama oder die vorgegebenen Optionen zur Anpassung scheinen in Systems Engineering Projekten am ehesten umsetzbar zu sein. Damit kann die Flexibilität in der Organisation zumindest bis zu einem gewissen Grad gesteigert werden.

### 5.2.5 Gleichzeitige Mitarbeit an möglichst wenigen Projekten

**Erklärung:** In den Methoden der agilen SW Entwicklung ist kein Multiprojektmanagement vorgesehen. Die Entwickler sollen einem Projekt möglichst vollzeitlich zugeteilt sein und maximal an zwei Projekten gleichzeitig arbeiten. Das geistige und emotionale Umstellen zwischen mehreren Projekten belastet die Produktivität der Entwickler sonst zu stark (Cockburn, 2002 S. 10). Auch wegen der starken Forcierung der personenbezogenen Werte und der Verwendung von nicht dokumentiertem Wissen sollen sich die Entwickler voll auf ein Projekt konzentrieren können.

**Empirische Erhebungen:** In den betrachteten Firmen waren die Mitarbeiter fast immer an mehreren Projekten gleichzeitig beschäftigt. Auch wenn dies problematisch sein kann, wenn mehrere Projekte zur gleichen Zeit die volle Arbeitskraft abverlangen, so bringt dies im Normalfall aber auch Vorteile. Einerseits natürlich aus wirtschaftlicher Sicht, weil Ressourcen gespart und Wartezeiten vermieden werden. Andererseits wird so auch ein Wissenstransfer zwischen verschiedenen Projekten ermöglicht.

**Bewertung:** Dieses Prinzip ist also durchaus zwiespältig zu sehen. In der Praxis sprechen die handfesteren Argumente (Wirtschaftlichkeit, Ressourcenausnutzung, Wissenstransfer) dagegen. Ein höhere Flexibilität der Mitarbeiter und eine bessere Effizienz sind für ein einzelnes Projekt zu erwarten, auf Unternehmungsebene kann sich aber durchaus der gegenteilige Effekt einstellen. Eine Möglichkeit zum Einsatz kann ein kritisch gewordenes Projekt (ausgelöst z.B. durch Veränderungen) sein, dessen Team von anderen Aufgaben befreit werden soll, damit es sich auf das kritische Projekt konzentrieren kann.

### 5.2.6 Pair Programming

**Erklärung:** Pair Programming ist eine XP Praktik, die beim Programmieren ein Zusammenarbeiten von 2 Programmierern an einem Computer fordert (Beck, et al., 2005 S. 6). Dabei schreibt ein Programmierer den Code, der andere überprüft ihn permanent und hat die Möglichkeit, Verbesserungsvorschläge zu machen oder Probleme anzusprechen. Dies soll folgende Vorteile bewirken (Beck, et al., 2005 S. 42):

- Die beiden Programmierer achten gegenseitig darauf, dass die Tätigkeit sich auf die zu erledigende Aufgabe fokussiert.
- Es findet ein häufiges Brainstorming über Verbesserungen statt.
- Ideen werden mit dem Partner abgeklärt.
- Kommt ein Programmierer bei einem Problem nicht weiter, übernimmt der andere die Initiative.
- Die Partner achten gegenseitig darauf, dass die Teampraktiken eingehalten werden.

Diese Methode hat eine starke Auswirkung auf die personellen Ressourcen, die für ein Projekt benötigt werden. Das ist auch der Grund, warum diese Methode von Managern oft abgelehnt wird, da diese einen doppelt so hohen Personaleinsatz vermuten. In einer Studie wurde aber gezeigt, dass sich der zusätzliche Personalaufwand im besten Fall auf 15 % reduzieren lässt, bei einer Verringerung der Entwicklungszeit von 40%. Außerdem wurden dadurch Fehler reduziert und die Qualität der entwickelten Software erhöht (Williams, et al., 2000 S. 4).

**Empirische Erhebungen:** In den untersuchten Firmen fand diese Praktik aufgrund der Ressourcenproblematik keinen Anklang, eine Verbesserung der Effizienz wurde bezweifelt.

**Bewertung:** Die Wirkungsweise des Pair Programmings zur Verbesserung von Effizienz und Effektivität kann durchaus in Zweifel gezogen werden. Der Zusammenhang mit Veränderungen und damit mit der Problemstellung dieser Arbeit ist auch unklar bzw. nicht direkt darstellbar. Deshalb wird diese Praktik vom Autor eher als Qualitätsmaßnahme, denn als Maßnahme zur Verbesserung der Agilität betrachtet.

## 5.3 Maßnahmen im Zusammenhang mit Mitarbeitern

Im folgenden Kapitel wird nicht auf die gesamte Theorie und jede Praktik zur positiven Beeinflussung von Mitarbeitern und Unternehmungskultur eingegangen. Das volle Spektrum aller Maßnahmen (Führungsstile, Mitarbeitermotivation, Ausbildungsprogramme, Kommunikationsmethoden etc.) würde den Umfang der Dissertation bei weitem sprengen. Es wird aber dargelegt, warum Mitarbeiter und Kultur so wichtig hinsichtlich Agilität sind und wo Ansatzpunkte bestehen, die Mitarbeiter zu fördern, um damit die Agilität zu steigern. Außerdem werden die häufig genannten, personenbezogenen Praktiken aus den agilen SW Entwicklungsmethoden vorgestellt.

### 5.3.1 Mensch im Mittelpunkt der Entwicklung

**Erklärung:** Die Mitarbeiter vollbringen die geistige Arbeit der Entwicklung. Sie sind es auch, die Veränderungen erkennen, bewerten und in der Entwicklung umsetzen. Außerdem gestalten sie Organisation, Prozesse und Produkte, weshalb ihre Fähigkeiten auch direkten Einfluss auf deren Flexibilität haben. Als fühlende Individuen sind sie aber auch empfindlich gegenüber einer Reihe von möglichen Problemen und oft auch abgeneigt gegenüber Veränderungen. Es hängen also viele wesentliche Aspekte der Agilität direkt von den Entwicklern ab. Im XP wird deshalb „Menschlichkeit“ als erstes Prinzip genannt (Beck, et al., 2005 S. 24).

Bei herkömmlichen Entwicklungsprozessen wird oft versucht, die Mitarbeiter an die Standards der Organisation anzupassen. Bei agilen Entwicklungsprozessen wird hingegen versucht, Kapital aus den individuellen Fähigkeiten der Mitarbeiter, aber auch aus den Fähigkeiten ganzer Teams zu schlagen (Cockburn, et al., 2001 S. 132).

Im Gegensatz zu herkömmlichen Methoden, bei denen das generierte Wissen in Prozessbeschreibungen, Trainingsunterlagen, Wissensdatenbanken und sonstiger Dokumentation gespeichert wird, ist bei agiler Entwicklung der Mensch der zentrale Wissensträger (Turner, et al., 2002 S. 158).

Weil Menschen von langen, abgekapselten Entwicklungsprozessen demotiviert werden, schlägt Mikio Aoyama einen auf Menschen fokussierten Entwicklungsprozess vor. Dabei werden die Entwickler auf Teams aufgeteilt, die jeweils Verantwortung für einen Teil des Systems erhalten. Dies soll Mitarbeitermotivation, Produktivität und Qualität steigern (Aoyama, 1998 S. 6).

Praktisch alle agilen SW Entwicklungsmethoden fordern also die Berücksichtigung personenbezogener Werte. Das Manifest für agile Softwareentwicklung fordert als erstes Prinzip, höheres Gewicht auf Personen und ihre Interaktionen statt auf Prozesse und Werkzeuge zu legen (Beck, et al., 2001).

**Empirische Erhebungen:** Ein Großteil der befragten Experten bezeichnete die Mitarbeiter als wichtigsten Erfolgsfaktor, nicht nur im Umgang mit Veränderungen, sondern generell für den Erfolg von Entwicklungsprojekten. Trotzdem wurden die Bedürfnisse der Mitarbeiter kaum als Kriterium bei der Gestaltung des Entwicklungsprozesses berücksichtigt. Im Mittelpunkt standen die Verkürzung der Entwicklungszeit, die Standardisierung der Prozesse, Rückverfolgbarkeit der Anforderungen im Prozess, Qualitätsmaßnahmen und ähnliches.

**Bewertung:** Die Anwendung dieses Prinzips könnte bei der Erreichung all dieser Ziele helfen und ist auf keinen Fall ein gegensätzliches Ziel. Berücksichtigt man es nicht, können Bedingungen geschaffen

werden, in denen die Mitarbeiter ihr Potential nicht ausschöpfen können oder wollen. Die Mitarbeiter können alle Teilfunktionen der Agilität steigern, vom Erkennen der Veränderungen, bis zum effizienten Umsetzen der Reaktionen darauf. Deshalb erscheint es einer der wichtigsten Punkte für ASE zu sein, darauf zu achten, dass die Entwickler bestmöglich ihre Arbeit verrichten können und durch organisatorische Regeln gefördert und keinesfalls behindert werden.

### 5.3.2 Ausbildung der Mitarbeiter

**Erklärung:** Da die agilen Methoden mehr auf die Fähigkeiten der Mitarbeiter setzen, als auf standardisierte Prozesse, sind die Fähigkeiten der einzelnen Mitarbeiter ein noch wichtigerer Erfolgsfaktor als bei herkömmlicher Entwicklung (Cockburn, et al., 2001 S. 131). Weil das Erkennen und Reagieren auf Veränderungen auch nach Erfahrung und Wissen verlangt, werden bei agiler Entwicklung in dieser Hinsicht besonders gute Mitarbeiter benötigt. Die Geschwindigkeit, mit der erfahrene Mitarbeiter arbeiten, kann auch um ein vielfaches höher sein, als die von unerfahrenen Mitarbeitern (Cockburn, 2002 S. 10). Deshalb verlangen agile Entwicklungsmethoden im Allgemeinen nach sehr kompetenten Entwicklungsteams (Wilson, et al., 2003 S. 793), (Boehm, et al., 2006 S. 52).

**Bewertung:** Auf Ausbildungsmaßnahmen wurde in der empirischen Studie nicht eingegangen. Es scheint aber gesichert zu sein, dass mit gezielten Ausbildungsmaßnahmen nicht nur der Wissensstand der Mitarbeiter, sondern damit auch die Agilität wesentlich erhöht werden kann. Z.B. kann ein Mitarbeiter mit mehr Fachwissen auf ein größeres Repertoire an Lösungsmöglichkeiten für eine auftretende Veränderung zurückgreifen. Größeres Methodenwissen kann zu einem effizienteren Entwickeln führen oder größeres interdisziplinäres Wissen kann, wenn Zusammenhänge zwischen den Disziplinen dadurch früher klar werden, ein früheres Erkennen von potentiellen Veränderungen ermöglichen. Weiter soll auf die Ausbildung nicht eingegangen werden. Sie soll nur als Prinzip verstanden werden, mit dem wesentlich auf die Agilität, als Fähigkeit der Mitarbeiter, Einfluss genommen werden kann.

### 5.3.3 Förderung von Motivation

**Erklärung:** Eine gesunde Work-Life-Balance, aus der auch eine motivierte, aktive und energievolle Arbeitsweise resultieren soll, wird im Extreme Programming als Praktik gefordert. Eine 40 Stunden Woche ist als Norm vorgesehen (Beck, et al., 2005 S. 6). Auch Anreizsysteme haben einen wesentlichen Einfluss auf die Mitarbeitermotivation. Bei einer Bezahlung rein nach Aufwand fehlt der Anreiz, die Ziele zu erreichen. Ein Anreizsystem welches die messbare Erfüllung von Kundenwünschen berücksichtigt, wird z.B. in der agilen Entwicklungsmethode von Tom Gilb vorgeschlagen (Gilb, 2005 S. 7f). Beides sind nur Beispiele, die zeigen sollen, dass die Mitarbeitermotivation auch eine Bedeutung für die Agilität hat. In den Prinzipien zum Manifest für agile SW Entwicklung, wird die Motivation ebenso erwähnt (Beck, et al., 2001). Auch das Lean SW Development Prinzip „Respect People“ zielt im Wesentlichen auf die Motivation ab (Poppendieck, et al., 2009 S. 36).

**Bewertung:** Ebenso wie die Ausbildung der Mitarbeiter, kann auch die Förderung von Motivation als Mittel zur Agilitätssteigerung gesehen werden. Dies kann z.B. dadurch funktionieren, dass motivierte Mitarbeiter sich verstärkt darum kümmern, Informationen aus der Entwicklungsumgebung aufzunehmen, zu kombinieren und dadurch potentielle Veränderungen früh zu erkennen. Ebenso ist von ihnen effizienteres Arbeiten zu erwarten, auch im Umgang mit Veränderungen, genauso wie der Wunsch

zur ständigen Verbesserung der Produkte oder Prozesse. Auf konkrete Motivationsmodelle und Maßnahmen zur Motivierung der Mitarbeiter soll hier nicht eingegangen werden.

### 5.3.4 Förderung von Teamwork

**Erklärung:** Eine enge Zusammenarbeit des Teams wird bei agiler Entwicklung als kritischer Erfolgsfaktor gesehen (Derby, 2007 S. 8). Jim Highsmith betrachtet die Zusammenarbeit als zentrale Fähigkeit eines Teams für die Erreichung von Ergebnissen. Als wichtigste Faktoren dafür nennt er Vertrauen und Respekt (Highsmith, 2010 S. 55). Auch der Projektleiter hat einen wesentlichen Einfluss auf die Zusammenarbeit im Team. Vor allem beim Einsatz zyklischer Entwicklung in kleinen Schritten treten schon sehr früh Konflikte zutage, für die der Projektleiter ein geeignetes Konfliktmanagement überlegen muss (McMahon, 2006 S. 26). Gute Zusammenarbeit beschleunigt die Weitergabe wichtiger Informationen und reduziert damit die Kosten der Informationsweitergabe (Cockburn, et al., 2001 S. 131). Mary und Tom Poppendieck sehen die Vereinigung von individuellen Fähigkeiten und Zusammenarbeit, als die entscheidenden Merkmale, die ein gutes Team ausmachen (Poppendieck, et al., 2009 S. 126).

**Bewertung:** Wie bei der Motivation soll die Förderung von Teamwork als Prinzip zur Steigerung der Agilität aufgezeigt werden, ohne auf konkrete Maßnahmen einzugehen.

### 5.3.5 Optimierung der Kommunikation

**Erklärung:** Es wurden bereits einige Praktiken und Prinzipien vorgestellt, welche die Kommunikation im Team (z.B. in der Organisation) oder zum Kunden verbessern sollen. Hier soll nun die direkte Förderung der Kommunikation im Team diskutiert werden.

Kommunikation, als essentielle Voraussetzung für effektive Zusammenarbeit, wird im XP als erstes Prinzip genannt (Beck, et al., 2005 S. 18). Hansson sieht effektive Kommunikation als wesentliches Element jeder agilen Entwicklung (Hansson, et al., 2006 S. 1298). Die Kosten der Informationsweitergabe, können z.B. durch Wahl des geeigneten Informationskanals reduziert werden (Cockburn, et al., 2001 S. 131).

Je nach Kommunikationskanal gibt es unterschiedliche Informationsarten, die übertragen werden können. Bei der direkten Kommunikation wird nicht nur die Sprache übertragen, sondern auch Gestik und Mimik. Außerdem kann die Sprache unterschiedlich betont werden und es kann immer eine unmittelbare Rückfrage stattfinden. Bei allen anderen Kommunikationskanälen gehen manche dieser Dinge verloren, die Reichhaltigkeit ist geringer. Das verringert die Effektivität dieser Kommunikationsarten (Cockburn, 2002 S. 12). In Abb. 20 sind für verschiedene Kommunikationsarten Reichhaltigkeit und Effektivität dargestellt.

Die Verwendung von direkter Kommunikation und von Whiteboards ist also viel effektiver als das Lesen von Dokumentation (Paper). Auch gute Zusammenarbeit und gutes Betriebsklima sind wichtig, um die Mitarbeiter zur schnellen Weitergabe von wichtigen Informationen zu ermutigen. Eine Voraussetzung zur Umsetzung dieser Maßnahmen ist allerdings die örtliche Nähe der Teammitglieder (Cockburn, et al., 2001 S. 131). In der Methode Crystal Clear beschreibt Cockburn unter dem Namen „Osmotische Kommunikation“ wie die Kommunikation in einem sehr kleinen Team im besten Fall funktionieren soll (Cockburn, 2005 S. 57ff).

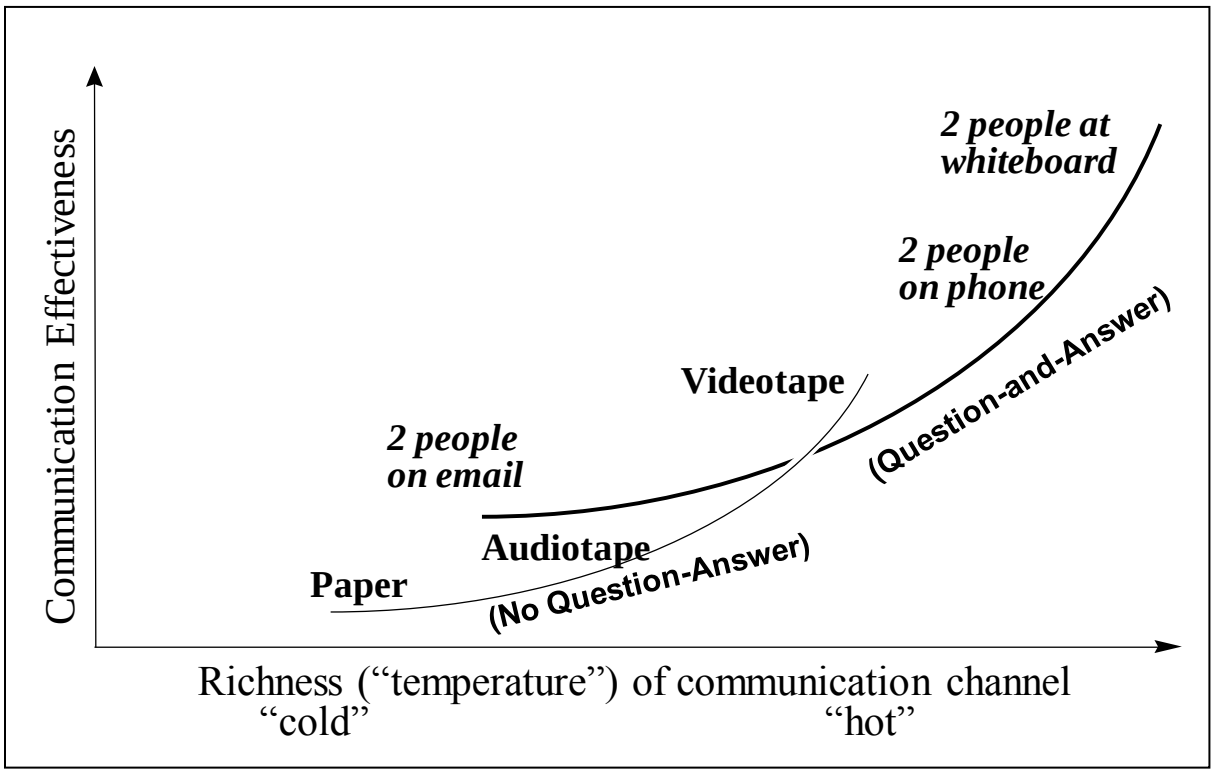


**Abb. 20: Effektivität und Reichhaltigkeit von Kommunikationsmitteln (Cockburn, 2002 S. 13)**

**Empirische Erhebungen:** In der empirischen Studie wurde die Kommunikation diskutiert, ohne zu sehr ins Detail zu gehen. Es waren vor allem Fragen der Organisation (Hierarchie) die als Einflussfaktoren für eine schnelle Kommunikation genannt wurden. Auf jeden Fall wurde die große Wichtigkeit der Kommunikation im Allgemeinen und speziell hinsichtlich Agilität bestätigt. Und in fast allen Unternehmungen werden Anstrengungen unternommen, die Kommunikation zu verbessern. In der Expertengruppe wurde der Unterschied in der Nomenklatur zwischen den Domänen Hardware und Software als potentielles Problem, speziell im Systems Engineering, hervorgehoben. Einige Begriffe werden unterschiedlich verwendet. Für einige Sachverhalte oder Dokumente gibt es auch unterschiedliche Namen (z.B. Lastenheft, Pflichtenheft, Anforderungskatalog, Anforderungsspezifikation, Produktspezifikation und einige mehr im Zusammenhang mit Dokumenten zu den Anforderungen). Der Bezug zur Agilität ist hier aber nicht so groß wie bei den zuvor dargestellten Sachverhalten.

**Bewertung:** Die Optimierung der Kommunikation ist also auch eine Möglichkeit zur Steigerung der Agilität. Vor allem für die schnelle Erkennung von Veränderungen ist sie sehr wichtig. Neben den zahlreichen Prinzipien und Praktiken, die in diesem und den vorigen Kapiteln aufgezeigt wurden, wird auf die Literatur zur Kommunikation verwiesen[5].

### 5.3.6 Wertschätzung von implizitem Wissen (Tacit Knowledge)

**Erklärung:** Implizites Wissen wird in den Managementwissenschaften vielfach als Wissen verstanden, welches noch nicht ausformuliert ist und auf eine Weiterverarbeitung zu explizitem Wissen wartet (z.B.

[5] Eine umfangreiche Aufstellung befindet sich unter http://de.wikipedia.org/wiki/Kommunikation.

durch Dokumentieren). Dieses Verständnis entspricht aber nicht dem ursprünglich von Michael Polani 1966 definierten Prinzip. Dieses beschreibt ein Wissen oder Können, das nicht ausformuliert werden kann, sondern nur in den Köpfen von Personen steckt. Es ist kein „Wissen, was“ sondern ein „Wissen, wie“ und ist notwendig für jede praktische Anwendung von theoretischem Wissen (Tsoukas, 2003).

Barry Boehm sieht implizites Wissen als eines der Hauptkonzepte agiler Entwicklung, weil dabei wichtiges Wissen in den Köpfen der Entwickler und nicht in den Dokumenten steckt (Boehm, et al., 2005 S. 32). Als Voraussetzung dafür sieht er ein hochqualifiziertes Team (Boehm, et al., 2006 S. 20). Auch Highsmith meint, dass schriftliche Dokumentation nicht ausreichend für die Interaktion von Entwicklern sein kann. Er glaubt, dass Dokumentation das benötigte Verstehen nur zu einem geringen Anteil vermitteln kann. Man kann deshalb nicht das Wissen des Teams und die ganze Entwicklungstätigkeit mit Dokumentation abbilden und sollte das implizite Wissen mehr wertschätzen (Highsmith, 2010 S. 287ff).

**Empirische Erhebungen:** In den Interviews hat sich gezeigt, dass sich Systementwickler dahingehend kaum Gedanken machen. Es wird sogar versucht, noch mehr zu dokumentieren. Z.B. weil während der Projekte Mitarbeiter die Firma verlassen können und ihr Wissen gesichert werden soll. Außerdem kann in großen Teams nicht immer Kontakt zwischen allen Teammitgliedern herrschen, weshalb versucht wird das Wissen durch Dokumentation für alle verfügbar zu machen, auch für eine Anwendung in späteren Projekten. Es wurde aber auch mehrfach von gescheiterten Versuchen berichtet, Entwicklungsprozesse möglichst detailliert zu dokumentieren. Danach wurde jeweils erfolgreicher in den Prozessen nur grob definiert, was darin zu tun ist und den Entwicklern mehr Freiheit beim „Wie?“ und auch bei der detaillierteren Festlegung des „Was?“ gegeben.

**Bewertung:** Dieses Prinzip wird in agilen SW Entwicklungsmethoden häufig erwähnt, aber selten hinsichtlich agiler Aspekte beleuchtet. Es scheint eher eine Erweiterung des Prinzips „Mensch im Mittelpunkt der Entwicklung“ zu sein und die Agilität nur indirekt zu betreffen. Es gibt den Mitarbeitern mehr Freiheit, effizient und effektiv zu handeln, anstatt sie zu zwingen, viel Dokumentation zu produzieren oder sich nach ihr zu richten, auch wenn das nicht sinnvoll ist. Weil die Weitergabe von implizitem Wissen wichtig für das Entwickeln von Systemen ist, spricht sich das Prinzip auch ganz klar für Entwicklungsbedingungen aus, die das ermöglichen. Diese sind, wie bei den agilen SW Methoden, kleine Teams mit guter Zusammenarbeit bei denen die Teammitglieder möglichst im gleichen Raum untergebracht sind.

### 5.3.7 Positive Einstellung gegenüber Veränderungen

**Erklärung:** In der Softwareentwicklung wird eine Akzeptanz oder positive Einstellung gegenüber Veränderungen als wichtige Grundvoraussetzung für agiles Entwickeln genannt (Beck, et al., 2001), (Beck, et al., 2005), (Boehm, et al., 2005 S. 32).

In einer Studie zu Systementwicklungsprojekten wurde die agilitätssteigernde Wirkung der „Bereitschaft zu Veränderungen“ auch quantitativ nachgewiesen (Maierhofer, et al., 2010 S. 68).

**Bewertung:** Bei diesem Prinzip sollte ebenfalls nur gezeigt werden, dass es einen Einfluss auf die Agilität hat. Wie man konkret die Einstellung von Mitarbeitern verändern kann, ist nicht Inhalt dieser Dissertation. Wie die Wirkung dieser Praktik erzielt wird, wurde in der Literatur nicht begründet. Der Autor vermutet aber, dass eine positive Einstellung die Mitarbeiter dazu veranlasst, Anstöße zu

Veränderungen nicht prinzipiell zurückzuweisen, sondern sie zu prüfen, gegebenenfalls zu akzeptieren und aktiv nach Verbesserungsmöglichkeiten zu suchen. Außerdem hängt die Motivation zur effizienten Behandlung von Veränderungen vermutlich auch mit der Einstellung ihnen gegenüber zusammen.

### 5.3.8 Improvisation

**Erklärung:** Als Improvisation soll hier die ungeplante Durchführung einer Aktion aufgrund einer ad-hoc Beurteilung bezeichnet werden. Aber auch das Abweichen von einer festgelegten Regelung, die man im konkreten Fall als nicht geeignet betrachtet. Auch ohne explizite Vorgehensweisen oder Techniken kann in der Entwicklung auf Veränderungen reagiert werden. Deshalb soll auch Improvisation als potentielle agile Praktik angesehen werden.

Laut einer Industriestudie wird in der Praxis auch viel mehr auf Improvisation gesetzt, als die Theorie zu herkömmlichen Entwicklungsmethoden behauptet (Hansson, et al., 2006 S. 1296f, 1309). Z.B. wird improvisiert, um bei offizieller Verwendung eines Wasserfall-Prozesses, trotzdem noch sehr spät in der Entwicklung Anforderungsänderungen möglich zu machen (Hansson, et al., 2006 S. 1303).

Improvisation kann in praktisch allen Bereich durchgeführt werden. Beispiele sind beim Produkt (technische Lösungen die nicht geplant wurden), im Entwicklungsprozess (ungeplante Tätigkeiten) oder in der Organisation (ungeplante Aufgabenverteilung).

**Empirische Erhebungen:** Auch in der Praxis wurden sowohl innerhalb der Expertengruppe, als auch bei den Interviews zahlreiche Beispiele gefunden, wo Dinge von den Entwicklern der Situation angepasst, anders gemacht wurden, als geplant. Bezüglich der definierten Entwicklungsprozesse wird es von den Entwicklern oft als normal erachtet, dass diese nur einen ungefähren Leitfaden darstellen und in zahlreichen Fällen nicht angewandt werden können. Zur erfolgreichen Erledigung der Entwicklungsarbeit ist es dann notwendig, zu improvisieren. Auch hinsichtlich der technischen Gestaltung des Produkts wurde bei auftretenden Veränderungen die Improvisation als sehr häufig Reaktionsmaßnahme genannt.

**Bewertung:** Es steht außer Frage, dass Improvisation in der Praxis eine häufige Vorgehensweise nach Veränderungen ist. Zur Steigerung der Agilität stellt sich aber die Frage, welche Einflussfaktoren zu einer möglichst guten Improvisation führen. Nachdem die Mitarbeiter die Durchführenden sind, liegt es nahe ihr Wissen, Können und ihre Motivation als Hauptfaktor zu suchen. Es muss aber auch die Möglichkeit bzw. einen Handlungsspielraum zur Improvisation geben, weshalb auch Anpassbarkeit oder Reserven in Prozessen, Produkten und Organisation notwendig sind. Eine Gefahr bei der Anerkennung von Improvisation als agile Praktik ist die mögliche Vernachlässigung von definierten Praktiken oder Plänen. Sie darf auch nicht mit der Anwendung von implizitem Wissen verwechselt werden, sondern ist nur der Versuch unerwartet aufgetretene Probleme zu meistern, für die es keine geplante Vorgehensweise gibt oder für die diese sich nicht eignet.

### 5.3.9 Angemessene Arbeitszeit

**Erklärung:** In einer älteren Version von Extreme Programming wurde eine 40-Stunden Woche explizit als Praktik gefordert. Falls doch notwendig, sollte man maximal für eine Woche länger arbeiten. Begründet wurde das durch die sinkende Motivation und geringere Leistungsfähigkeit von überlasteten Mitarbeitern (Paulk, 2001 S. 3). In der aktuellen Version von XP wurde der Namen der Praktik auf

„Energievolle Arbeit“ geändert. Darin wird gefordert, dass Mitarbeiter nicht länger arbeiten sollen, als sie dies mit voller Konzentration tun können (Beck, et al., 2005 S. 41).

**Bewertung:** Nachdem die Mitarbeiter als sehr wichtiger Einflussfaktor für die Agilität erkannt wurden, soll mit dieser Praktik noch ein wichtiger Faktor für deren Leistungsfähigkeit dargestellt werden. Eine tiefergehende Untersuchung angemessener Arbeitszeiten wird in dieser Arbeit aber nicht durchgeführt.

## 5.4 Maßnahmen im Entwicklungsprozess

Hier werden Prozesspraktiken und –prinzipien aufgezeigt, die eine agilere Vorgehensweise in der Entwicklung unterstützen sollen. Z.B. solche, die ein frühes Erkennen von Änderungen fördern oder die Effizienz der Entwicklung und speziell den Umgang mit Veränderungen verbessern sollen. Aber auch Prinzipien, die den Prozess selbst flexibel bzw. anpassbar machen sollen, werden hier aufgezeigt, ebenso wie Maßnahmen im Prozess, die sich mit der flexiblen Gestaltung des Produkts beschäftigen.

### 5.4.1 Prozessgestaltung durch Team

**Erklärung:** Wie schon in 5.2.4 angekündigt, soll es dem Team nicht nur gestattet werden, die Organisation selbst zu gestalten, sondern auch den Prozess. Das Entwicklungsteam selbst soll also die Verantwortung für die Entwicklungsprozesse tragen. Wird der Prozess von Personen gestaltet, die zwar Experten der Theorie sind, aber denen die Sichtweise eines Entwicklers fehlt, besteht Gefahr für die Anwendbarkeit des Prozesses (Aoyama, 1998 S. 10).

Eine weitere Gefahr besteht auch in der Verwendung von Standardprozessen. Auch wenn für konkrete Projekte jeweils Anpassungen stattfinden, können doch unnötige Prozessschritte enthalten sein. Speziell wenn versucht wird, den Prozess mit agilen Praktiken zu erweitern, kann das die Steigerung der Agilität wieder verhindern. Boehm und Turner sehen es deshalb als vorteilhaft an, wenn die Prozesse vom Team von Grund auf entwickelt werden. Damit soll sichergestellt werden, dass nur jene Prozessschritte ausgewählt werden, die eine effektive und effiziente Projektdurchführung gewährleisten (Boehm, et al., 2005 S. 31f).

Die Agilität darf aber nicht als Ausrede vom Team verwendet werden, um unzureichende oder nicht vorhandene Prozessdefinitionen zu rechtfertigen. Ein gut definierter Prozess wird als Basis für den Erfolg von agiler Entwicklung gesehen (Aoyama, 1998 S. 10). Das Team hat also sicherzustellen, dass die Prozesse ausreichend definiert sind und auch eingehalten werden.

**Empirische Erhebungen:** Weder bei den Interviews, noch in der S²QI Expertengruppe konnte eine Umsetzung dieses Prinzips in der Praxis gefunden werden. Sowohl Kunden als auch Management wollen definierte und nachvollziehbare Prozesse. Oft wird auch eine Standardisierung angestrebt. Prozesse werden auch in diesem Sinne zertifiziert und Mitarbeiter, die in mehreren Projekten gleichzeitig arbeiten, sollen sich anhand der standardisierten Prozesse leichter orientieren können.

Gefunden wurden aber Versuche, Flexibilität in den Entwicklungsprozess zu bringen. Z.B. wird bei manchen Firmen im Projekt Kick-Off der Prozess je nach Projektgröße angepasst. Auch wurden Prozessanpassungen im Projektverlauf erwähnt. Diese wurden aber nicht vom Team selbst, sondern

immer unter der Verantwortung des Prozessmanagers durchgeführt. Die Flexibilität des Prozesses wird in Kapitel 5.4.13 diskutiert.

**Bewertung:** Betrachtet man ein einzelnes Projekt, so kann man durchaus einige Vorteile in der Prozessgestaltung durch das Team erkennen. Dieses führt die Entwicklung durch, beschäftigt sich täglich mit dem Prozess und es ist davon auszugehen, dass es selbst am besten weiß, wie das Projekt effektiv und effizient umgesetzt werden kann. Außerdem kann das Team den Prozess bei Veränderungen am schnellsten anpassen, wenn es dazu auch berechtigt ist. Betrachtet man aber die Unternehmung als Ganzes, so erscheint auch eine Umkehrung dieser Vorteile ins Negative möglich. Über viele Projekte hinweg kann oder soll nämlich ein standardisierter Prozess für mehr Effizienz sorgen. Die Mitarbeiter kennen diesen und können jedes neue Projekt sofort damit beginnen. Ein Austausch von Mitarbeitern ist leichter möglich und das Management erhofft sich auch eine bessere Planbarkeit. Bei großen Projekten wäre es außerdem schwierig und langwierig, wenn sich das Team jedes Mal einen neuen Prozess überlegen müsste. Die Umsetzung dieses Prinzips muss also vorher sehr genau überprüft werden bzw. erscheint nur in bestimmtem Kontext sinnvoll zu sein.

### 5.4.2 Simultaneous Engineering

**Erklärung:** Diese, auch unter dem Namen „Concurrent Engineering“ bekannte, Vorgehensweise lässt Prozesse parallel ablaufen, die normalerweise nacheinander abgearbeitet werden würden (Abb. 21).

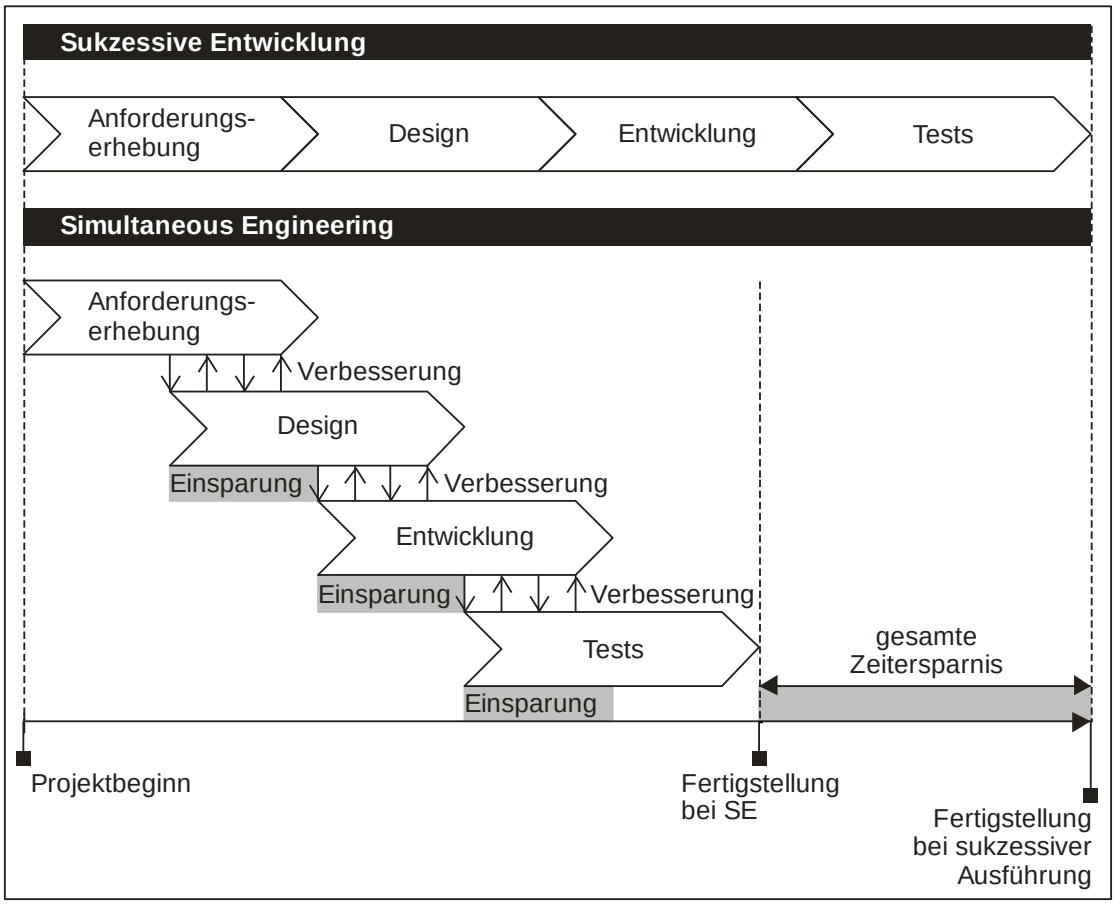


**Abb. 21: Simultaneous Engineering, Grafik: Maierhofer (2009 S. 31)**

Damit will man im Wesentlichen drei Ziele erreichen (Ehrlenspiel, et al., 2007 S. 44ff):

- Zeitgewinn: Wie in Abb. 21 gezeigt, werden Prozesse so weit als möglich parallelisiert, damit sich ein Zeitgewinn zur sukzessiven Abwicklung einstellt. Um korrekt entscheiden zu können, welche Arbeiten der verschiedenen Phasen vorgezogen werden sollen, muss ein durchgängiger Informationsaustausch zwischen allen beteiligten Personen stattfinden. Ein zentrales Team ist für die Kommunikation, Planung der Abläufe und Fortschrittsüberwachung zuständig.
- Kostenminimierung: Durch die verkürzte Entwicklungszeit ist mit geringeren Gesamtkosten für die Entwicklung zu rechnen. Vor allem auch deshalb, weil Änderungen aufgrund des Informationsaustausches früher bekannt werden und dadurch kostengünstiger implementiert werden können. Kurzfristig ist natürlich mit höherem Ressourcenaufwand zu rechnen.
- Qualitätsverbesserung: Dem Auftraggeber kann durch die rasche praktische Umsetzung seiner Aufgabenstellung in relativ kurzer Zeit ein vorläufiges Ergebnis präsentiert werden. Anhand des Kundenfeedbacks können falsch erfasste Anforderungen korrigiert und so ein besseres Endprodukt erstellt werden. Voraussetzung ist eine kontinuierliche Zusammenarbeit mit dem Kunden und ehrliches Feedback zu den Zwischenergebnissen.

Laut Alistair Cockburn ist Simultaneous Engineering als ein grundlegendes Prinzip agiler Ansätze zu betrachten, weil nur so Veränderungen auch noch in späten Phasen der Entwicklung eingebracht werden können, ohne dass frühere Phasen bereits vorher abgeschlossen wurden und deshalb wieder neu gestartet werden müssen. Als Voraussetzung ist allerdings eine effektive (schnelle, reichhaltige und billige) Kommunikation zu sehen (Cockburn, 2002 S. 9f).

Haberfellner und de Weck machen aber darauf aufmerksam, dass diese Vorgehensweise auch zu einem erhöhten Risiko für Fehlinvestitionen und Nacharbeit führen kann, weil hier Ergebnisse einer Entwicklungsphase an die nächste weitergegeben werden, bevor die erste abgeschlossen ist (Haberfellner, et al., 2005 S. 6).

**Empirische Erhebungen:** In der Praxis wurde Simultaneous Engineering als etablierte Vorgehensweise gefunden, es wurden dabei dieselben Vorteile wie in der Literatur genannt.

**Bewertung:** Die Agilität kann durch Simultaneous Engineering gesteigert werden, indem es Informationen aus späteren Entwicklungsphasen früher verfügbar und dadurch potentielle Änderungen schneller erkennbar macht. Eine gewisse Sequenz der Tätigkeiten muss aber beachtet werden, da sonst Nacharbeit die erwarteten Vorteile zunichtemacht. Man kann ein sehr schnell aufeinanderfolgendes und wiederkehrendes Analysieren, Planen, Realisieren und Testen in Entwicklungszyklen auch als paralleles Entwickeln betrachten. Dann kann Simultaneous Engineering als ein grundlegendes Prinzip des AEZ gesehen werden. Aber auch unabhängig einer zyklischen Entwicklung bietet Simultaneous Engineering eine Möglichkeit zur Agilitätssteigerung.

***Simultane Entwicklung von Hardware und Software***

Ein Aspekt des Simultaneous Engineering, der für die gemeinsame Entwicklung von HW und SW wichtig ist, soll hier noch erwähnt werden. Es geht dabei um die forciert gleichzeitige Entwicklung von HW und SW, gelegentlich wird auch von „Software first“ gesprochen. Unter vielen heutzutage entwickelten Systemen kann kaum noch Hardware verstanden werden, „auf der etwas Sofware läuft“. Zahlreiche

Systeme sind viel mehr Software mit „ausreichend Hardware, damit die Software darauf lauffähig ist“. Software liefert dabei oft das entscheidende Potential eines Systems, macht dessen Flexibilität aus, erlaubt schnelle Reaktion auf Veränderungen in der Umwelt und repräsentiert den wesentlichen Teil des Wertes eines Systems. Deshalb ist es nicht mehr sinnvoll, so wie es bei herkömmlicher Entwicklung oft vorkommt, dass zuerst die Hardware entwickelt wird und die Softwareentwickler dann von den Hardwareentwicklern regelrecht beauftragt werden, das System zu vervollständigen. Viel besser erscheint es, die Software gleichzeitig mit der Hardware, womöglich sogar davor zu entwickeln. Nur so kann man den vollen Nutzen aus der Flexibilität und Anpassbarkeit der Software ziehen, diese und das Gesamtsystem weniger komplex machen und die Systemarchitektur besser abstimmen (Turner, 2007 S. 14). Im Rahmen einer agilen Entwicklung sollen also HW und SW simultan entwickelt werden.

### 5.4.3 Entwicklung in kleinen Schritten

**Allgemeine Erklärung:** In den Methoden der agilen SW Entwicklung wird das Vorgehen oft als zyklisch, inkrementell oder iterativ beschrieben. Diese Vorgehensweisen werden als spezielle Ausprägungsformen des hier verallgemeinerten Prinzips der Entwicklung in kleinen Schritten verstanden, worauf weiter unten eingegangen wird.

Entwicklung in kleinen Schritten bedeutet den Gesamtaufwand der Entwicklung in kleinere Teile aufzuteilen. In erster Linie hat das natürlich organisatorische Gründe. Erfüllt diese Aufteilung gewisse Merkmale, kann sie aber auch die Agilität steigern. Zur Übersicht werden diese Merkmale in Abb. 22 dargestellt. Genaue Definitionen und Erklärungen folgen danach. In der Abbildung werden auch die Ausprägungsformen dieser Merkmale dargestellt, die notwendig sind, um den AEZ abzubilden. In dieser Ausprägungsform soll die Entwicklung in kleinen Schritten (nach gängigem Verständnis) die bestmögliche Steigerung der Agilität gewährleisten (siehe 4.3.4).

Bevor das Prinzip der Entwicklung in kleinen Schritten nach diesen Merkmalen analysiert wird, sollen noch die Ziele seiner Anwendung (bzw. Vorteile) aus der Literatur wiedergegeben werden. Danach werden je Merkmal auch empirischen Beobachtungen dargestellt und eine Bewertung durchgeführt. Am Ende des Kapitels folgt noch eine allgemeine Bewertung.

Für soziale Systeme wies der Philosoph Karl Popper schon 1944 auf den Vorteil hin, den kleine gegenüber großen, revolutionären Entwicklungsschritten bieten. Es können dabei die Auswirkungen leichter beobachtet und Fehler korrigiert werden. Im SE soll man aber Problemfelder durchaus weiter fassen und zuerst umfassende Lösungskonzepte erarbeiten. Die Realisierung sollte aber in kleinen Schritten erfolgen, die leicht rückgängig gemacht werden können, wenn sie sich als unzweckmäßig erweisen (Haberfellner, et al., 2002 S. 23).

In der Literatur zur traditionellen SE Methodik wird auch auf die Notwendigkeit von iterativen Zyklen in der Entwicklung hingewiesen (Büchel, 1969 S. 381f). Entwicklung ist eben kein linearer Vorgang, der mit einmaligem Durchlaufen eines Entwicklungsprozesses von einem Problem zu einer Lösung führt. Es müssen immer wieder Rückschritte gemacht werden, z.B. wenn neu gewonnene Informationen es erforderlich machen. In der Softwareentwicklung wird iterative Entwicklung deshalb seit vielen Jahrzehnten angewandt und nicht erst seit Bekanntwerden der agilen Methoden (Larman, et al., 2003).

| **Merkmal** | **Erklärung** | **Ausprägung im AEZ** |
|---|---|---|
| Nutzbare Zwischenergebnisse | Kann der Kunde die Zwischenergebnisse bereits nutzen, kann er zumindest ein Feedback dazu abgeben? Oder können die Ergebnisse der einzelnen Entwicklungsschritte nur intern oder überhaupt nicht bewertet werden? | Jeder Entwicklungsschritt soll vom Kunden (Anwender) genutzt werden können, weil dabei das beste Feedback zu erwarten ist. |
| Schrittgröße/Zyklusdauer | Welchen Umfang sollen die Entwicklungsschritte haben bzw. wie lang sollen die Zyklen dauern? | Größe („klein“) und Dauer („kurz“) entsprechend den Zielen des AEZ. |
| Zyklische Entwicklung | Erfolgen in jedem Entwicklungsschritt sämtliche Prozessschritte (Analyse, Planung, Realisierung und Test) oder sind die Entwicklungsschritte nur Wiederholungen und Rückschritte hinsichtlich einzelner Prozessschritte? Ebenso ist ein kleinerer Zyklus (z.B. nur Analyse und Planung) für die Entwicklung denkbar, gefolgt von einer Realisierungsphase, o.Ä. | Es ist geplant, in jedem Schritt sämtliche Prozessschritte (Analyse, Planung, Realisierung, Test) durchzuführen. |
| Art der Unterteilung | Möglichkeiten sind u.a.:<br>• Inkrementell (Das System wird schrittweise erweitert.)<br>• Iterativ (Der Prozess wird mehrfach wiederholt.)<br>• Phasen (Zeitliche Einteilung z.B. nach Detaillierungsgrad der Lösungen.)<br>• Releases (Software-Versionen mit definiertem Funktionsumfang.)<br>• Systemkomponenten | Inkrementell und iterativ bzw. jede Unterteilung, die eine Entwicklung nach den anderen Ausprägungen des AEZ (Nutzbare Zwischenergebnisse, Schrittgröße/Zyklusdauer, zyklische Entwicklung, Vorausplanung der Unterteilung) erlaubt. |
| Expliziter Prozess | Ist die Entwicklung in kleinen Schritten ein gewollter, definierter Prozess oder entsteht sie nur zufällig und ungewollt nach Fehlern, Missverständnissen, schlechter Abstimmung etc.? | Ein explizit gewollter und im Vorhinein definierter Prozess. |
| Vorausplanung der Unterteilung | Wird der Inhalt der einzelnen Schritte zu Beginn des Projekts definiert oder ergibt er sich im Laufe der Entwicklung? | Der genaue Inhalt eines Schrittes wird durch die vorhergehenden Schritte beeinflusst und erst zu Beginn des Schrittes definiert. |
| Periodizität | Soll die Zykluszeit konstant sein? | Konstante Zykluszeit. |

**Abb. 22: Merkmale der Entwicklung in kleinen Schritten**

Aufgrund der Unmöglichkeit, alle Eventualitäten einer Entwicklung vorhersehen zu können, meint Gilb, dass zu detailliertes Vorausplanen nicht sinnvoll und es besser ist, in kleinen Schritten zu arbeiten. Dadurch sollen die Ziele auch schneller erreicht werden. Der Kundennutzen kann in kleinem Ausmaß ev. schon nach dem ersten Zyklus hergestellt werden. Außerdem kann man selbst früh zu praktischer Erfahrung mit dem zu entwickelnden System gelangen (Gilb, 2008 S. 7).

Kaindl nennt folgende Vorteile für die iterative und inkrementelle Softwareentwicklung im Gegensatz zu einer Entwicklung in einem großen Schritt (Wasserfallprozess) (Kaindl, 2005 S. 4):

- Unklare und veränderliche Anforderungen sind besser handhabbar.
- Eine laufende Integration geschieht von selbst.
- Ab einem gewissen Zeitpunkt steht immer eine verwendbare Version zur Verfügung.
- Entwicklungsrisiken werden reduziert.
- Fehlerkosten werden reduziert, weil die Fehler schneller entdeckt werden.

Folgendes wird von Kaindl aber als Herausforderung einer Entwicklung in kleinen Schritten gesehen (Kaindl, 2005 S. 4):

- Eine stabile Architektur wird benötigt.
- Es muss eine konkrete Vision für die Entwicklung existieren.
- Es wird ein dynamisches Projektmanagement benötigt.
- Die Abnahmetests werden komplizierter.

Cockburn und Highsmith sehen die Entwicklung in kleinen Schritten als Basisprinzip jeder agilen Entwicklung und weisen darauf hin, dass hiermit die Zeitdauer zwischen Entscheidungen und dem Sichtbarwerden der Konsequenzen der Entscheidungen verkürzt wird (Cockburn, et al., 2001 S. 131). Außerdem kann durch frühes und häufiges Liefern (von Teilen des Systems) die Zeit zwischen der Aufnahme eines Kundenwunsches und der Bestätigung, dass dieser richtig verstanden wurde, reduziert werden (Dove, et al., 2008 S. 5).

Eine Entwicklung in kleinen Schritten ist auch in psychologischer Hinsicht vorteilhaft. Es liegt in der Natur vieler Menschen unter dem Druck einer nahenden Deadline besonders effizient und zielgerichtet zu arbeiten. Bei vielen kleinen Schritten, für die auch jeweils eine Deadline gesetzt wird, spürt man den Zeitdruck viel häufiger. Wichtig dabei ist aber, dass die verfügbare Zeit in den einzelnen Schritten ausreichend für die gestellten Aufgaben ist (Malotaux, 2006 S. 3), (Boehm, et al., 2005 S. 32).

Durch die sich wiederholenden Tätigkeiten, kann mit einem Vergleich zwischen geplanten und abgeschlossenen Tätigkeiten auf die Arbeitsgeschwindigkeit des Teams geschlossen werden. Das kann auch getan werden, falls die Arbeit einer Iteration als nicht zielführend erachtet und verworfen wurde (McMahon, 2006 S. 28). Auch der Entwicklungsprozess wird getestet, Schwachstellen im Prozess treten schneller zutage und er kann nach jedem Entwicklungsschritt verbessert werden (Gilb, 2006 S. 9f).

Die ersten Inkremente sind dabei nicht als „quick and dirty“ Prototypen zu verstehen. Jedes Inkrement soll die Qualitäts-Standards der Branche erfüllen. Dadurch sollen folgende Vorteile erreicht werden: Die grundlegende Architektur wird dahingehend getestet, ob mit ihr auch höhere Qualitätslevel erreicht werden können. Die frühen Inkremente können schon richtigen Benutzern zugänglich gemacht und von diesen getestet werden. Die wahren Kosten für die Entwicklung und die Hardware-Komponenten, die

benötigt werden, um den Qualitäts-Standard zu erreichen, können besser abgeschätzt werden. Projektverzögerungen aufgrund von nachträglich zu erreichenden Qualitäts-Standards werden unwahrscheinlicher (Gilb, 2006 S. 10f).

Hinsichtlich Risikobeherrschung sind zwei Vorteile erkennbar. Einerseits wird das Risiko einer Fehlentwicklung reduziert, weil man immer nur den Aufwand für einen Zyklusdurchgang riskiert und danach das entwickelte Inkrement einer Prüfung unterziehen kann. Andererseits erhält man wegen des früheren Vorhandenseins von Zwischenergebnissen auch früher Informationen über das zu erwartende Systemverhalten und damit auch über potentielle Risiken. Man hat also auch mehr Zeit um sich auf Risiken, die nach Fertigstellung des Systems schlagend werden können, vorzubereiten (Malotaux, 2006 S. 6).

Dieses Prinzip bietet also zahlreiche Möglichkeiten zur Agilitätssteigerung. Deshalb ist es kaum verwunderlich, dass es in der Ausprägung des AEZ oft als konkrete Umsetzung agiler Entwicklung gesehen wird, wie in Kapitel 4.3 beschrieben wurde. Dazu müssen aber bestimmte Merkmale erfüllt sein, wie bereits in Abb. 22 gezeigt wurde. Um zu verdeutlichen wie Entwicklung in kleinen Schritten und AEZ zusammenhängen, soll folgende Darstellung dienen:

***AEZ = Entwicklung in keinen Schritten (nach bestimmten Merkmalen)***

***+ Flexibles Anforderungsmanagement***

***+ Priorisierung der Entwicklungsschritte***

***+ Kundeneinbindung (Feedback)***

***+ Timeboxing***

Flexibles Anforderungsmanagement, Priorisierung, Kundeneinbindungen und Timeboxing sind auch separat denkbar, deshalb werden sie in anderen Kapiteln diskutiert. Im Weiteren wird auf die Merkmale der Entwicklung in kleinen Schritten eingegangen:

***Nutzbare Zwischenergebnisse***

**Erklärung:** Scrum fordert, dass jeder Entwicklungsschritt lieferbar und die Funktionalität sofort implementierbar sein soll, auch wenn keine Lieferung vorgesehen ist (Schwaber, 2004 S. 12f).

Gilb meint, dass jedes Inkrement zumindest einen kleinen Kundennutzen liefern soll (Gilb, 2008 S. 7).

Bedoll weist darauf hin, dass auch ein Bedarf des Kunden nach Zwischenergebnissen bestehen muss. In vielen Fällen wird der Kunde sich nicht mit unfertigen Produkten beschäftigen wollen, auch wenn diese einen Teilnutzen liefern können (Bedoll, 2003 S. 30).

Ein großes Problem für eine inkrementelle Lieferung kann die Beschaffenheit des Systems mit sich bringen. 2% des Systems können oft total nutzlos sein (z.B. 2% eines Fahrzeuges) weil sich die Funktion des Systems erst durch das Zusammenwirken aller Komponenten ergibt. Hier gibt es z.B. die Möglichkeit, neue Entwicklungen in einem alten, vorhandenen System zu implementieren. Außerdem ist bei inkrementeller Lieferung nicht unbedingt eine materielle Zerstückelung des Systems gemeint. Vielmehr geht es darum, den Nutzen inkrementell zu liefern. Und dieser kann auch entstehen, wenn das System noch nicht als Ganzes materiell vorhanden ist (Gilb, 2008 S. 8f).

**Empirische Erhebungen:** Die Umsetzbarkeit von kleinen Entwicklungsschritten, die nutzbar sein sollen, wurde auch von den Interviewpartnern als großes Problem gesehen. Es wurde aber darauf hingewiesen, dass der Nutzen nicht beim Endanwender entstehen muss, sondern auch in den Informationen bestehen kann, die der Kunde für weitere eigene Entwicklungstätigkeiten benötigt. Z.B. können Simulationsmodelle von Fahrzeugkomponenten als Zwischenergebnisse nicht vom zukünftigen Fahrer verwendet werden, wohl aber vom Entwickler des Gesamtfahrzeugs.

**Bewertung:** In ihrer Endanwendung nutzbare Zwischenergebnisse scheinen am besten zur Generierung der Informationen geeignet zu sein, die für die Realisierung der genannten Vorteile der schrittweisen Entwicklung notwendig sind. Man kann aber auch alleine den Informationszuwachs als Nutzen sehen. Damit lässt sich die Richtung der weiteren Entwicklung auch steuern, wenn die AEZ Forderung nach vom Anwender nutzbaren Zwischenergebnissen nicht erfüllt werden kann. Hier besteht also ein Potential, den AEZ auch in einer abgeschwächten Version einzusetzen, wenn es die Umstände nicht anders erlauben. Auch scheint die Entwicklung in kleinen Schritten die Agilität steigern zu können, wenn die Zwischenergebnisse nicht vom Endanwender genutzt werden können.

***Größe der Schritte/Zyklusdauer***

**Erklärung:** In der agilen Softwareentwicklung wird oft von 2-4 Wochen als Zyklusdauer gesprochen (Beck, et al., 2005 S. 46), (Schwaber, 2004 S. 7f). Aber auch eine Woche oder 4 Monate für einen Extremfall, bei dem eine sehr komplexe Funktion entwickelt wurde, werden genannt (Bedoll, 2003 S. 32f).

In der „Evolutionary Project Management" Methode wird vorgeschlagen, die Anforderungen in Inkrementen zu entwickeln, die jeweils ca. 2% des Budgets ausmachen sollen. Die Aufteilung der Anforderungen auf die Inkremente soll nach dem Gesichtspunkt erfolgen, dass nach jedem Inkrement zumindest einem Stakeholder die Erfüllung einer Anforderung aus dem Anforderungskatalog präsentiert werden kann. Die Anforderungen sollen dabei nach Nutzen und Wert priorisiert abgearbeitet werden (Gilb, 2005 S. 1).

Durch die Begrenzung auf 2% des Gesamtbudgets als Richtwert, wird auch das Risiko auf diesen Wert begrenzt. Schlägt ein Entwicklungsschritt fehl, verliert man eben nur 2% und nicht ein ganzes Projektbudget (Gilb, 2008 S. 4f). Die 2% Regel ist nicht nur auf Basis des Budgets anwendbar, sondern kann auch auf Basis der geplanten Projektdauer angewandt werden. Damit wird verhindert, dass eine Fehlentwicklung sich über einen zu langen Zeitraum hinweg zieht. Wie beim Budget auch, sind 2% ein Richtwert, der je nach Erfordernissen verändert werden kann (Gilb, 2008 S. 5).

**Empirische Erhebungen:** In der Praxis wurden bei den Systementwicklern keine Entwicklungsschritte gefunden, die in solch kurzen Zeitdauern auch nutzbare Zwischenergebnisse generierten. Einteilungen erfolgten normalerweise nach Phasen, die sich über mehrere Monate hinweg zogen. Oder nach den Komponenten des Systems, die aber wiederum nicht nacheinander schrittweise entwickelt wurden.

**Bewertung:** Die gefundenen Schrittgrößen bzw. Zyklusdauern scheinen kaum brauchbar zur Realisierung der genannten Vorteile der schrittweisen Entwicklung. Die für die Softwareentwicklung genannten Dauern und Größen erscheinen in der Systementwicklung wiederum schwer realisierbar. Es kann hier aber keine allgemeine Aussage gemacht werden, welche Vorteile der Entwicklung in kleinen Schritten, sich mit welchen Dauern/Größen verwirklichen lassen bzw. wie weit sich die Agilität damit steigern lässt.

Wie stark sich die Entwicklungsschritte verkleinern lassen, ist abhängig von Produkt und Projekt. Es ist durchaus denkbar, dass in einigen Fällen der gewünschte Grad an Agilität erreicht werden kann, auch wenn die Entwicklungsschritte deutlich größer als in der SW-Entwicklung angesetzt werden.

***Zyklische Entwicklung***

**Erklärung:** Dieses Merkmal beschäftigt sich mit der Frage, ob in den einzelnen Entwicklungsschritten jeweils die gleiche Vorgehensweise angewendet wird und welche Tätigkeiten sie beinhaltet. In den agilen SW-Entwicklungsmethoden sind einheitliche Zyklen vorgesehen, die immer alle Phasen der Entwicklung bis zur Lieferung an den Kunden beinhalten (Schwaber, 2004 S. 5f) (Cockburn, 2005 S. 163).

**Empirische Erhebungen:** In der Praxis wurden zwar Zyklen gefunden, die sämtliche Aktivitäten enthielten, z.B. Musterstände von Steuergeräten oder Versionen von Prototypen, aber keine in den beschriebenen Zeitdauern. Kürzere Zyklen wurden nur innerhalb der Planungsphasen gefunden, in welchen aber keine Realisierungstätigkeiten gesetzt wurden.

**Bewertung:** Um nutzbare Zwischenergebnisse zu produzieren, müssen jeweils alle Phasen der Entwicklung in einem Zyklus vorhanden sein. Für die Entwicklung von Hardware ist dabei die Produktion, also die physikalische Herstellung in kurzen Zeitdauern das Problem. Bei längeren Zeitdauern ist wiederum fraglich, wie weit die Agilität damit gesteigert werden kann. Das Weglassen der Realisierungsphase kann auch zu einem Zyklus führen, mit dem eine schrittweise Entwicklung der HW möglich wird. Vor allem wenn man den Nutzen aus den generierten Informationen ziehen kann (z.B. Simulationsmodelle), wie vorher beschrieben. Es gibt also auch bei diesem Merkmal Ausprägungsformen, mit denen zwar nicht der AEZ verwirklicht, aber dennoch die Agilität gesteigert werden kann.

***Art der Unterteilung***

**Erklärung:** Die Unterteilung des Entwicklungsumfangs in kleine Schritte kann nach vielen Gesichtspunkten erfolgen. Möglichkeiten sind die Unterteilung in Phasen, die schon in der traditionellen SE Methodik erwähnt wird (2.1.5) oder die Einteilung in Software-Versionen oder Releases. Ebenso kann das System in seine Komponenten oder Einzelteile aufgeteilt werden. Diese Unterteilung würde nach der Definition von Cockburn auch der inkrementellen Entwicklung entsprechen (Cockburn, 2007):

- Inkrementelle Entwicklung ist eine Strategie zur Unterteilung des Systems.
- Iterative Entwicklung ist eine Strategie zur Überarbeitung von Teilen des Systems (Rework).

Eine andere Betrachtungsweise für inkrementelle und iterative Entwicklung ist (Kaindl, et al., 2010 S. 2):

- Inkrementelle Entwicklung: Erweiterung des Systems in kleinen Portionen im Zuge einer Iteration.
- Iterative Entwicklung: Wiederholungen im Prozess.

Der Begriff “Inkrement” wird von den Autoren der zweiten Definition auch im Sinne der, in einer Iteration durchgeführten, Änderungen am System verwendet bzw. im Sinne des Entwicklungsumfangs, der in einer Iteration geleistet wird (Kaindl, et al., 2010).

Dies entspricht auch dem vorherrschenden Verständnis dieser beiden Begriffe in der Softwareentwicklung, wo die Unterscheidung der beiden Vorgehensweisen an Bedeutung verloren hat.

Softwareentwicklungsmethoden, die eine zyklische Entwicklung in kleinen Schritten vorsehen, werden deshalb auch unter dem Begriff „Iterative and Incremental Development“ zusammengefasst (Larman, et al., 2003), wie bereits in 4.3.3 erwähnt wurde.

**Empirische Erhebungen:** In der empirischen Studie wurden alle Arten von Aufteilungen vorgefunden, bis auf die zuletzt beschriebene nach dem Verständnis des IID. Es war aber niemals die Agilität das entscheidende Ziel für die Aufteilung, sondern technische und organisatorische Gründe.

**Bewertung:** Wie schon erwähnt, sind die verwendeten Phaseneinteilungen meist zu lang, um die beschriebenen Vorteile einer Entwicklung in kleinen Schritten (nach AEZ Prinzipien) umsetzen zu können. Dasselbe gilt für die Einteilung in Software-Versionen oder Releases. Bei einer Einteilung in Systemkomponenten, hängt es davon ab, ob diese schrittweise nacheinander entwickelt werden. Nur dann können die gewonnenen Informationen eines Schritts für den nächsten verwendet werden. Dies ist aber oft nicht möglich, weil die einzelnen Komponenten in gegenseitiger Beziehung stehen und nur gemeinsam entwickelt werden können. Auch sind damit meist keine kurzen Entwicklungszyklen möglich. Außerdem können bereits hergestellte Hardwarekomponenten oft nur schwer angepasst werden, sollten spätere Entwicklungsschritte es notwendig machen.

Von den genannten Unterteilungsstrategien bleiben also nur die inkrementelle und die iterative Entwicklung für die Umsetzung des AEZ. Diese beiden Strategien erscheinen im Licht der praktischen Entwicklung eines komplexen Systems aber nur als theoretische Idealfälle. Die Einzelteile der Systeme aber auch die Tätigkeiten zu ihrer Entwicklung weisen gewöhnlich viele Abhängigkeiten auf, die in der Entwicklung berücksichtigt werden müssen. Es ist aber praktisch unmöglich im Vorhinein die Reihenfolge zu bestimmen, in der diese Interdependenzen bearbeitet werden müssen. Deshalb entstehen oft Vorgehensweisen, die nicht mehr als rein inkrementell oder iterativ bezeichnet werden können. Das erklärt auch, warum im IID kein Wert darauf gelegt wird, jeden Entwicklungsschritt entweder als Inkrement oder als Iteration zu identifizieren.

Für den AEZ ist eine genaue Definierung der Art der Aufteilung nach diesem Gesichtspunkt auch nicht wichtig. Entscheidend ist, dass die anderen Merkmale der Entwicklung in kleinen Schritten so ausgeprägt werden können, dass eine positive Beeinflussung der Agilität möglich wird. Eine iterative und inkrementelle Entwicklung (nicht nach den separaten Definitionen sondern im Sinne des IID) wurde deshalb für die Beschreibung der Art der Aufteilung im AEZ gewählt, weil darunter im Allgemeinen Vorgehensweisen verstanden werden, die eine stetige Weiterentwicklung und Verbesserung des Systems erkennen lassen, was hinsichtlich Agilität dem Sinn des AEZ entspricht.

Soll nicht der AEZ umgesetzt werden, sondern eine andere Form der Entwicklung in kleinen Schritten, ist bei der Aufteilung auf die gewünschten Ausprägungsformen der anderen Merkmale zu achten. Die Tatsache, dass HW-Komponenten auch physikalisch hergestellt werden müssen, stellt dabei ein besonderes Erschwernis dar. Je nach Art des Systems kann es sehr schwierig sein, eine Aufteilung zu finden, die z.B. eine zyklische Entwicklung (inkl. Realisierung also Produktion der Hardwarekomponenten) von nutzbaren Zwischenergebnissen ermöglicht. Aber nicht nur die Merkmale der Entwicklung in kleinen Schritten sind hier zu berücksichtigen, sondern auch die Art der Fertigung. Z.B. können bei einer Einzelfertigung die Zwischenschritte bereits Teile des späteren fertigen Systems darstellen. Bei einer Serienfertigung ist dies nicht möglich, die Zwischenschritte können hier nur Teile der

Entwicklungstätigkeit sein bzw. als Prototypen oder Simulationsmodelle das später in Serie gefertigt System repräsentieren.

Bei der Aufteilung des Entwicklungsumfangs ist also zu überprüfen, was mit der Entwicklung in kleinen Schritten bezweckt werden soll, wie diese dazu umgesetzt werden soll und welchen Einfluss die Art von Projekt und Produkt darauf hat.

***Expliziter Prozess***

**Erklärung:** Explizit vorgesehen ist in der traditionellen SE Methodik eine Einteilung in Phasen (2.1.5). Möglichkeiten für Wiederholungszyklen oder Rückschritte werden im Entwicklungsprozess auch definiert, sind aber nicht explizit vorgesehen und werden nur genutzt, wenn z.B. Ziele nicht erfüllt werden können, Varianten bis zum Auswahlzeitpunkt noch nicht fertig ausgearbeitet wurden oder die Lösung noch unbefriedigend ist (Haberfellner, et al., 2002 S. 96ff).

Von den agilen Softwareentwicklungsmethoden hingegen wird ein Vorgehen in kleinen Schritten explizit gefordert und im Prozess dargestellt (Beck, et al., 2005 S. 33, 62), (Schwaber, 2004 S. 5f), (Poppendieck, et al., 2009 S. 183ff), (Gilb, 2005 S. 187).

**Empirische Erhebungen:** In den empirischen Untersuchungen konnte kein Entwicklungsprozess gefunden werden, der explizit ein Vorgehen in kleinen Schritten forderte, das auch den bereits genannten anderen Prinzipien entspricht. Implizit wurde die Entwicklung in kleinen Schritten in zahlreichen Ausprägungsformen gefunden. Sie wurde aber zumeist aus organisatorischen oder technischen Gründen durchgeführt und nicht zur Steigerung der Agilität. Hinsichtlich Veränderungen wurde sie sogar eher als Unzulänglichkeit gesehen. Z.B. werden Iterationsschleifen nur als notwendig erachtet, wenn unsorgfältig gearbeitet wurde, die Abstimmung zwischen Mitarbeitern oder mit dem Kunden schlecht war oder Fehler passiert sind.

**Bewertung:** Um die oben genannten Ziele erreichen zu können, ist die Entwicklung in kleinen Schritten fast immer als expliziter Prozess auszuführen. Geschieht ein Iterieren nur nach unsorgfältigem Arbeiten, Fehlern oder schlechter Abstimmung, so kann man der schrittweisen Entwicklung nicht die ganze Agilität zuschreiben, die für die Korrektur notwendig war bzw. entspricht das auch nicht der Definition von Iterationen. In solchen Fällen muss man sich fragen, ob die Reaktion nicht eher als Improvisation einzustufen ist. Führt die Reaktion zum gewünschten Ergebnis, kann zwar behauptet werden, dass genügend Agilität vorhanden war. Das liegt aber möglicherweise daran, dass z.B. die Mitarbeiter ausreichend befähigt waren oder das Produkt genügend Flexibilität aufwies. Der Prozess hat die Agilität dann lediglich zugelassen aber nicht durch die Entwicklung in kleinen Schritten erzeugt oder unterstützt. Das entspricht definitiv nicht dem AEZ. Ob man in diesem Fall von einer agilen Praktik der „Entwicklung in kleinen Schritten“ sprechen will, oder von einer agil durchgeführten Improvisation ist eine Frage der Definition.

Ein expliziter Prozess in kleinen Schritten schließt natürlich auch weitere, implizite Iterationen nicht aus. Es soll nur darauf hingewiesen werden, dass die explizite Anwendung für den wesentlichen Nutzen des AEZ verantwortlich ist und je nach Definition auch für das Prinzip der Entwicklung in kleinen Schritten.

In der Diskussion um die explizite Prozessgestaltung könnte man sich auch noch fragen, was ein Entwicklungsprozess bewirken würde, der zwar in kleinen Schritten durchgeführt wird, aber nicht explizit

definiert ist. Da dieser Fall aber eine ad-hoc Entwicklung darstellt und keine Vorgehensmethodik (was die Definition von Systems Engineering ist) wird er hier auch nicht diskutiert. Für eine Bewertung dieser Art von Entwicklung zur Herstellung von SW siehe Kettunen, et al. (2005 S. 593ff).

Es soll hier auch noch erwähnt werden, dass es Prinzipien und Praktiken gibt, die sich mit dem Zulassen bzw. Ermöglichen von impliziten kleinen Entwicklungsschritten im Prozess (und natürlich auch von anderen agilen Praktiken) beschäftigen. Diese sind z.B. eine flexible Prozessgestaltung (5.4.13) oder Pufferzeiten (5.4.10).

***Vorausplanung der Unterteilung***

**Erklärung:** Wie schon mehrfach beschrieben, werden die Inhalte der Entwicklungsschritte in den agilen SW-Entwicklungsmethoden je nach Priorität am Anfang eines jeden Schritts bestimmt und nicht zu Projektbeginn vorausgeplant (Schwaber, 2004 S. 6), (Cockburn, 2005 S. 163ff), (Boehm, et al., 2005 S. 32).

**Empirische Erhebungen:** In den empirischen Erhebungen wurden fast nur Unterteilungen gefunden, bei denen versucht wurde, den Entwicklungsumfang zu Beginn aufzuteilen. Es wurde auch versucht, die Planung der einzelnen Umfänge bereits zu Projektbeginn detailliert durchzuführen.

**Bewertung:** Im AEZ wurde dieses Merkmal nicht beschrieben, da es sich aus den anderen ergibt. Hier soll aber festgestellt werden, dass eine Unterteilung des Entwicklungsumfangs im Vorhinein nur beschränkt agilitätssteigernd wirken kann. Die Informationsgewinnung kann zwar funktionieren, d.h. man hat gewonnene Informationen im nächsten Schritt zur Verfügung. Man nimmt sich aber die Möglichkeiten den Inhalt oder Umfang des nächsten Schritts dahingehend anzupassen. Das soll aber nicht gegen eine Grobplanung der einzelnen Schritte zu Projektbeginn sprechen. Diese sollen aber inhaltlich veränderbar bleiben und auch so behandelt werden.

***Periodizität***

**Erklärung:** In der agilen Softwareentwicklung werden durchwegs konstant bleibende Zeitdauern für die Zyklen empfohlen (Beck, et al., 2005 S. 46), (Schwaber, 2004 S. 7f), (Aoyama, 1998 S. 7).

Wenn für einen zusammenhängenden Teil der Software mehr als ein Zyklus gebraucht wird, dann kann dieser Teil über 2 Zyklen hinweg entwickelt werden (Aoyama, 1998 S. 7).

Einige Inkremente, wie z.B. Bestellungen von Zulieferern dauern vermutlich länger als normale Entwicklungszyklen. Auch diese können in eine inkrementelle Entwicklung eingebunden werden, indem man sie rechtzeitig initiiert und in der Wartezeit andere Entwicklungsschritte durchführt (Gilb, 2008 S. 8).

**Empirische Erhebungen:** In der Praxis wurden keine entsprechenden periodischen Zyklen in der Entwicklung von HW gefunden. Vom Expertenpanel wurde die Synchronisierung von SW Zyklen (die mit agilen Methoden hergestellt wird) mit den viel längeren Zyklusdauern von HW als aktuelles Problem genannt (das wird unter 5.4.5 diskutiert).

**Bewertung:** Hier stellt sich die Frage, ob Periodizität der Schritte für Agilität notwendig ist. Die beschriebenen Vorteile der häufigen Deadlines und der Messbarkeit der Arbeitsgeschwindigkeit sprechen dafür. Ebenso wie der Wunsch, einen zyklischen Prozess explizit darzustellen. Der Autor hält aber auch eine Bestimmung der Dauer je Zyklus zum Zyklusbeginn für möglich.

**Allgemeine Bewertung:** In den empirischen Erhebungen konnte keine Entwicklung gefunden werden, die dem AEZ entspricht. Es konnte aber auch keine Entwicklung gefunden werden, die nicht in irgendeinem Sinne in kleine Schritte aufgeteilt wurde. Meist geschah dies aber aus organisatorischen Gründen und nicht zur Förderung der Agilität. Iterationsschleifen wurden auch eher als unerwünscht betrachtet, und nur für notwendig gehalten, weil die Komplexität des Systems hoch ist, sich gewisse Dinge nicht einfach berechnen lassen und Vieles zwischen den Entwicklern abgestimmt werden muss.

Ohne Iterationen kann man ein komplexes Produkt aber nur entwickeln, wenn man die wesentlichen Designparameter im Vorhinein schon kennt. Eine Entwicklung ist aber auch dadurch definiert, dass das Wissen über das System erst aufgebaut werden muss und vor Entwicklungsbeginn nicht gegeben ist.

Agile Entwicklungsmethoden berücksichtigen diese Tatsache und versuchen aus der Notwendigkeit des Iterierens eine Tugend zu machen. Um die Vorteile daraus zu realisieren, muss man es anscheinend aber bewusst einsetzen. Die dazu notwendigen Ausprägungsformen der Merkmale der Entwicklung in kleinen Schritten wurden beschrieben. Die Ausprägungsformen, die hinsichtlich Agilität den größten Nutzen erwarten lassen, wurden zusammen mit anderen Prinzipien zum AEZ kombiniert. Bei einigen Merkmalen lassen aber auch andere, „schwächere“ Ausprägungsformen noch einen Agilitätsgewinn erwarten.

Im Unterschied zur SW müssen Hardwareteile auch materiell hergestellt werden. Deshalb kann die Beschaffenheit des Systems als größtes Hindernis für eine Entwicklung in kleinen Schritten im Systems Engineering betrachtet werden.

Die zahlreichen genannten Vorteile lassen erahnen, warum unter „agiler Entwicklung“ oft die Anwendung dieses Prinzips (in der Ausprägung des AEZ) verstanden wird. Bei richtiger Durchführung kann die Entwicklung in kleinen Schritten beinahe alle Teilfunktionen der Agilität fördern. Das macht sie für den ASE Ansatz auch ganz besonders interessant. In den Kapiteln 6 und 7 wird weiter analysiert, in welchem Kontext dieses Prinzip eingesetzt und wie es im ASE angewandt werden kann.

### 5.4.4 Kontinuierliche Integration

**Erklärung:** Dies ist eine gemeinsame Praktik der agilen Methoden (Turner, 2007 S. 13) und besagt, dass neu entwickelte Funktionen so häufig wie möglich in das, sich in Entwicklung befindende, System integriert werden sollen (Beck, et al., 2005 S. 6). Es gibt am Ende des Entwicklungsprozesses keine speziell ausgewiesene Phase in der die einzelnen Komponenten integriert werden, sondern jeder Entwickler soll seine Änderungen so schnell wie möglich (mindestens einmal am Tag) in das Gesamtsystem integrieren (Paulk, 2001 S. 3).

**Empirische Erhebungen:** In der Praxis konnte das auch teilweise gefunden werden, wobei es als Sache des Konfigurationsmanagements betrachtet wurde. Als wichtig wurde es erachtet, weil gewöhnlich Zusammenhänge zwischen den Komponenten bestehen und für die Entwicklung einer Komponente immer der neueste Stand der anderen Komponenten berücksichtigt werden soll.

**Bewertung:** Wie das Konfigurationsmanagement gestaltet werden soll, damit eine möglichst häufige Integration möglich ist, ist nicht Inhalt dieser Dissertation. Das Prinzip erscheint aber wertvoll hinsichtlich Agilität, da es dafür sorgt, dass über Module oder Komponenten hinweg, neue Informationen zum Entwicklungsstand so schnell wie möglich verfügbar werden. So können Fehler oder durchzuführende Änderungen schneller erkannt werden. Es kann auch die Effizienz steigern, weil die Gefahr kleiner wird,

dass eine Komponente hinsichtlich eines alten Entwicklungsstandes einer anderen Komponente entwickelt wird und das danach für den neuen Stand wiederholt werden muss.

### 5.4.5 Anchor Point Milestones

**Erklärung:** Eine kontinuierliche Integration ist nicht immer möglich, vor allem wegen der Unterschiede zwischen HW und SW. In Kapitel 5.4.3 wurde bereits angesprochen, dass die Entwicklung dieser beiden Komponenten unterschiedliche Zykluszeiten aufweisen kann. Es kann z.B. vorkommen, dass die Software nach einer agilen Methode in 2-Wochen Zyklen entwickelt wird, die Hardware aber in Phasen mit viel längeren Zykluszeiten. Die Integration von HW und SW bzw. die Synchronisation von HW- und SW-Entwicklung kann dann zu gemeinsamen Meilensteinen stattfinden. Boehm nennt diese „Anchor Point Milestones“ (Boehm, 1996). Damit können die oft zahlreichen, gleichzeitig auftretenden Aktivitäten (Simultaneous Engineering) synchronisiert, stabilisiert und auf Risiken überprüft werden (Boehm, et al., 2008 S. 3).

Diese Meilensteine sind auch gute Punkte zur Demonstration des Entwicklungsstandes beim Kunden. Im Unterschied zu Meilensteinen bei herkömmlicher Entwicklung soll der Kunde das gelieferte (Teil-) Produkt aber nicht zum allerersten Mal sehen, sondern im Rahmen der agilen Entwicklung durch die enge Zusammenarbeit schon Bescheid wissen, was er in etwa beim Meilenstein geliefert bekommt. Trotzdem sind Meilensteine auch bei agiler Entwicklung und vor allem bei großen Projekten sehr wichtig, um sicherzustellen, dass die Zusammenarbeit funktioniert (McMahon, 2006 S. 29).

Zur besseren Abstimmung der einzelnen Systemkomponenten und zur Erhöhung der Flexibilität empfiehlt Kaindl nicht auf Anchor Point Milestones zu warten, sondern die Architektur von HW und SW gemeinsam zu gestalten und in jedem Zyklus gemeinsam zu verbessern (Kaindl, 2005 S. 5ff).

**Empirische Erhebungen:** Im Expertenpanel wurde die Verwendung dieser Meilensteine zur Synchronisierung einer agilen SW-Entwicklung mit einer traditionell durchgeführten Entwicklung von HW diskutiert. Sie wird schlichtweg als notwendig erachtet, sollten unterschiedliche Zykluszeiten bestehen. Als praktisches Problem wurde auch das unterschiedliche Vokabular der HW- und SW-Entwickler genannt, wie schon unter 5.3.5 beschrieben.

**Bewertung:** Die Anchor Point Milestones sind ein Hilfsmittel zur Synchronisierung der unterschiedlichen Entwicklungsgeschwindigkeiten von Systemkomponenten und kein direktes Mittel zur Steigerung von Agilität. In einem ASE können sie als notwendig erachtet werden, wenn das System in kleinen Schritten entwickelt werden soll.

### 5.4.6 Test-Driven Development

**Erklärung:** Bei Test-Driven-Development wird immer zuerst ein Testfall geschrieben und erst dann wird mit der Entwicklung der Funktion begonnen, die den Test bestehen soll (Beck, et al., 2005 S. 6), (Malotaux, 2006 S. 2). Modul- oder Methodentests werden dabei vom Entwickler oder auch vom Kunden vor oder am Anfang eines Entwicklungszyklus entworfen, was kurze Iterationszyklen ermöglichen und forcieren soll (Boehm, et al., 2005 S. 32). Dazu können in der Softwareentwicklung automatisierte Tests verwendet werden (Kettunen, et al., 2005 S. 603). Test-Driven Development wird als gemeinsame Praktik der agilen SW-Entwicklungsmethoden gesehen (Turner, 2007 S. 13).

Eine Schwierigkeit dieser Praktik ist das Zusammenspiel zwischen Entwickler und Tester. Zur besseren Abstimmung schlägt McMahon deren örtliche Nähe vor (McMahon, 2006 S. 29).

In einem Systems Engineering mit iterativer Entwicklung erscheint auch das Test-Driven Development machbar. Als Hilfsmittel können dabei Simulationen oder Hardware-in-the-Loop Umgebungen dienen, in denen die noch nicht vorhandenen HW-Komponenten durch Simulationsmodelle ersetzt werden (Turner, 2007 S. 13).

**Empirische Erhebungen:** Im Expertenpanel wurde darauf hingewiesen, dass die Testkriterien geändert werden müssen, wenn sich Anforderungen ändern. Die Praktik kann also auch zu Mehraufwand führen, wenn Testfälle mehrfach ausgearbeitet werden müssen.

**Bewertung:** Dieses Prinzip kann im Rahmen einer iterativen Entwicklung, als Hilfsmittel angesehen werden, die Effizienz zu steigern. Es wird damit sichergestellt, dass genau so lange iterativ weiterentwickelt wird, bis die gestellten Anforderungen erfüllt sind. Eine direkte agile Wirkung im Sinne der Bewältigung von Veränderungen konnte nicht festgestellt werden.

### 5.4.7 Prototyping/Simulation

**Erklärung:** Wie schon bei einigen anderen Prinzipien und Praktiken, soll hier nicht auf technische Details eingegangen werden, sondern auf die Bedeutung dieser Praktiken für Agilität. Über Prototyping und Simulation wurde in dieser Arbeit bereits geschrieben. Es kann nämlich das Herstellen von nutzbaren Zwischenergebnissen bei der Entwicklung in kleinen Schritten (oder auch im AEZ) als Prototyping-Ansatz gesehen werden bzw. die Simulation als Ersatz der materiellen Prototypen. Nun sollen die beiden Praktiken unabhängig davon diskutiert werden.

Prototyping ist nicht nur eine gute Methode um die Erfüllung der Anforderungen zu überprüfen, sondern fördert auch die Kommunikation zwischen den Kunden/Anwendern und den Entwicklern. Oft wird dem Kunden nämlich erst beim Erproben eines Systems klar, was er eigentlich haben will und erst dann kann er brauchbares Feedback geben (Song, et al., 2005 S. 164), (Hansson, et al., 2006 S. 1300, 1303). Natürlich dienen Prototypen auch zur Bestimmung der Tauglichkeit der bisher durchgeführten Entwicklung (Gilb, 2005 S. 10).

Ein Prototyp liefert dem Kunden mehr Information, als ein auf Papier ausgearbeitete Design. Oft besteht aber die Schwierigkeit, ein geeignetes Prototyping- Werkzeug zu finden, welches die Möglichkeit der schnellen Herstellung mit der Nutzbarkeit des damit entwickelten Prototyps kombiniert (Bedoll, 2003 S. 31). Unter dem Namen „Rapid Prototyping“[6] werden einige Verfahren zusammengefasst, die hier ev. Abhilfe schaffen können.

Kaindl fasst die Verwendung von Prototypen zu zwei Hauptzwecken zusammen (Kaindl, 2005 S. 3):

a) Abklären von Anforderungen an die Funktionalität oder die Benutzer-Schnittstelle.
b) Ermittlung der Leistungsfähigkeit und/oder Machbarkeit.

Er unterscheidet bei Prototypen auch, ob diese bereits Teil des zu entwickelnden Systems sind („Evolutionäres Prototyping“) oder nur der Entwicklung dienen und danach entsorgt werden („Wegwerf-

[6] Eine Aufzählung ist unter http://de.wikipedia.org/wiki/Rapid_Prototyping zu finden.

Prototyping“). Ersteres ist speziell in der Softwareentwicklung in Verwendung. Bei Hardware sind aber die Anpassbarkeit der Prototypen und die Fertigungskosten ein Problem, weshalb im Normalfall Wegwerf-Prototypen verwendet werden, die sich in Größe, Aussehen, Material und Funktionalität vom Endprodukt unterscheiden (Kaindl, et al., 2010 S. 6f).

Der AEZ ist damit als spezielle Form des evolutionären Prototyping zu sehen, bzw. auch die Entwicklung in kleinen Schritten, wenn die entsprechenden Merkmale dahingehend ausgeprägt sind. Sind keine für den Anwender nutzbaren Zwischenergebnisse machbar, können bei einer Entwicklung in kleinen Schritten aber auch Wegwerf-Prototypen eine Agilitätssteigerung bewirken.

**Empirische Erhebungen:** In der S²QI Expertengruppe wurde die Simulation als Ersatz für das gewöhnlich teurere Prototyping hervorgehoben, da diese auch zu einem wesentlichen Informationsgewinn führen und damit als brauchbares Zwischenergebnis einer schrittweisen Entwicklung gesehen werden kann. Im Vergleich zu materiellen Prototypen kann die Simulation auch bei komplexen Systemen zu relativ geringen Kosten durchgeführt werden und hat auch einen weiteren Vorteil bei Variantenstudien, wo sie durch die Veränderung einzelner Parameter einen schnellen Vergleich zahlreicher Varianten ermöglicht. Sowohl Prototyping als auch Simulation sind in den betrachteten Industrien etabliert. Je nach Art des Systems werden aber nur sehr wenige Prototypen erstellt, aufgrund hoher Kosten, langer Herstellzeiten, schlechter Anpassbarkeit und nicht gegebener Wiederverwendbarkeit. Deshalb wird auch versucht, die angestrebte Gewinnung von Informationen ersatzweise durch Simulation zu erreichen.

**Bewertung:** Als inhärenter Teil des AEZ ist mit Prototyping der größte Gewinn an Agilität zu erwarten. Aber auch innerhalb traditioneller Entwicklung kann damit die Agilität gesteigert werden, durch Lernen und Gewinnung von Informationen zu Anforderungen und Systemverhalten. Ebenso wie bei den nutzbaren Zwischenergebnissen für AEZ und Entwicklung in kleinen Schritten ist die Art des Systems ein wesentlicher Faktor für die Machbarkeit, Art, Anpassbarkeit und wirtschaftlich sinnvolle Anzahl von Prototypen. Simulation ist als kostengünstigere Alternative zu sehen, wenn sich damit die gewünschten Informationen ebenso und in der erforderlichen Genauigkeit generieren lassen.

### 5.4.8 Timeboxing

**Erklärung:** Eine Timebox wird als spezielle Art von Meilenstein gesehen. Die Meilensteine, welche bei traditioneller Entwicklung verwendet werden, sind Termine, zu denen gewisse Ergebnisse vorliegen müssen und die verschoben werden, falls die Ergebnisse nicht erreicht wurden. Eine Timebox hingegen ist ein Zeitrahmen, der unbedingt eingehalten werden muss, auch wenn die gewünschten Ergebnisse nicht erfüllt wurden (Oesterreich, 2006 S. 61).

Bei herkömmlicher Zeitplanung werden normalerweise die Anforderungen zuerst definiert und dann wird dafür eine wahrscheinliche Zeitdauer eingeplant. Sind am Ende nicht alle Anforderungen erfüllt, wird die Frist meist verlängert (man könnte auch von „Featureboxing“ sprechen). Beim Timeboxing wird zuerst eine Zeitdauer fixiert und erst dann werden die Anforderungen bestimmt, die darin zu erfüllen sind. Sind am Ende der Zeitdauer nicht alle Anforderungen erfüllt, so endet diese Phase dennoch, eben mit weniger erfüllten Anforderungen. Durch richtige Priorisierung der Tätigkeiten kann Timeboxing helfen, das bestmögliche Ergebnis zu erhalten, wenn die Ressource Zeit kritisch ist (Malotaux, 2006 S. 4). Nachdem natürlich im Endeffekt alle Anforderungen zu erfüllen sind, kann Timeboxing seine Stärke vor allem bei einer Entwicklung in kleinen Schritten ausspielen. In einer agilen Entwicklung ist das frühe

Erhalten von Feedback wichtiger, als dass die Zwischenergebnisse schon alle definierten Anforderungen erfüllen. Kommt es zu Problemen, erfolgt keine Terminverschiebung, sondern der Umfang des zu erstellenden Entwicklungsschrittes wird angepasst. Dadurch können sich die Beteiligten zumindest auf die festgelegten Termine verlassen (Seibert, 2007 S. 44).

Diese Praktik wurde als Teil vieler agiler Softwareentwicklungsmethoden auch in den AEZ übernommen. Es kann auch nur so sichergestellt werden, dass die gleichbleibende Zyklusdauer sichergestellt wird.

Bei Scrum wird Timeboxing aber nicht nur für die Sprints, sondern auch für Meetings verwendet. Es dürfen die dafür festgelegten Zeitdauern also nicht überschritten werden (Schwaber, 2004 S. 135ff). Gleich wie in den Sprints soll das die Mitarbeiter dazu bringen, sich auf das Wesentliche zu konzentrieren, möglichst einfache Lösung anzustreben und die Meetings effizient zu gestalten (Schwaber, 2004 S. 37,52).

**Bewertung:** In der Systementwicklungspraxis konnte das Timeboxing nicht gefunden werden. Es kann als Hilfspraktik für die Optimierung der Entwicklung in kleinen Schritten angesehen werden. Es sorgt dafür, dass der Informationsaustausch (Kundenfeedback) in der gewählten Zykluszeit auch stattfindet. Im Zusammenhang mit Veränderungen steigert es die Agilität nicht direkt, kann aber die Effizienz erhöhen. Das scheint auch für Besprechungen interessant zu sein. Als Ersatz für die Terminhandhabung bei Meilensteinen traditioneller Entwicklung scheint es nicht geeignet zu sein. Der Sinn dieser ist das Erfüllen bestimmter Qualitätsanforderungen zu Phasenende und nicht mit der Entwicklung in kleinen Schritten zu vergleichen.

Am Rande sei erwähnt, dass auch ein kostenabhängiges „Boxing“ durchgeführt werden kann. Man legt z.B. 2% des Gesamtbudgets als Budget für den nächsten Zyklus fest und versucht damit so viele Anforderungen wie möglich zu erfüllen (Gilb, 2005 S. 4f).

### 5.4.9 Laufende Planung

**Erklärung:** Bei herkömmlicher Entwicklung wird davon ausgegangen, dass man das gesamte Entwicklungsprojekt zu Beginn planen kann und soll. Bei agiler Entwicklung, wo Veränderungen als unvermeidbar angesehen werden, ist eine laufende Planung zur Berücksichtigung neuer Informationen vorgesehen. Nach Kettunen wird z.B. ein Planungshorizont von 2-3 Wochen empfohlen (Kettunen, et al., 2005 S. 593).

Auch Gilb empfiehlt, nicht zu weit zu planen. Langfristige Ziele müssen natürlich erreicht werden, aber kurzfristige Fakten müssen sofort zu Reaktionen führen (Gilb, 2005 S. 10).

In einer Fallstudie von Hansson wird gezeigt, dass es sehr gut funktioniert, wenn es nur eine grobe Gesamtplanung gibt und die detaillierte Planung kurzfristig zwischen Entwicklungsteam und Kunden vereinbart wird (Hansson, et al., 2006 S. 1308). Deshalb wird als Strategie empfohlen, einen detaillierten Plan für die nächsten Wochen zu haben, einen groben Plan für die nächsten Monate und vage Ideen für die Zukunft (Hansson, et al., 2006 S. 1299).

Die ständig weitergeführte Detailplanung bedeutet aber nicht, dass der Planungsaufwand zu Beginn eines agilen Projektes viel kleiner als bei einem herkömmlichen Projekt sein muss. Es gibt auch bei agilen Projekten sehr viele Dinge die trotzdem zu Beginn des Projektes durchgeführt werden müssen wie z.B.:

die Terminplanung für die zyklischen Entwicklungsschritte, die Priorisierung der Aufgaben oder ein ausführliches Risk Assessment (Turner, 2002 S. 140).

Im Extreme Programming werden 2 Planungszyklen vorgeschlagen, auf wöchentlicher und quartalsweiser Basis mit folgenden Inhalten (Beck, et al., 2005 S. 46ff):

- Wöchentlicher Zyklus:
  - Projektfortschritt verfolgen, speziell den Fortschritt der letzten Woche.
  - Mit den Kunden die Anforderungen bestimmen, die in der kommenden Woche zu entwickeln sind.
  - Die Anforderungen in Entwicklungsaufgaben zerlegen, mit den Entwicklern die Zuteilung dieser Aufgaben bestimmen und den Zeitaufwand abschätzen.
- Quartalweiser Zyklus:
  - Engpässe an Ressourcen identifizieren, speziell jene, die nicht vom Team beeinflusst werden können.
  - Ausbesserungsarbeiten initiieren.
  - Grobe Themen auswählen und zugehörige Anforderungen bestimmen, die in diesem Quartal entwickelt werden sollen.
  - Einen Überblick verschaffen, wie das Projekt zur Unternehmung passt.

**Empirische Erhebungen:** In der Praxis wurde der Wunsch, alles zu Projektbeginn planen zu können, weit verbreitet gefunden, vor allem von Seiten des Managements. Das soll auch erreicht werden durch den expliziten Einsatz von Systems Engineering. Laufende Neuplanung findet natürlich trotzdem statt, weil auf Veränderungen reagiert werden muss. Die Ursache der Änderung von Plänen wird in einer unsorgfältigen oder unrealistischen Planung gesehen oder es wird den Entwicklern unterstellt, nicht schnell genug oder zu fehlerhaft gearbeitet zu haben. Auch beim Kunden oder den Lieferanten wird oft die Schuld gesucht. Fast immer verschlechtern Planänderungen die Stimmung bei Projektbeteiligten.

**Bewertung:** Es scheint, dass in der Planung in der Praxis oft zu perfektionistisch gearbeitet und keine Rücksicht auf Veränderungen genommen wird, die im Laufe der Entwicklung eines komplexen Systems auftreten können. Wenn man akzeptiert, dass Änderungen unvermeidlich oder zumindest wahrscheinlich sind, bieten die agilen Methoden eine Reihe von Ideen zur besseren Planung. Diese empfehlen nicht weniger zu planen. Sie sehen den Planungsaufwand nur an sinnvolleren Zeitpunkten vor. Änderungen können so leichter behandelt werden und es wird effizienter geplant. Im AEZ bzw. bei einer Entwicklung in kleinen Schritten ist die zyklische Detailplanung inhärent.

### 5.4.10 Pufferzeiten

**Erklärung:** Nachdem Effizienz auch eine Grundvoraussetzung für Agilität ist, stellt sich die Frage, wie weit diese Zeitreserven die Effizienz der Entwicklung beeinträchtigen dürfen. Auf jeden Fall können sie die Flexibilität erhöhen, um beliebige Entwicklungstätigkeiten durchführen zu können, die aufgrund von Veränderungen oder Fehlern notwendig wurden und zusätzliche Entwicklungszeit benötigen. Die Idee von Extreme Programming, in jeden Zyklus auch immer ein paar niedrig priorisierte Anforderungen einzuplanen (Beck, et al., 2005 S. 48), erscheint sehr interessant, weil hier zumindest irgendein Nutzen fabriziert wird, wenn die Pufferzeit nicht benötigt wird. Falls doch, werden nur weniger wichtige Anforderungen verschoben, welche dann in einem späteren Schritt nachgeholt werden können.

**Empirische Erhebungen:** In der Praxis liegt es im Ermessen des Projektleiters und an den Möglichkeiten, die der Kunde bietet, Pufferzeiten einzuplanen. Auf jeden Fall ist dies gängige Praxis, um die Flexibilität der Ressource Zeit bzw. der Mitarbeiter zu erhöhen. Pufferzeiten werden normalerweise nicht an die Mitarbeiter kommuniziert, da sonst die Gefahr besteht, dass diese erst das Ende der Pufferzeit als Deadline betrachten und langsamer entwickeln.

**Bewertung:** Betrachtet man die Planung von Tätigkeiten von Wissensarbeitern tiefgründiger, stellt sich überhaupt die Frage nach der Definition von Pufferzeiten. Nicht jeder Gedankengang kann geplant werden und es ist bei Ideen nicht absehbar, wann diese auftreten. Die Planungspraxis der Systementwickler basiert im Wesentlichen auf Erfahrungswerten. Womöglich sind dadurch schon implizit Pufferzeiten in die Planung von Entwicklungsprojekten gekommen, die das ständige Wahrnehmen neuer Informationen, iteratives Entwickeln, Berücksichtigen von Feedback und Reagieren auf Veränderungen erlauben und damit eine gewisse Grundagilität in jede Entwicklung bringen, ohne die eine jede Entwicklung zum Scheitern verurteilt wäre. Dieser Frage soll hier aber nicht weiter nachgegangen werden. Pufferzeiten werden als Möglichkeit der Flexibilitätssteigerung gesehen, die sich aber auch nachteilig auf die Effizienz auswirken kann.

### 5.4.11 Laufende Situationsanalyse

**Erklärung:** Bei traditioneller Entwicklung, in der das Umfeld als statisch angenommen wird, wird die Situationsanalyse meist zu Beginn angesetzt. Bei einer Entwicklung in dynamischen Umfeldern sollen aber auch potentielle Veränderungen während der Entwicklung berücksichtigt werden. Für existierende Prozessmodelle empfehlen Haberfellner und de Weck daher die Situationsanalyse nicht nur zu Beginn, sondern auch später, mit der Option Anforderungen noch verändern zu können, anzuwenden (Haberfellner, et al., 2005 S. 8).

Wissen über externe und interne Ereignisse, die nach einer Reaktion verlangen, ist eine Grundbedingung für Agilität. Dazu müssen die Veränderungen wahrgenommen und analysiert werden, um dann in den Entwicklungsprozess einfließen zu können. In der Unternehmung muss dafür ein Informations-Netzwerk vorhanden sein, das relevante Daten möglichst genau und rechtzeitig jenen Stellen zur Verfügung stellt, die daraus das notwendige Wissen für agiles Handeln generieren können (Dove, 2005 S. 5).

**Empirische Erhebungen:** In der Praxis wurden hierfür kaum explizite Maßnahmen gefunden. Wenn der Kunde nicht mit neuen Informationen an die Projektleitung herantritt, wird es den Mitarbeitern überlassen, solche Informationen aufzuschnappen und zu bewerten. Wenn Mitarbeiter neue Informationen für relevant halten, können sie diese im Rahmen der normalen Kommunikation (Regelmeetings, direkte Kommunikation mit Projektleiter …) ins Projekt einbringen.

**Bewertung:** Das Erkennen und Bewerten von Veränderungen ist einer der Hauptpunkte der Literatur zur „Agile Enterprise“ (siehe Kapitel 2.3). Im Agile Systems Engineering kann dafür die Situationsanalyse verwendet werden, nach denselben Kriterien wie im herkömmlichen SE (Haberfellner, et al., 2002 S. 109ff). Es wird aber zusätzlich empfohlen die Situationsanalyse während der Entwicklung häufiger oder laufend durchzuführen. Eine Entwicklung in kleinen Schritten oder der AEZ bieten eine gute Möglichkeit, dies explizit und regelmäßig zu tun. Zu berücksichtigen ist natürlich der Ressourcenaufwand für die häufigen Analysetätigkeiten.

### 5.4.12 Tägliche/häufige Meetings

**Erklärung:** Bei der agilen Softwareentwicklungsmethode Scrum wird für die einzelnen Entwicklungsschritte ein 4-Wochen-Zyklus vorgeschlagen. Zur laufenden Koordination finden aber täglich kurze Besprechungen zum Arbeitsfortschritt und zu aufgetretenen Problemen statt (Schwaber, 2004 S. 135). Durch den Vergleich des Arbeitsfortschritts soll abgeschätzt werden, ob die Aufgaben des Entwicklungsschrittes bis zum Ende der 4 Wochen abgearbeitet werden können. Erscheint dies nicht möglich, können niedrig priorisierte Aufgaben gestrichen werden. Ist es vorhersehbar, dass alle Aufgaben vor dem Ende der 4 Wochen erledigt sind, können neue Aufgaben hinzugefügt werden (Seibert, 2007 S. 42).

Bei Extreme Programming wird nach einer wöchentlichen Planung verlangt (Beck, et al., 2005 S. 6). In Fallstudien zur Softwareentwicklung wird von wöchentlichen aber auch von täglichen Meetings berichtet (Hansson, et al., 2006 S. 1300, 1302, 1304).

In den Meetings soll immer nur über zukünftige Arbeit gesprochen und diese priorisiert werden. Die Meetings sind nicht dazu da, um nach Entschuldigungen für nicht oder schlecht erledigte alte Aufgaben zu suchen (Malotaux, 2006 S. 6).

**Empirische Erhebungen:** Bei den empirischen Erhebungen konnten regelmäßige Meetings im kürzesten Abstand von einer Woche gefunden werden. Ein weiterer Informationsaustausch fand natürlich auch häufiger statt, aber ungeplant.

**Bewertung:** Diese täglichen oder häufig geplanten Meetings können als im Prozess implementierte Verbesserungsmaßnahme für die Kommunikation, zum Wissensaustausch und zur Fortschrittskontrolle gesehen werden. Wichtig ist dabei, dass sie nur von kurzer Dauer sind (Ressourcenaufwand) und die eigentliche Arbeitszeit möglichst wenig beeinträchtigen.

### 5.4.13 Flexible Prozessgestaltung

**Allgemeine Erklärung:** Um auf Veränderungen reagieren zu können, die eine Änderung der Vorgehensweise verlangen, darf der Entwicklungsprozess nicht zu starr gestaltet sein. Nachdem Prozesse nur gedankliche Konstrukte sind, sind sie prinzipiell leicht änderbar. Bei den folgenden Prinzipien geht es deshalb speziell darum, genau definierte, etablierte und teilweise auch standardisierte Prozesse leicht änderbar zu machen, sollten Umstände im Projekt es erfordern.

***Lean***

Eine in der agilen Softwareentwicklung aufgekommene Forderung dazu ist es, den Prozess lean zu gestalten, nach den Prinzipien des Toyota Produktionssystems (Womack, et al., 1991) welche für die Softwareentwicklung im Lean Software Development adaptiert wurden (Poppendieck, et al., 2009).

Eine leichte Veränderlichkeit des Prozesses soll dadurch hergestellt werden, dass der Prozess nur Tätigkeiten beinhaltet, die Nutzen stiften und damit möglichst einfach gestaltet ist. Aktivitäten, die nicht unbedingt benötigt werden und kaum Nutzen stiften, sollen aus dem Entwicklungsprozess entfernt werden (Turner, 2007 S. 13). Ein wesentlicher Punkt davon betrifft die Dokumentation, die etwas ausführlicher in Kapitel 5.4.14 betrachtet werden soll.

***Modularität***

Das Prinzip der Modularität (siehe auch 5.5.2) kann nicht nur auf Produkte, sondern auch auf den Prozess angewandt werden, um diesen flexibler zu gestalten. Ein konkreter Anwendungsfall wurde in der S²QI Gruppe gefunden, wo in einer Firma die Prozesse so modular gestaltet waren, dass jeweils zu Projektbeginn die für das Projekt benötigten Module ausgewählt werden konnten. Dabei wurden die Anzahl der Quality Gates, die Reporting Prozesse, die Anzahl der Entwicklungszyklen oder auch die Teamkonfiguration festgelegt (Stelzmann, et al., 2010 S. 6).

***Prozessskelett***

Unter dem Begriff „Prozessskelett“ wird ein Prinzip vorgeschlagen, bei dem nur die wichtigsten Tasks auf höherer Ebene vorgegeben werden. Darunter kann das Team den Prozess selbst gestalten (Kettunen, et al., 2005 S. 587). Aoyama schlägt die Unterteilung des Entwicklungsprozesses in Hauptprozesse vor, die wiederum in Teilprozesse unterteilt werden sollen. Jeder Teilprozess soll dabei von einem Entwickler innerhalb einer Woche abgearbeitet werden können, um eine Systemeinheit entwickeln zu können (Aoyama, 1998 S. 7).

***Vor- bzw. Rückgriffe (Wiederholungszyklen)***

Eine Prozessgestaltung, die gedankliche Vor- bzw. Rückgriffe leicht ermöglicht oder unterstützt, kann als flexibel betrachtet werden. Vorgriffe sollen es ermöglichen, zielgerichteter zu arbeiten. Wiederholungszyklen sind notwendig wenn z.B. Ziele nicht erfüllt werden können, Varianten bis zum Auswahlzeitpunkt noch nicht fertig ausgearbeitet wurden oder die Lösung noch nicht zufriedenstellend ist (Haberfellner, et al., 2002 S. 96ff).

**Empirische Erhebungen:** In den Interviews wurde nicht die Agilität als Grund für eine flexible Gestaltung der Entwicklungsprozesse genannt. Vielmehr war es der Wunsch, kleinere Projekte rationeller durchführen zu können, bei denen es nicht notwendig ist, alle Prozessanforderungen von großen Projekten umzusetzen. Diese Anforderungen entstanden durch Standardisierungen der Entwicklungsprozesse, die von der Geschäftsführung zum besseren Management oder von den Kunden verlangt wurden. Auch wegen zu erlangender Zertifizierungen (z.B. nach der ISO 9000, CMMI oder SPICE), die eine erfolgreiche Entwicklung sicherstellen, vorhersehbar und die Erfüllung der Anforderungen an das Produkt nachverfolgbar machen sollen. Speziell bei sicherheitskritischen Produkten ist dies von hoher Wichtigkeit. Diese Prozessanforderungen sind damit aber nicht nur ein Effizienzproblem bei kleinen Projekten, sondern wirken sich auch auf die Agilität aus. Ein Hauptproblem wird im Bereich der Prozessdokumentation gesehen, dieses Thema wird in Kapitel 5.4.14 separat besprochen. Strenge Prozessregeln wurden auch als Hindernis für die Kreativität der Mitarbeiter gesehen. Als Lösungsmöglichkeit hinsichtlich der Prozessdefinition wurden Anpassungsmöglichkeiten im Kick-Off Meeting genannt, in dem standardisierte Prozesse in einigen Punkten verändert werden können. Dies geschieht nach Kriterien wie z.B. Teamgröße, Kritikalität, der Möglichkeit, Kundenfeedback während der Entwicklung zu bekommen, erwarteter Stabilität der Anforderungen, etc.

**Allgemeine Bewertung:** Bei der Flexibilität des Entwicklungsprozesses geht es also primär um die Frage, wie diese noch aufrecht erhalten werden kann, trotz der zahlreichen Anforderungen, die eine komplexe Systementwicklung an den Prozess stellt. Die Wichtigkeit dieses Prinzips liegt aber nicht nur in der Änderbarkeit der Vorgehensweise, sondern auch bei den Mitarbeitern. Ein Prozess, der die

Vorgehensweise zu streng regelt, kann die agile Handlungsfähigkeit der Entwickler beschränken. Vor allem bei einer Entwicklung sicherheitskritischer Produkte, die dennoch nach Agilität verlangt, kann es zu einem Zielkonflikt kommen. Allgemeine Prinzipien die Abhilfe schaffen können, wurden vorgestellt. Im Einzelfall (je nach Projekt, Produkt, agiler Methode und Prozesszertifizierung) muss aber eine genauere Analyse durchgeführt werden. Beispiele hierzu finden sich in (Paulk, 2001), (Turner, et al., 2002), (Fritzsche, et al., 2007) und (Kähkönen, et al., 2004).

### 5.4.14 Beschränkung auf notwendige Dokumentation

**Erklärung:** Dieses Prinzip sollte automatisch umgesetzt werden, wenn der Prozess lean gestaltet wird. Es kann aber auch unabhängig davon versucht werden, den Dokumentationsaufwand gering zu halten.

Wie vorher beschrieben, sind Prozessanforderungen oft für einen großen Dokumentationsaufwand verantwortlich. Bei traditioneller Entwicklung werden Dokumente erstellt, um den Projektfortschritt darzustellen, weil dies oft nicht durch Zwischenergebnisse getan werden kann. Bei agiler Entwicklung ist der Projektfortschritt wegen der Entwicklung in kleinen, zyklischen Schritten deutlicher sichtbar. Dokumente müssen deshalb nur dann erstellt werden, wenn sie wirklich bedeutsam sind und dringend benötigt werden (Hansson, et al., 2006 S. 1298).

Der Arbeitsaufwand für eine normale Dokumentation kann durchaus beträchtlich sein. Sie ist unflexibel und es ist mühsam, sie ständig auf dem aktuellen Stand zu halten. Mithilfe automatisch generierter Dokumentation kann in der Softwareentwicklung Abhilfe geschaffen werden. Diese ist aber nur bedingt tauglich, weil sie normalerweise nur vom Programmierer selbst interpretiert werden kann (Karlström, et al., 2006 S. 216). Bei der Softwareentwicklung kann die Dokumentation auch schritthaltend mit der Programmierung innerhalb des Codefiles stattfinden, was in einem Fallbeispiel in einer Form getan wurde, die es allen Entwicklern ermöglichte, den Code zu verstehen und zu verändern (Hansson, et al., 2006 S. 1308). Dies ist auch notwendig für das Prinzip „Product Team Ownership“ (siehe Kapitel 5.4.15).

Cockburn empfiehlt die Dokumentation durch persönliche Kommunikation und Verwendung von Whiteboards so weit wie möglich zu ersetzen und damit die Kosten der Informationsweitergabe zu reduzieren (Cockburn, et al., 2001 S. 131). Durch direkte Zusammenarbeit und Kommunikation gelangen die Teammitglieder zu einem besseren Projektverständnis, als bei Weitergabe von Dokumentation erreicht werden kann (Highsmith, 2002 S. 5).

In seinen „Agile Documentation Guidelines“ weist Highsmith darauf hin, dass der Zweck nicht die Dokumentation an sich, sondern Wissenstransfer und Verständnis ist (Highsmith, 2010 S. 290).

Bedoll zeigt in einem Fallbeispiel, wie bei kleinen Entwicklerteams, die in direktem Kontakt mit ebenfalls kleinen Teams von Endnutzern standen, der Prozess vereinfacht und damit der Dokumentationsaufwand wesentlich reduziert werden konnte (Bedoll, 2003 S. 32).

**Empirische Erhebungen:** Hinsichtlich Prozessanforderungen sind die empirischen Erhebungen aus Kapitel 5.4.13 auch hier relevant. Zusätzlich wurde die Einschulung von neuen Teammitgliedern als Grund für eine ausführliche Dokumentation genannt. Hier gilt es aber zu bedenken, ob nicht durch persönliche Weitergabe des Wissens und Diskussion die Integration ins Team besser funktioniert, als durch das Lesen der Dokumentation.

**Bewertung:** Es gibt also einige Möglichkeiten den Dokumentationsaufwand zu reduzieren. Wichtiger ist es, die Dokumentation in einem umfassenden System aus Wissen, Verstehen und Kommunikation zu betrachten. Damit ist es möglich, diese Elemente gemeinsam zu optimieren, da diese auch alle in Beziehung zur Agilität stehen. Das erklärt auch, wieso in der Literatur zur Agilität so oft auf die Dokumentation eingegangen wird. Beachtet werden müssen dabei aber Dokumentationsregeln für sicherheitskritische Produkte. Auch bei großen und örtlich verteilten Teams, wo nicht immer direkt kommuniziert werden kann, ist mehr Dokumentation notwendig.

### 5.4.15 Product Team Ownership

**Erklärung:** Die auch „Collective Ownership“ genannte Praktik (Paulk, 2001 S. 3) fordert in den agilen SW-Entwicklungsmethoden, dass jedes Mitglied des Teams auf sämtliche Module des Produkts Zugriff hat und diese verbessern darf.

Im XP wird diese Praktik „Shared Code“ genannt (Beck, et al., 2005 S. 66). Es wird aber auch darauf hingewiesen, dass sich das Team kollektiv für Projekt und Produkt verantwortlich fühlen muss. Sonst besteht die Gefahr, dass unzweckmäßige Änderungen durchgeführt werden oder Teile des Codes in einem, für die anderen Entwickler nicht weiter bearbeitbaren Zustand belassen werden (Beck, et al., 2005 S. 66).

Vorteile werden in einer Fallstudie gezeigt, wo Mitarbeiter krank waren oder aus der Firma ausschieden und die weitere Entwicklung gut funktionierte, weil die Entwickler auch Zugang zum Code der anderen hatten und diesen auch verändern durften (Hansson, et al., 2006 S. 1300, 1306).

In der Softwareentwicklung, wo die Teammitglieder normalerweise alle eine Ausbildung in derselben Disziplin haben, mag dies auch vertretbar sein.

**Bewertung:** In den empirischen Erhebungen wurde dieses Prinzip nicht gefunden. In der Systementwicklung, mit großen multidisziplinären Teams, muss die Anwendbarkeit dieses Prinzips hinterfragt werden. Für den Fall des Ausscheidens von Mitarbeitern sollte es reichen, wenn eine andere Person (Vertretung) die Verantwortung für das betroffene Modul übernehmen kann oder diese schon kollektiv hatte.

### 5.4.16 Emerging Architecture

**Erklärung:** Unter Architektur wird hier die prinzipielle Struktur eines Lösungskonzepts (Gesamtkonzept, Detailkonzept) verstanden, welche die Eigenschaften einer Lösung maßgeblich bestimmt und als Orientierungshilfe für die weitere Bearbeitung dienen soll. Das Prinzip der „Emerging Architecture“ besagt, dass die Architektur nicht in einer frühen Phase festgelegt werden soll, wie dies in traditionellen Entwicklungsmethoden üblich ist, sondern im Laufe der Entwicklung immer weiter entwickelt wird.

Dove meint, dass sich das Verständnis der Anforderungen während der Entwicklung oft so weit verändert, dass es dann nicht mehr kompatibel mit einer am Anfang festgelegten Produktarchitektur ist (Dove, et al., 2008 S. 5).

Bei traditioneller Entwicklung wird die Systemarchitektur im Anschluss an die Analyse der Kundenanforderungen entwickelt und bleibt für die weitere Entwicklung dann mehr oder weniger unverändert. Es wird davon ausgegangen, dass das grundlegende Konzept des Produkts bald festgelegt

werden kann. Laut Turkington wird bei agiler Entwicklung aber akzeptiert, dass sich auch die Architektur ständig weiterentwickelt, wenn sich im Laufe der Entwicklung neue Erkenntnisse ergeben (Turkington, 2007 S. 8).

Kruchten sieht zwar Projekte mit hohem Innovationsgrad und hoher Rate an Änderungen als Anwendungsgebiet agiler Methoden, meint aber, dass die Architektur zumindest im Wesentlichen von Anfang an bekannt sein sollte (Kruchten, 2010 S. 6.4, 6.6). Cockburn erachtet ein „Skelett" einer Architektur als notwendig, um die technische Arbeit und die Organisation zu lenken. Die anfängliche Architektur kann damit als Leitfaden gesehen werden, der sich aber ändern kann. Große Änderungen an der Architektur können aber auch bei Einsatz agiler Methoden hohe Kosten verursachen und ein Problem darstellen (Cockburn, 2005 S. 83ff).

**Empirische Erhebungen:** In der Praxis konnte die Anwendung dieses Prinzips für Systeme nicht gefunden werden. Änderungen, die auch die Architektur betreffen, kommen zwar auch in späteren Phasen vor, verursachen aber meist hohe Kosten und werden eher als Fehler angesehen.

**Bewertung:** Betrachtet man „Emerging Architecture" als absichtliche Nicht-Festlegung einer Architektur zu Beginn, kann man die Flexibilität eventuell steigern, riskiert aber auch eine chaotische und ineffiziente Entwicklung. Sinnvoller erscheint das vorläufige Festlegen der Architektur, unter Berücksichtigung der Anpassbarkeit. Mögliche Prinzipien dafür werden in 5.5.1 und 5.5.2 beschrieben. Man kann dann aber nicht mehr von einem beabsichtigten Prinzip einer „Emerging Architecture" sprechen. Dieses erscheint schon in der Softwareentwicklung als zweifelhaft und für die Systementwicklung als kaum realisierbar.

### 5.4.17 Entwicklung von Varianten

**Erklärung:** Dieses Prinzip wird im traditionellen Systems Engineering empfohlen und besagt, dass man sich zuerst einen umfassenden Überblick über mögliche Lösungsvarianten beschaffen soll, bevor man sich für eine entscheidet und diese dann detaillierter ausarbeitet (Haberfellner, et al., 2002 S. 33ff).

Es wird eine bessere Beurteilung des Problems ermöglicht, wie auch der gewählten Lösung. Das Problem wird aus der Sicht verschiedener Lösungen mit Blick auf deren Eignung betrachtet, wodurch das Problem besser verständlich wird. Und die Lösungsvarianten werden vergleichend beurteilt, was deren Charakteristik deutlicher macht (Haberfellner, et al., 2002 S. 33ff).

Im Agile Systems Engineering kann das Variantenprinzip mehreren Zwecken dienen. Einerseits fördert es die Gewinnung von Informationen im Bereich der möglichen Lösungen und bietet auch die Möglichkeit, den tatsächlichen Kundenanforderungen näher zu kommen, indem man den Kunden in die Variantenauswahl einbindet. Andererseits steht das über die nicht ausgewählten Varianten bereits gewonnene Wissen zur Verfügung, falls die ausgewählte Variante bei einer Veränderung verworfen werden muss. Damit kann man schneller auf eine Ersatzvariante zugreifen. Gilb sieht die Varianten auch als Backup-Lösungen und damit das Variantenprinzip als Möglichkeit zur Risiko-Reduzierung (Gilb, 2005 S. 191f).

Es ist auch möglich, mehrere Varianten parallel zu entwickeln (bis hin zur Erstellung von Prototypen für jede Variante), was die Flexibilität hinsichtlich der Lösungsauswahl erhöht. Dies würde aber auch eine enorme Ressourcenbelastung darstellen. Eine Möglichkeit zur Offenhaltung mehrerer Lösungsvarianten bietet das Set-Based Design, das in Kapitel 5.4.18 näher beschrieben wird.

**Empirische Erhebungen:** In der Praxis wird das Variantenprinzip oft überhaupt nicht beachtet. Vielfach bleibt man bei der ersten gefundenen und als machbar erachteten Lösung. Falls diese sich später als nur bedingt tauglich herausstellt, wird versucht, eine taugliche Lösung mit Hilfe von Workarounds herzustellen. Begründet wird dies mit den nicht vorhandenen Ressourcen zur Ausarbeitung mehrerer Varianten.

**Bewertung:** Laut Haberfellner et al. (2002 S. 36) stellt das aber den falschen Ansatz dar, da der Aufwand für Entwurf und Auswahl der besten Lösung geringer sein sollte, als für die Workarounds bei einer weniger guten Variante. Mit den genannten Vorteilen, stellt das Prinzip eine interessant Möglichkeit zur Informationsgewinnung und Steigerung der Flexibilität dar. Der Ressourcenaufwand ist aber ebenso zu berücksichtigen.

### 5.4.18 Set-Based Design

**Erklärung:** In dynamischen Umgebungen, mit sich ständig ändernden Märkten und Kundenwünschen, besteht die Forderung nach der Möglichkeit, das Produkt möglichst spät noch verändern zu können. Die Zeit zwischen der Festlegung des endgültigen Designs und der Markteinführung des Produkts soll möglichst kurz sein, um die Kundenanforderungen möglichst aktuell befriedigen zu können (Fricke, et al., 2005 S. 343).

Beim Set Based Design wird so lange mit einer Menge möglicher Lösungsvarianten gearbeitet, bis eine zu fällende Entscheidung die Anzahl der möglichen Varianten reduziert. Die Entscheidung kann dann fallen, wenn genug Informationen zur Verfügung stehen, um den Nutzen der einzelnen Varianten zu bestimmen. Sie kann aber auch durch Fristen (z.B. Beschaffung von Anlageteilen) oder durch einen Mangel an Ressourcen, um mehrere Varianten weiterzuentwickeln, ausgelöst werden. Das Lösungskonzept ergibt sich dann aus der sukzessiven Einengung des Lösungsspielraumes. Dadurch können die Entwickler mit dem Bewusstsein arbeiten, dass man die letztendlich gültige Variante zu Projektbeginn noch nicht gefunden haben muss. Außerdem wird dadurch ein größerer Lösungsraum ergründet, der spätere Änderungen erleichtert (Haberfellner, et al., 2005 S. 9).

Das Set-Based Design ist keine genau definierte Vorgehensweise, sondern bietet ein großes Spektrum an Ausprägungsformen, die in verschiedenen Anwendungsbereichen eingesetzt werden können. Ein sehr bekannter Fall ist die Vorgehensweise des japanischen Automobilproduzenten Toyota, die auch mit „Set-Based Concurrent Engineering“ bezeichnet wird (Ward, et al., 1995) und in Abb. 23 dargestellt ist.

Die Schritte dieser Vorgehensweise sind folgendermaßen zu interpretieren (Bernstein, 1998 S. 47ff):

1. Es wird ein Bereich (“Design Space”) mit denkbaren Lösungsvarianten erstellt, diese werden aber nicht diskret dargestellt, sondern bilden eine ungenau abgegrenzte Menge an möglichen Lösungen. Die abgebildeten “Specialities” sind Teilbereiche der Entwicklung (wie z.B. die in Abb. 24 gezeigten Fachbereiche, Entwicklungsabteilungen oder Systemkomponenten), in welchen begonnen wird, den Lösungsbereich zu analysieren.
2. Die Entwicklungsabteilungen untersuchen innerhalb des Design Space, welche Lösungsbereiche aus ihrer Sicht machbar erscheinen. Gemeinsam versuchen sie einen überlappenden Bereich zu finden.

3. Rund um den gefundenen Überlappungsbereich wird versucht, Lösungen zu finden, die allen Fachbereichen der Entwicklung machbar erscheinen. Es wird nach wie vor immer von Lösungsbereichen und nicht von einzelnen, diskreten Lösungen gesprochen.
4. Nicht machbare oder unvorteilhafte Optionen werden sukzessive ausgeschieden, der Lösungsbereich wird verkleinert, während versucht wird, den Überlappungsbereich zu vergrößern.
5. Der Detaillierungsgrad der verbleibenden Lösungen wird laufend erhöht, während weiter Varianten ausgeschieden werden, bis nur mehr eine Lösung übrig bleibt.

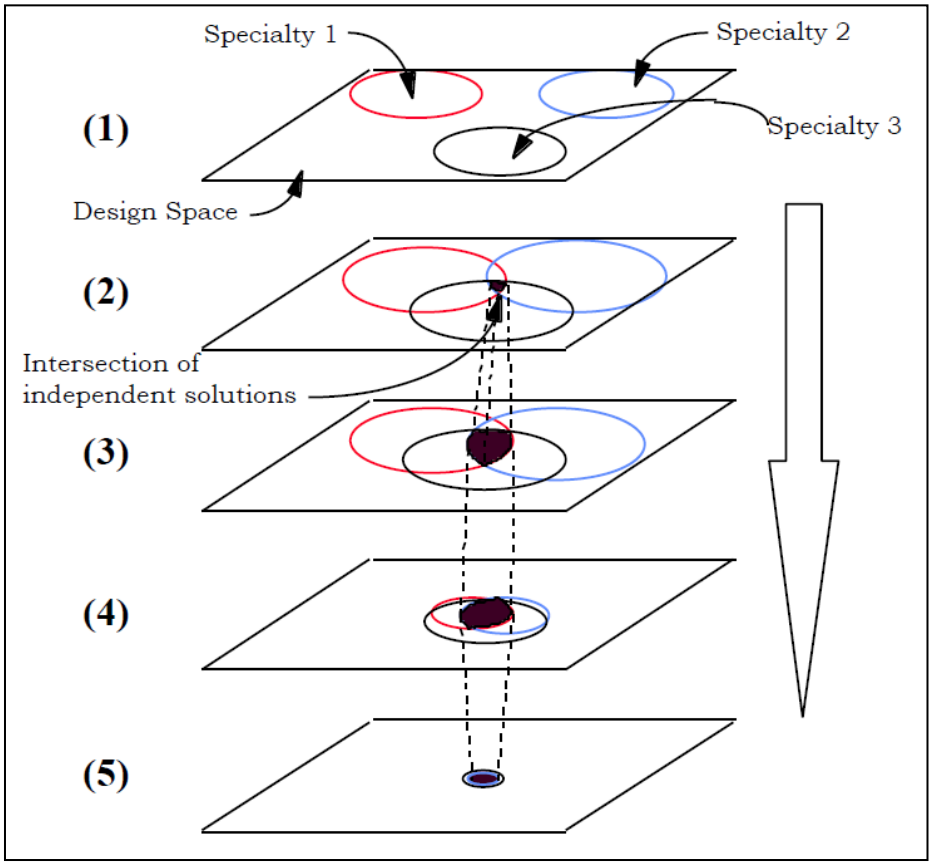


**Abb. 23: Set-Based Concurrent Engineering (Bernstein, 1998 S. 49)**

Beim Set-Based Concurrent Engineering wird also mit einer größeren, durch einen Lösungsbereich definierten Anzahl an denkbaren Lösungsvarianten begonnen. Je Fachbereich (Abteilung, Systemkomponente) wird dieser Lösungsbereich hinsichtlich Machbarkeit analysiert. Die Lösungsvielfalt wird nach und nach reduziert, bis die allgemein beste Gesamtlösung gefunden wird (Ward, et al., 1995 S. 49).

Die kontinuierliche Reduktion von Varianten im Entwicklungsprozess von Toyota bis zur Festlegung einer machbaren Lösung für jeden Fachbereich (Abteilung, Systemkomponente) ist in Abb. 24 skizziert.

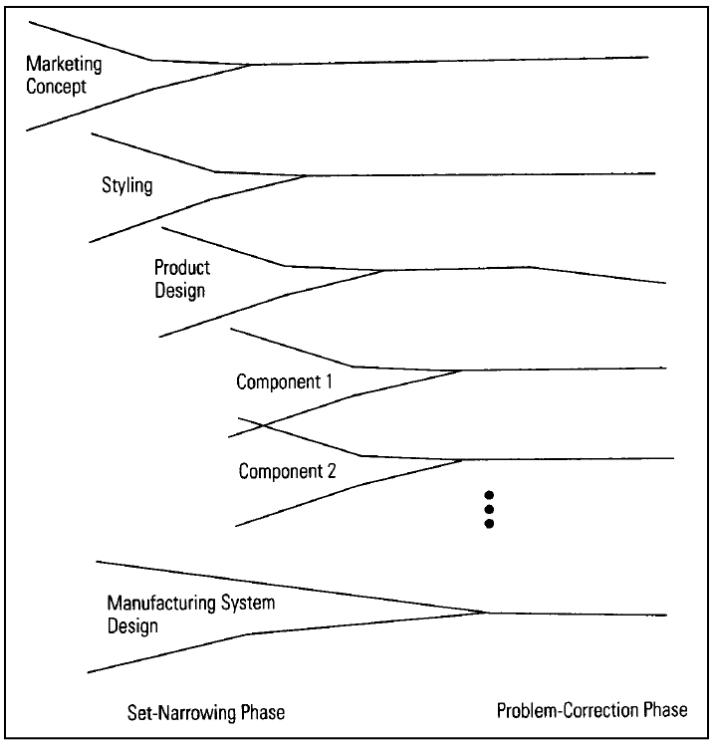


**Abb. 24: Toyotas Set-Based Concurrent Engineering Prozess (Ward, et al., 1995 S. 49)**

Sobek et al. erachten sehr gut ausgebildete Mitarbeiter mit viel Erfahrung als notwendiges Kriterium für diese Art der Entwicklung (Sobek, et al., 1999 S. 72). Das Set-Based Concurrent Engineering ist nämlich eher als Prinzip, denn als standardisierte Vorgehensweise zu betrachten. Es ist bei jedem Projekt unterschiedlich anzuwenden, wobei das Verständnis der Mitarbeiter gefragt ist (Sobek, et al., 1999 S. 81). Die Effizienz dieser Vorgehensweise ist ebenso fragwürdig. Das Identifizieren vieler Lösungsmöglichkeiten zu Beginn ist nämlich nicht genug, um die Varianten auch für die Entwicklung nutzbringend darzustellen. Um Daten für die Entscheidungshilfe zu generieren, müssen auch Prototypen entwickelt oder Simulationsmodelle erstellt werden (Sobek, et al., 1999 S. 74ff). Set-Based Concurrent Engineering kann aber auch eine ganze Reihe an Vorteilen bieten (Ward, et al., 1995 S. 58ff):

- Die Kommunikation wird zuverlässiger und effizienter: Spricht man über Lösungsbereiche und nicht über diskrete Lösungen, so müssen viele Details nicht diskutiert werden, die sich später wieder ändern können.
- Das Simultaneous Engineering wird unterstützt: Späte Phasen (z.B. Entwicklung der Fertigungsprozesse) können früher und effektiver integriert werden.
- Es stehen schon früh Daten (z.B. Festigkeitsberechnungen) für Designentscheidungen zur Verfügung.
- Das Lernen der Organisation wird gefördert: Die gewonnenen Erkenntnisse aus dem Weg, der für eine Lösungsfindung notwendig war, können wiederverwendet werden, inkl. des Wissens wie die Abstimmung der Entwicklungsbereiche funktioniert hat.
- Die Suche nach einem gesamthaften Optimum wird gefördert: Viele Optionen werden durchdacht und auch radikale Innovationen können mit geringerem Risiko entwickelt werden, weil bei einem Fehlschlag andere Varianten zur Verfügung stehen.

Im Lean SW Development werden andere Ausprägungsformen des Set-Based Designs beschrieben. Z.B. kann die Entwicklung auf mehrere Teams aufgeteilt werden, wobei eines eine Basis-Variante des Produkts entwickeln soll, die mit großer Zuverlässigkeit funktioniert. Andere Teams entwickeln zusätzlich gewünschte Funktionalität, deren Zuverlässigkeit aber schwieriger zu erreichen ist. In das Produkt werden jene Funktionen integriert, die bei Erreichen der Entwicklungsdeadline zuverlässig verfügbar sind. Die nicht fertig entwickelten Funktionen können ev. in zukünftigen Projekten weiterentwickelt werden (Poppendieck, et al., 2009 S. 160ff).

In einer weiteren Ausprägungsform entwickeln mehrere Teams unterschiedliche Basis-Varianten einer Software, die dem Kunden präsentiert und von diesem bewertet werden. Danach werden auf Basis der vielversprechendsten Funktionen wiederum mehrere Varianten entwickelt. Die Varianten werden sukzessive reduziert, bis sich eine optimale Lösung herauskristallisiert (Poppendieck, et al., 2009 S. 162).

Haberfellner und de Weck beschreiben eine Ausprägungsform des Set-Based Design, bei der die Entscheidung für das Gesamtkonzept hinausgezögert werden soll, die Realisierung aber schon angestoßen werden muss, um eine rechtzeitige Fertigstellung zu gewährleisten. Dazu sollen Realisierungsschritte, die für möglichst viele Varianten gelten, zuerst durchgeführt werden, während das Gesamtkonzept noch weiterentwickelt werden kann. Die in Abb. 25 dargestellte Vorgehensweise für das Set-Based Design beinhaltet folgende Punkte (Haberfellner, et al., 2005 S. 9f):

- Es werden die verschiedenen Varianten für Gesamtkonzepte entworfen (hier V1 bis V5).
- Für jede Variante werden die erforderlichen Realisierungsschritte (RS) ermittelt.
- Die Realisierung startet mit jenen Schritten, die für möglichst viele Varianten gelten.
- Mit der Zeit fallen alle Varianten weg, die mit den durchgeführten Realisierungsschritten nicht mehr sinnvoll fertiggestellt werden können (in Abb. 25 sind das V1 und V2).
- Zu beachten ist dabei, dass man durch willkürliche Festlegung der Realisierungsschritte nicht implizit Varianten eliminiert.

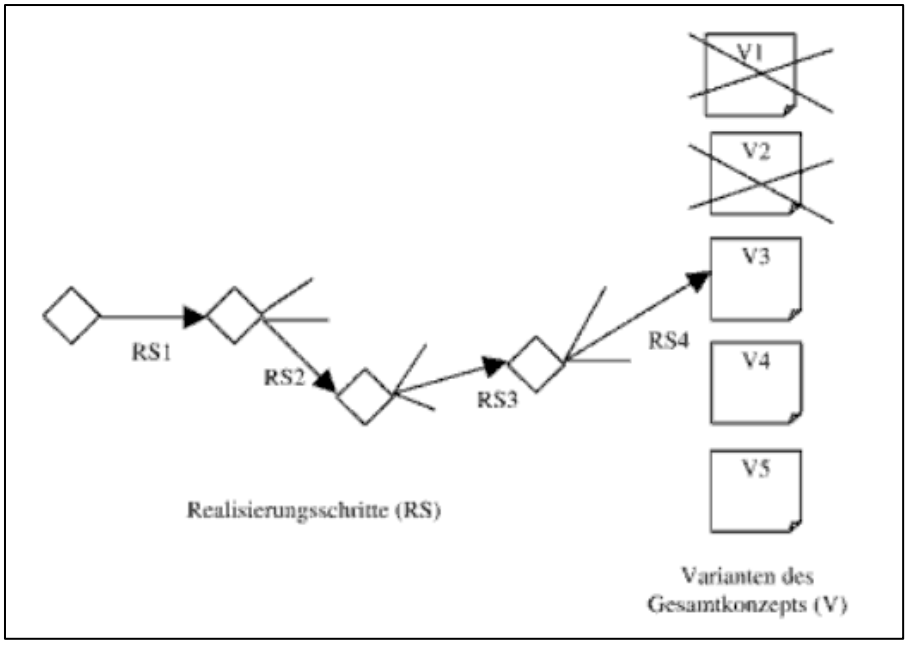


**Abb. 25: Set-Based Design (Haberfellner, et al., 2005 S. 9)**

Durch diese Methode kann man der Unsicherheit einer frühen Projektphase Rechnung tragen und die Entscheidung für die letztendlich gültige Lösungsvariante hinausschieben, sie kann aber als Nachteil einen suboptimalen Projektablauf bedeuten (Haberfellner, et al., 2005 S. 10). Außerdem muss hinterfragt werden, ob im jeweiligen Systementwicklungsprojekt die Realisierungsschritte in beliebiger Reihenfolge abgearbeitet werden können, damit man auch mit jenem starten kann, der in den meisten Varianten benötigt wird.

Als grundlegende Vorgehensweise der Entwicklung kann das Set-Based Design für den Umgang mit Veränderungen als prinzipielle Alternative zum AEZ gesehen werden. Beim AEZ wird ein Konzept des Systems entwickelt und laufend angepasst, bis es den Kundenanforderungen entspricht. Wegen der Anpassbarkeit des Systems ist die gleichzeitige Entwicklung alternativer Varianten nicht notwendig. Für den Fall, dass die Anpassbarkeit aber nicht gegeben ist oder Entscheidungen hinausgezögert werden sollen, die kaum rückgängig gemacht werden können, stellt der AEZ aber keine brauchbare Vorgehensweise dar. Hier kann das Set-Based Design eingesetzt werden, um die Flexibilität zu erhöhen und die Informationsgewinnung über System und Anforderungen über einen längeren Zeitraum zu ermöglichen (Poppendieck, et al., 2009 S. 160ff).

**Empirische Erhebungen:** In der empirischen Studie konnte das Set-Based Design nicht in einer expliziten Form gefunden werden. Es wurden aber einige Kritikpunkte hinsichtlich des Prinzips geäußert. Es sei schwer vorstellbar, dass Mitarbeiter wirklich mit einer größeren Menge an Varianten arbeiten wollen. Einerseits weil das Denken in Lösungsmengen schwierig ist und Mitarbeiter eher immer an konkrete Lösungen denken. Andererseits weil häufig Standardlösungen in Unternehmungen vorhanden sind und oft versucht wird eine Lösung durchzusetzen, die für den eigenen Entwicklungsbereich möglichst wenig Aufwand mit sich bringt. Auch der nötige Aufwand für die Entwicklung einer Anzahl an Varianten (Prototypen …) wurde für gewöhnliche Systementwicklungsprojekte als nicht durchführbar erachtet. Deshalb werden größere Mengen an Lösungsvarianten nur ganz zu Beginn der Entwicklung als realistisch betrachtet. Aber schon nach dem ersten Abstimmungsgespräch ist zu vermuten, dass sich jeder Entwicklungsbereich nur mehr um jene Variante kümmern wird, die als bester Kompromiss zwischen den Abteilungen erscheint, um den Aufwand zu minimieren. Eventuell sind dann noch ein paar wenige Varianten zusätzlich machbar, die aber kaum mehr der Idee eines Lösungsbereichs entsprechen, sondern nur durch die leichte Abänderbarkeit eines oder weniger Parameter bestehen. Für dieses Prinzip werden deshalb sehr gute Mitarbeiter als erforderlich angesehen und auch eine Kultur, die eine große Variantenvielfalt schätzt. Dazu darf aber kein Kostendruck herrschen, da die Entwicklung mehrerer Varianten zumindest kurzfristig die Kosten erhöht.

**Bewertung:** Das Prinzip erscheint sehr interessant, speziell wenn die Art des Systems eine schlechte Anpassbarkeit bedingt und die Entwicklung in kleinen Schritten (bzw. der AEZ) nicht möglich ist. Für diesen Fall kann Set-Based Design in einigen Punkten Ersatz für die Entwicklung in kleinen Schritten bieten. Die frühe Informationsgewinnung wird durch die frühe Kommunikation aller Teilbereiche der Entwicklung unterstützt. Die bei der Entwicklung in kleinen Schritten geforderten Zwischenergebnisse können durch Prototypen zu den Lösungsvarianten ersetzt werden, was aber natürlich einen hohen Aufwand verursacht. Eine Einbindung des Kunden ist auch denkbar. Der Umgang mit Veränderungen wird zwar nicht direkt verbessert, da aber nicht von Beginn an mit konkreten Lösungen gearbeitet wird, müssen diese auch nicht konkret gestaltet werden und dadurch können Änderungen im

Entwicklungsverlauf vermieden werden. Die konkrete Gestaltung wird auf einen Zeitpunkt verschoben, wo mehr Informationen zur Verfügung stehen und Änderungen unwahrscheinlicher sind. Die Flexibilität wird im Gegensatz zur Entwicklung in kleinen Schritten nicht im Produkt erreicht, sondern in der Verfügbarkeit vieler Varianten. Hinsichtlich Effizienz und Effektivität ist Set-Based Design aber kritisch zu betrachten.

### 5.4.19 Adaptive Product Development Process nach Levardy

**Erklärung:** Hierbei handelt es sich um ein Optimierungsverfahren auf Basis der Design Structure Matrix bei dem mit Hilfe der Monte Carlo Simulation versucht wird, die Teilschritte von anpassbaren Prozessen in der optimalen Reihenfolge auszuführen (Levardy, et al., 2009). Dadurch soll die Anzahl der Iterationen verringert werden. Das Verfahren klingt interessant, doch meint der Autor selbst, dass es noch einige Zeit brauchen wird, bis die Anwendbarkeit in der Industrie gegeben ist (Levardy, et al., 2009 S. 222).

### 5.4.20 Förderung des Lernens

**Erklärung:** Das ständige Lernen sollte jedem Systems Engineering Prozess innewohnen, schließlich werden Systeme entwickelt, die in genau dieser Form vorher noch nicht entwickelt wurden. Es muss also während der Entwicklung ein Lernprozess stattfinden, wie das System bestmöglich zu entwickeln ist (Turner, 2007 S. 12). Um diesen Lernprozess zu optimieren, sollen nun Tätigkeiten gefunden werden, die ein ständiges, aktives Lernen zum Produkt und den Prozessen forcieren.

Die meisten Praktiken dazu aus der agilen Softwareentwicklung ähneln Lessons Learned[7] Workshops. Nur finden sie nicht wie diese nur am Ende eines Projekts statt, sondern viel häufiger.

Im Extreme Programming findet die sogenannte Reflexion beim wöchentlichen und beim monatlichen Meeting statt. Sie kann und soll aber auch ungeplant stattfinden und speziell die Arbeitsweise analysieren und verbessern (Beck, et al., 2005 S. 29f).

Bei Scrum ist ein genau geregeltes „Sprint Retrospective Meeting“ am Ende jedes Entwicklungsschritts vorgesehen, in dem über Positives im letzten Zyklus, als auch über Verbesserungsmöglichkeiten diskutiert wird (Schwaber, 2004 S. 138f).

In Crystal Clear werden Reflexions-Workshops vorgeschlagen, die danach fragen, was beibehalten werden soll, wo es häufig Probleme gibt und was im nächsten Inkrement ausprobiert werden sollte (Cockburn, 2005 S. 97ff).

Innerhalb solcher Verbesserungs-Meetings kann auch versucht werden, die Planung ständig zu verbessern (Boehm, et al., 2005 S. 32). Z.B kann durch einen Vergleich zwischen geplanter und abgeschlossener Tätigkeit innerhalb eines Entwicklungsschritts auf die Arbeitsgeschwindigkeit des Teams geschlossen werden. Dieses Feedback ist dabei nicht vom Erfolg des Zyklus abhängig, sondern kann auch generiert werden, falls dieser verworfen wird (McMahon, 2006 S. 28). Es wird also das Feedback eines Planungszyklus verwendet, um die Planung des nächsten Zyklus zu verbessern, wie schon bei der Entwicklung in kleinen Schritten erwähnt wurde.

[7] Erklärung unter http://de.wikipedia.org/wiki/Lessons_Learned

Die Entwicklung in kleinen Schritten bietet auch die Möglichkeit, Produkteigenschaften und Kosten zu quantifizieren und nach jedem Schritt Abweichungen zum geplanten Level festzustellen. Daraus können wertvolle Informationen zur Entwicklung gewonnen werden, die für die weitere Planung von Nutzen sein können (Gilb, 2006 S. 7f).

Malotaux empfiehlt speziell Problemstellungen, bei denen ein großer Lerneffekt zu erwarten ist, in einen frühen Entwicklungszyklus zu reihen (Malotaux, 2006 S. 2). Dadurch wird dieses Wissen für den ganzen weitern Projektablauf verwendbar.

**Empirische Erhebungen:** In der Praxis wurden in den Entwicklungsprozessen keine speziell auf das Lernen abzielenden Praktiken gefunden. Nur traditionelle Lessons Learned Workshops am Ende von Projekten wurden durchgeführt.

**Bewertung:** Die benötigten Ressourcen für Verbesserungs-Workshops sind überschaubar. Es scheint also durchaus überlegenswert diese in den Prozess zu integrieren und regelmäßig durchzuführen.

## 5.5 Maßnahmen im Zusammenhang mit dem System

Eine flexible d.h. veränderbare Systemarchitektur ist für viele Autoren eine Grundbedingung für agile Entwicklung z.B. (Gilb, 2005 S. 7). Es gibt dazu zahlreiche Prinzipien, die in den folgenden Unterkapiteln dargestellt werden. Es werden aber auch Möglichkeiten gezeigt, wie die Agilität in anderen Aspekten unterstützt werden kann. Dieses Kapitel soll damit eine Übersicht möglicher Prinzipien geben, die im Zusammenhang mit dem Produkt für eine höhere Agilität im SE sorgen können. Das Aufzeigen konkreter technischer Maßnahmen ist aber nicht Teil dieser Arbeit. Deshalb werden die meisten Prinzipien nur kurz und theoretisch aus der Literatur erklärt.

### 5.5.1 Einfachheit

**Erklärung:** Einfachheit wird im XP als einer der übergreifenden Werte genannt (Beck, et al., 2005 S. 18f). Im Lean SW Development ist die Vermeidung von Unnötigem (worunter auch Zusatzfunktionen des Produkts gesehen werden) das erste Prinzip (Poppendieck, et al., 2009 S. 23).

In der SW Entwicklung sind auch Begriffe wie YAGNI[8] („You Aren't Going to Need It") und KISS[9] (z.B. „Keep it small and simple" oder „Keep it simple, stupid!") entstanden. Es soll dabei immer die einfachste Lösung gewählt werden und keine Funktion implementiert werden, bevor man sich nicht sicher ist, dass diese erforderlich ist (Boehm, et al., 2005 S. 32).

Es wird versucht, Komplexität aus dem System zu nehmen und das Verhältnis von sinnvollen Funktionen zu nutzlosen und gefährlichen Funktionen möglichst groß zu gestalten. Ein ideales System besteht ausschließlich aus sinnvollen Funktionen. Richtlinien für die Gestaltung eines möglichst simplen Systems sind minimale Anzahl von Schnittstellen, minimale Anzahl an Sekundär-Funktionen und Fokus auf bereits existierende Ressourcen (im System oder in der System-Umgebung) (Fricke, et al., 2005 S. 348f).

**Bewertung:** Einfachheit soll also Effizienz und Effektivität erhöhen. Außerdem erhöht sie auch indirekt die Flexibilität bzw. Anpassbarkeit des Produkts. Im Gegensatz zu einem einfachen Produkt bestehen bei

[8] Erklärung unter http://de.wikipedia.org/wiki/YAGNI
[9] Erklärung unter http://de.wikipedia.org/wiki/KISS-Prinzip

einem komplexen Produkt nämlich mehr Beziehungen zwischen seinen Einzelteilen, die bei Änderungen zu berücksichtigen sind. Damit kann im Allgemeinen gesagt werden, dass Änderungen an einfachen Systemen leichter durchzuführen sind, als an komplexen. Ein Hindernis für die Einfachheit können Sicherheitsaspekte sein, die nach komplexeren Lösungen verlangen.

### 5.5.2 Anpassbarkeit

**Allgemeine Erklärung:** Die Einfachheit ist bereits eine erste Möglichkeit, das Produkt leichter anpassbar zu machen. Es folgen nun weitere Prinzipien, welche die Anpassbarkeit des Systems, bzw. seiner Architektur erhöhen können. Vorher soll aber noch kurz dargestellt werden, welche Änderungen am System durchgeführt werden können. Dove unterteilt diese in proaktive und reaktive Änderungen (Dove, 2001 S. 88):

- Proaktiv: Proaktive Änderungen werden von der Unternehmung selbst ausgelöst. Man will mit ihnen neue Werte generieren bzw. Innovationen umsetzen.
    - Erschaffung (und Eliminierung): Etwas (z.B. ein Modul) wird neu erzeugt oder auch ausgeschieden.
    - Verbesserung: Etwas, das bereits vorhanden ist, wird verbessert (z.B. hinsichtlich seiner Fähigkeiten).
    - Migration: Es wird ein grundlegender Wandel durchgeführt (z.B. wird die, das System unterstützende, Infrastruktur geändert oder ein Generationswechsel in der Technologie durchgeführt).
    - Modifizierung (Hinzufügen/Entfernen von Leistungsvermögen): Es wird eine bestimmte und einmalige Fähigkeit hinzugefügt oder entfernt.
- Reaktiv: Diese Änderungen werden durch ein Ereignis ausgelöst, auf welches reagiert werden soll. Das Ereignis muss aber nicht ein Problem sein, sondern kann auch eine Chance oder Gelegenheit darstellen.
    - Fehlerbehebung: Korrektur einer Fehlfunktion o.Ä.
    - Variation: Anpassungen bei der Anwendung, um auf Abweichungen zu reagieren.
    - Erweiterung (und Verringerung) der Leistungsfähigkeit: Eine vorhandene Fähigkeit wird vergrößert oder verkleinert.
    - Rekonfiguration: Umgestaltung bzw. Reorganisation vorhandener Elemente und ihrer Beziehungen.

Damit man genügend Flexibilität hat, Veränderungen gegenüberzutreten, sollen die Änderungen am System möglichst leicht, einfach und schnell durchgeführt werden können und dazu soll das System nach folgenden Prinzipien gestaltet werden:

***Unabhängigkeit***

Jede Systemfunktion soll durch einen unabhängigen Entwurfsparameter (technisches Prinzip, Systemgröße, Komponente ...) hergestellt werden. Das Verändern eines Entwurfsparameters soll keinen Einfluss auf verwandte Entwurfsparameter haben (Fricke, et al., 2005 S. 349f).

***Modularität***

System-Funktionen werden in einzelnen Modulen angehäuft. Diese sind dadurch gekennzeichnet, dass Verbindungen innerhalb der Module sehr stark, während Verbindungen zwischen den Modulen sehr lose sind. Dieses Prinzip erleichtert die Wiederverwendung einzelner Module (Elemente oder Komponenten) in unterschiedlichen Systemen z.B. bei Plattform-Strategien (Fricke, et al., 2005 S. 350f). Änderungen an einem Modul haben demnach wenig Einfluss auf die weiteren Module. Dadurch kann man die oben beschriebenen Änderungsarten leichter auf der Ebene einzelner Module durchführen.

***Integrierbarkeit***

Hiermit wird die Kompatibilität der Module zueinander angesprochen, welche durch die Verwendung einheitlicher Schnittstellen verbessert werden soll. Diese Kompatibilität wird von Fricke et al. als unabdingbar für zusammenhängende Systeme in sich schnell verändernden Umgebungen gesehen (Fricke, et al., 2005 S. 352f).

Deshalb verlangt z.B. Extreme Programming auch nach Standards für die Kommunikation zwischen den Modulen (Paulk, 2001 S. 3). Es können so neue Module leichter und schneller in ein bestehendes System eingefügt werden, wenn Änderungen das erforderlich machen.

***Nicht-hierarchische Integration***

Die Schnittstellen befinden sich alle auf demselben hierarchischen Level. Dadurch wird direkte, flexible und schnelle Kommunikation möglich, mit der man schnell neue Verknüpfungen aufbauen oder alte abbrechen können soll (Fricke, et al., 2005 S. 353f).

***Skalierbarkeit***

Es soll damit eine Unabhängigkeit von der Größe/Kapazität erreicht werden. Dazu sollen entweder einzelne Elemente vergrößert und verkleinert werden können. Oder es soll das Verknüpfen mehrerer Elemente eines Typs möglich sein, um die Kapazität anzupassen (Fricke, et al., 2005 S. 353f).

***Verstärkter Softwareeinsatz***

Software ist in komplexen Systemen oft der entscheidende Faktor, der Flexibilität möglich macht (Boehm, et al., 2008 S. 2f), (Capell, et al., 2008 S. 3). Software liefert oft nicht nur das wesentliche Potential und den Wert eines Systems, sondern macht dessen Flexibilität aus und erlaubt schnelle Reaktionen auf Veränderungen in der Umwelt. Man soll deshalb versuchen, den vollen Nutzen aus der Flexibilität und Anpassbarkeit der Software zu ziehen. Damit kann man auch die Systemarchitektur besser abstimmen und das Gesamtsystem weniger komplex machen (Turner, 2007 S. 14).

Es soll also versucht werden, Funktionen bei denen man spätere Änderungen vermutet, durch Software darzustellen, wenn dies möglich ist. Darauf muss bei der Architekturgestaltung des Systems Rücksicht genommen werden. Wie schon beim Simultaneous Engineering (5.4.2) angesprochen, soll die SW früh in die Entwicklung einbezogen werden und nicht nach der HW entwickelt werden.

Auch nach der Produkteinführung kann die Software noch für eine erhöhte Anpassbarkeit des Systems sorgen. Durch SW-Updates[10] ist es einfach und kostengünstig möglich, noch gewisse Änderungen am Produkt durchzuführen. Z.B. um Funktionen hinzuzufügen oder Fehler auszubessern.

**Allgemeine Bewertung:** Es gibt also einige Möglichkeiten, das System bzw. die Systemarchitektur anpassbar zu machen. Wie die praktische Ausführung dieser Prinzipien aussieht, ist natürlich stark von der Art des Systems abhängig, auch ob die Umsetzung überhaupt möglich bzw. wirtschaftlich sinnvoll ist. Eine Gestaltung des Systems hinsichtlich Anpassbarkeit kann auch hohen Aufwand mit sich bringen und damit Effizienz und Effektivität in Frage stellen. Die Anpassbarkeit des Produkts ist auf jeden Fall aber ein wichtiger Faktor für die Agilität. Nicht nur für die Flexibilität, die man zur Umsetzung von Änderungen am Produkt benötigt, sondern auch für die Machbarkeit von Praktiken wie der Entwicklung in kleinen Schritten oder dem AEZ.

Die Anpassbarkeit des Systems ist aber nicht nur ein Kriterium für die Agilität während der Entwicklungsphase, sondern auch für mögliche Änderungen während der Nutzungsphase. Speziell wichtig ist diese Flexibilität für Systeme, die mit großen Investitionen verbunden sind, eine lange Lebensdauer haben (>10 Jahre), bei denen während dieser Zeit signifikante Änderungen der Anforderungen auftreten können und wo die Kosten zu groß sind, um bei jeder Anforderungsänderung ein neues System zu entwickeln (Haberfellner, et al., 2005 S. 8, 13).

### 5.5.3 Förderung nutzbarer Zwischenergebnisse

**Allgemeine Erklärung:** Zur Umsetzung des AEZ muss das System während der Entwicklung leicht anpassbar sein. Das bedeutet aber noch nicht, dass die Zwischenergebnisse zur besseren Bewertbarkeit auch vom Kunden genutzt werden können. Die Nutzbarkeit von Zwischenergebnissen hängt zwar im Wesentlichen von der Art des Systems und der Form der Aufteilung des Entwicklungsumfangs bei der Entwicklung in kleinen Schritten ab. Es wurden aber auch einige Prinzipien gefunden, die eventuell eine Hilfestellung bieten können, um die Zwischenergebnisse so gestalten zu können, dass diese vom Kunden genutzt werden können. Die folgenden Prinzipien sollen es ermöglichen, das System leichter in einer Form aufteilen zu können, in der die Einzelteile (Zwischenergebnisse) dennoch eine nutzbare Funktion darstellen können.

***Autonomie***

Eine autonome Gestaltung von Objekten (Modulen) soll dazu führen, dass diese ihre grundlegenden Funktionen unabhängig vom System, in das sie eingebettet sind, durchführen können (Fricke, et al., 2005 S. 353). Damit kann der Kunde z.B. ein separates Modul testen (bevor das Gesamtsystem realisiert ist), unter Umständen auch eingebettet in ein altes, kompatibles System.

***Selbst-Organisation***

Hierbei soll sich die Interaktion von einzelnen Modulen selbstständig anpassen oder zwischen den Modulen ausgehandelt werden (Madni, 2008 S. 2). Das kann bei einer Entwicklung in kleinen Schritten dafür sorgen, dass der Integrationsaufwand neu hinzugefügter Module geringer wird (Z.B. wenn eine neue Komponente wie bei einem PC angeschlossen wird und sich die dafür notwendigen Treiber

[10] Erklärung unter http://de.wikipedia.org/wiki/Update

automatisch installieren (Plug&Play)). Damit wird es kostengünstiger möglich, neue nutzbare Zwischenergebnisse zu einem bereits vom Kunden genutzten Teilsystem hinzuzufügen.

***Dezentralisation***

Es soll versucht werden, die Kontrolle dezentral zu verteilen. Entscheidungen sollen dort getroffen werden, wo am meisten Wissen dafür zur Verfügung steht. Dadurch soll ein schnelles Anpassen ermöglicht werden (Fricke, et al., 2005 S. 354f). Bei einer zentralen Regeleinheit, müsste diese immer schon vorhanden sein, wenn einzelne, davon abhängige Module getestet werden sollen. Kann sich aber jedes Modul selbst regeln, um seine Funktion darzustellen, so kann es unabhängig von der zentralen Regeleinheit genutzt werden.

**Allgemeine Bewertung:** Wie bei der Anpassbarkeit sind hier natürlich wieder die Kosten für die Herstellung dieser Funktionalitäten zu berücksichtigen. Ihre Vorteile gehen aber auch wieder über die Entwicklungsphase hinaus und können auch während der Anwendungsphase genutzt werden.

### 5.5.4 Reserven im Potential des Produkts

**Erklärung:** Man entwickelt das Produkt mit höherem Leistungspotential, als für notwendig erachtet bzw. vom Kunde gefordert wird (Pich, et al., 2002 S. 1010). Sollte der Kunde später eine höhere Leistung wünschen, ist diese leicht herzustellen bzw. schon vorhanden. Die Flexibilität ist auch größer, wenn der Entwickler Faktoren, die einen Einfluss auf das Leistungspotential haben, falsch berechnet hat. Dann kann dieser Fehler leicht korrigiert werden. Damit können Dimensionierungsfehler genauso wie geänderte Leistungsanforderungen leichter kompensiert werden. Unter Umständen kann die Entwicklung auch in zukünftigen Projekten verwendet werden, in denen ein höheres Leistungspotential gefordert wird.

**Bewertung:** Die Anwendung dieses Prinzips ist vor allem aus wirtschaftlichen Gründen schwierig, weil sie leicht in Konflikt mit Effizienz und Effektivität geraten kann.

***Redundanzen***

Unter Umständen können auch im System vorhandene Redundanzen ähnlich wie Reserven im Potential genutzt werden.

Redundanzen ermöglichen neben der größeren Sicherheit auch Kapazitäts-, Funktionalitäts- und Leistungserhöhungen, wenn diese plötzlich benötigt werden (Fricke, et al., 2005 S. 355). Es kann also auf Veränderungen, die solche Erhöhungen erfordern, schneller reagiert werden. Das Vorsehen von Redundanzen bedeutet aber auch wieder einen Ressourcenaufwand.

Bei „funktionaler Redundanz“ versucht man dieselbe Funktion auf unterschiedlichen Wegen (mit unterschiedlichen Modulen/Funktionen) zu erreichen, um keine physikalische Redundanz (zwei Mal das gleiche Bauteil) zu benötigen (Madni, 2008 S. 2). Dadurch kann eventuell der benötigte Ressourcenaufwand niedriger gehalten werden.

### 5.5.5 Wiederverwendung von Modulen

**Erklärung:** Es soll hierbei ein Repertoire an Modulen für möglichst viele Funktionen aufgebaut werden (Dove, et al., 2008 S. 4). Dazu müssen die Module so definiert sein, dass sie auch in unterschiedlichen Anwendungen verwendet werden können (Madni, 2008 S. 2).

Bei diesem Prinzip geht es in erster Linie um die schnelle Einsatzmöglichkeit der vorhandenen Module. Die Agilität wird also weniger durch eine Erhöhung der Flexibilität, als durch die schnelle Reaktionsfähigkeit auf Veränderungen unterstützt, die erfolgen kann, sollten die erforderlich gewordenen Funktionalitäten durch Wiederverwendung abgedeckt werden können.

**Bewertung:** Durch die Wiederverwendung von Modulen kann der Aufwand innerhalb eines Entwicklungsprojekt sicher wesentlich verringert werden, wenn für alle Veränderungen schon passende Module bereitstehen. Der Aufbau des Repertoires an Modulen und das Sicherstellen ihrer Integrierbarkeit verursachen aber natürlich wieder Aufwand über die Projekte hinweg. Es ist hier also der Aufwand für das Repertoire an Modulen dem damit generierten Nutzen gegenüberzustellen. Enge Grenzen für die zu erwartenden Veränderungen und einer hoher Grad an Wiederverwendbarkeit sprechen für die Anwendung dieses Prinzips.

### 5.5.6 Produktplattform/Product Lines

**Erklärung:** Automobil-Plattformen[11] sind gekennzeichnet durch einen gemeinsamen Aufbau (Fahrgestell, Motor, Getriebe …) für mehrere Fahrzeugmodelle, die unterschiedlich aussehen (Karosserie, Innenraumgestaltung …). Computer-Plattformen[12] verwenden z.B. eine gemeinsame Architektur oder eine einheitliche Programmiersprache, damit Softwareprogramme auf Prozessoren verschiedener Hersteller laufen können.

Eine Software Product Line wird durch eine Reihe von Softwareprogrammen charakterisiert, die eine Anzahl von gemeinsamen Funktionen teilen, welche die Kundenwünsche eines Marktsegments befriedigen und die auf Basis derselben Kernfähigkeiten entwickelt wurden (Clements, et al., 2002).

Für diese Arbeit ist es nicht notwendig, Plattformen oder Product Lines weiter zu analysieren. Es soll hier das gemeinsame Grundprinzip dieser Konzepte betrachtet werden. Dieses besteht darin, dass ein Basissystem entwickelt wird, von dem konkrete Produkte abgeleitet werden können.

In der Arbeit der S²QI Gruppe wurde dieses Prinzip schon als möglicher agiler Erfolgsfaktor genannt. Es wird aber darauf hingewiesen, dass Flexibilität nur in einem bestimmten Bereich erhöht wird. Im Allgemeinen schränkt man die Flexibilität ein, da man sich auf ein Basissystem festlegt und Ressourcen dafür investiert. Durch die Wiederverwendung wird aber die Effizienz der folgenden Entwicklungen erhöht (Stelzmann, et al., 2010 S. 7).

Die Agilität wird erhöht, weil man innerhalb der Möglichkeiten des Basis-Systems relativ schnell und einfach Änderungen vornehmen kann, in Verbindung mit der Wiederverwendung von Modulen. Deshalb kann man sich auch relativ spät dafür entscheiden, welche Module verwendet werden sollen. Es sind

[11] Nähere Erklärung unter http://de.wikipedia.org/wiki/Plattform_%28Automobil%29
[12] Nähere Erklärung unter http://de.wikipedia.org/wiki/Plattform_%28Computer%29

auch sehr kurze Markteinführungszeiten neuer Produkte möglich, wenn die Basis einmal besteht (Leitner, et al., 2011 S. 11).

Die Flexibilität in der Architektur des Basis-Systems wird als wichtiger Erfolgsfaktor für Plattformen bzw. Product Lines gesehen (Stelzmann, et al., 2010 S. 253). Es ist hier aber auch wieder auf den Aufwand für die Entwicklung zu achten, um die Wirtschaftlichkeit zu gewährleisten.

**Bewertung:** Die Verwendung von Plattformen bzw. Product Lines kann unter gewissen Umständen als agilitätssteigerndes Prinzip gesehen werden. Man muss dafür in einer Branche tätig sein, in der die Produkte ähnlich sind. Und der Aufwand für die Entwicklung der Basis ist durch eine entsprechende Anzahl von davon abgeleiteten Produkten zu rechtfertigen. Dann ist es möglich, schnell auf neue Marktanforderungen zu reagieren und neue, von der Basis abgeleitete, Produkte zu entwerfen. Flexibilität besteht aber nur in dem durch die Basis vorgegebenen Rahmen.

### 5.5.7 Refactoring

**Erklärung:** Software wird zur besseren Lesbarkeit und Veränderbarkeit umstrukturiert, ohne die Funktion zu verändern (Boehm, et al., 2005 S. 32). Ohne das Verhalten des Systems zu ändern, werden Doppelfunktionen entfernt, die Kommunikation verbessert, Vereinfachungen vorgenommen und die Flexibilität erhöht (Paulk, 2001 S. 3).

Weil durch Änderungen immer wieder die Komplexität erhöht wird, ist im Lean SW Development ein regelmäßiges Refactoring vorgesehen, um die Komplexität wieder zu verringern (Poppendieck, et al., 2009 S. 164ff).

**Bewertung:** Refactoring ist eine technische Maßnahme, die in agilen SW Entwicklungsmethoden vorgeschlagen wird, um Software nach Änderungen wieder einfacher und damit wieder anpassbarer zu machen.

### 5.5.8 Systemfeedback

**Erklärung:** Unter Systemfeedback sollen Informationen verstanden werden, die das Produkt während seiner Anwendung generiert und an den Entwickler übermittelt. Diese Informationen können vom Entwickler z.B. bei Serviceaktionen abgerufen oder auch laufend vom System übermittelt werden, z.B. über Internet oder Mobilfunk. Der Entwickler kann diese Informationen dann zur Verbesserung des bestehenden Systems oder zur Entwicklung neuer Systeme verwenden.

In einem Fallbeispiel beschreiben Aschbacher et al. ein solches System, das seine Auslastung über eine Internetverbindung an den Entwickler zurücksendet. Der Entwickler konnte dadurch zielgerichtet neue Angebote für Produkte bzw. Services machen, die genau den Bedürfnissen des Kunden entsprachen (Aschbacher, et al., 2010 S. 7ff).

**Bewertung:** Dieses Prinzip stellt ein interessantes Konzept zur Datengewinnung dar. Nachteil ist, dass das System schon in Verwendung sein muss, weshalb die Daten nur mehr für Anpassungen oder für zukünftige Produkte genutzt werden können. Vorteil ist, dass die Daten direkt aus der Verwendung stammen und damit das Verhalten des Systems und die Anforderungen des Kunden sehr genau abbilden können. Neue Informationen und Veränderungen können auf diese Art schnell und direkt erkannt werden. Durch die Abstimmung der Produkteigenschaften auf die gemessenen Bedürfnisse des Kunden

kann die Effektivität sichergestellt werden. Herausforderungen können aber die Implementierung von Sensoren und Übertragungseinheit sein. Ebenso muss der Kunde mit dieser Art von Datensammlung einverstanden sein. Das kann ev. erreicht werden, indem man ihm plausibel macht, wieso sich diese Vorgehensweise auch zu seinem Vorteil auswirkt.

## 5.6 Zusammenfassung

In diesem Kapitel wurden eine Reihe von Prinzipien und Praktiken zur Steigerung der Agilität dargestellt und bewertet. Wie zu erwarten war, wurde kein Prinzip und keine Praktik gefunden, welche(s) direkt und unmittelbar das Gesamtkonzept bzw. die Fähigkeit der Agilität im Systems Engineering erhöht. Viel mehr dienen die einzelnen Praktiken dazu, gewisse Teilfunktionen der Agilität zu steigern, welche in Kapitel 4.2 zur Charakterisierung des ASE dargestellt wurden.

Zur sinnvollen und agilitätssteigernden Anwendung können bzw. müssen die Prinzipien und Praktiken kombiniert werden. Auf viele Zusammenhänge wurde bereits hingewiesen. Ebenso wurden Hindernisse bzw. Herausforderungen hinsichtlich der praktischen Anwendbarkeit identifiziert. Viele Prinzipien und Praktiken erscheinen nur unter gewissen Umständen sinnvoll, z.B. wenn der Kunde zur Einbindung gewillt ist, wenn das Produkt in kleinen Schritten entwickelt werden kann oder es kein sicherheitskritisches Produkt ist. Außerdem wurde in einigen Praktiken ein Zielkonflikt zwischen Flexibilität und Wirtschaftlichkeit festgestellt.

Ein aus allen Prinzipien und Praktiken kombinierter ASE Ansatz kann also nicht eine im Allgemeinen bessere Methodik für SE darstellen. Er kann nur je nach Bedarf und Voraussetzungen einen besseren Umgang mit Veränderungen im SE unterstützen. Deshalb sollen im Kapitel 6 die Anwendungsgebiete analysiert werden, damit der ASE Ansatz für seine potentiellen Anwendungsfälle optimiert werden kann. Bzw. soll ein kontextabhängiges Auswahlverfahren entworfen werden, dass je nach Einsatzgebiet die passenden Praktiken empfiehlt.

Die zu den Prinzipien und Praktiken gesammelten Erkenntnisse werden danach in Kapitel 7 für die Erstellung einer Vorgehensmethodik verwendet, welche den wesentlichen Teil des Lösungskonzepts dieser Arbeit darstellt. Ein Überblick über alle Prinzipien und Praktiken wird in Abb. 32 gegeben, nachdem in Kapitel 6 eine Kategorisierung für die Anwendungsfälle erstellt wurde. In Kapitel 7.4 wird auch dargestellt, wie je nach Anwendungsfall möglichst brauchbare Praktiken ausgewählt werden sollen.

# 6 Der richtige Kontext für Agile Systems Engineering

Hier sollen die dritte und vierte Forschungsfrage, nach dem Bedarf an Agilität und den Voraussetzungen dafür, beantwortet werden. Dies erfolgt vor der Darstellung der Vorgehensmethodik des ASE Ansatzes, weil diese darauf bereits Rücksicht nehmen soll. Wie in Kapitel 5 gezeigt, sind Nutzen und Voraussetzungen der Prinzipien und Praktiken sehr verschieden und es scheint keine allgemeingültige Vorgehensweise zu geben, welche in allen Anwendungsfällen eine optimal agile Lösung bieten kann. Der Ansatz soll das berücksichtigen und für verschiedene Anwendungsfälle Möglichkeiten aufzeigen, die Agilität zu steigern. Ein praktischer Anwender des ASE Ansatzes muss sich dazu fragen, in welchem Kontext er die Agilität erhöhen will. Einflussfaktoren dafür sind einerseits der Bedarf an Agilität im Allgemeinen (also nach der Fähigkeit mit Veränderungen umgehen zu können) und im Speziellen der Bedarf nach gewissen Teilfunktionen der Agilität (also z.B. nach bessere Situationskenntnis, höhere Flexibilität, …). Andererseits ist zu berücksichtigen, welche Voraussetzungen die Projektumgebung bietet.

Auch in der Softwareentwicklung gibt es eine Diskussion über den richtigen Kontext für die Anwendung der agilen Methoden. Dabei wurden zahlreiche Faktoren genannt, die einen Einfluss darauf haben sollen, ob agile Methoden erfolgversprechender sind als traditionelle Entwicklungsmethoden. Nachdem zahlreiche Ideen aus den Methoden der agilen SW-Entwicklung in diese Arbeit übernommen wurden, soll nun aufgezeigt werden, in welchem Kontext der SW-Entwicklung diese Methoden sinnvoll angewendet werden können.

## 6.1 Kontextfaktoren für agile Softwareentwicklung

Wie in Kapitel 2.2.3 beschrieben, werden in der Crystal Methodenfamilie je nach Teamgröße und Kritikalität unterschiedliche Entwicklungsprozesse empfohlen. Der Faktor der **Teamgröße** soll dabei allgemein für die Komplexität der Kommunikation und Koordination stehen. Die **Kritikalität** beschreibt den Schaden, der bei einem Projektfehlschlag oder Produktfehler entstehen kann und geht bis zum Verlust von Menschenleben (siehe Abb. 11). Je kleiner Team und Kritikalität sind, umso agiler sind die empfohlenen Methoden (Cockburn, 2005 S. 253ff).

Boehm und Turner analysieren die Anwendungsgebiete für agile Methoden im Vergleich zu traditionellen Methoden ausführlich (Boehm, et al., 2006 S. 25ff) und nennen zur Unterscheidung 5 Hauptfaktoren (Boehm, et al., 2006 S. 55):

- **Größe**: Agile Methoden passen zu kleinen Projekten und Teams. Traditionelle Methoden wurden speziell für große Projekte und Teams entwickelt.
- **Kritikalität**: Die Anwendung agiler Methoden für sicherheitskritische Produkte ist ungeprüft. Einige traditionelle Methoden wurden speziell dafür entworfen.
- **Dynamik**: Agile Methoden sind für dynamische Umgebungen gemacht, traditionelle Methoden für eher statische.
- **Mitarbeiter**: Agile Methoden brauchen während der gesamten Projektlaufzeit hochqualifizierte Mitarbeiter. Traditionelle Methoden brauchen einige hochqualifizierte Mitarbeiter zu Beginn, danach reichen weniger und es können auch einige weniger qualifizierte oder neu eingestellte, unerfahrene Mitarbeiter verkraftet werden.

- **Kultur**: Agile Methoden eignen sich für Mitarbeiter, die viele Freiheiten bevorzugen aber verantwortungsbewusst sind und dabei erfolgreich arbeiten. Traditionelle Methoden werden von Mitarbeitern bevorzugt, die klare Rollen, Richtlinien und Prozeduren schätzen.

Bei einer detaillierteren Betrachtung ergeben sich für Boehm und Turner die in Abb. 26 aufgezählten Charakteristika für die erfolgversprechendsten Anwendungsgebiete der verschiedenen Methoden.

| **Charakteristik** | **Agile Methoden** | **Traditionelle Methoden** |
|---|---|---|
| **Hauptziel** | Schnelle Erzeugung von Nutzen, gute Reaktion auf Veränderungen. | Berechenbarkeit, Stabilität, Hohe Sicherheit. |
| **Größe** | Kleine Teams und Projekte. | Große Teams und Projekte. |
| **Umgebung** | Turbulent, viele Veränderungen, Fokus auf das Projekt. | Stabil, wenige Veränderungen, Fokus auf Projekt und Organisation. |
| **Kundenbeziehungen** | Kunde vor Ort, Fokus auf priorisierte Inkremente. | Interaktion mit Kunden nach Bedarf, Fokus auf Vertrag. |
| **Planung und Kontrolle** | Verinnerlichte Pläne, qualitative Kontrolle. | Dokumentierte Pläne, quantitative Kontrolle. |
| **Kommunikation** | Implizites Wissen, das zwischen den Mitarbeitern ausgetauscht wird. | Explizit dokumentiertes Wissen. |
| **Anforderungen** | Priorisierte informelle Stories (siehe z.B. 2.2.2) und Testfälle, welche unvorhersehbaren Veränderungen ausgesetzt sind. | Formalisierte Anforderungen die vorhersehbar entwickelt werden. |
| **Entwicklung** | Einfache Gestaltung, kleine Entwicklungsschritte; Refactoring wird als kostengünstig betrachtet. | Aufwendige Gestaltung, größere Entwicklungsschritte; Refactoring wird als kostspielig betrachtet. |
| **Testen** | Ausführbare Testfälle definieren die Anforderungen. | Dokumentierte Testpläne und Prozeduren. |
| **Kunden** | Zugeordnete, qualifizierte Kunden-vertreter sind vor Ort. | Qualifizierte Kundenvertreter sind nicht immer vor Ort. |
| **Entwickler** | Mindestens 30% hochqualifizierte Vollzeit-Mitarbeiter, keine schlecht qualifizierten. | 50% hochqualifizierte Mitarbeiter zu Beginn, danach 10%; 30% weniger qualifizierte Mitarbeiter sind verkraftbar. |
| **Kultur** | Mitarbeiter arbeiten erfolgreicher bei vielen Freiheitsgraden. | Mitarbeiter arbeiten erfolgreicher in einem Rahmenwerk von Richtlinien und Prozeduren. |

**Abb. 26: Anwendungsgebiete-Vergleich der SW-Entwicklungsmethoden (Boehm, et al., 2006 S. 51f)**

Kruchten betrachtet die Kontextfaktoren getrennt für die Unternehmung (Umgebungsfaktoren) und das Projekt. Auf Unternehmungsebene sieht er folgende Faktoren (Kruchten, 2010 S. 6.3f):

- **Geschäftsbereich (Branche)**: Was sind die Charakteristika der Branche bzw. ihrer Produkte? Software für Internetanwendungen kann z.B. viel eher agil entwickelt werden als Software für Luft- und Raumfahrt.
- **Stückzahl**: Wird ein System in Einzelfertigung entwickelt oder soll es in Serie produziert werden?
- **Reife der Organisation**: Wie viel Erfahrung hat die Unternehmung in der Entwicklung von Software? Wie reif sind die Prozesse und wie fähig die Mitarbeiter?
- **Innovationslevel**: Werden neue Ideen und Technologien entwickelt oder arbeitet man eher in traditionellen Marktsegmenten?
- **Kultur**: Sowohl Unternehmungskultur als auch nationale Kultur spielen eine Rolle für die Akzeptanz und den Erfolg agiler Praktiken.

Betrachtet man ein konkretes Projekt, so sind es laut Kruchten folgende Faktoren, die den richtigen Kontext für agile Methoden definieren sollen (Kruchten, 2010 S. 6.4):

- **Größe:** Wie vorher schon beschrieben.
- **Gleichbleibende Architektur:** Auch für agile Methoden ist eine früh definierte und gleichbleibende Systemarchitektur vorteilhaft.
- **Geschäftsmodell:** Interne Entwicklung (agile Methoden sehr gut geeignet), kommerzielles Produkt, Auftragsentwicklung, Open-Source Software, etc.
- **Teamverteilung:** Wie viele Teams sind beteiligt? Wie weit sind die Mitarbeiter voneinander entfernt? Bei agilen Methoden sollen Mitarbeiter am gleichen Ort arbeiten.
- **Änderungsrate:** Welchen Änderungen ist das Marktumfeld unterworfen? Welche Risiken gibt es? Agile Methoden für hohe Änderungsraten.
- **Lebenszyklus des Systems:** Soll ein neues System entworfen oder ein bestehendes weiterentwickelt werden? Agile Methoden für neue Systeme.
- **Kritikalität:** Wie vorher schon beschrieben.
- **Projektsteuerung:** Was sind Start- und Abbruchsbedingungen für Projekte, wer entscheidet bei Fehlschlägen? Agile Methoden benötigen einfache Regeln.

Highsmith fasst die von mehreren Autoren genannten Faktoren zu 3 Gruppen zusammen (Highsmith, 2010 S. 323ff):

- **Projektfaktoren (Komplexität und Unsicherheit)**: Eine hohe Komplexität im Projekt verlangt nach traditionellen Methoden, hohe Unsicherheit nach agilen Methoden. Der Konflikt in der Wahl der Methoden ist demnach bei Projekten mit hoher Komplexität und großen Unsicherheiten am größten.
- **Kulturelle Faktoren**: Hier verweist Highsmith auf eine ältere Quelle (Highsmith, 2002 S. 323ff) in der er Kulturen, die auf Kompetenzen und Zusammenarbeit aufbauen, als Anwendungsgebiete für Agilität sieht. Kulturen welche auf Kontrolle aufbauen, sieht er als Anwendungsgebiet für traditionelle Methoden. Zusätzlich nennt er auch noch Kulturen, die auf der Kreativität von Einzelpersonen aufbauen und wo wenig Zusammenarbeit vorherrscht. Dafür hält er keine Methode für geeignet, sondern eher eine ad-hoc Entwicklung.

- **Faktoren der Unternehmungsführung und Richtlinieneinhaltung:** Der Einsatz von agilen Methoden kann erschwert sein, wenn mehr Wert auf Kontrolle und Konformität, als auf die eigentliche Entwicklungstätigkeit und deren Ergebnisse gelegt wird.

Außerdem sieht Highsmith in den **Marktphasen** ein Kriterium für die Anwendungsgebiete der unterschiedlichen Methoden. In frühen Phasen mit hohem Forschungsaufwand wird kaum eine Methode benötigt und oft ad-hoc entwickelt. In turbulenten Phasen mit großem Marktwachstum sieht er das Anwendungsgebiet der agilen Methoden und in der Marktsättigungsphase das Anwendungsgebiet traditioneller Methoden (Highsmith, 2002 S. 330).

Wilson und Mooz betrachten agile Methoden als anwendbar, wenn (Wilson, et al., 2003 S. 794):

- die **Komplexität** der Schnittstellen gering ist,
- die Effektivität der **Kommunikation** hoch ist,
- die **Anzahl der Stakeholder** gering ist und
- die **Mitarbeiter** hoch qualifiziert sind.

Die Autoren widersprechen sich in ihren Aussagen also kaum. In den verschiedenen Zusammensetzungen der Faktorenlisten kann man lediglich eine unterschiedliche Prioritätenbildung erkennen. Im Allgemeinen kann man aber ein recht homogenes Bild für das Anwendungsgebiet agiler Methoden in der Softwareentwicklung zeichnen. Dieses kann ungefähr beschrieben werden mit kleinen, nicht sicherheitskritischen Projekten, in denen hochqualifizierte Mitarbeiter, im Team und mit dem Kunden, gut zusammenarbeiten um innovative, aber nicht zu komplexe Produkte zu entwickeln, deren Anforderungen unsicher sind bzw. vielen Änderungen während der Entwicklung unterliegen.

Dieses Bild bezieht sich aber lediglich auf die Anwendbarkeit der in Kapitel 2.2 beschriebenen Softwareentwicklungsmethoden, wie z.B. Scrum, Extreme Programming oder Crystal. Diese sind jeweils Sammlungen zahlreicher, teilweise ineinandergreifender Prinzipien und Praktiken. Wie in Kapitel 5 aufgezeigt, können viele Prinzipien und Praktiken nur unter gewissen Voraussetzungen angewandt werden. Eine Methode als Sammlung von zusammenhängenden Prinzipien und Praktiken kann demnach nur angewandt werden, wenn die Voraussetzungen für sämtliche Prinzipien und Praktiken erfüllt sind. Dies scheint der Grund dafür zu sein, welcher das Anwendungsgebiet der agilen Methoden so weit einschränkt.

Andererseits gibt es aber auch Faktoren, die nicht oder kaum beschränkend wirken können. Z.B. wurde eine hohe Änderungsrate als Grund für die Anwendung von agilen Methoden genannt, eine niedrige Änderungsrate kann aber kaum ein Hindernis für die Anwendung agiler Methoden darstellen. Es kann also unterschieden werden, zwischen Faktoren die einen Bedarf an Agilität begründen können und solchen, die Voraussetzungen für die Anwendung agiler Methoden darstellen.

## 6.2 Kontextfaktoren für Agile Systems Engineering

Bedarf und Voraussetzungen für Agilität sind auch Fragestellungen, die für den ASE Ansatz beantwortet werden sollen. Dieser ist aber nicht mit einer Methode der agilen SW-Entwicklung vergleichbar und soll auch nicht auf eine spezielle Art von Projekten beschränkt sein. Er soll viel umfassender

Analysemöglichkeiten anbieten und allgemeine Möglichkeiten aufzeigen, die Agilität zu steigern, sollte dies im Systems Engineering erforderlich sein.

Die Kontextbeschreibung für diesen Ansatz geschieht deshalb unter einer anderen Prämisse, als die Faktorensuche für die Anwendungsgebiete der agilen SW-Entwicklungsmethoden. Sie soll einerseits einem Entwickler die Möglichkeit geben, zu erkennen, welche Probleme mit dem ASE Ansatz gelöst werden können. Damit kann er feststellen, ob er überhaupt einen Bedarf an Agilität hat und ob dieser Ansatz für ihn interessant ist. Andererseits soll er über die Machbarkeit der Empfehlungen des ASE Ansatzes informiert werden bzw. über die Voraussetzungen die dafür bestehen. Wie in Kapitel 5 schon gezeigt wurde, sind viele Prinzipien und Praktiken nur unter gewissen Bedingungen machbar. Nachdem der ASE Ansatz zwar die Auswahl von Praktiken und Prinzipien erleichtern soll, aber nicht durch eine genau definierte Sammlung davon repräsentiert wird (wie die agilen SW-Entwicklungsmethoden), gibt es auch keine Voraussetzungen für den ASE Ansatz als Ganzes. Der ASE Ansatz soll vielmehr aufgrund des spezifischen Bedarfs und der gegebenen Voraussetzungen selektiv Prinzipien und Praktiken anbieten, die erfolgversprechend erscheinen.

In der Einleitung wurde das Problemfeld, welches Agilität erforderlich macht, mit einem dynamischen Umfeld beschrieben, in dem es häufig zu Veränderungen kommt. Hier soll nun detaillierter darauf eingegangen werden, wann ein Bedarf für Agilität besteht bzw. welche Gründe für die Anwendung des ASE Ansatzes sprechen. Zur praktischen Umsetzung der Empfehlungen des ASE Ansatzes, werden dann je nach Bedarf, eine oder mehrere Praktiken oder Prinzipien aus Kapitel 5 empfohlen. Wie erwähnt unterliegen diese aber gewissen Voraussetzungen, welche damit auch einen Einfluss auf den richtigen Kontext für ASE haben.

### 6.2.1 Empirische Erhebungen zu Voraussetzungen für Agilität

Als erster Schritt sollen nun diese Voraussetzungen dargestellt werden, welche für die Anwendung der agilen Praktiken und Prinzipien im SE in der empirischen Studie gefunden wurden. Diese wurden zumeist schon in Kapitel 5 erwähnt. Hier sollen sie zusammengefasst, kategorisiert und benannt werden. Einige davon korrelieren mit Kontextfaktoren der agilen SW-Entwicklung. Andere wurden ausschließlich in den empirischen Erhebungen gefunden und können deshalb als Voraussetzungen speziell für die agile Entwicklung von Hardware betrachtet werden. Es wird jeweils auf Prinzipien und Praktiken verwiesen, für welche diese Voraussetzungen gelten können. Die Betonung liegt hier aber auf dem Wort „können“ da dies nur allgemeine Aussagen sind und für jeden praktischen Einzelfall zu überprüfen sind.

***Art des Systems***

Da hier keine Systeme betrachtet werden, die nur aus Software bestehen, ist die Art bzw. Beschaffenheit des Systems wohl als größter Unterschied zur agilen Softwareentwicklung zu betrachten und dieser hat auch einen großen Einfluss auf die Machbarkeit von agilen Methoden. Die Gesprächspartner in der empirischen Studie haben immer wieder auf die Unterschiede in der Entwicklung von Hardware und Software hingewiesen. Eine geeignete Art des Systems kann als Voraussetzung für das Prinzip der Entwicklung in kleinen Schritten, aber auch vieler damit zusammenhängender Prinzipien und Praktiken gesehen werden. Im Vergleich zur Softwareentwicklung ist diese Voraussetzung viel schwieriger zu erfüllen, da Hardware auch materiell hergestellt werden muss. Je nach Ausprägungsform der Entwicklung in kleinen Schritten gibt es unterschiedliche Voraussetzungen. Statt einer Entwicklung, bei

der die Zwischenergebnisse der Hardware schon Teile des späteren Systems sein sollen, ist bei Serienfertigung nur eine Herstellung von Prototypen möglich, die in der Regel nicht identisch mit den späteren Serienprodukten sind. Sollen die Zwischenergebnisse bereits Teile des späteren Systems sein, ist die Anpassbarkeit früh realisierter Komponenten ein Kriterium, damit spätere Entwicklungserkenntnisse noch zu Änderungen dieser führen können. Bei Serienfertigung ist hingegen die Anpassbarkeit der Produktionslinie zu berücksichtigen, sollte die Planung dieser, schon während der Produktentwicklungsphase starten. Egal ob die Zwischenergebnisse Teil des späteren Systems sein sollen oder nicht, ist auch die Frage nach ihrer Nutzbarkeit zu stellen. Nicht bei jeder Art von System können Teile davon dem Kunden zum Test übergeben werden. Bei Hardware-Komponenten sind auch Herstelldauern und –kosten ein Kriterium, ebenso wie die Kosten und Zeitdauern der Testläufe. Der Komplexitätsgrad des Systems kann auch Einfluss auf die umsetzbaren Schrittdauern haben. Bei hoher Komplexität sind lange Analyse- und Planungsphasen zu erwarten, bevor man Teile umsetzen und testen kann. Bei folgenden Prinzipien und Praktiken ist beispielweise zu überprüfen, ob die Art des Systems eine Entwicklung in der erforderlichen Ausprägungsform zulässt, damit die Praktik sinnvoll umgesetzt werden kann:

- AEZ (4.3.4).
- Flexibles Anforderungsmanagement (5.1.4).
- Entwicklung in kleinen Schritten (5.4.3).
- Kontinuierliche Integration (5.4.4).
- Test-Driven Development (5.4.6).
- Prototyping/Simulation (5.4.7).
- Emerging Architecture (5.4.16).

***Kundeneinbindung***

Bei den Prinzipien, welche das Feedback des Kunden verwenden, um die weitere Richtung der Entwicklung zu bestimmen, muss dieser auch bereit sein, dementsprechend mitzuwirken. Das ist laut Aussagen der Interviewpartner aber nicht immer der Fall, vor allem wenn er die anfänglichen Anforderungen als ausreichend und weiteres Feedback nur als unnützen Aufwand ansieht. Prinzipien und Praktiken, für welche die Kundeneinbindung als Voraussetzung gelten kann, sind z.B.:

- AEZ (4.3.4)
- Fokus auf tatsächlichen Kundennutzen (5.1.1).
- Vertrauensvolle Zusammenarbeit statt übertriebene Vertragsverhandlungen (5.1.2).
- Kunde in Entwicklung einbinden (Feedback) (5.1.3).
- Priorisierung (5.1.5).
- Entwicklung in kleinen Schritten (je nach gewünschter Ausprägungsform) (5.4.3).
- Systemfeedback (5.5.8).

***Kritikalität***

Für die Entwicklung sicherheitskritischer Produkte gibt es spezielle Normen und Standards (z.B. IEC 61508), die dafür sorgen sollen, dass sicherheitskritische Produktanforderungen zuverlässig und nachvollziehbar entwickelt werden. Der flexible Umgang mit Anforderungen und die reduzierte

Dokumentation bei manchen agilen Prinzipien und Praktiken sind u.U. damit nicht in Einklang zu bringen. Das kann z.B. gelten für:

- AEZ (4.3.4).
- Flexibles Anforderungsmanagement (5.1.4).
- Wertschätzung von implizitem Wissen (Tacit Knowledge) (5.3.6).
- Improvisation (5.3.8).
- Prozessgestaltung durch Team (5.4.1).
- Entwicklung in kleinen Schritten (5.4.3).
- Flexible Prozessgestaltung (5.4.13).
- Beschränkung auf notwendige Dokumentation (5.4.14).
- Emerging Architecture (5.4.16).
- Einfachheit (5.5.1).
- Anpassbarkeit (5.5.2).

***Kultur***

Die agilen Prinzipien und Praktiken können teilweise nicht nur als ungewöhnlich im Vergleich zu traditionellen Methoden betrachtet werden, sondern erfordern auch spezielle Verhaltensweisen der beteiligten Personen auf Entwickler- und Kundenseite. Z.B. fordern einige Praktiken von den Entwicklern mehr Verantwortung zu übernehmen oder einen besonders offenen, intensiven und auf Respekt und Zusammenarbeit aufbauenden Umgang miteinander. Eine geeignete Unternehmungskultur (oder auch nationale Kultur) kann deshalb Voraussetzung von z.B. folgenden Prinzipien sein:

- Team Selbstorganisation (5.2.4).
- Pair Programming (5.2.6).
- Mensch im Mittelpunkt der Entwicklung (5.3.1).
- Wertschätzung von implizitem Wissen (Tacit Knowledge) (5.3.6).
- Positive Einstellung gegenüber Veränderungen (5.3.7).
- Prozessgestaltung durch Team (5.4.1).
- Timeboxing (5.4.8).
- Tägliche/häufige Meetings (5.4.12).
- Product Team Ownership (5.4.15).
- Set-Based Design (5.4.18).

***Mitarbeiterqualifikation***

Agile Prinzipien stellen das Wissen und Können der Entwickler über den Prozess. Sie legen viel Verantwortung in die Hände der Entwickler und lassen ihnen viele Freiheiten. Deshalb ist die Qualifikation der Mitarbeiter mehr noch als bei traditionellen Entwicklungsmethoden als Erfolgskriterium zu sehen, z.B. bei:

- Team Selbstorganisation (5.2.4).
- Wertschätzung von implizitem Wissen (Tacit Knowledge) (5.3.6).
- Improvisation (5.3.8).
- Prozessgestaltung durch Team (5.4.1).
- Beschränkung auf notwendige Dokumentation (5.4.14).

- Product Team Ownership (5.4.15).
- Set-Based Design (5.4.18).

***Teamgröße und Verteilung***

Einige Praktiken beruhen auf enger Zusammenarbeit und schneller und direkter Kommunikation. So ist weniger Dokumentation notwendig, es können höhere Entwicklungsgeschwindigkeiten erreicht werden und das Team kann durch intensivierten Gedankenaustausch zu einem besseren Verständnis des Systems gelangen. Speziell bei folgenden Prinzipien und Praktiken kann das als Voraussetzung gelten, damit diese sinnvoll umsetzbar sind:

- Örtliche Zusammenlegung des Teams (5.2.2).
- Team Selbstorganisation (5.2.4).
- Wertschätzung von implizitem Wissen (Tacit Knowledge) (5.3.6).
- Prozessgestaltung durch Team (5.4.1).
- Laufende Planung (5.4.9).
- Tägliche/häufige Meetings (5.4.12).
- Beschränkung auf notwendige Dokumentation (5.4.14).
- Product Team Ownership (5.4.15).

***Preisgestaltung***

In den empirischen Erhebungen wurden vor allem Fixpreisprojekte gefunden. Gewöhnlich werden dabei sowohl der Preis für die Entwicklungstätigkeit, als auch für das Produkt je Mengeneinheit (bei Serienfertigung) im Vorhinein vereinbart (auch wenn das bei Änderungen zur Nachverhandlung des Preises führen kann). Einige agile Prinzipien und Praktiken rechnen aber prinzipiell mit Änderungen und legen keinen Wert auf exakte Definition der Anforderungen zu Projektbeginn. Das kann wiederum Probleme bei der Vereinbarung eines Fixpreises verursachen. Deshalb ist z.B. bei folgenden Prinzipien und Praktiken die Preisgestaltung zu bedenken:

- AEZ (4.3.4).
- Fokus auf tatsächlichen Kundennutzen (5.1.1).
- Vertrauensvolle Zusammenarbeit statt übertriebene Vertragsverhandlungen (5.1.2).
- Flexibles Anforderungsmanagement (5.1.4).
- Entwicklung in kleinen Schritten (5.4.3).
- Emerging Architecture (5.4.16).

***Wirtschaftlichkeit***

In der Charakterisierung des ASE Ansatzes dieser Arbeit wurden Effizienz und Effektivität berücksichtigt, da diese auch in dynamischen Wettbewerbsumfeldern als Kriterien gelten. Einzelne agile Prinzipien und Praktiken, die z.B. für mehr Flexibilität oder Situationskenntnis sorgen sollen, können im Projektkontext aber durchaus einen negativen Einfluss auf Effizienz oder Effektivität und damit auf die Wirtschaftlichkeit haben. Z.B. bei den folgenden Prinzipien ist dieser mögliche Zielkonflikt zu berücksichtigen:

- Gleichzeitige Mitarbeit an möglichst wenigen Projekten (5.2.5).
- Pair Programming (5.2.6).
- Prozessgestaltung durch Team (5.4.1).

- Prototyping/Simulation (5.4.7).
- Pufferzeiten (5.4.10).
- Tägliche/häufige Meetings (5.4.12).
- Entwicklung von Varianten (5.4.17).
- Set-Based Design (5.4.18).
- Anpassbarkeit (5.5.2).
- Reserven im Potential des Produkts (5.5.4).

Damit wurden die wesentlichen Faktoren zusammengefasst, die von den Gesprächspartnern in der empirischen Studie als Voraussetzungen für agile Prinzipien und Praktiken genannt wurden. Auch wurden diese Faktoren als Gründe dafür genannt, warum agile Praktiken in der Systementwicklung (noch) nicht eingesetzt werden.

Man muss diese Faktoren aber nicht streng als Voraussetzungen betrachten. Zumindest sollte man sie, bei einem Versuch die agilen Praktiken im SE anzuwenden, aber als Risiken ansehen. Wie man unter Umständen diese Risiken verringern kann, wird in 6.4 beschrieben.

### 6.2.2 Empirische Erhebungen zum Bedarf an Agilität

Bevor Faktoren für den Bedarf an Agilität aufgezeigt werden, soll auf das in der Einleitung dargestellte Problemfeld aus Unsicherheiten, Veränderungen, Komplexität und kurzen Entwicklungszeiten verwiesen werden. Jeder Systementwickler, der in diesem Umfeld erfolgreich Systeme entwickeln will, hat demnach Bedarf an Agilität. Laut der Definition dieser Arbeit sind aber die Veränderungen das entscheidende Kriterium, welches nach einer Steigerung der Agilität verlangt. Komplexität, kurze Entwicklungszeiten und Unsicherheiten im Markt, bei Technologien oder im Systemverhalten werden als Ursachen für Veränderungen gesehen. Einen Bedarf an Agilität hat deshalb genaugenommen ein Systementwickler, der seinen Erfolg steigern will und der Veränderungen als wesentliches Kriterium erkannt hat, das seinen Erfolg beschränkt oder mindert. Dies stellt den grundlegenden Bedarfsfall dar, für den der ASE Ansatz entwickelt wurde.

Andere Vorteile, die häufig für agile Softwareentwicklungsmethoden genannt werden, wie zufriedenerer Mitarbeiter, werden in dieser Arbeit nur als Zusatznutzen gesehen bzw. als Nutzen der nur in indirektem Zusammenhang mit Agilität steht. Damit ist gemeint, dass z.B. die Mitarbeiterzufriedenheit durch Einsatz agiler statt traditioneller Methoden steigen kann oder auch, dass auf Mitarbeiterzufriedenheit spezieller Wert gelegt wird, weil erwartet wird, dass diese dann agiler handeln. Ist aber die Erhöhung der Mitarbeiterzufriedenheit das primäre Ziel, so ist nicht die Agilität als unmittelbare Lösungsmöglichkeit zu sehen. Damit stellt eine gewünschte Steigerung der Mitarbeiterzufriedenheit auch keinen Bedarfsgrund für Agilität als Fähigkeit oder die Anwendung des ASE Ansatzes dar. Wohl aber kann die Mitarbeiterzufriedenheit ein Bedarfsgrund für eine einzelne (agile) Praktik sein.

Um das in der Einleitung beschriebene Problemfeld genauer darzustellen, werden nun beispielhafte Änderungen genannt, welche in den empirischen Erhebungen und sonstigen Gesprächen gefunden wurden:

- Der Kunde ändert während der Entwicklung die Produktanforderungen, weil sich seine Bedürfnisse geändert haben.

- Marktvolumina ändern sich, Bedürfnisse am Markt für bestimmte Produkte entstehen, verschwinden oder ändern sich.
- Aufgrund der Komplexität eines Produkts sind dem Kunden die Produktanforderungen zu Projektbeginn nicht klar und er ändert sie bzw. definiert sie genauer, nachdem er Zwischenergebnisse der Entwicklung begutachtet hat.
- Es ist eine Änderungen des Produktionsverfahrens (und damit auch von Designparametern des Produkts) notwendig, als indirekte Folge einer Anforderungsänderung (z.B. Stückzahlerhöhung).
- Ein Bauteillieferant geht in Konkurs. Das Ersatzbauteil hat unterschiedliche Spezifikationen und macht Änderungen am System notwendig.
- Konkurrenzdruck und der Wunsch nach schneller Markteinführung führen zu einem Start des Entwicklungsprojekts, obwohl die Produktanforderungen noch unklar sind. Im Laufe der Entwicklung werden diese klarer und weichen von den anfänglichen Annahmen ab.
- Neue Gesetze verhindern den Einsatz eines Systems oder lassen ein anderes interessanter erscheinen.
- Während der Entwicklung machen neue Vorschriften Änderungen an Produkt oder Prozess notwendig. Z.B. kann eine neue Recycling-Richtline den Einsatz von bestimmten Materialien verhindern oder eine neue Sicherheitsrichtlinie in einer Prozessänderung münden.
- Neue Technologien werden verfügbar. Diese können Änderungen am Produkt erwünscht machen oder dieses auch obsolet werden lassen. Eventuell entstehen dadurch auch neue Bedürfnisse am Markt.
- Der Innovationsgrad des Produkts ist sehr hoch oder es besteht wenig Erfahrung mit einer verwendeten Technologie. Änderungen sind aufgrund des anfänglich geringen Wissensstandes wahrscheinlich.
- Ressourcen werden knapp bzw. Marktpreise für Ressourcen ändern sich und machen aus wirtschaftlichen Gründen Änderungen am Produkt notwendig.
- Anfängliche Fehler oder schlechte Koordination in der Entwicklung verlangen nach Änderungen zur Korrektur.
- Mitarbeiter verlassen das Team, neue kommen hinzu. Änderungen in der Organisation werden notwendig.
- Der Kunde (Auftraggeber oder Anwender) ändert sich. Der neue Kunde verlangt nach Änderungen am Produkt.
- Der Eigentümer der Unternehmung ändert sich und verlangt nach Änderungen an der Organisation.
- Etc.

Auch wenn der ASE Ansatz vermutlich zu allen Arten von Veränderungen gewisse Hilfestellungen anbieten kann, ist er natürlich auf bestimmte Veränderungen fokussiert. In Abb. 27 wird dieser Fokus dargestellt, der notwendig war, um den Inhalt dieser Dissertation nicht ausufern zu lassen. Ein besonderer Erklärungsbedarf besteht dabei vor allem für den „Mechanismus des Auftretens“:

Wie schon für den AEZ skizziert (Abb. 18), können die Veränderungen hinsichtlich ihres Auftretens unterschieden werden, was in Abb. 27 mit „Mechanismus des Auftretens“ bezeichnet wird. Unter Veränderungen innerhalb der Entwicklung sind solche zu verstehen, die direkt durch die Entwicklungstätigkeit ausgelöst werden, z.B. durch neue Erkenntnisse über das Systemverhalten, die

eine Änderung der Produktanforderungen erforderlich machen. Beispielsweise wird im AEZ durch frühes und häufiges Testen von realisierten Entwicklungsschritten und Einholung von Kundenfeedback ein frühes Erkennen dieser Veränderungen forciert. Solche Veränderungen können als Teil der gewöhnlichen Entwicklungstätigkeit gesehen werden, um von den ursprünglich formulierten Produktanforderungen zu einem funktionstüchtigen und nutzbaren System zu gelangen.

Veränderungen außerhalb der Entwicklung entstehen unabhängig davon, z.B. sind dies unerwartete Anforderungsänderungen des Kunden, die nichts mit seinen ursprünglichen Wünschen zu tun haben, veränderte Absatzmöglichkeiten oder Gesetzesänderungen (siehe auch Haberfellner et al. (2002 S. 89f)).

Damit können Veränderungen innerhalb und außerhalb der Entwicklung als grundverschieden hinsichtlich ihrer Ursache und des Mechanismus ihres Auftretens bezeichnet werden. Der ASE Ansatz soll beide Arten berücksichtigen, hinsichtlich eines effizienten und effektiven Umgangs damit und der dafür notwendigen Flexibilität. Hinsichtlich der Erkennung von Veränderungen ist er aber auf die Veränderungen innerhalb der Entwicklung fokussiert und berücksichtigt solche außerhalb der Entwicklung kaum (dafür wären z.B. Marktstudien, Vorhersagetechniken u.Ä. notwendig).

| **Kategorie** | **Mögliche Ausprägungen** | **Fokus des ASE Ansatzes** |
|---|---|---|
| Zeitpunkt der Veränderung | Alle Lebenszyklusphasen des Systems. | Die Zeitdauer des Entwicklungsprojekts; Veränderungen die z.B. in Marktstudien davor oder in der Anwendungsphase danach auftreten, werden kaum berücksichtigt. |
| Chance/Gefahr | Chance und/oder Gefahr. | Hauptsächlich werden Gefahren für den Erfolg der Entwicklung betrachtet aber auch Chancen die während des Entwicklungsprojekts noch wahrgenommen werden können. |
| Verursacher der Veränderung | Kunde, Anwender, Markt, Gesetzgeber, Lieferanten, Partner, Mitarbeiter, Eigentümer, Management, Gesellschaft, etc. | Alle; in den empirischen Erhebungen wurde aber zumeist über vom Kunden verursachte Veränderungen gesprochen. |
| Auswirkung auf | Produkt, Vorgehensweise (Entwicklungsprozess), Organisation, Mitarbeiter, Fertigung, andere Produkte, Prozesse, Abteilungen, etc. | Spezieller Fokus auf Änderungen am Produkt aber auch in der Vorgehensweise, Organisation und bei den Mitarbeitern. |
| Vorhersehbarkeit | Vorhersehbar oder nicht vorhersehbar hinsichtlich des Zeitpunkts, des Inhalts oder des Verständnisses der Auswirkung der Veränderung. | Alle möglichen Arten bei denen mindestens ein Parameter unvorhersehbar ist. Ist alles vorhersehbar, benötigt man keine Agilität sondern nur einen guten Plan. |
| Mechanismus des Auftretens | Außerhalb oder innerhalb der Entwicklung. | Beide; das Erkennen der Veränderungen wird aber nur innerhalb der Entwicklung betrachtet. |

**Abb. 27: Kategorisierung von Veränderungen und Fokus des ASE Ansatzes**

Innerhalb dieses Fokus bzw. dieser Grenzen sollen nun die zuvor genannten Beispiele für Veränderungen zu Faktoren zusammengefasst werden, die den Bedarf an Agilität im SE beeinflussen. Dabei wurde versucht, keine unnötigen Kategorien zu erschaffen, sondern die Faktoren nur so weit zu untergliedern, wie dies für die Bestimmung des Bedarfs notwendig ist.

***Veränderungen außerhalb der Entwicklung***

Häufige und für den Projekterfolg relevante Veränderungen außerhalb der Entwicklung bedingen generell einen Bedarf an Agilität. Der ASE Ansatz dieser Arbeit soll Hilfestellungen bieten, diesen Bedarf innerhalb der in Abb. 27 aufgezeigten Grenzen abzudecken.

***Produktspezifikationen***

Werden die Anforderungen vom Kunden zu Projektbeginn nur ungenau oder unvollständig definiert, bzw. werden vom Entwickler solche unzulänglichen Anforderungen akzeptiert, besteht ebenfalls ein Bedarf an Agilität. Genauso muss berücksichtigt werden, dass die niedergeschriebenen Anforderungen in ihrem Wortlaut nicht zu 100% mit den tatsächlichen Anforderungen (Kundenwünschen) übereinstimmen können, wie in 5.1.1 beschrieben wurde. Der ASE Ansatz soll helfen, mit den dadurch zu erwartenden Änderungen besser umgehen zu können. Diese treten vermutlich auf, wenn der Kunde mit Entwicklungsergebnissen konfrontiert wird und sind deshalb als innere Veränderungen zu betrachten.

***Art des Systems***

In der Art des Systems (speziell in Neuartigkeit und Komplexität) können ebenso unzulänglich formulierte Anforderungen begründet sein. Dies liegt aber in der Natur der Sache und kann weder Kunden noch Entwicklern angelastet werden. Dabei ist es einfach nicht möglich das Systemverhalten exakt vorherzusagen und Änderungen können auftreten, wenn im Laufe der Entwicklung mehr Informationen dazu verfügbar werden. Diese sind also ebenso Veränderungen innerhalb der Entwicklung.

Es soll aber darauf hingewiesen werden, dass der ASE Ansatz keine allgemeine Lösung für neuartige oder komplexe Produkte darstellt, sondern nur damit in Zusammenhang stehende Veränderungen behandelt.

### 6.2.3 Bedarf und Voraussetzungen für Agile Systems Engineering

Die Forschungsfragen 3 und 4 werden nun in zusammengefasster Form in Abb. 28 beantwortet, in der die Grundlagen aus dem Kontext der agilen Softwareentwicklungsmethoden erweitert werden, um die Erkenntnisse aus den empirischen Erhebungen, über die Faktoren für den Bedarf und die Voraussetzungen des ASE Ansatzes. Der Bedarf ist dabei hinsichtlich der Fähigkeit Agilität, wie sie in dieser Arbeit definiert wurde, zu sehen. Die Voraussetzungen gelten immer nur für einzelne Prinzipien oder Praktiken.

Mit diesen Faktoren kann der richtige Kontext für den ASE Ansatz definiert werden. Einige der übergreifenden Kontextfaktoren der agilen Softwareentwicklungsmethoden wurden weggelassen, da sie sich ohnehin in anderen Faktoren wiederspiegeln (z.B. spiegelt sich der Geschäftsbereich in den äußeren Veränderungen, der Kritikalität oder in der Preisgestaltung wieder). Andere wurden zusammengefasst, weil sie ähnliche Auswirkungen und Zusammenhänge haben und dadurch auch immer gemeinsam zu betrachten sind. Wie man konkret geeignete Prinzipien und Praktiken je nach Anwendungsfall (Bedarf

und Voraussetzungen) auswählt, wird im Rahmen der agilen Vorgehensmethodik in Kapitel 7.4 dargestellt.

| **Kontextfaktor** | **Bedarf an Agilität im Allgemeinen** | **Voraussetzung für einzelne agile Praktiken und Prinzipien** |
|---|---|---|
| **Veränderungen** | Häufige Veränderungen außerhalb der Entwicklung. | |
| **Produktspezifikationen** (Anforderungen an das Produkt) | Ungenaue, unvollständige oder mehrdeutige Produktspezifikationen zu Projektbeginn. | |
| **Art des Systems** | Hohe Komplexität und hoher Neuigkeitsgrad des Produkts. | Anpassbarkeit und niedrige Komplexität des Systems und Machbarkeit, Nutzbarkeit und Testbarkeit von Zwischenergebnissen. |
| **Kundeneinbindung** | | Ein Kunde, der zur Einbindung in die Entwicklung und zum Geben von Feedback gewillt ist. |
| **Kritikalität** | | Ein niedriges Sicherheitsrisiko des Produkts, sonst können umzusetzende Prozessvorschriften (Dokumentation) Agilität verhindern. |
| **Kultur** | (Kann auch Bedarfsgrund sein, wenn kulturelle Probleme im Umgang mit Veränderungen bestehen.) | Eine geeignete Kultur in der Unternehmung (z.B. hinsichtlich guter Zusammenarbeit, der Übernahme von Verantwortung oder dem Umgang mit vielen Freiheitsgraden). |
| **Mitarbeiterqualifikation** | | Eine besonders hohe Qualifikation der Mitarbeiter. |
| **Teamgröße und Verteilung** (Kommunikation) | | Kleine Teams und örtliche Nähe der Mitglieder bzw. eine sehr effiziente Kommunikation. |
| **Preisgestaltung** | | Eine flexible Preisgestaltung (kein Fixpreisprojekt). |
| **Wirtschaftlichkeit** | (Eine gewünschte Steigerung von Effizienz und Effektivität im Umgang mit Veränderungen kann auch als Bedarfsgrund gesehen werden.) | Möglichkeit zur effizienten und effektiven Umsetzung im Projektkontext (wesentlich durch die anderen Faktoren beeinflusst). |

**Abb. 28: Kontextfaktoren für Agile Systems Engineering**

## 6.3 Zusätzliche Bedarfsgründe für agile Praktiken

In dieser Arbeit wird Agilität speziell unter dem Gesichtspunkt einer Verbesserung im Umgang mit Veränderungen betrachtet, worunter ihre Hauptfunktion gesehen wird. Es gibt aber durchaus auch Bedarfsgründe für agile Prinzipien und Praktiken, die nicht direkt im Zusammenhang mit Veränderungen stehen. Der Nutzen, den agilen Prinzipien und Praktiken darstellen können und welcher nicht direkt in Zusammenhang mit Veränderungen steht, soll im Weiteren als Zusatznutzen bezeichnet werden. Möglicher zusätzlicher Nutzen wurde in den jeweiligen Unterkapiteln von Kapitel 5 schon beschrieben, hier sollen noch einmal die wesentlichsten Punkte zusammengefasst werden, die bei agilen Praktiken häufiger anzutreffen sind:

- **Kundenzufriedenheit**: Die Kundenzufriedenheit kann z.B. erhöht werden durch Verfolgung des tatsächlichen Kundennutzens, eine frühe Bereitstellung von Kundennutzen oder bessere Einhaltung von Terminen.
- **Mitarbeiterzufriedenheit**: Durch die gezielte Förderung der Mitarbeiter zu Agilitätszwecken kann auch ihre Zufriedenheit erhöht werden. Ihre Motivation kann z.B. durch die Übertragung von mehr Verantwortung gesteigert werden. Mit einem Wechsel von einer traditionellen zu einer agilen Vorgehensweise kann auch die Mitarbeiterzufriedenheit gesteigert werden, wenn die Unternehmungskultur besser dazu passt.
- **Zusammenarbeit:** Hiermit ist eine Verbesserung der Zusammenarbeit sowohl im Entwicklungsteam als auch zwischen Entwicklern und Kunden gemeint.
- **Risikoreduzierung:** Speziell durch das Entwickeln in kleinen Schritten kann eine Reduzierung der Projektrisiken durchgeführt werden, weil jeweils nach der Entwicklung eines kleinen Teils des Entwicklungsumfangs eine Überprüfung durchgeführt wird. Es ist aber auch jede andere Reduzierung von Projekt- oder Produktrisiken gemeint.

Bei der Auswahl von agilen Praktiken im Rahmen der ASE Methodik soll der mögliche Zusatznutzen ebenso berücksichtigt werden, siehe Kapitel 7.4.

## 6.4 Möglichkeiten bei nicht erfüllten Voraussetzungen

Wie bereits beschrieben, können zahlreiche Voraussetzungen für den Einsatz von agilen Praktiken bestehen. Unter Umständen stellen diese Voraussetzungen im jeweiligen Anwendungsfall aber nicht absolute Ausschlussgründe dar, sondern können umgangen werden. Die Voraussetzungen sind dann als Risiken zu betrachten.

Es sollen hier Lösungsansätze für die Erfüllung/Umgehung von Voraussetzungen bzw. für die Reduzierung der Risiken in der Anwendung agiler Praktiken aufgezeigt werden. Wenn in einer Unternehmung oder für ein Projekt ein Bedarf nach einer höheren Agilität vorhanden ist, die Voraussetzungen für dementsprechende Maßnahmen aber nicht gegeben sind, dann sind drei grundsätzliche Strategien denkbar:

- **Erfüllen der Voraussetzungen**: Es kann versucht werden, die Voraussetzungen für die erforderliche Praktik zu schaffen. Dies ist natürlich nur bei bestimmten Voraussetzungsfaktoren denkbar. Z.B. kann versucht werden einen Kunden, der nicht zur Einbindung gewillt ist, zu

überreden. Oder es kann versucht werden (langfristig) einen Kulturwandel in der Unternehmung herbeizuführen, wenn das der Hindernisgrund für den Einsatz von agilen Praktiken war.

- **Anwendung alternativer Praktiken**: In einigen Fällen existiert eine alternative Praktik mit keinen/anderen Voraussetzungen mit welcher derselbe Nutzen erzielt bzw. derselbe Teilaspekt der Agilität gefördert werden kann. Die Suche und Auswahl geeigneter Praktiken je Bedarf und Voraussetzungen ist ein wesentlicher Inhalt der ASE Vorgehensmethodik und wird in Kapitel 7.4 genauer erklärt.
- **Unkonventionelle Anwendung der Praktiken**: Viele der bekannten Praktiken unterliegen spezifischen Voraussetzungen, da für sie ganz bestimmte Ausprägungsformen und strenge Regeln vorgesehen sind. In vielen Fällen kann aber auf diese Regeln verzichtet werden, die Praktik in einer anderen Ausprägungsform angewandt und dennoch ein Gewinn hinsichtlich Agilität erzielt werden. Nachdem das Ziel des ASE nicht die Anwendung einer bestimmten Methode, sondern die Steigerung der Agilität als Fähigkeit ist, sollen alle Ausprägungsformen von Praktiken betrachtet werden, welche die Agilität steigern können. Auch für die Gestaltung der agilen SE Vorgehensmodelle (7.5) werden die Praktiken in einer unkonventionellen Weise verwendet.

In den folgenden Unterkapiteln werden geordnet nach den Voraussetzungsfaktoren Lösungsmöglichkeiten im Sinne der eben genannten 3 Strategien besprochen. Dabei wird jeweils auf die wichtigsten Praktiken eingegangen, welche diesen Voraussetzungen unterliegen. Danach werden speziell für die Entwicklung in kleinen Schritten, als strukturgebendes Prinzip des AEZ, alternative Ausprägungsformen diskutiert. Nachdem der AEZ als das am weitesten verbreitete Verständnis agiler SW-Entwicklung identifiziert wurde, soll damit für dieses wichtige Prinzip versucht werden, eine allgemeine Machbarkeit im SE herzustellen. Die Zusammenstellung der Lösungsansätze ist nicht umfassend, weitere Möglichkeiten sind ebenso denkbar. Ebenso muss angemerkt werden, dass die Lösungsansätze natürlich immer nur unter bestimmten Umständen machbar sind, auch wenn darauf nicht immer eingegangen wird.

### 6.4.1 Art des Systems

**Allgemeine Voraussetzungen:** Anpassbarkeit und niedrige Komplexität des Systems und Machbarkeit, Nutzbarkeit und Testbarkeit von Zwischenergebnissen.

**Mögliche Probleme:** Die Art des Systems kann ein großes Hindernis für den Einsatz agiler Praktiken darstellen, vor allem wegen der oft langwierigen und kostspieligen Herstellung der Hardware-Komponenten und der schlechten Änderbarkeit dieser. Ebenso kann es schwierig sein Zwischenergebnisse herzustellen, welche für den Kunden einen Nutzen stiften können. Z.B. ist es fragwürdig wie Nutzen gestiftet werden soll, mit 2% eines Fahrzeuges oder einer Industrieanlage.

**Erfüllen der Voraussetzungen:** Die Art des Systems ist natürlich durch die Anforderungen des Kunden bestimmt und damit ist es oft kaum möglich die Voraussetzungen zu erfüllen, wenn die Systemart die erwünschte agile Praktik nicht zulässt. Eventuell kann aber mit den unter 5.5.1 und 5.5.2 beschriebenen Prinzipien versucht werden, das System so zu gestalten, dass Änderungen leichter machbar sind oder, dass es in kleineren Schritten entwickelt und hergestellt werden kann. Z.B. kann versucht werden, das System möglichst einfach und modular zu gestalten und möglichst viele Designparameter durch Software

abzubilden. Damit sollte es dann leichter möglich sein zumindest bestimmte Änderungen einzubringen. Mit den in 5.5.4 beschriebenen Prinzipien kann versucht werden, die Zwischenergebnisse so zu gestalten, dass es leichter möglich ist, diese auch nutzenbringend zu verwenden. Z.B. wenn Module so gestaltet werden, dass sie ihre Funktion unabhängig vom Gesamtsystem erbringen können.

Soll, wie in 5.5.5 und 5.5.6 beschrieben, die Agilität durch die Möglichkeit des schnellen Austauschs von Modulen bei Plattformstrategien gesteigert werden, sind die in 5.5.2 und 5.5.3 genannten Prinzipien ebenso Möglichkeiten zur Systemgestaltung, wenn zuvor keine Umsetzung der Plattformstrategie möglich erschien, aufgrund der schlechten Anpassbarkeit der Systemarchitektur.

**Alternative Praktiken:** Sollte der AEZ aufgrund der Art des Systems nicht in vollem Umfang angewandt werden können, so kann möglicherweise eine Entwicklung in kleinen Schritten nach anderen Ausprägungsformen verwendet werden. Auch das Set-Based Design kann als prinzipielle Alternative gesehen werden. Nachdem mit diesen Prinzipien die grundlegende Struktur des Entwicklungsprozesses festgelegt wird, wird diese wichtige Thematik separat in Kapitel 7.5 besprochen, wo alternative Vorgehensmodelle dargestellt werden.

Als einzelne Praktiken zur Agilitätssteigerung können noch Prototyping und Simulation genannt werden, durch deren Einsatz auch bestimmte Informationen in der Entwicklung generiert werden können, wenn die Entwicklung in kleinen Schritten oder der AEZ nicht umsetzbar sind.

**Unkonventionelle Anwendung:** In vielen Fällen ist die Entwicklung in kleinen Schritten nur deshalb nicht möglich, weil von bestimmten (zumeist aus der agilen SW-Entwicklung bekannten) Regeln ausgegangen wird. In einer unkonventionellen Anwendungsform kann mit der Entwicklung in kleinen Schritten aber sehr wohl die Agilität gesteigert werden, auch wenn bestimmt Regeln nicht eingehalten werden (siehe 6.4.9).

**Weiteres:** Sind unterschiedliche Zykluszeiten wegen der gleichzeitigen Entwicklung von Hardware und Software das Problem, kann durch Synchronisation der Zyklen mit Anchor Point Milestones (5.4.5) Abhilfe geschaffen werden.

### 6.4.2 Kundeneinbindung

**Allgemeine Voraussetzung:** Ein Kunde, der zur Einbindung in die Entwicklung und zum Geben von Feedback gewillt ist.

**Mögliche Probleme:** Durch Kundeneinbindung soll z.B. der Fokus auf die tatsächlichen Kundenanforderungen gelegt werden (5.1.1) oder es sollen die Anforderungen priorisiert werden (5.1.5). Im AEZ wird das Feedback des Kunden genutzt, um die Richtung der Entwicklung zu lenken. Bei Auftragsentwicklung kann hier ein Problem entstehen, wenn der Kunde nicht zur Einbindung gewillt ist, weil er hierbei nur die Kosten und nicht den Nutzen sieht. Wird ein Produkt entwickelt, dass erst danach am Markt angeboten werden soll, ist während der Entwicklung keine unmittelbare Ansprechperson vorhanden.

**Erfüllen der Voraussetzungen:** Will man die Kundeneinbindung als Voraussetzung für bestimmte Praktiken erfüllen, so kann man versuchen den Kunden durch Erklärung der Vorteile zu einer Mitarbeit

zu bewegen. Am Markt besteht die Möglichkeit nach Testpersonen zu suchen bzw. Informationen aus dem Marketing zu erhalten, worauf in dieser Arbeit aber nicht weiter eingegangen wird.

**Alternative Praktiken:** Stehen weder Kunden noch Anwender zur Verfügung kann auch ein „Product Owner“ eingesetzt werden, der die Sichtweise des Kunden in der Entwicklung vertritt (5.1.6). Befinden sich bereits Vorläufer des zu entwickelnden Systems in Verwendung, können eventuell Daten daraus als Feedback genutzt werden (siehe 5.5.8).

**Unkonventionelle Anwendung:** Bei Anwendung des AEZ kann eventuell auf das Feedback des Kunden verzichtet werden und durch Feedback von zugezogenen internen oder externen Experten/Beratern ersetzt werden.

Scheitert die Priorisierung der Anforderungen am Willen des Kunden, kann diese ebenso intern durchgeführt werden. Internen Maßnahmen hängen stark von der Qualifikation und Erfahrung der Mitarbeitern ab und ihrer Fähigkeit, die tatsächlichen Anforderungen der Kunden antizipieren zu können.

### 6.4.3 Kritikalität

**Allgemeine Voraussetzung:** Ein niedriges Sicherheitsrisiko des Produkts.

**Mögliche Probleme:** Sicherheitskritische Projekte werden nicht als Anwendungsgebiet der agilen Softwareentwicklungsmethoden gesehen, da die geringfügigen Dokumentationsvorschriften keine nachvollziehbare Entwicklung der sicherheitskritischen Anforderungen sicherstellen. Außerdem können bei flexiblem Anforderungsmanagement diese Anforderungen leicht „verloren“ oder geändert werden. Ein weiterer Problemfall sind Zwischenergebnisse, welche vom Kunden genutzt werden sollen. Der Kunde darf dabei natürlich nicht gefährdet werden, weshalb auch die Zwischenergebnisse hinsichtlich Sicherheit zu überprüfen sind, was einen beträchtlichen Aufwand bedeuten kann.

**Erfüllen der Voraussetzungen:** Eventuell kann durch sicherheitstechnische Maßnahmen am Produkt die Sicherheitsanforderungsstufe[13] am Produkt verringert werden, was zu leichter erfüllbaren Prozessvorschriften führen kann. Bei einem tatsächlich sicherheitskritischen Produkt ist es aber nicht möglich bzw. gesetzlich erlaubt, die Entwicklung nach nicht sicherheitskritischen Gesichtspunkten durchzuführen um diese Voraussetzung zu erfüllen.

**Alternative Praktiken:** Agile Praktiken, welche aufgrund der Sicherheitskritikalität nicht durchführbar sind, können selten durch andere agile Praktiken ersetzt werden, sondern sind viel mehr durch traditionelle, aus Sicherheitsgründen starr ausgelegte und besser nachverfolgbare Methoden zu ersetzen.

**Unkonventionelle Anwendung:** Einige agile Praktiken können verwendet werden, wenn zusätzlich sicherheitskritische Prozessanforderungen umgesetzt werden. Ein unabsichtliches „Verlieren“ oder Ändern der sicherheitskritischen Produktanforderungen kann durch spezielle Priorisierung behoben werden (Stelzmann, et al., 2010 S. 253). Bei Zwischenergebnissen ist die Sicherheit speziell zu berücksichtigen, wenn diese an den Kunden übergeben werden sollen. Eine nicht ausreichende Dokumentation kann durch die sicherheitsrelevante Dokumente erweitert werden. Beispiele aus der

[13] Siehe http://de.wikipedia.org/wiki/Sicherheitsanforderungsstufe

Softwareentwicklung hierzu finden sich in (Paulk, 2001), (Turner, et al., 2002), (Fritzsche, et al., 2007) und (Kähkönen, et al., 2004). Bei diesen Maßnahmen ist aber zu überprüfen, wie weit sie die Effizienz des Prozesses beeinträchtigen und welcher Nutzen hinsichtlich Agilität dann noch besteht.

### 6.4.4 Kultur

**Allgemeine Voraussetzung:** Eine für die jeweilige Praktik geeignete Kultur.

**Mögliche Probleme:** Sowohl intern als auch beim Kunden können die Kultur, bzw. gewisse Gewohnheiten zu Problemen bei der Anwendung von agilen Praktiken führen. Z.B. ist der Wunsch, alles im Voraus planen zu wollen, weit verbreitet und Änderungen werden oft als Fehler gesehen. Alternative Vorgehensmodelle werden mit Argwohn betrachtet, da ein sequentielles Abarbeiten von Phasen, welches in Ausbildung und Darstellung von firmenspezifischen Prozessmodellen weit verbreitet ist, als unumgänglich betrachtet wird. Viele weitere Ressentiments kommen in der Praxis ebenso vor.

**Erfüllen der Voraussetzungen:** Die besten und umfangreichsten Möglichkeiten bestehen hier wohl im Erfüllen der Voraussetzungen, also im Kulturwandel, Ändern von Gewohnheiten und Beseitigen von Ressentiments, auch wenn dies schwierig sein und lange dauern mag. Mitarbeiter und Kunden müssen dazu überzeugt werden, dass in der Entwicklung von komplexen und neuartigen Systemen nicht immer alles (Systemverhalten, Entwicklungszeiten, Kosten, …) vorhergesehen und geplant werden kann. Änderungen und Wiederholungszyklen sollen nicht als Fehler sondern als natürlicher Bestandteil der Entwicklung gesehen werden. Mitarbeiter sollen dazu bewogen werden, Verantwortung zu übernehmen, an der Organisation mitzuwirken und Freiheiten sinnvoll zu nutzen. Für die Entwicklung von Varianten oder beim Set-Based Design soll ein Verständnis geschaffen werden, dass mehrere Varianten nicht nur zusätzlichen Aufwand bedeuten, sondern einen großen Nutzen hinsichtlich Informationsgewinnung und Flexibilität haben können.

**Alternative Praktiken und unkonventionelle Anwendung:** Natürlich besteht die Möglichkeit, dass die kulturellen Voraussetzung nur hinsichtlich einer bestimmten Praktik oder einer bestimmten Ausprägung gegenüber bestehen. Das kann aber nur im konkreten Fall genauer betrachtet werden und soll hier nicht weiter diskutiert werden.

### 6.4.5 Mitarbeiterqualifikation

**Allgemeine Voraussetzung:** Eine besonders hohe Qualifikation der Mitarbeiter.

**Mögliche Probleme:** Agile Praktiken legen weniger Wert auf genau Prozessvorschriften und mehr auf die Fähigkeiten der Entwickler, weshalb mangelnde Qualifikation und Erfahrung der Mitarbeiter natürlich zu Problemen in der Anwendung dieser Praktiken führen können.

**Erfüllung der Voraussetzungen:** Hier ist natürlich die Ausbildung der Mitarbeiter als Lösungsmöglichkeit zu sehen, genauso wie das Rekrutieren von neuen, hochqualifizierten Mitarbeiter. Beides ist aber oft nicht kurzfristig möglich. Bei einem aktuell zu startenden Projekt kann also nur auf die vorhandenen Mitarbeiter mit ihrem Qualifikationsprofil zugegriffen werden. Dabei ist aber durchaus vorstellbar, dass für Projekte, in denen ein erhöhter Bedarf an Agilität festgestellt wurde, vermehrt hochqualifizierte Mitarbeiter eingesetzt werden. Und die weniger qualifizierten, bzw. unerfahrenen Mitarbeiter eher in

Projekten mit weniger Bedarf an Agilität, bis diese mehr Erfahrung erlangt haben oder durch Ausbildungsmaßnahmen auf den Einsatz in agilen Projekten vorbereitet wurden (5.3.2).

**Alternative Praktiken und unkonventionelle Anwendung:** Ob alternative Praktiken oder andere Ausprägungsformen mehr den Qualifikationen der Mitarbeiter entsprechen, ist im konkreten Fall zu untersuchen. Eine Reihung der Praktiken hinsichtlich „Schwierigkeitsgrad“ wird in dieser Arbeit nicht durchgeführt.

### 6.4.6 Teamgröße und Verteilung

**Allgemeine Voraussetzungen:** Kleine Teams und örtliche Nähe der Mitglieder.

**Mögliche Probleme:** Bezüglich Teamgröße und Verteilung ist vor allem die Effizienz der Kommunikation zu beachten, die für viele agile Praktiken notwendig ist. Zu große und stark verteilte Teams können hier Probleme bereiten.

**Erfüllen der Voraussetzungen:** Große Teams können aufgeteilt werden (5.2.1), verteilte Teams sollen nach Möglichkeit örtlich zusammengelegt werden (5.2.2).

Highsmith meint aber, dass wenn die Voraussetzungen nicht erfüllt werden können, Agilität trotzdem oft auch in großen, verteilten Teams unabdingbar ist. Er hält dann eine genaue Betrachtung der Gestaltung von Organisation, Zusammenarbeit, Entscheidungsfindung und Wissensaustausch für notwendig und sieht die Selbstorganisation der Teams und Disziplin als wichtige Faktoren (Highsmith, 2010 S. 271ff). Teamgröße und Verteilung sind also in einem größeren Kontext zu betrachten, wenn es um die Machbarkeit von agilen Praktiken geht. Die Kommunikation als wichtigstes Kriterium kann auch auf anderem Wege optimiert werden (5.3.5).

**Alternative Praktiken und unkonventionelle Anwendung:** Auch wenn agile Praktiken leichter bei kleinen, nicht verteilten Teams umzusetzen sind, sind diese Faktoren kaum ein absoluter Ausschlussgrundgrund für agile Praktiken. In vielen Fällen sollten die Praktiken anwendbar sein, wenn sie durch traditionelle Vorgehensweisen ergänzt werden, z.B. durch eine in Maßen durchgeführte Ergänzung der Dokumentation oder eine Unterstützung der Selbstorganisation von oben herab.

### 6.4.7 Preisgestaltung

**Allgemeine Voraussetzung:** Eine flexible Preisgestaltung.

**Mögliche Probleme:** Einige agile Praktiken sind nicht mit der Festsetzung eines Fixpreises zu Projektbeginn kompatibel, weil sie nicht auf einer genauen Definition von Anforderungen und Entwicklungsumfang zu Beginn basieren. Deshalb werden agile Softwareentwicklungsmethoden auch nicht für Fixpreisprojekte empfohlen. In der Systementwicklung sind Fixpreisprojekte aber üblich. Soll dennoch Flexibilität gegenüber Änderungen bestehen, kann es zu Problemen mit einigen dafür notwendigen Praktiken kommen.

**Erfüllen der Voraussetzungen:** Es wäre denkbar den Kunden zu einem alternativen Preismodell zu überreden, welches eine faire Bezahlung der Erfüllung der tatsächlichen Kundenanforderungen vorsieht. Dazu wurde aber für diese Arbeit aber keine Recherche betrieben.

**Alternative Praktiken:** Ist der AEZ oder die Entwicklung in kleinen Schritten nicht anwendbar, kann hier ev. auf Set-Based Design umgestiegen werden. Dabei kann im Rahmen der Machbarkeitsuntersuchung auch der Preis als Kriterium gelten und Varianten, welche nicht mit dem vereinbarten Fixpreis machbar sind, werden ausgeschieden.

**Unkonventionelle Anwendung:** Für Scrum wird als mögliche Abhilfemaßnahme eine genauere Analyse der Kundenanforderungen zu Beginn empfohlen, um damit einen ausführlicheren und genaueren Product-Backlog zu formulieren. Damit kann ein Preis festgesetzt und die Entwicklung danach dennoch mit Scrum durchgeführt werden, auch wenn die Anforderungen in der weiteren Entwicklung nicht mehr so flexibel gehandhabt werden können (Schwaber, 2004 S. 147ff). Mit dieser Vorgehensweise könnte man also den AEZ bzw. die Entwicklung in kleinen Schritten trotz Fixpreis umsetzbar machen.

### 6.4.8 Wirtschaftlichkeit

**Allgemeine Voraussetzung:** Möglichkeit zur effizienten und effektiven Umsetzung der jeweiligen Praktik.

**Mögliche Probleme:** Die Wirtschaftlichkeit hängt sehr stark von der gegebenen Situation und damit auch von den anderen Voraussetzungen ab. Z.B. können je nach Art des Systems (Kosten für Herstellen, Testen und Ändern) verschiedene Praktiken oder auch verschiedene Vorgehensmodelle (siehe 7.5) mehr oder weniger wirtschaftlich sein. Aber auch die Art der Fertigung und die herzustellende Stückzahl haben einen Einfluss darauf, welche Vorgehensweisen in der Entwicklung wirtschaftlichen Sinn machen. Bei Veränderungen wird die Thematik noch komplexer. Je nachdem welche Veränderungen eintreten, kann oft erst im Nachhinein bestimmt werden, welche Vorgehensweise die wirtschaftlichste gewesen wäre. In vielen Fällen gibt es auch einen Unterschied zwischen der Vorgehensweise, welche für das aktuelle Projekt und jener, welche für die gesamte Unternehmung das wirtschaftliche Optimum darstellt. Z.B. wenn durch Standardisierung Effizienzgewinne erzielt wurden, die aber in einem bestimmten Projekt nicht umgesetzt werden können. Damit ist die Wirtschaftlichkeit eine sehr komplexe Thematik, die im Einzelfall untersucht werden muss. Hier sollen zu möglichen Problemstellungen ein paar allgemeine Gedanken widergegeben werden.

**Erfüllen der Voraussetzungen:** Sind AEZ bzw. Entwicklung in kleinen Schritten nicht wirtschaftlich umsetzbar, kann durch Gestaltung von Produkt und Entwicklungswerkzeugen versucht werden, die Kosten für das Herstellen von Zwischenergebnissen, das Testen und für Änderungen zu verringern. Das Einführen neuer Prototyping-Verfahren (5.4.7) kann für kostengünstigere und besser nutzbare Zwischenergebnisse sorgen. Tests können automatisiert werden. Einfachheit (5.5.1) und Anpassbarkeit (5.5.2) des Produkts können für kostengünstigere Änderungsmöglichkeiten sorgen.

**Alternative Praktiken:** Für jede Situation kann sich eine unterschiedliche Vorgehensweise bzw. Strategie im Umgang mit Veränderungen als die wirtschaftlichste herausstellen. Einige dieser Strategien werden im Folgenden diskutiert:

- **AEZ bzw. Entwicklung in kleinen Schritten (4.3.4), (5.4.3):** Die Anwendung dieser Prinzipien ist vermutlich dann am wirtschaftlichsten, wenn die Zwischenergebnisse schnell und kostengünstig hergestellt, getestet und geändert werden können und wenn die Anforderungen zu Beginn unklar waren oder sich häufig ändern.

- **Set-Based Design (5.4.18):** Ebenso bei unklaren Anforderungen oder vielen Änderungen kann sich das Set-Based Design als wirtschaftlichste Vorgehensweise herausstellen, vor allem dann wenn Herstellung und Änderung von Produkt und Produktionslinie große Kosten verursachen.
- **Flexibilität in Produkt oder Produktionslinie (5.5.2), (5.5.1), (5.5.4):** Können mögliche Änderungen auf einen oder wenige Designparameter eingeschränkt werden, so kann auch eine traditionelle Vorgehensweise - bei der aber speziell für diese Designparameter Flexibilität eingeplant wird - die wirtschaftlichste sein. Z.B. kann für eine bestimmte Spezifikation des Produkts ein höherer Wert eingeplant werden, der als Reserve hinsichtlich einer bestimmten Änderung zu sehen ist.
- **Entwickeln mehrerer Varianten (5.4.17):** Sind Änderungen teuer aber es ist die Notwendigkeit gegeben, zu einem gewissen Zeitpunkt eine kurzfristige Auswahlmöglichkeit zwischen Varianten zu haben, so kann die gleichzeitige Entwicklung mehrerer Varianten die wirtschaftlichste Alternative sein.
- **Plattformstrategie und Wiederverwendung von Modulen (5.5.5), (5.5.6):** Besteht die Herausforderung an die Agilität darin, rasch nach Bekanntwerden der Anforderungen neue Produkte herstellen zu können, die sich in ihrer Grundstruktur aber ähneln, so können diese Prinzipien die größte Wirtschaftlichkeit bieten.
- **Pufferzeiten (5.4.10):** Es kann auch eine traditionelle Entwicklung bei den entsprechenden Umständen die wirtschaftlichste Lösung darstellen. Die Situation wird dabei im Vorhinein möglichst genau analysiert und die Produktspezifikationen werden möglichst genau festgelegt. Für wenige und kleinere Änderungen kann eine Pufferzeit im Projekt eingeplant werden.

Aus einigen dieser grundlegenden Strategien (von denen auch weitere denkbar sind) wurden agile Vorgehensmodelle abgeleitet (siehe 7.5), welche alternativ im SE angewandt werden können.

Hinsichtlich Wirtschaftlichkeit sind aber nicht nur diese grundlegenden Strategien zu diskutieren, sondern es kann auch für einzelne Praktiken nach wirtschaftlicheren Alternativen gesucht werden. Z.B. können in einigen Fällen teure Prototypen durch kostengünstigere Simulationen ersetzt werden.

**Unkonventionelle Anwendung:** Ob eine unkonventionelle Anwendung einzelner Praktiken zu besserer Wirtschaftlichkeit führen kann, ist im konkreten Einzelfall zu überprüfen.

### 6.4.9 Entwicklung in kleinen Schritten

Nachdem die verschiedenen Voraussetzungen allgemein diskutiert wurden, soll nun ganz speziell auf Möglichkeiten bei nicht erfüllten Voraussetzungen für die Entwicklung in kleinen Schritten eingegangen werden. Nach den Ausprägungsformen des AEZ ist diese von zahlreichen Voraussetzungen betroffen. Sie bietet hinsichtlich Agilität aber auch große Chancen. Als strukturelle Vorgehensweise für die Entwicklung ist sie eine Alternative zum Wasserfallprozess und es wurde neben ihr und dem Set-Based Design keine weitere Vorgehensweise für ASE gefunden, die der Entwicklung eine solche Grundstruktur geben kann. Sollte die Entwicklung in kleinen Schritten zu risikoreich sein, ist es also schwer ein vergleichbares Prinzip als Ersatz zu finden. Auch die Erfüllung aller Voraussetzungen, so dass die Entwicklung in kleinen Schritten in allen Merkmalen nach den Ausprägungsformen des AEZ umgesetzt werden kann, ist in vielen Systementwicklungsprojekten nur schwer machbar. In 5.4.3 wurden für die Merkmale schon Beispiele gegeben, wie diese in einer anderen Ausprägungsform, als der vom AEZ geforderten, dennoch Nutzen

hinsichtlich der Agilität bringen können. Diese Merkmale sollen nun weiter untersucht werden. Dabei werden je Merkmal die Ausprägungsformen des AEZ als Maximalkriterium hinsichtlich Agilität dargestellt. Weiters werden je Merkmal Minimalkriterien genannt, welche gerade noch die Chance zur Agilitätssteigerung bieten können. Und dazwischen werden je Merkmal auch Möglichkeiten dargestellt, mit denen eine teilweise Umsetzung des AEZ möglich sein soll, bzw. eine Anwendung in einer Form, mit welcher die Agilität zumindest moderat gesteigert werden kann. Wie aus der Entwicklung in kleinen Schritten ein konkretes SE-Vorgehensmodell erstellt werden kann, unter Zuhilfenahme der folgenden Hinweise, wird später in Kapitel 7.5.2 beschrieben.

***Nutzbare Zwischenergebnisse***

**Ausprägung im AEZ:** Jeder Entwicklungsschritt soll vom Kunden (Anwender) genutzt werden können, weil dabei das beste Feedback zu erwarten ist.

**Minimalkriterium:** Mit dem Einholen von Feedback auf Zwischenergebnisse soll eine Lenkungsfunktion für die Entwicklung umgesetzt werden. Der Sinn der Nutzbarkeit der Zwischenergebnisse ist, dass diese vom Kunden möglichst gut hinsichtlich der Übereinstimmung mit den tatsächlichen Kundenanforderungen bewertet werden können. Änderungen, die zur Korrektur des Unterschieds zwischen der Interpretation der niedergeschriebenen Anforderungen und der tatsächlichen Anforderungen zustande kommen, können damit zu einem früheren Zeitpunkt bearbeitet oder vermieden werden. Dies sind aber nicht die einzigen Änderungen, die im Rahmen einer agilen Vorgehensweise berücksichtigt werden sollen. Für Änderungen mit anderer Ursache, z.B. wenn bei der Produktionsplanung ein Designparameter des Produkts als nicht machbar eruiert wird oder für Änderungen außerhalb der Entwicklung ist es nicht wichtig, ob der Kunde die Zwischenergebnisse in ihrer Endanwendung testen konnte. Es muss hier lediglich eine Beurteilbarkeit hinsichtlich der möglichen Ursachen für Änderungen gewährleistet sein. Die Lenkungsfunktion des Feedbacks kann dann natürlich auch nur hinsichtlich der beurteilbaren Kriterien umgesetzt werden. Als Minimalkriterium für das Merkmal der Nutzbarkeit der Zwischenergebnisse kann also eine Beurteilbarkeit hinsichtlich der zu behandelnden Änderungen gesehen werden.

**Möglichkeiten:** Anstatt des Nutzens für den späteren Anwender kann also auch die Gewinnung jeglicher Entwicklungserkenntnisse aus den Zwischenergebnissen als Feedback für die Lenkung der Entwicklung verwendet werden. Damit kann die Entwicklung auch ohne Einbindung des Kunden gelenkt werden. Die Zwischenergebnisse müssen auch keine Teile des späteren Systems, sondern können auch Prototypen, Simulationsmodelle oder auch nur Berechnungen oder Konstruktionszeichnungen sein. Alle Informationen, welche das Endergebnis zu einem früheren Zeitpunkt in der Entwicklung beurteilbar machen, können hilfreich hinsichtlich Agilität sein. Die Gestaltung der Zwischenergebnisse soll natürlich trotzdem so weit wie möglich ihrer Endanwendung entsprechen, weil dadurch die beste und umfangreichste Beurteilbarkeit zu erwarten ist.

Abgesehen von der Gestaltung der Zwischenergebnisse bieten sich auch Möglichkeiten im Prozess. So ist es nicht unbedingt als nötig zu erachten, dass jedes einzelne Zwischenergebnis nutzbar ist. Eine Lenkungsfunktion in der Entwicklung kann auch erreicht werden, wenn nur manche Zwischenergebnisse in einer nutzbaren Form gestaltet werden. Natürlich kann damit die Agilität auch nur in einem geringeren Ausmaß gesteigert werden.

Wenn die Zwischenergebnisse schon Teile des späteren Systems sind und das Problem darin besteht, dass diese separat nicht nutzbar sind, besteht untern Umständen eine weitere Möglichkeit. Wenn ein altes, kompatibles System vorhanden ist, können diese eventuell darin integriert und genutzt werden.

***Schrittgröße/Zyklusdauer***

**Ausprägung im AEZ:** Größe („klein“) und Dauer („kurz“) entsprechend den Zielen des AEZ.

**Minimalkriterium:** Ein Minimalkriterium, welches die größtmögliche Schrittgröße bzw. die längst mögliche Zyklusdauer darstellen würde, kann nicht im Allgemeinen quantifiziert werden. Grenzen hierfür sind für konkrete Projekte abhängig vom geforderten Grad an Agilität zu finden.

**Möglichkeiten:** Auch wenn dadurch die Ziele des AEZ nur mehr teilweise erreichbar sind, können die Schritte länger/größer angesetzt werden, als sie in den agilen SW Entwicklungsmethoden vorgeschlagen werden. Es können die Zyklusdauern für Systemkomponenten auch unterschiedlich angesetzt werden, z.B. für SW kürzer und für HW länger.

***Zyklische Entwicklung***

**Ausprägung im AEZ:** Es ist geplant, in jedem Schritt sämtliche Prozessschritte (Analyse, Planung, Realisierung und Test) durchzuführen.

**Minimalkriterium:** Der AEZ fordert eine zyklische Entwicklung - es soll in jedem Zyklus ein Teilumfang der Entwicklung geplant, realisiert und getestet werden. Hat man bereits akzeptiert, dass nicht nach jedem Schritt ein vom Anwender nutzbares Zwischenergebnis zur Verfügung stehen muss, so gibt es kaum ein Minimalkriterium für die zyklische Entwicklung. Lediglich eine geregelte Entwicklung (sinnvolle Aneinanderreihung von Phasen/Schritten) kann als solches gesehen werden.

**Möglichkeiten:** Wie im traditionellen SE Vorgehensmodell (Abb. 8) können Entwicklungs- und Realisierungsphasen getrennt bleiben und die Realisierung (speziell Fertigung der Hardware) in einem linear durchgeführten, großen Schritt erfolgen. In der Entwicklung kann dann aber trotzdem in kleinen, zyklischen Schritten aus Analyse, Planung und Test vorgegangen werden. Der Schritt der Planung enthält, wie in 4.3.4 beschrieben, auch die technische Planung, also Berechnungen, Konstruktionszeichnungen, Simulationsmodelle, Prototypen usw. Die Tests bzw. Beurteilungen in der Entwicklungsphase beziehen sich dann eben nur auf diese Zwischenergebnisse und nicht auf realisierte Teile des Systems.

Eine weitere Möglichkeit ist die Durchführung unterschiedlicher Zyklen, z.B. können die Entwicklungsschritte in schneller Frequenz erfolgen und nur nach jedem dritten Entwicklungsschritt wird ein Realisierungsschritt durchgeführt.

***Art der Unterteilung***

**Ausprägung im AEZ:** Inkrementell und iterativ bzw. jede Unterteilung, die eine Entwicklung nach den anderen Ausprägungen des AEZ (Nutzbare Zwischenergebnisse, Schrittgröße/Zyklusdauer, zyklische Entwicklung, Vorausplanung der Unterteilung) erlaubt.

**Minimalkriterium:** Als Minimalkriterium kann hier eine Unterteilung gesehen werden, mit der sich auch zumindest die Minimalkriterien der anderen Merkmale erreichen lassen. Die Unterteilung muss also Zwischenergebnisse ermöglichen, die zur Lenkung der Entwicklung bewertbar sind. Die damit umsetzbaren Schrittgrößen/Zyklusdauern müssen dem gewünschten Grad an Agilität entsprechen. Eine

geregelte Entwicklung muss möglich sein. Und es muss eine gewisse Anpassbarkeit der Entwicklungsrichtung von Schritt zu Schritt möglich sein.

**Möglichkeiten:** Wie in 5.4.3 bereits erwähnt, sind Möglichkeiten u.a. Inkremente, Iterationen, Phasen, Releases und Systemkomponenten. Die Unterteilung ist aber immer so zu wählen, dass die geforderten Ausprägungsformen der anderen Merkmale umgesetzt werden können und der gewünschte Grad an Agilität erreicht werden kann. Z.B. kann mit einer Entwicklung mit wenigen Releases, in welcher das Feedback von Release zu Release für Änderungen genutzt wird, auch die erforderliche Agilität erreicht werden, wenn diese nicht besonders hoch ist.

***Expliziter Prozess***

**Ausprägung im AEZ:** Ein explizit gewollter und im Vorhinein definierter Prozess.

**Minimalkriterium:** Wie in 5.4.3 bereits erklärt, wird hier die Entwicklung in kleinen Schritten als expliziter Prozess gesehen. Kleine implizite Schritte (ungeplant durchgeführt) wurden als Improvisation definiert. Als Minimalkriterium für einen expliziten Prozess wird also die Abgrenzung zur Improvisation gesehen.

**Möglichkeiten:** Auch bei einer expliziten Entwicklung in kleinen Schritten gibt es natürlich auch implizite kleine Schritte. Diese sollen wertgeschätzt und dürfen nicht verhindert werden. Dazu soll der Prozess eine hohe Flexibilität aufweisen. Die Entwicklung soll eine sinnvolle Kombination von expliziten und impliziten Prozessschritten darstellen.

***Vorausplanung der Unterteilung***

**Ausprägung im AEZ:** Der genaue Inhalt eines Schrittes wird durch die vorhergehenden Schritte beeinflusst und erst zu Beginn des Schrittes definiert.

**Minimalkriterium:** Abhängig vom geforderten Grad an Agilität und der zu bewältigenden Änderungen, muss eine gewisse Anpassbarkeit der Entwicklungsrichtung von Schritt zu Schritt möglich sein.

**Möglichkeiten:** Auch im AEZ werden natürlich die Schritte im Groben geplant, sie bleiben aber anpassbar. Man kann auch das System im Groben fixieren und die Details anpassbar gestalten. Das kann dadurch umgesetzt werden, indem man z.B. lediglich die Systemarchitektur zu Beginn fixiert.

***Periodizität***

**Ausprägung im AEZ:** Konstante Zyklusdauer.

**Minimalkriterium:** Eine agilitätssteigernde Umsetzung der Entwicklung in kleinen Schritten mit variablen Zyklusdauern ist ebenso denkbar, weshalb hier kein Minimalkriterium benannt werden kann.

**Möglichkeiten:** Z.B. kann die Zyklusdauer zu Beginn jeden Schrittes frei vereinbart werden oder es kann auch eine Regeldauer geben, die nur im Bedarfsfall verändert wird.

## 6.5 Zusammenfassung

In diesem Kapitel wurde der richtige Kontext für Agile Systems Engineering diskutiert. Um einen Überblick über mögliche Faktoren dafür zu bekommen, wurden zuerst die Anwendungsgebiete der agilen Softwareentwicklungsmethoden analysiert. Danach wurden die Ergebnisse der empirischen Erhebungen zum richtigen Kontext für ASE dargestellt.

Damit sollten die Forschungsfragen 3 und 4, nach dem Bedarf und den Voraussetzungen von Agilität beantwortet werden. In Abb. 28 wurde dazu überblicksartig dargestellt, welche Bedarfsfaktoren die Beschäftigung mit Agilität im SE - und damit den ASE Ansatz dieser Arbeit - interessant erscheinen lassen. Ebenso wurden Voraussetzungen aufgezählt, welche für bestimmte Werkzeuge (Praktiken) des ASE Ansatzes gelten können.

In Kapitel 6.3 wurde zusätzlicher Nutzen aufgezeigt, der bei der Anwendung von agilen Praktiken häufig erzielt werden kann, aber nicht in direktem Zusammenhang mit Agilität steht.

Außerdem konnten in Kapitel 6.4 einige Möglichkeiten zum Umgang mit den Voraussetzungen angeführt werden. Es wurde dabei gezeigt, welche Möglichkeiten bestehen, bestimmte Voraussetzungen für agile Praktiken zu erfüllen. Ebenso wurden für einige Fälle, in denen Voraussetzungen die Anwendung einer bestimmten Praktik verhindern, alternative Praktiken diskutiert, welche den agilen Nutzen zumindest teilweise erfüllen können. In der im nächsten Kapitel erstellten Vorgehensmethodik wird die Auswahl geeigneter Praktiken ebenso ein wichtiger Punkt sein. Genauso wie die Nutzbarmachung agiler Praktiken durch eine unkonventionelle Anwendung, welche auch in 6.4 diskutiert wurde und in Kapitel 7.5 bei der Gestaltung von agilen SE Vorgehensmodellen zur Anwendung kommen wird.

Dieses Kapitel hat damit wichtige Grundlagen für die Vorgehensmethodik im nächsten Kapitel geschaffen und Rahmenbedingungen für eine erfolgreiche Anwendung des ASE Ansatzes aufgezeigt.

# 7 Die Agile Systems Engineering Vorgehensmethodik

Die bisherigen Erkenntnisse können als grundlegende Bausteine des Ansatzes zur Lösung der Problemstellung dieser Arbeit betrachtet werden. Das Herstellen aller Zusammenhänge und die Beschreibung der Vorgehensweise bei der Anwendung des ASE Ansatzes erfolgt nun in diesem Kapitel. Eine Zusammenfassung der bisher gewonnenen, grundlegenden Erkenntnisse findet dazu in Kapitel 7.1 statt. Danach wird mit einer Ziel-Mittel-Hierarchie (7.2) gezeigt, wie die Agilität mit ihren Teilfunktionen, übergreifenden Maßnahmen in den verschiedenen Eingriffsbereichen und schließlich mit den agilen Praktiken als Mittel dafür zusammenhängt, um das Ziel der erfolgreichen Bewältigung von Veränderungen im SE zu erreichen. In 7.3 werden die Zusammenhänge der möglichen Maßnahmen in den Eingriffsbereichen Kunde, Organisation, Mitarbeiter, Entwicklungsprozess und System aufgezeigt. Kapitel 7.4 bietet dann eine Auswahlmatrix der dafür zur Verfügung stehenden Praktiken an. In 7.5 werden Vorgehensmodelle vorgestellt, die als Beispiele für die Gestaltung von agilen Entwicklungsprozessen gesehen werden können. Damit werden in diesem Kapitel jene Komponenten einer SE Methodik zusammengefügt, welche die BWI-Hall Methodik um Agilität erweitern sollen (ein abschließender Überblick dazu wird im Fazit in Abb. 39 gegeben).

## 7.1 Zusammenfassung der bisherigen Erkenntnisse

Es sollen hier kurz die dem Ansatz zugrundeliegenden Erkenntnisse zusammengefasst werden, wobei auf die Kapitel verwiesen wird, in denen diese näher erklärt wurden.

- Agilität ist eine Fähigkeit (4.1).
- „Agile Systems Engineering“ erscheint als Bezeichnung für den Lösungsansatz zur Problemstellung dieser Arbeit angemessen, wenngleich in der sonstigen Literatur die Agilität nicht immer so stark auf den Umgang mit Veränderungen bezogen ist (1.1, 4.2).
- ASE kann nicht ein „besseres SE“ darstellen, sondern nur Hilfestellungen im Umgang mit Veränderungen anbieten, sollte dafür ein Bedarf bestehen (4.2, 6).
- Die Agilität kann im SE durch die Förderung ihrer Teilfunktionen gesteigert werden (4.2):
    - Situationskenntnis
    - Kontinuierliche Verbesserung
    - Bewältigung von Veränderungen
    - Flexibilität
    - Effizienz
    - Effektivität
- Man kann die Agilität in den Eingriffsbereichen Kunde, Organisation, Mitarbeiter, Entwicklungsprozess und System fördern (5).
- Dazu stehen zahlreiche Prinzipien und Praktiken zur Verfügung (5).
- Die einzelnen Praktiken fördern aber immer nur bestimmte Teilfunktionen der Agilität, außerdem haben sie gewissen Voraussetzungen bzw. unterliegen sie gewissen Risiken. Erst durch genaue Analyse aller Einflussfaktoren, können die richtigen Maßnahmen hinsichtlich einer Verbesserung der Agilität getroffen werden (5, 6.2.3).

## 7.2 Ziel-Mittel-Hierarchie

Mit dem Ziel-Mittel-Denken sollen die Gesamtzusammenhänge von Zielen und Mitteln über mehrere Ebenen dargestellt und Begründungszusammenhänge für untergeordnete Sollaussagen geliefert werden (Haberfellner, et al., 2002 S. 139ff). In Abb. 29 wird eine Ziel-Mittel-Hierarchie für den ASE Ansatz dieser Arbeit gezeigt, mit der Ursachen und Wirkungen besser voneinander unterschieden und die Zusammenhänge zwischen den Ebenen des Ansatzes verständlich gemacht werden sollen.

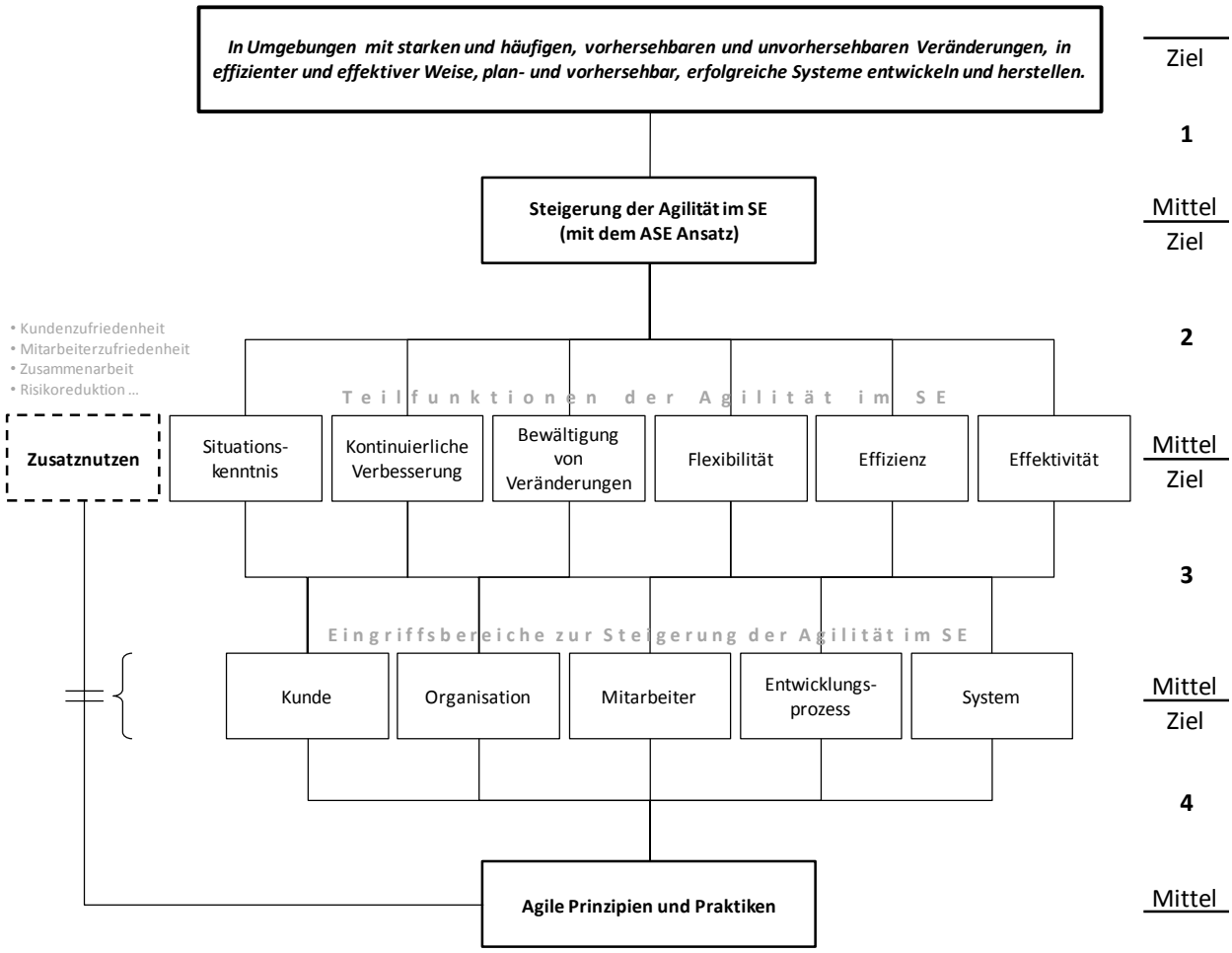


**Abb. 29: Ziel-Mittel-Hierarchie für ASE**

Das Ziel auf erster Ebene ist das Ziel dieser Arbeit, das auch als Ziel des ASE Ansatzes gesehen werden kann. Das Mittel zur Erreichung dieses Ziels ist die Steigerung der Agilität im SE mithilfe des ASE Ansatzes. Hiermit soll also ausgedrückt werden, für welches Problemfeld der ASE Ansatz geschaffen wurde.

Betrachtet man auf zweiter Ebene die Agilitätssteigerung als Ziel, so ist als Mittel dafür die Steigerung oder Verbesserung ihrer Teilfunktionen zu sehen, welche in Abb. 14 bzw. in 4.2 aufgezeigt wurden. Damit soll dargestellt werden, dass die Agilität als Fähigkeit nicht unmittelbar durch eine konkrete Methode umgesetzt oder gesteigert werden kann. Dies kann nur durch die Optimierung ihrer

Teilfunktionen im SE geschehen. Analysiert man ein Projekt oder eine Unternehmung hinsichtlich Agilität, so stellen diese Teilfunktionen die Ansatzpunkte dar, bei denen nach Mängeln gesucht werden kann bzw. für welche Verbesserungspotentiale aufgezeigt werden sollen, wenn eine Steigerung der Agilität angestrebt wird.

Wird eine Verbesserung der Teilfunktionen der Agilität auf der dritten Ebene als Ziel gesehen, so sind Maßnahmen bei Kunden, in der Organisation, bei Mitarbeitern, im Entwicklungsprozess und beim System die Mittel dafür. Diese Eingriffsbereiche wurden bisher nur genutzt, um eine Einteilung für die Prinzipien und Praktiken in Kapitel 5 zu finden. Um die Zusammenhänge zwischen den Praktiken und den Teilfunktionen der Agilität besser darstellen zu können, werden die Bereiche nun auch für Aussagen auf dieser zusätzlichen Ebene dazwischen genutzt. Die Zusammenhänge werden in 7.3 genauer erklärt. Durch die Einführung dieser Ebene soll auch ausgesagt werden, dass es in unterschiedlichen Eingriffsbereichen Möglichkeiten gibt, die Agilität im SE zu verbessern.

Werden auf Ebene 4 die Maßnahmen in den eben erwähnten Eingriffsbereichen als Ziel gesehen, so sind es die in Kapitel 5 beschriebenen Prinzipien und Praktiken, welche die Mittel dafür darstellen. Damit ist die Erklärung der Zusammenhänge bei konkreten Maßnahmen angelangt, für welche damit folgendes ausgesagt werden soll: Es existieren zahlreiche Prinzipien und Praktiken, die in verschiedenen Eingriffsbereichen angewandt werden können, um auf unterschiedliche Art und Weise (je nach Teilfunktion) die Agilität im SE zu beeinflussen.

Unabhängig vom Ziel auf höchster Ebene wird auch noch der Zusatznutzen (siehe 6.3) der Praktiken gezeigt. Dieser ist vergleichbar mit den Teilfunktionen der Agilität, wird auch über Anwendung der Praktiken in den Eingriffsbereichen erzeugt und ist deshalb als Ziel auf Ebene 3 dargestellt.

Mit den dargestellten Zusammenhängen soll die Ziel-Mittel-Hierarchie auf prinzipieller Ebene das wesentliche Verständnis für Agilität im SE erzeugen, welches notwendig ist, um je nach Problemstellung geeignete Lösungsmaßnahmen zu finden. Vergleicht man dieses Verständnis mit den Komponenten des BWI-Hall Ansatzes (Abb. 7), so kann man es durchaus im Rahmen einer nun „agilen SE-Philosophie“ neben Systemdenken und Vorgehensmodell im gedanklichen Überbau des SE Modells ansetzen. Im nächsten Kapitel wird nun weiter ausgeführt, wie man mit den Maßnahmen in den Eingriffsbereichen die Teilfunktionen der Agilität beeinflussen kann und welche Zusammenhänge dabei bestehen.

## 7.3 Inhalte und Zusammenhänge des ASE

Wie erwähnt, kann zur Verbesserung einer Teilfunktion der Agilität nicht direkt auf eine anzuwendende Praktik geschlossen werden bzw. ist es schwierig das sinngemäß durchzuführen. Deshalb wurden in der Ziel-Mittel-Hierarchie (Abb. 29) dazwischen noch die Eingriffsbereiche eingefügt, um die Zusammenhänge besser darstellen zu können. Z.B. kann kaum von einer Entwicklung in kleinen Schritten direkt auf eine bessere Situationskenntnis geschlossen werden. Als Zwischenschritt ist hier der Entwicklungsprozess zu betrachten, der dadurch z.B. einen früheren Wissenserwerb über das Systemverhalten möglich macht. Damit kann dann die Verbesserung der Situationskenntnis begründet werden.

Um dieses Beispiel fortzuführen, soll eine Priorisierung der Anforderungen nach erwartetem Wissensgewinn angenommen werden. Werden die Entwicklungsschritte in der Reihenfolge dieser

Priorisierung durchgeführt, kann der Effekt des frühen Wissenserwerbs noch verstärkt werden. Damit kann eine weitere Verbesserung der Situationskenntnis aber auch der Zusammenhang zwischen den Praktiken der Entwicklung in kleinen Schritten und der Priorisierung begründet werden.

Durch die Aussagen zu den auf Ebene 3 als Mittel dargestellten Eingriffsbereichen wird also nicht nur begründet, wie die Praktiken die Teilfunktionen der Agilität verbessern sollen, sondern es werden auch die Zusammenhänge zwischen den Praktiken dargestellt. Damit sind diese Aussagen entscheidend für das Verständnis, das zur praktischen Anwendung des ASE Ansatzes benötigt wird. Auf eine grafische Darstellung aller Zusammenhänge zwischen den Praktiken wurde verzichtet, weil dies aufgrund der hohen Komplexität (große Anzahl an Praktiken und Zusammenhänge) nicht in einer übersichtlichen Form zu bewerkstelligen gewesen wäre. Stattdessen werden im Folgenden je Eingriffsbereich die Zusammenhänge erklärt, die darin zur Verbesserung der Teilfunktionen der Agilität führen können.

Für jede Teilfunktion werden in jedem Eingriffsbereich die wichtigsten Praktiken zur konkreten Umsetzung erwähnt. Zu beachten ist, dass in Kapitel 5 die Praktiken nach ihrem primären Eingriffsbereich eingeteilt wurden. Die Auswirkungen der Praktiken können aber durchaus mehrere Bereiche betreffen, weshalb einige der Maßnahmen in den folgenden Unterkapiteln mehrfach genannt werden.

Die Maßnahmen werden im Folgenden nur soweit begründet, dass die Zusammenhänge klar werden. Für eine genauere Erklärung wird auf die Darstellung der Prinzipien und Praktiken in Kapitel 5 verwiesen. Es werden im Folgenden auch nur die positiven bzw. agilitätssteigernden Auswirkungen beschrieben, um die potentiellen Umfänge des ASE Ansatzes aufzuzeigen. Voraussetzungen und Risiken wurden je Praktik ebenso schon in Kapitel 5 diskutiert und werden später auch noch in einer Übersicht in 7.4 dargestellt.

### 7.3.1 Kunde

In diesem Bereich wird das Zusammenspiel mit allen Personen betrachtet, für welche das Produkt entwickelt wird, egal ob diese extern (Kunde/Anwender) sind oder aus derselben Unternehmung stammen (interne Auftraggeber). Die Frage lautet, welche Möglichkeiten im Zusammenhang mit diesen bestehen, um die Teilaspekte der Agilität zu steigern?

**Situationskenntnis:** Durch die Wünsche des Kunden wird die Situation maßgeblich beeinflusst. Eine gute und laufend aktualisierte Situationskenntnis kann also nur unter Einbeziehung des Kunden erreicht werden. Um Missverständnisse zu vermeiden, soll auf gute Kommunikation und Zusammenarbeit Wert gelegt werden. Dies ist auch notwendig, um die tatsächlichen Anforderungen des Kunden zu ergründen. Entwickelt man nur nach den anfänglich niedergeschriebenen Anforderungen, sind später Änderungen in Richtung der tatsächlichen Kundenanforderungen (Kundenwunsch) zu erwarten. Man soll auch versuchen, zu verstehen, warum beim Kunden Änderungswünsche auftreten können. Durch die Situationskenntnis kann man spätere Änderungen antizipieren und eventuell früher in der Entwicklung berücksichtigen oder vermeiden. Durch ein frühes Demonstrieren von Entwicklungsergebnissen (Prototypen, Simulation), kann man vom Kunden besseres Feedback hinsichtlich der Anforderungserfüllung erhalten. Auch durch Varianten, die man vom Kunden bewerten lässt, oder durch eine vom Kunden durchgeführte Priorisierung kann man die Kenntnis über seine genauen Anforderungen verbessern. Für Änderungen, die außerhalb der Entwicklung entstehen, ist auch eine gute Zusammenarbeit mit dem Kunden anzustreben, um möglichst früh davon zu erfahren.

**Kontinuierliche Verbesserung:** Wenn akzeptiert wird, dass ab einem gewissen Grad von Komplexität und Neuigkeit des Produkts die genauen Anforderungen nicht zu Beginn fixierbar sind, kann die Entwicklung als laufende Verbesserung in Richtung der tatsächlichen Kundenwünsche gesehen werden. Dazu ist der Kunde wieder mit den Maßnahmen einzubinden, die schon bei der Situationskenntnis beschrieben wurden (Bewertung von Prototypen, Simulationsmodellen, …).

**Bewältigung von Veränderungen:** Anforderungsmanagement und Änderungsmanagement sind mit dem Kunden für einen effizienten Umgang mit Änderungen zu definieren. Dies wird auch durch die Kommunikation und Zusammenarbeit mit dem Kunden beeinflusst. Preisgestaltung (Wie werden Änderungen verrechnet?) und dazu notwendige Nachverhandlungen von Vertragsdetails können den Umgang mit Veränderungen erschweren.

**Flexibilität:** Wie oben erwähnt, sind es vor allem Anforderungsmanagement und Preis, wo im Bereich des Kunden Flexibilität gewonnen werden kann. Im Sinne des Set-Based Designs kann man auch bei der Abstimmung von Anforderungen Flexibilität gewinnen, wenn man nicht über diskrete Lösungen, sondern über Lösungsbereiche spricht.

**Effizienz:** Durch eine gute Regelung von Kommunikation und Zusammenarbeit mit dem Kunden kann die Effizienz erhöht werden. Durch Prozessanforderungen (z.B. Dokumentation) beeinflusst der Kunde die Effizienz der Entwicklung. Deshalb soll verstanden werden, welchen Zweck der Kunde mit diesen Anforderungen verfolgt, um die Umsetzung so effizient wie möglich gestalten zu können.

**Effektivität:** Nachdem die Erfüllung der Kundenwünsche der Zweck der Entwicklung ist, kann jede Maßnahme zur besseren Erfüllung dieses Zwecks auch als Effektivitätssteigerung gesehen werden, sollte der Ressourcenaufwand dafür nicht zu hoch sein. Alle bereits beschriebenen Maßnahmen zur besseren Erfüllung der tatsächlichen Kundenanforderungen können also auch die Effektivität steigern. Bei einer Entwicklung in Reihenfolge der Priorität der Anforderungen kann die Effektivität ev. auch weiter gesteigert werden, wenn der wesentliche Nutzen schon vor Projektende erbracht und auf die Entwicklung der unwichtigen Anforderungen verzichtet wird.

### 7.3.2 Organisation

Einerseits können Veränderungen eine Anpassung der Organisation notwendig machen. Andererseits stellt sich die Frage, wie man die Organisation gestalten kann, um die Teilfunktionen der Agilität zu verbessern.

**Situationskenntnis:** Für eine gute und aktuelle Kenntnis der Situation ist eine gute und häufige Informationsverteilung (Kommunikation) notwendig. Es ist aber ebenso notwendig, dass das Wissen und die Erfahrung zur Verfügung steht, um die Informationen interpretieren zu können und ihre Relevanz zu erkennen. In kleinen Teams funktioniert die Kommunikation normalerweise effizienter. In großen Teams steht aber mehr Erfahrung und Wissen zur Verfügung (besonders in interdisziplinären Teams). Eine zweckmäßige Teamzusammensetzung ist Bestandteil der Organisation und kann die Situationskenntnis erleichtern und verbessern.

**Kontinuierliche Verbesserung:** Einerseits kann die Organisation das Zielobjekt der laufenden Verbesserungen sein. Andererseits soll die Organisation aber auch eine laufende Verbesserung der

Produkte fördern. Das kann sie vor allem mit geeigneter Kommunikation (Besprechungen, gezielte Weitergabe von Informationen) und einem geeigneten Wissensmanagement tun.

**Bewältigung von Veränderungen:** Für einen guten Umgang mit Änderungen am Produkt sind auch organisatorische Fragen wie Verantwortlichkeiten zu klären. Eine schnelle Kommunikation der Änderungen ist ebenso eine Frage der Organisation. An der Organisation selbst durchgeführte Änderungen sollen aber auch effizient gehandhabt werden. Dies ist z.B. zu erwarten, wenn das Team die Organisation selbst verändern darf. Die Organisation soll auch sicherstellen, dass sämtliches Wissen (und damit die Personen, welche dieses besitzen), das zur Bewältigung der Veränderungen notwendig ist, abrufbar ist.

**Flexibilität:** Die Möglichkeit eines schnellen Hinzuziehens von Mitarbeitern, deren Fähigkeiten nach einer Veränderung rasch notwendig geworden sind, kann auch als Flexibilität der Organisation betrachtet werden, ebenso wie sonstige Anpassungen, die während eines Projekts noch durchgeführt werden können, z.B. Änderungen bei Kapazitäten oder Verantwortung. Die Organisation kann aber auch die Flexibilität der Mitarbeiter beeinflussen, z.B. Eigeninitiative und Improvisation fördern oder auch erschweren oder verhindern.

**Effizienz:** Die Organisation kann die Effizienz der Entwicklung wesentlich beeinflussen. Z.B. kann durch flache Hierarchien und kurze Kommunikationswege (kleine, örtlich nicht getrennt Teams, wenig Dokumentation) die Effizienz gesteigert werden. Ebenso können bei dezentraler Wissensverteilung auch Entscheidungen dezentral schneller getroffen werden. Normalerweise weiß das Team am besten, wie es effizient arbeitet und sollte deshalb Mitsprache bei der Organisationsgestaltung bekommen.

**Effektivität:** Über den geringeren Ressourcenaufwand, der mit organisatorischen Maßnahmen erreicht werden kann, wird auch die Effektivität gesteigert.

### 7.3.3 Mitarbeiter

Genaugenommen sind die Mitarbeiter in der Entwicklung die einzigen „Wesenheiten“, die Agilität als Fähigkeit besitzen können. Die Förderung der Mitarbeiter kann alle Teilfunktionen der Agilität steigern. Deshalb sollen im Rahmen einer agilen Entwicklung die Mitarbeiter in den Mittelpunkt gestellt werden. Zumindest soll ein agiles Verhalten der Mitarbeiter nicht verhindert werden.

**Situationskenntnis:** Die Mitarbeiter nehmen die Informationen für Kenntnis und Verständnis der Situation auf. Sie sollen motiviert werden, das auch auf Eigeninitiative zu tun und relevante Informationen in die Entwicklung einzubringen. Zur Analyse der Informationen ist Wissen und Erfahrung notwendig, z.B. um den tatsächlichen Kundenwunsch zu erkennen. Auch Interdisziplinarität kann große Vorteile im Erkennen der Situation und Antizipieren zukünftiger Veränderungen bringen. Zusammenarbeit und Kommunikation der Mitarbeiter sollen gefördert werden. Nutzbare Zwischenergebnisse oder Variantenvergleich bieten nicht nur dem Kunden, sondern auch den Mitarbeitern die Gelegenheit, möglichst früh über das spätere Systemverhalten zu lernen.

**Kontinuierliche Verbesserung:** Die Mitarbeiter sollen eine kontinuierliche Verbesserung als Sinn jeder Entwicklung ansehen und motiviert werden, dies in Eigeninitiative umzusetzen. Änderungen sollen als gewöhnlicher Teil der Entwicklung angesehen werden und nicht als Fehler. Die Mitarbeiter sollen auch motiviert werden zu lernen und sich selbst ständig zu verbessern. Dazu soll auch der

Erfahrungsaustausch gefördert werden. Ebenso sollen die Mitarbeiter dazu motiviert werden, die Organisation und den Prozess laufend zu verbessern.

**Bewältigung von Veränderungen:** Der Umgang mit Änderungen (Änderungsmanagement) soll so gestaltet werden, dass er von den Mitarbeitern nicht als lästige Bürokratie angesehen wird. Wenn nötig und zielführend soll auch Improvisation ermöglicht werden. Die Übertragung von Verantwortung für Prozesse und Organisation an das Team macht Änderungen in diesen Bereichen einfacher. Von gut qualifizierten und erfahrenen Mitarbeitern ist generell zu erwarten, dass diese mit Änderungen besser umgehen können. Ebenso können sie spätere Änderungen eventuell schon antizipieren (z.B. hinsichtlich des tatsächlichen Kundennutzens) und dadurch vermeiden bzw. früher in der Entwicklung berücksichtigen.

**Flexibilität:** Die Mitarbeiter sollen selbst flexibel sein (z.B. Bereitschaft haben, zur Übernahme neuer Rollen oder Tätigkeiten, zum Erlernen neuen Wissens oder an anderen Orten zu arbeiten). Ausbildung kann auch als Maßnahme zur Steigerung der Flexibilität der Mitarbeiter gesehen werden, da sie dadurch mehr Entwicklungsaufgaben lösen können sollten. Wissen und Erfahrung der Mitarbeiter ist auch entscheidend für die flexible Gestaltung von Organisation, Prozessen und Produkten.

**Effizienz:** Die Effizienz der Mitarbeiter soll durch eine gute Organisation und gute Prozesse gefördert und nicht durch sinnlose Prozessanforderungen behindert werden. Die Mitarbeiter sollen dazu ein Mitspracherecht bei Prozessen haben. Wesentlich für die Effizienz der Mitarbeiter ist auch eine gute Abstimmung von Kommunikation und Dokumentation. Zusammenarbeit und die Verbreitung von implizitem Wissen sollen gefördert werden. Die Kommunikation kann auch effizienter ablaufen, wenn im Sinne des Set-Based Design am Anfang der Entwicklung über Lösungsbereiche und nicht über konkrete Lösungen gesprochen wird.

Eine höhere Effizienz ist natürlich auch von erfahrenen und hochqualifizierten Mitarbeitern zu erwarten. Motivierte Mitarbeiter, die sich auf ein Projekt konzentrieren können (angemessene Arbeitszeit, nicht zu viele Projekte gleichzeitig) arbeiten normalerweise auch effizienter. Aus psychologischen Gründen kann man durch häufige Deadlines (Entwicklung in kleinen Schritten) auch die Effizienz steigern. Insgesamt scheinen die Mitarbeiter der wesentliche Faktor für die Effizienz der Entwicklung zu sein. Deshalb soll zur Effizienzsteigerung die ganze Entwicklung auf die Mitarbeiter ausgelegt werden.

**Effektivität:** Es gelten hier dieselben Punkte wie bei der Effizienz, die für eine niedrigere Ressourcenbelastung sorgen sollen. Zusätzlich können die Mitarbeiter die Effektivität noch steigern, indem sie die tatsächlichen Wünsche des Kunden erkennen und dahingehend entwickeln.

### 7.3.4 Entwicklungsprozess

Hier soll dargestellt werden, was hinsichtlich des Entwicklungsprozesses getan werden kann, um die Teilfunktionen der Agilität zu unterstützen. Mit dem Prozess kann man die Agilität aber nicht nur unterstützen, nicht angemessene Prozesse können die Agilität der Mitarbeiter auch behindern. Deshalb können auch dahingehende Abhilfemaßnahmen als agilitätssteigernde Maßnahmen gesehen werden.

**Situationskenntnis:** Zur Verbesserung der Situationskenntnis können im Prozess (sich wiederholende) Schritte vorgesehen werden, die eine Analyse der Situation zum Inhalt haben. Der Prozess kann auch so

gestaltet sein, dass er Zusammenarbeit im Team und zwischen (multidisziplinären) Teams, Kommunikation (tägliche/häufige Meetings,…) und Wissensaustausch (auch von implizitem) fördert.

Der Prozess soll es möglich machen, Fehler möglichst früh zu erkennen und Informationen über das spätere Systemverhalten möglichst früh zu generieren. Dazu sollen Zwischenergebnisse, die einen hohen Erkenntnisgewinn erwarten lassen, mit höherer Priorität entwickelt und getestet werden. Einbindung von Kunden oder Anwendern und nutzbare Zwischenergebnisse fördern den Erkenntnisgewinn zusätzlich. Aber auch andere spätere Prozessphasen, die Änderungen notwendig machen können, wie z.B. die Produktion, sollen früh berücksichtigt werden.

Der Prozess kann auch so gestaltet sein, dass Entscheidungen lange offengehalten werden, z.B. mit Set-Based Design. Dadurch hat man mehr Zeit, um Kenntnisse über Machbarkeit und Leistungsfähigkeit des Produkts oder die Situation im Allgemeinen zu erlangen. Der Erkenntnisgewinn über das Produkt kann mit Prototypen, Simulationen und Varianten unterstützt werden. Durch kontinuierliche Integration stehen immer die aktuellen Entwicklungsstände aller Komponenten für die Analyse zur Verfügung.

Der Prozess kann aber auch Kenntnisse über sich selbst fördern, z.B. kann bei Entwicklung in kleinen Schritten durch Vergleich der jeweils erledigten Aufgaben auf die Prozessgeschwindigkeit geschlossen werden.

**Kontinuierliche Verbesserung:** Der Prozess kann explizit so gestaltet werden, dass z.B. durch Entwickeln in kleinen Schritten eine kontinuierliche Verbesserung erzielt wird. Mit einer Einbindung des Kunden kann eine kontinuierliche Verbesserung in die Richtung des tatsächlichen Kundennutzens gelenkt werden. Beim Set-Based Design ist auch eine kontinuierliche Weiterentwicklung möglich, nämlich in Richtung des am besten machbaren Designs. Ist der Prozess nicht explizit so ausgelegt, dass er eine kontinuierliche Verbesserung unterstützt, so soll er die kontinuierlichen Verbesserungsversuche der Mitarbeiter zumindest nicht behindern wie dies z.B. ein Wasserfallprozess mit zu viel Bürokratie häufig tut.

Der Prozess soll aber auch das kontinuierliche Lernen der Mitarbeiter, die laufende Verbesserung des Prozesses selbst und die der Organisation unterstützen. Dazu können häufige Lessons Learned Workshops vorgesehen sein.

**Bewältigung von Veränderungen:** Gute Prozesse für das Änderungsmanagement sollen einen effizienten Umgang mit Änderungen erlauben. Durch eine Priorisierung der Änderungen, kann man die Effizienz im Umgang mit ihnen weiter verbessern. Änderungen sollen gut und schnell kommuniziert werden. Wenn nicht alles detailliert im Voraus geplant ist, können Änderungen später auch leichter eingebracht werden. Änderungen am Prozess funktionieren schneller, wenn das Team dazu ermächtigt ist.

**Flexibilität:** Der Entwicklungsprozess selbst soll flexibel sein, falls unerwartete Situationen nach einer Änderung der Vorgehensweise verlangen. Zumindest sollen notwendige Änderungen nicht prinzipiell verhindert werden. Eine einfache oder modulare Gestaltung des Prozesses und eine Ermächtigung des Teams zur Änderung fördern dies. Hinsichtlich Änderungen am Produkt soll der Prozess ein flexibles Anforderungsmanagement unterstützen. Durch Set-Based Design kann er Flexibilität gegenüber späten Änderungen bieten. Auch Varianten oder Pufferzeiten können Flexibilität bringen.

**Effizienz:** Der Prozess soll effizientes Arbeiten der Mitarbeiter ermöglichen. Er soll einfach sein und keine unnötigen Schritte enthalten (z.B. nicht notwendige Dokumentation). Er soll auch eine effiziente Kommunikation fördern, z.B. wird bei Set-Based Design lange nur über Lösungsbereiche gesprochen, weshalb nicht immer jedes Detail ausdiskutiert werden muss. Auch die Planung kann effizienter werden, wenn zu Beginn nicht alle Details geplant werden, vor allem wenn diese ohnehin Änderungen unterworfen sind. Die Effizienz kann auch mit Test-Driven Development erhöht werden, weil dadurch nur genau so weit entwickelt wird, bis die Testkriterien erfüllt sind. Durch die psychologische Wirkung können häufige Deadlines die Effizienz der Mitarbeiter erhöhen. Timeboxing forciert die Einhaltung der Deadlines und kann auch die Effizienz von Meetings verbessern. Nachdem die Mitarbeiter normalerweise wissen, wie sie am effizientesten arbeiten, sollte man sie in die Gestaltung des Prozesses einbeziehen.

**Effektivität:** Auf der Aufwandsseite wird die Effektivität durch dieselben Maßnahmen unterstützt wie die Effizienz. Um die Effektivität weiter zu steigern, kann der Prozess auch speziell auf die Erreichung der tatsächlichen Kundenanforderungen ausgelegt werden, z.B. durch Entwicklung in kleinen Schritten, die vom Kunden getestet und bewertet werden und Nutzung des Feedbacks zur weiteren Richtungsbestimmung der Entwicklung. Aber auch andere Qualitätsmaßnahmen, die das Endergebnis näher an die Kundenwünsche heranbringen, steigern die Effektivität.

### 7.3.5 System

Hier soll die Frage beantwortet werden, was man im Zusammenhang mit dem Produkt tun kann, um die Teilaspekte der Agilität zu verbessern. Die Flexibilität welche in der Entwicklung zur Verfügung steht, ist natürlich wesentlich von der Anpassbarkeit des Produkts abhängig. Häufig ist das Produkt auch ein Kriterium für die Machbarkeit agiler Praktiken. Deshalb werden hier auch Möglichkeiten aufgezählt, das System so zu gestalten, damit agile Praktiken (z.B. die Entwicklung in kleinen Schritten) umgesetzt werden können.

**Situationskenntnis:** Gestaltet man das System so, dass Zwischenergebnisse nutzbar sind, ist der Kunde zu einem besseren Feedback fähig. Eventuell kann man auch Systemfeedback, also Informationen aus bereits in Verwendung befindlichen Systemen, für eine bessere Situationskenntnis verwenden.

**Kontinuierliche Verbesserung:** Die Machbarkeit der kontinuierlichen Verbesserung am Produkt hängt wesentlich von der Produktgestaltung ab (Anpassbarkeit). Eventuell kann es sogar noch nach der Einführung und anhand von erhaltenem Systemfeedback weiter verbessert werden.

**Bewältigung von Veränderungen:** Ein leichter anpassbares Produkt lässt auch ein einfacheres Änderungsmanagement zu. Eventuell ist bei Änderungen gar keine zusätzliche Entwicklung notwendig, weil das Produkt einfach „umgeschaltet“ werden kann (auch möglich, wenn Reserven im Leistungspotential vorhanden sind).

Durch Wiederverwendung von Modulen oder bei Plattformen kann in einem gewissen Bereich auch schnell auf Veränderungen reagiert werden.

**Flexibilität:** Die Möglichkeiten die das Produkt zur Anpassung bietet, sind wesentlich für die Flexibilität, die im SE zur Verfügung steht. Erreicht werden soll sie durch eine anpassbare Gestaltung der Systemarchitektur. Das Produkt soll auch möglichst einfach sein (Software kann dazu auch im Laufe der

Entwicklung durch Refactoring nach Änderungen wieder einfacher gestaltet werden). In einem gewissen Bereich erhöhen auch Wiederverwendung und Plattformstrategien die Flexibilität. Ebenso die Möglichkeit aus Varianten wählen zu können, wenn schnell für eine Lösung entschieden werden muss. Auch Reserven im Potential des Produkts erhöhen die Flexibilität.

**Effizienz:** Die Produktgestaltung kann auch die Effizienz der Entwicklung verbessern, z.B. bei Einfachheit. Wiederverwendung oder Plattformstrategien können je nach Projekt auch die Effizienz erhöhen, wenn schon angemessene und bereits entwickelte Lösungen zur Verfügung stehen. Dies gilt natürlich nur für den Einzelfall und muss im Allgemeinen überprüft werden.

**Effektivität:** Das Produkt unterstützt die Effektivität, wenn es genau die tatsächlichen Kundenwünsche erfüllt, sonst aber so einfach wie möglich ist und Ressourcen spart. Um sich den tatsächlichen Kundenwünschen anzunähern, soll das Produkt so gestaltet sein, dass Zwischenergebnisse zur Lenkung der Entwicklungsrichtung verwendet werden können. Eventuell kann man die Effektivität durch die Nutzung von Systemfeedback verbessern, weil man auch dadurch besser erkennen kann, in welche Richtung entwickelt werden soll.

### 7.3.6 Zusammenfassung

Das Ziel dieser Ausführungen war die Erläuterung der Zusammenhänge zwischen den agilen Praktiken untereinander und zu den Teilfunktionen der Agilität.

Es sollte damit aber auch ein Verständnis für das große Repertoire an Möglichkeiten geschaffen werden, welches sich zur Verbesserung der Agilität bietet und welches der ASE Ansatz zur Verfügung stellen soll. Dies kann deshalb mit den Maßnahmen in den Eingriffsbereichen gut getan werden, weil hier ein Überblick über das ganze Repertoire des Ansatzes gegeben werden kann, ohne dass konkrete Praktiken bestimmt werden müssen. Dieses Repertoire ist zusammengefasst noch einmal in Abb. 30 dargestellt.

Versucht man diese Maßnahmen in Beziehung zu den Komponenten des BWI-Hall Ansatzes (Abb. 7) zu setzen, so sind sie unter dem Problemlösungsprozess neben der Systemgestaltung und dem Projektmanagement als möglicher „Umgang mit Änderungen“ zu betrachten (siehe Abb. 39).

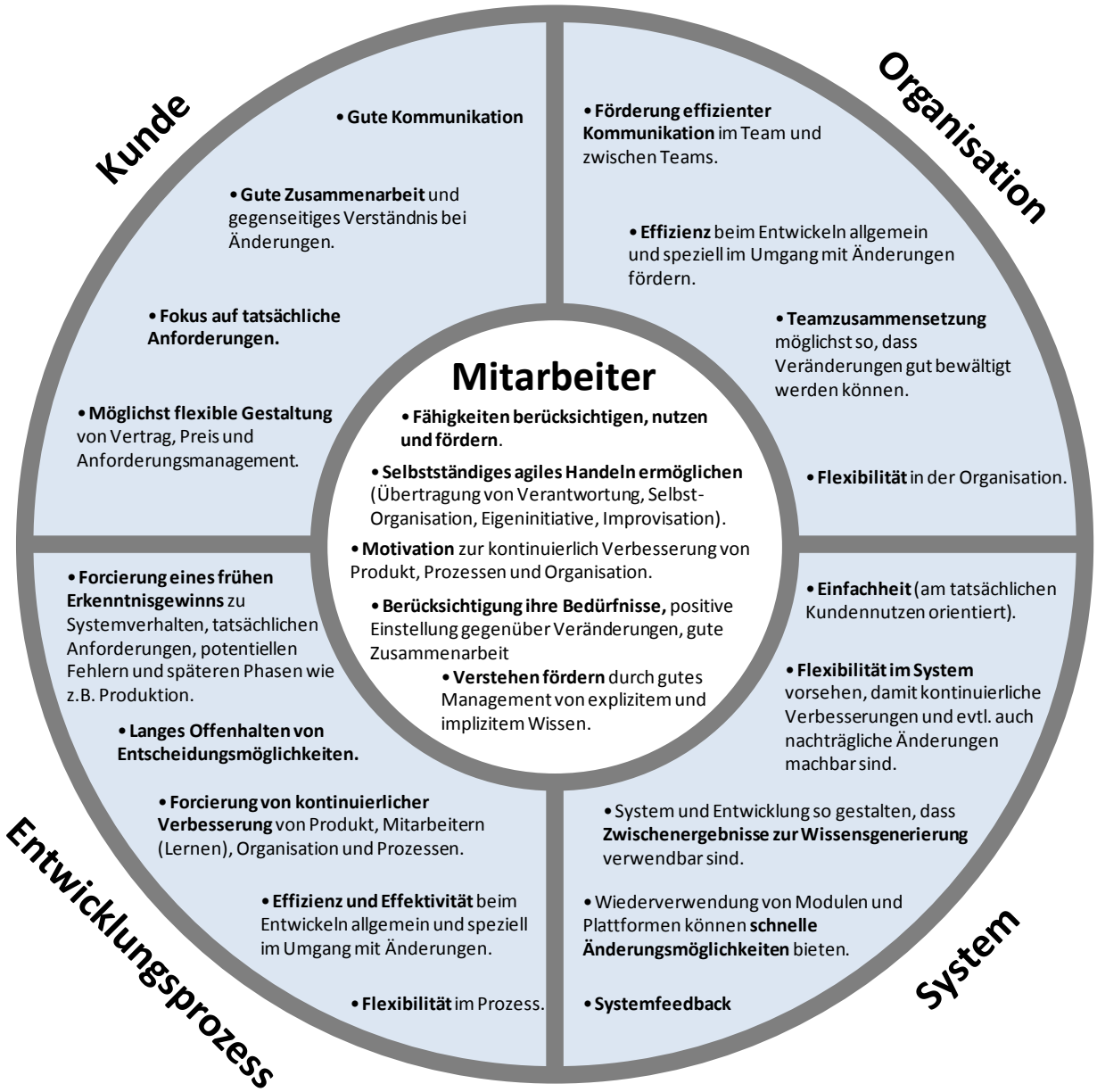


**Abb. 30: Umfang des Agile Systems Engineering Ansatzes**

## 7.4 Auswahl der Praktiken für ASE

Wie bereits jeweils für die Prinzipien und Praktiken angeführt und in Kapitel 6.2 zusammengefasst, können die Prinzipien und Praktiken vielen Voraussetzungen unterliegen bzw. stellen viele Punkte ein Risiko in der Anwendung dieser dar. Ebenso dienen die einzelnen Prinzipien und Praktiken nur bestimmten Zwecken bzw. fördern nur bestimmte Teilfunktionen der Agilität.

In einem konkreten Anwendungsfall ist also genau zu analysieren welcher Bedarf an Agilität besteht und welche ihrer Teilfunktionen dazu zu verbessern sind. Ebenso ist zu ermitteln, welche Voraussetzungen und Risiken in der Entwicklung und ihrer Umgebung bestehen. Erst damit kann bestimmt werden, welche Prinzipien oder Praktiken sinnvoll zur Steigerung der Agilität angewandt werden können. Um dafür ein Entscheidungshilfewerkzeug zu gestalten, wurden die Faktoren für Bedarf und Voraussetzungen als die 2 Dimensionen einer Matrix dargestellt. Damit können die unterschiedlichen Anwendungsfälle für die Prinzipien und Praktiken grob kategorisiert werden (siehe Abb. 31).

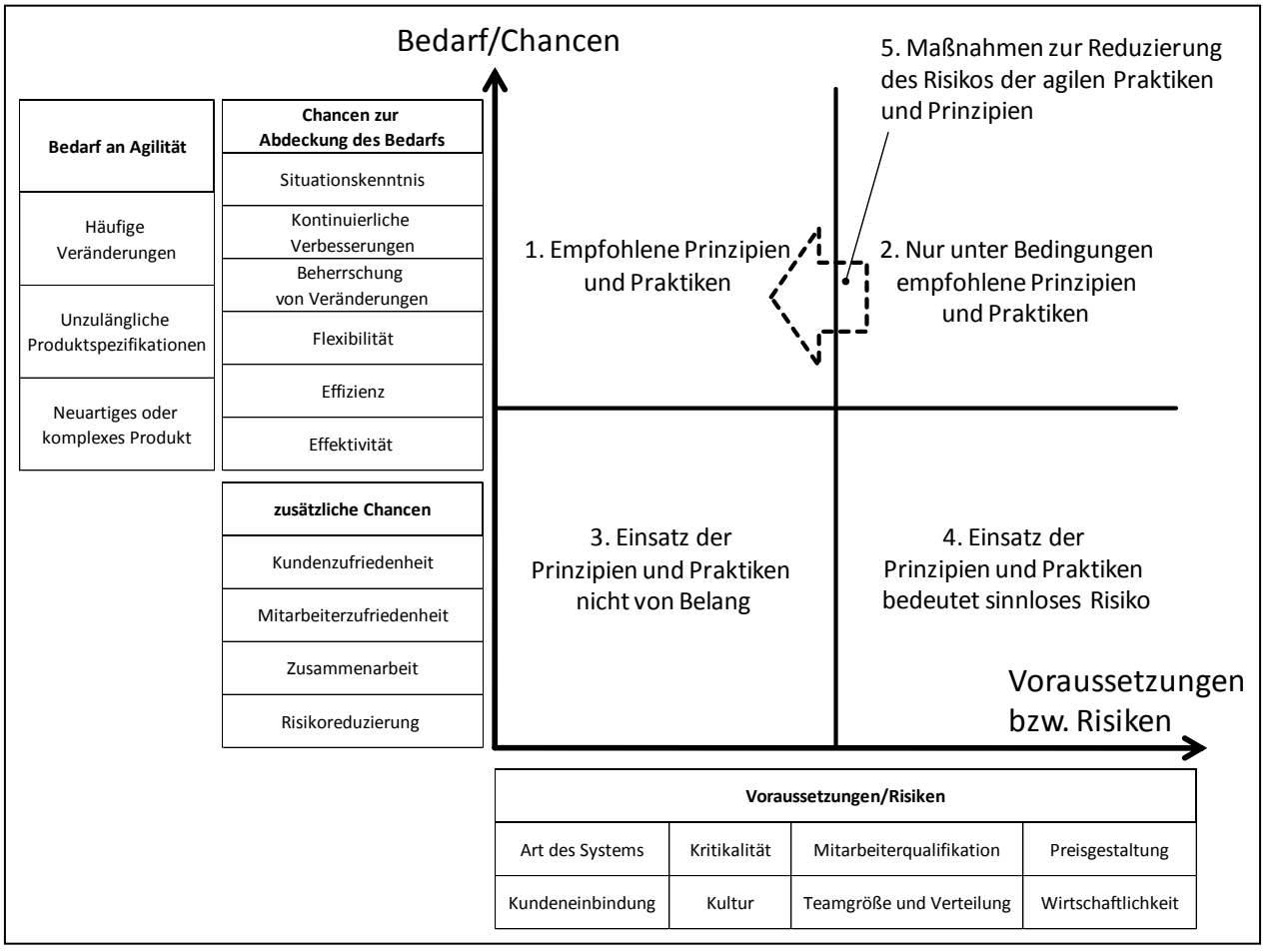


**Abb. 31: Anwendungsmatrix für Prinzipien und Praktiken des ASE**

In dieser „Anwendungsmatrix“ soll der Bedarf an Agilität auf die einzelnen Teilfunktionen der Agilität herunter gebrochen werden. Nachdem kein Prinzip und keine Praktik die Agilität als Ganzes steigern kann, sollen im konkreten Anwendungsfall jene Teilaspekte der Agilität identifiziert werden, deren Steigerung in der aktuellen Situation des Projekts oder der Unternehmung am sinnvollsten hinsichtlich

einer Agilitätssteigerung erscheinen. Damit kann dann nach Praktiken und Prinzipien gesucht werden, welche diese Teilaspekte unterstützen. Ebenso sollen das Projekt und die Unternehmung auf Voraussetzungen und Risiken untersucht werden. Lässt man diese in die Anwendungsmatrix einfließen, so können jene Prinzipien und Praktiken ausgefiltert werden, deren Risiko nicht mehr akzeptabel ist. Als weiteres Kriterium zur Auswahl können auch zusätzliche, nicht direkt in Zusammenhang mit Veränderungen stehende Chancen (Zusatznutzen, siehe Kapitel 6.3) in Betracht gezogen werden.

Das Ziel dieser Anwendungsmatrix ist also nicht die genaue Definition der Anwendungsgebiete, sondern die Erstellung eines Werkzeugs, das Entscheidungshilfen bei der praktischen Anwendung des ASE Ansatzes bieten soll. Die Bereiche, in welche die Prinzipien und Praktiken damit eingeordnet werden können, stellen sich in Abb. 31 folgendermaßen dar:

1. Diese Prinzipien bieten eine hohe Chance einen (Teil-) Nutzen hinsichtlich Agilität zu erreichen. Sie können andererseits mit geringem Risiko eingesetzt werden. Damit werden sie vom ASE Ansatz bedingungslos empfohlen.
2. Hier besteht ein erhöhtes Risiko. Wenn möglich sollen andere Prinzipien und Praktiken verwendet werden, die den gleichen Nutzen stiften können. Stehen keine anderen zur Verfügung, sind Chancen und Risiken genau abzuwägen.
3. Prinzipien und Praktiken, die hier eingeordnet wurden, sind belanglos. Eventuell kann ein Zusatznutzen ihren Einsatz rechtfertigen.
4. Diese Prinzipien und Praktiken sollen vermieden werden.
5. Die Erklärung dazu erfolgte bereits in Kapitel 6.4.

Um nach einer erfolgten Analyse des Kontexts, die Prinzipien und Praktiken rasch in die Bereiche der Anwendungsmatrix einordnen zu können, wurden die Erkenntnisse zu den Chancen und Risiken der Prinzipien und Praktiken in einer Auswahlmatrix zusammengestellt (siehe Abb. 32).

Darin wird angegeben, welche agilen Teilfunktionen im Endeffekt von den Praktiken gefördert werden können. Ebenso werden die Zusatzfunktionen genannt, welche die Praktiken bieten können. Beides sind Chancen, die von den Praktiken geboten werden können. Es wird auch gezeigt ob sich diese Chancen hinsichtlich Veränderungen außerhalb oder innerhalb der Entwicklung bieten, wie in Kapitel 6.2.2 erklärt wurde. Bei den Veränderungen innerhalb der Entwicklung wurde keine genauere Untergliederung hinsichtlich der Ursache mehr durchgeführt, da die Wirkmechanismen der Praktiken sich dahingehend auch nicht unterscheiden.

Selbstverständlich werden auch die Risiken bzw. Voraussetzungen für die Praktiken aufgezeigt. Diese sind genau zu analysieren, da sie ein ernsthaftes Problem für die Anwendbarkeit von Praktiken darstellen können. Eventuell können die in 6.4 beschriebenen Maßnahmen Abhilfe verschaffen, ansonsten ist aber zu erwarten, dass je nach Systementwicklungsprojekt zahlreiche Praktiken nicht durchführbar sind.

Eine genaue quantitative Bestimmung der Grenzen zwischen den Bereichen in Abb. 31 kann im Allgemeinen nicht durchgeführt werden. Lediglich Praktiken, für die weder ein Bedarfsgrund, noch eine Voraussetzung gegeben ist, können definitiv in Bereich 3 eingeordneten werden. Jeder einzelne Bedarfsgrund und jede einzelne Voraussetzung kann abhängig von der Situation eine andere Einteilung rechtfertigen oder auch nicht. Deshalb ist dies für jedes Projekt neu zu bewerten. Die Darstellung des

Zusammenhangs einer Praktik mit einem Faktor in Abb. 32 soll dazu dienen, die Aufmerksamkeit auf die Untersuchung des Faktors zu lenken.

Als Grundlage für die Bewertung der Chancen (zur Abdeckung des Bedarfs an einer bestimmten agilen Teilfunktion) und Risiken (welche nicht erfüllte Voraussetzungen darstellen) je Praktik dienten die Erkenntnisse aus der Literatur und den empirischen Erhebungen, die in Kapitel 5 dargestellt wurden.

Der AEZ wurde in die Auswahlmatrix ebenso aufgenommen. Er stellt die Summe der Praktiken der Entwicklung in kleinen Schritten, der Kundeneinbindung, des flexiblen Anforderungsmanagements, der Priorisierung und des Timeboxings dar. Damit soll aufgezeigt werden, wie umfassend diese Methode, die das in der Softwareentwicklung vorherrschende Verständnis einer agilen Entwicklung darstellen soll, die Funktionalitäten der Agilität abbilden kann. Als Summe mehrerer Praktiken und wegen den speziellen Ausprägungsformen der Entwicklung in kleinen Schritten, ist sie aber auch von sehr vielen Voraussetzungen abhängig.

Die Entwicklung in kleinen Schritten wird in der Matrix nach den für den AEZ notwendigen Ausprägungsformen bewertet. Deshalb unterliegt sie ebenso zahlreichen Risiken. Auf eine Bewertung je nach Ausprägung der Merkmale wurde verzichtet. Die Entwicklung in kleinen Schritten kann aber in sehr vielen Fällen große Chancen für die Agilität bieten. Und fast für jede Voraussetzung kann sie in einer vom AEZ abweichenden Ausprägungsform angewandt werden. Deshalb wird auf jeden Fall eine Überprüfung der Anwendbarkeit dieser, in Kapitel 5.4.3 beschriebenen, Praktik empfohlen. In 6.4.9 wird näher darauf eingegangen, wie sie zumindest zur Erreichung eines Teilnutzens angewandt werden kann, wenn bestimmte Voraussetzungen nicht gegeben sind. Schon aus organisatorischen Gründen ist zu erwarten, dass der Entwicklungsumfang für ein komplexes System in irgendeiner Art auf kleine Schritte aufgeteilt wird. Die Hinweise in den beiden eben genannten Kapiteln können auch genutzt werden, um Möglichkeiten zu finden, wie mit der vorhandenen Aufteilung eventuell die Agilität verbessert werden kann.

Es sei auch noch erwähnt, dass diese Auflistung der Praktiken und Prinzipien nicht vollständig sein muss. Sie ist durch sorgfältige Recherche nach Praktiken, welche im Zusammenhang mit Agilität genannt wurden, entstanden. Es ist aber durchaus möglich, dass in Zukunft noch mehr dafür anwendbare Praktiken entwickelt werden oder auch jetzt schon existieren. Ebenso muss erwähnt werden, dass die Informationen aus der Matrix nur einen Überblick für eine schnelle Vorauswahl liefern können. Für Detailinformationen wird auf die Beschreibung der Praktiken in Kapitel 5 verwiesen.

Vergleicht man die agilen Prinzipien und Praktiken mit den Komponenten des BWI-Hall Ansatzes (Abb. 7), so sind diese auf unterster Ebene als „agile Techniken“ neben jenen für Systemgestaltung und Projektmanagement zu bezeichnen. Das Auswahlverfahren an sich kann aber durchaus auch einer agilen SE-Philosophie auf höchster Ebene zugerechnet werden (Abb. 39).

| | | Veränderungen | | Chancen (Erfüllung agiler Teilfunktionen) | | | | | | | Risiken (Voraussetzungen) | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Kapitel | Prinzipien und Praktiken für ASE | außerhalb der Entwicklung | innerhalb der Entwicklung | Situationskenntnis | kontinuierliches Verbessern | Bewältigung von Veränderungen | Flexibilität | Effizienz | Effektivität | Zusatznutzen oder Sonstiges<br>K = Kundenzufriedenheit<br>M = Mitarbeiterzufriedenheit<br>Z = Zusammenarbeit<br>R = Risikoreduzierung | Art des Systems | Kundeneinbindung | Kritikalität | Kultur | Mitarbeiterqualifikation | Teamgröße und Verteilung | Preisgestaltung | Wirtschaftlichkeit |
| 4.3.4 | Agiler Entwicklungszyklus (AEZ) | x | x | x | x | x | | x | x | K, R | x | x | x | x | x | x | x | |
| 5.1 | **Kunde** | | | | | | | | | | | | | | | | | |
| 5.1.1 | Fokus auf tatsächlichen Kundennutzen | | x | x | | | | | x | K | | x | | | x | | x | |
| 5.1.2 | Vertrauensvolle Zusammenarbeit statt übertriebene Vertragsverhandlungen | x | x | | | x | x | | | Z | | x | | x | | | x | |
| 5.1.3 | Kunden in Entwicklung einbinden (Feedback) | | x | x | x | | | | x | K, Z, R | x | x | | x | | | | |
| 5.1.4 | Flexibles Anforderungsmanagement | x | x | | | x | x | | | | x | | x | x | x | x | x | |
| 5.1.5 | Priorisierung | x | x | x | | x | | | x | K | | x | | | | | | |
| 5.1.6 | Einsatz eines Product Owners | | x | x | x | | | | x | repräsentiert Kunden | | | | x | x | | | x |
| 5.2 | **Organisation** | | | | | | | | | | | | | | | | | |
| 5.2.1 | Aufteilung großer Teams | | x | x | | | x | x | | Z | | | | | | | | |
| 5.2.2 | Örtliche Zusammenlegung des Teams | | x | x | | | | x | | Z | | | | x | | x | | |
| 5.2.3 | Dynamische Teamauswahl nach benötigten Fähigkeiten | x | x | | | x | x | | | | | | | x | x | | | |
| 5.2.4 | Team Selbstorganisation | x | x | | x | x | x | x | | M, Z | | | x | x | x | x | | x |
| 5.2.5 | Gleichzeitige Mitarbeit an möglichst wenigen Projekten | x | x | | | | x | x | | M | | | | | | | | x |
| 5.2.6 | Pair Programming | | | | | | | x | x | Z, R | | | | x | | x | | x |
| 5.3 | **Mitarbeiter** | | | | | | | | | | | | | | | | | |
| 5.3.1 | Mensch im Mittelpunkt der Entwicklungsprozesse | x | x | x | x | x | x | x | | M, Z | | | x | x | x | x | | |
| 5.3.2 | Ausbildung der Mitarbeiter | x | x | x | x | x | x | x | | R | | | | x | | | | x |
| 5.3.3 | Förderung von Motivation | x | x | x | x | x | x | x | | M | | | | | | | | |
| 5.3.4 | Förderung von Teamwork | | x | x | | | | x | | Z | | | | | | | | |
| 5.3.5 | Optimierung der Kommunikation | | x | x | | x | | x | | Z | | | | | | | | |
| 5.3.6 | Wertschätzung von implizitem Wissen | | | | | | | x | x | M, Z | | | x | x | x | x | | |
| 5.3.7 | Positive Einstellung gegenüber Veränderungen | x | x | | x | x | | x | | M | | | | x | | | | |
| 5.3.8 | Improvisation | x | x | | | x | | | | | | | x | x | x | x | | |
| 5.3.9 | Angemessene Arbeitszeit | | | | | | | x | | M | | | | | | | | x |
| 5.4 | **Entwicklungsprozess** | | | | | | | | | | | | | | | | | |
| 5.4.1 | Prozessgestaltung durch Team | x | x | | | x | x | x | | M, Z | | | x | x | x | x | | x |
| 5.4.2 | Simultaneous Engineering | | x | x | | | | x | | Z | | | | x | | | | |
| 5.4.3 | Entwicklung in kleinen Schritten (AEZ Ausprägungen) | x | x | x | x | x | | x | x | K, R | x | | x | x | x | x | x | |
| 5.4.4 | Kontinuierliche Integration | | x | x | | | | x | | | x | | x | | | | | |
| 5.4.5 | Anchor Point Milestones | | | | | | | | | Z, Synchro-nisierung | | | | | | | | |
| 5.4.6 | Test-Driven Development | | | | | | | x | | | x | | | | | | | |
| 5.4.7 | Prototyping/Simulation | | x | x | | | | | x | K, R | x | | | | | | | x |
| 5.4.8 | Timeboxing | | | | | | | x | | | x | | x | x | | | | |
| 5.4.9 | Laufende Planung | x | x | | | x | | x | | M | | | | x | | x | | |
| 5.4.10 | Pufferzeiten | x | x | | | x | x | | | R | | | | | | | | x |
| 5.4.11 | Laufende Situationsanalyse | x | | x | | | | | | R | | | | | | | | x |
| 5.4.12 | Tägliche/häufige Meetings | | x | x | | | | | | Z | | | | x | | x | | x |
| 5.4.13 | Flexible Prozessgestaltung | x | x | | | x | x | x | | M | | | x | x | x | | | |
| 5.4.14 | Beschränkung auf notwendige Dokumentation | | | | | | | x | x | M | | | x | | | x | | |
| 5.4.15 | Product Team Ownership | | | | | x | x | | | | x | | x | x | x | x | | |
| 5.4.16 | Emerging Architecture | x | x | | x | | x | | | | x | | x | | x | | x | |
| 5.4.17 | Entwicklung von Varianten | x | x | x | | | x | | | | | | | | | | | x |
| 5.4.18 | Set-Based Design | x | x | x | x | | x | x | x | Z, R | | | | x | x | | | x |
| 5.4.19 | Adaptive Product Development Process (Levardy) | | x | | | x | | x | | | x | | | | | | | |
| 5.4.20 | Förderung des Lernens | | | x | x | | | | | R | | | | | | | | |
| 5.5 | **System** | | | | | | | | | | | | | | | | | |
| 5.5.1 | Einfachheit | x | x | | | | x | x | x | | | | x | | | | | |
| 5.5.2 | Anpassbarkeit | x | x | | x | x | x | | | | x | | x | | | | | x |
| 5.5.3 | Förderung nutzbarer Zwischenergebnisse | | | x | | | | | x | K | x | | x | | | | | x |
| 5.5.4 | Reserven im Potential des Produkts | x | x | | | x | x | | | R | x | | | | | | | x |
| 5.5.5 | Wiederverwendung von Modulen | x | x | | | x | x | x | | | x | | | | | | | x |
| 5.5.6 | Produktplattform/Product Line | x | x | | | x | x | x | | | x | | | | | | | x |
| 5.5.7 | Refactoring | | | | | | x | | | | x | | | | | | | |
| 5.5.8 | Systemfeedback | x | | x | x | | | | x | K, R | x | x | | | | | | x |

**Abb. 32: Auswahlmatrix für Prinzipien und Praktiken des ASE**

## 7.5 Vorgehensmodelle für ASE

Analog zur Darstellung des SE Vorgehensmodells aus dem BWI-Hall Ansatz (Abb. 8) soll nun auch für den ASE Ansatz ein Vorgehensmodell gestaltet werden. Aufgrund der sehr unterschiedlichen Voraussetzungen, die in Systementwicklungsprojekten herrschen können, konnte aber kein allgemeingültiges Modell gefunden werden. Als grundlegende Struktur für ein agiles Vorgehensmodell scheint die Entwicklung in kleinen Schritten in Frage zu kommen, die auch in den Methoden der agilen Softwareentwicklung als Vorgehensmodell verwendet wird. Eine im Sinne des AEZ umgesetzte und damit als besonders agil anzusehende Entwicklung in kleinen Schritten ist aber von zahlreichen Voraussetzungen abhängig. Als einzig alternative Grundstruktur, die ebenfalls eine gute Beherrschung von Veränderungen erwarten lässt, wurde das Set-Based Design gefunden. Vor allem in der mit Simultaneous Engineering kombinierten Toyota-Vorgehensweise des Set-Based Concurrent Engineering.

Es ist aber ebenso denkbar das, durch Top-Down Prinzip und Phasen strukturierte, BWI-Hall Vorgehensmodell um agile Prinzipien zu erweitern und als Vorgehensmodell für ASE zu verwenden. Das gilt vermutlich auch für andere traditionelle Vorgehensmodelle, die bei Systementwicklern verwendet werden.

Hier soll nun ein Überblick gegeben werden, nach welchen grundlegenden Strukturen Vorgehensmodelle gestaltet werden können. Dazu werden die eben erwähnten Möglichkeiten dargestellt. Als erstes soll ein zur Agilitätssteigerung modifiziertes BWI-Hall Vorgehensmodell aufgezeigt werden, gefolgt von einem Vorgehensmodell, das auf einer Entwicklung in kleinen Schritten basiert und einem Vorgehensmodel auf der Basis der Set-Based Designs. Alle sollen möglichst allgemein, d.h. nicht auf ein bestimmtes Fachgebiet oder eine bestimme Branche bezogen, gestaltet werden, so dass sie einen möglichst großen Anwendungsbereich haben.

Es werden jeweils die wesentlichen Charakteristika dieser Modelle diskutiert und die Ziele nach denen sie gestaltet wurden. Ein Vergleich der Modelle wird danach in Kapitel 7.5.4 durchgeführt. Zur besseren Erklärung der praktischen Anwendung dieser Modelle wird auf das Fallbeispiel in Kapitel 8 verwiesen.

### 7.5.1 Hinsichtlich Agilität modifiziertes BWI-Hall Vorgehensmodell

*Prämisse: Es soll ein Modell gestaltet werden, welches in seinem Anwendungsbereich nicht oder kaum gegenüber dem traditionellen BWI-Hall Modell beschränkt ist, aber versucht so viele Maßnahmen (aus Abb. 30) wie möglich, zur Agilitätssteigerung umzusetzen.*

Auch wenn unter agiler Entwicklung zumeist eine dem AEZ ähnliche Vorgehensweise verstanden wird, kann auch mit dem BWI-Hall Modell (siehe 2.1.5) eine einigermaßen agile Entwicklung umgesetzt werden. Wendet man dieses Modell nicht stur und starr an, lässt sich die Agilität erhöhen, z.B. durch Wiederholungszyklen (Haberfellner, et al., 2002 S. 97ff). Es soll nun ein Vorgehensmodell dargestellt werden, das auf dem BWI-Hall Modell basiert und alle machbaren Möglichkeiten zur Steigerung der Agilität ausnutzt. Dieses wird zur Agilitätssteigerung auch so weit modifiziert, wie es der vorgesehene Anwendungsbereich zulässt. Dieser soll so definiert werden, dass das modifizierte BWI-Hall Modell im Vergleich zum traditionellen keinen zusätzlichen Voraussetzungen unterliegt. Wie schon diskutiert wurde, sind viele Praktiken von Voraussetzungen betroffen, die in Systementwicklungsprojekten oft nicht gegeben sind. Das hier dargestellte „Agile BWI-Hall Modell“ soll sich deshalb auf jene Praktiken

beschränken, welche für dieselben Einsatzgebiete geeignet sind, wie das traditionelle BWI-Hall Modell. Innerhalb dieses Bereichs soll es aber modifiziert werden, um den Veränderungen innerhalb und außerhalb der Entwicklung bestmöglich Rechnung zu tragen.

Als Grundstruktur wird also die des traditionellen BWI-Hall Vorgehensmodells verwendet (Abb. 8). Es wird das Top-Down Prinzip angewandt und eine Gliederung in Phasen vorgesehen. Eine Entwicklung von Varianten und stufenweise Auswahl dieser wird ebenso empfohlen. Damit wurden die Module 1-3 beibehalten. In den einzelnen Phasen ist aber nicht vorgesehen, den Problemlösungszyklus (Modul 4) nur ein einziges Mal oder mit Wiederholungszyklen ein paar wenige Male implizit durchlaufen zu lassen. Stattdessen wird er von vorneherein explizit zyklisch angesetzt, wobei neben den expliziten Zyklen natürlich auch implizite stattfinden können (Erklärung explizit/implizit in 5.4.3). Es kann dadurch innerhalb der einzelnen Phasen eine Entwicklung in kleinen Schritten umgesetzte werden, die zumindest teilweise den Kriterien des AEZ entspricht. Damit soll ein früheres Erkennen von potentiellen späteren Änderungen forciert werden. Außerdem sind gewisse Änderungen, der Phase entsprechend, auch noch leichter einzubringen. Wiederholungszyklen werden auch nicht als Fehler angesehen, sondern als gewöhnliche Entwicklung akzeptiert bzw. sogar positiv als Aktivitäten zur Verbesserung von Lösungen gesehen. Die wesentliche Modifizierung des BWI-Hall Vorgehensmodells aus Abb. 8 betrifft also den PLZ (Modul 4), der nun explizit zyklisch behandelt wird.

Die in den jeweiligen Phasen stattfindende Entwicklung in kleinen Schritten ist dabei nicht streng geregelt. Nutzbare Zwischenergebnisse und die Einbindung des Kunden werden nicht gefordert, sondern sind je nach Möglichkeit umzusetzen. Die Zyklusdauern können je nach Komponente unterschiedlich sein oder auch je Zyklus variieren (z.B. wenn unterschiedliche Entwicklungsaufwände dies erfordern). Die Realisierung des Systems erfolgt nach den Planungsphasen. Zwischenergebnisse werden in Form von Simulationen, Wegwerf-Prototypen o.Ä. geliefert. Es wird die Entwicklung in kleinen Schritten also in sehr vielen Punkten abweichend vom AEZ durchgeführt und für einige Merkmale wird nur versucht das Minimalkriterium, welches für eine Steigerung der Agilität notwendig ist, zu erreichen (siehe 6.4.9).

Insgesamt kann man das Modell dadurch beschreiben, dass agile Praktiken soweit umgesetzt werden sollen, wie möglich ist. Die nicht notwendige Starrheit wurde entfernt und eine Anpassung an die gewöhnliche Arbeitsweise in der Entwicklung vorgenommen. Und diese funktioniert in gewisser Weise immer in kleinen Schritten. Es wird aber nicht das traditionelle Verständnis des SE in Frage gestellt, das eine möglichst genaue Abklärung der Anforderungen zu Beginn und eine lineare, phasenweise Entwicklung bis zur Fertigstellung des Produkts vorsieht. Die starke Einbindung des Kunden und eine besondere Art des Systems (z.B. nachträgliche Anpassbarkeit) sind nur notwendig wenn dies situationsbedingt ausdrücklich gefordert und vereinbart wird. Die Vereinbarung von Fixpreisen und die nachverfolgbare Entwicklung und Dokumentation von sicherheitskritischen Anforderungen sind leicht möglich. Die Implementierung von Änderungen kann aber, vor allem wenn sie einen Rückgriff auf frühere Phasen erfordert, doch einen beträchtlichen Aufwand darstellen.

In Abb. 33 soll dieses Modell nun skizziert werden. Die Bezeichnungen der Elemente in diesem Modell und ihre Inhalte orientieren sich stark an den Bezeichnungen aus dem traditionellen BWI-Hall Vorgehensmodell (Haberfellner, et al., 2002). Teilweise wurden auch Anpassungen hinsichtlich des in der Industrie vorherrschenden Verständnisses und Sprachgebrauchs vorgenommen. Die meisten

Bezeichnungen wurden aber beibehalten, da sie die, für die Entwicklung von sehr unterschiedlichen Systemen, notwendigen Sachverhalte sehr gut verallgemeinert darstellen.

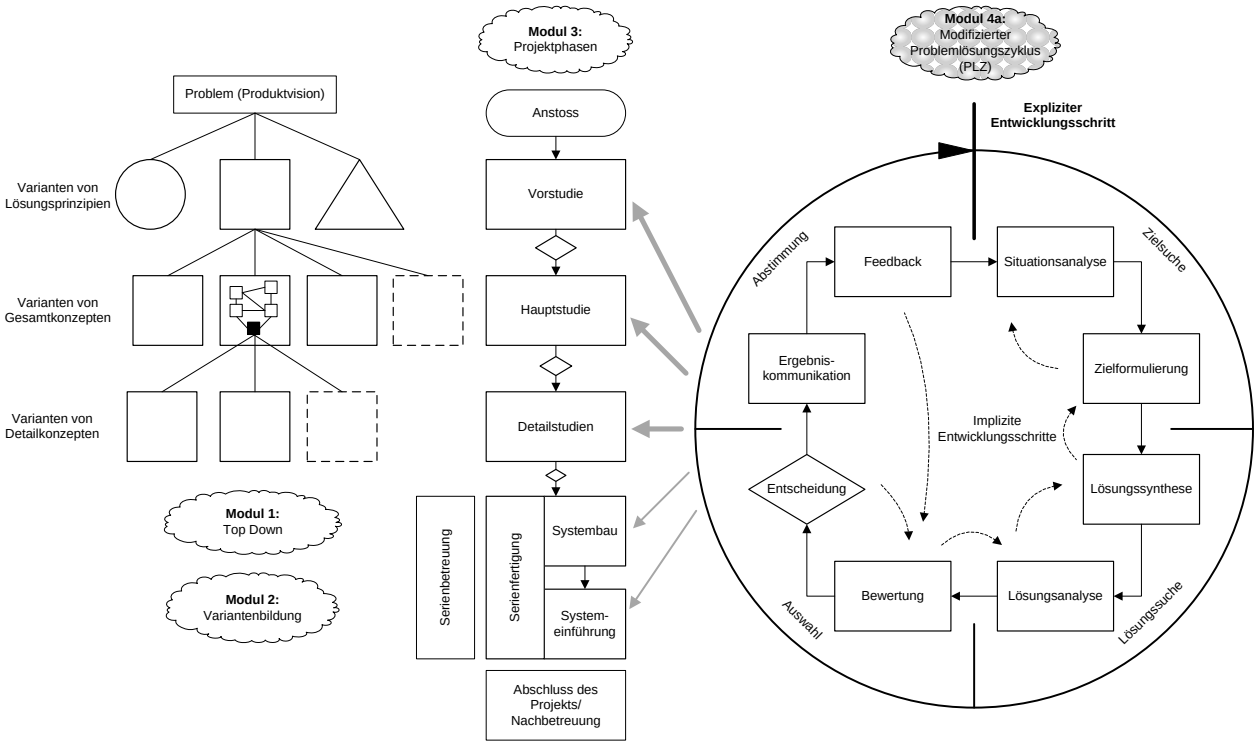


**Abb. 33: Agiles BWI-Hall Vorgehensmodell**

Die Systementwicklung nach dem agilen BWI-Hall Modell beginnt mit einem Problem. das mit einem durch eine Vision beschriebenen Produkt gelöst werden soll. In den Planungsphasen (Vorstudie, Hauptstudie, Detailstudie) soll die Entwicklung mit zunehmendem Detaillierungsgrad durchgeführt werden. Es soll auch das Variantenprinzip umgesetzt werden. Der Problemlösungszyklus soll hier aber nicht linear mit Wiederholungszyklen nur im Bedarfsfall dargestellt werden. Eine zyklische Darstellung entspricht vielmehr der tatsächlichen Vorgehensweise jeder Entwicklung und unterstützt die Agilität besser. Durch diese Darstellung soll sofort klar werden, dass Entwicklungstätigkeit immer wiederholende Schritte enthält und diese kein Fehler sein müssen. Durch die explizite Ausprägung der Entwicklung in kleinen Schritten sollen andere ihrer Vorteile (z.B. die in 5.4.3 beschriebene stimulierende Wirkung häufiger Deadlines) zumindest teilweise erreicht werden. Auch wenn keine nutzbaren Zwischenergebnisse zur Verfügung stehen, kann intern die Abstimmung verbessert werden, wenn eine zyklische Kommunikation der Ergebnisse vorgesehen ist.

Durch die zyklisch angesetzte Entwicklung wird die Situationsanalyse laufend durchgeführt. Die Zielformulierung wird hier so verstanden, dass die Ziele ausgewählt werden sollen, die im laufenden Schritt zu erreichen sind. Lösungssynthese und Lösungsanalyse werden, wie im traditionellen BWI-Hall Modell, als kreativer und destruktiver (Analyse der Machbarkeit) Schritt der Lösungssuche verstanden. Bewertung und Entscheidung dienen der Lösungsauswahl für den jeweiligen Entwicklungsschritt. Nach der Lösungsauswahl wurde noch die „Abstimmung“ eingeführt, um das Kommunizieren des

Zwischenergebnisses und das Einholen von Feedback hervorzuheben. Die Abstimmung kann extern zum Kunden hin erfolgen, z.B. um die tatsächlichen Anforderungen besser abzustimmen, wenn dies möglich und erwünscht ist. Sie kann aber auch intern erfolgen zwischen den Entwicklungsbereichen (z.B. mechanische HW, elektronische HW, SW, Produktionsdesign, etc.). Das einmalige, zeitlich geplante Durchlaufen dieses Zyklus wird als „expliziter Entwicklungsschritt“ bezeichnet. Es können aber auch beliebige Rückgriffe und Wiederholungszyklen ungeplant durchgeführt werden, die hier als „implizite Entwicklungsschritte bezeichnet werden.

Die Fertigung des Systems erfolgt nach den Planungsphasen. In der Serienfertigung wird das gewöhnlich als Abschluss des Entwicklungsprojekts gesehen und mit SOP (Start of Production) bezeichnet. In der Einzelfertigung kann man noch den Systembau und die Systemeinführung unterscheiden. Eventuelle weitere Entwicklungstätigkeiten (Servicetätigkeiten, Rückholaktionen, Software-Updates, etc.) während der Serienfertigung oder auch noch danach werden mit Serienbetreuung und Nachbetreuung bezeichnet.

Eine Entwicklung nach diesem Vorgehensmodell kann die Agilität nur beschränkt steigern. Dafür ist es für praktisch alle Anwendungsfälle geeignet, in denen auch das traditionelle BWI-Hall Vorgehensmodell angewandt werden kann. Ein wichtiger Punkt für die umsetzbare Agilitätssteigerung ist die geschickte Planung von expliziten Entwicklungsschritten.

### 7.5.2 Auf einer Entwicklung in kleinen Schritten basierendes AEZ-SE Vorgehensmodell

*Prämisse: Es soll ein Modell gestaltet werden, das möglichst den vollen Umfang aller Maßnahmen aus Abb. 30 unterstützt, auch wenn es zahlreichen Voraussetzungen unterliegt (insbesondere ist dabei notwendig, dass die Art des Systems eine Entwicklung in kleinen Schritten, möglichst nach AEZ Ausprägung, erlaubt).*

In diesem zweiten agilen Vorgehensmodell soll versucht werden, den AEZ so weit wie möglich im Systems Engineering umzusetzen. Es soll deshalb im weiteren „AEZ-SE Vorgehensmodell“ genannt werden. Die Grundstruktur dieses Modells ist eine Entwicklung in kleinen Schritten, bei welcher der Entwicklungsumfang in explizite Entwicklungsschritte aufgeteilt wird. Diese startet nachdem in einer anfänglichen Projektplanungsphase ein flexibler Anforderungskatalog (im Sinne eines Product-Backlogs) erstellt wurde. Auch wenn der AEZ möglichst umfassend umgesetzt werden soll, werden hier einige der in Kapitel 6.4 beschriebenen Möglichkeiten zur Umgehung von Voraussetzungen ausgenutzt. Um den Eigenschaften von Hardware Rechnung zu tragen, ist nicht vorgesehen, in jedem Zyklus auch eine Realisierung durchzuführen. Deshalb wird in der Situationsanalyse entschieden, ob im nächsten Schritt eine Realisierung oder ein planender Entwicklungsschritt durchgeführt werden soll. Auf jeden Fall ist aber vorgesehen, diese Schritte alternierend durchzuführen und nicht die Realisierungsschritte erst nachdem sämtliche Planungsschritte durchgeführt wurden. Das würde wieder dem vorher beschriebenen agilen BWI-Hall Vorgehensmodell entsprechen. Hier soll aber auf jeden Fall Feedback auf realisierte Zwischenergebnisse zur Informationsgewinnung für die weitere Entwicklung verwendet werden. Deshalb soll versucht werden so oft wie möglich ein Zwischenergebnis zu realisieren und dieses dem Kunden zu demonstrieren.

Eine Kommunikation der Zwischenergebnisse kann nach beiden Arten von Entwicklungsschritten durchgeführt werden (auch Planungsergebnisse können mit dem Kunden abgestimmt werden), kann aber auch ausgelassen werden, wenn dies nicht sinnvoll ist. Die Entwicklung endet, wenn bei der Ergebniskommunikation vom Kunden festgestellt wird, dass alle Anforderungen erfüllt sind. Bei Einzelfertigung ist das System dann auch schon realisiert, weil dies abwechselnd mit den Planungsphasen durchgeführt wurde. Es ist bei diesem Vorgehensmodell aber auch denkbar, nur einen Prototypen in kleinen Schritten herzustellen und danach eine Serienfertigung zu starten. Die Produktionsplanung ist dabei als einer oder mehrere explizite Entwicklungsschritte zu sehen. Dieses Vorgehensmodell forciert eine Verfolgung des tatsächlichen Kundennutzens, da es vom Feedback des Kunden gelenkt wird. Die Vorgehensweise macht auch ein sehr flexibles Anforderungsmanagement und eine leichte Einbringung von externen Veränderungen möglich. Durch frühe Realisierungsschritte können auch Informationen genutzt werden, die sonst erst spät in der Entwicklung verfügbar wären. Der Prozess ist im Detail sehr flexibel, da er keine genaue Abfolge von Entwicklungstätigkeiten oder Phasen vorsieht. In Abb. 34 wird das Modell grafisch dargestellt.

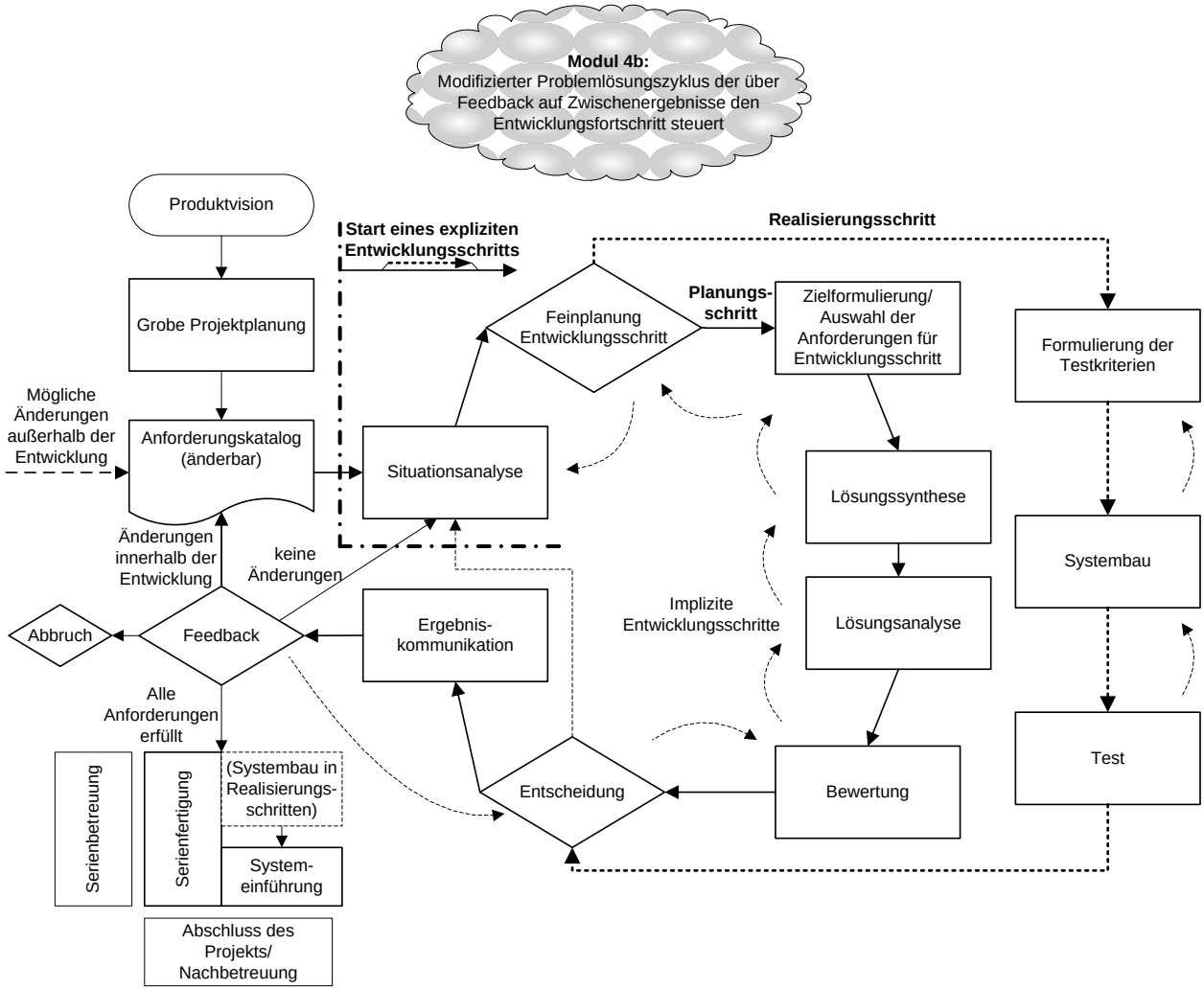


**Abb. 34: AEZ-SE Vorgehensmodell**

Der Schritt der Situationsanalyse ist hier etwas umfangreicher. Der Entwicklungsablauf ist nämlich nicht vorgegeben, sondern im Wesentlichen abhängig vom Feedback auf die Ergebniskommunikation. Deshalb muss dieses genau analysiert werden, um entscheiden zu können, was als nächstes zu tun ist.

Soll ein Planungsschritt durchgeführt werden, sind die Anforderungen auszuwählen, die als nächstes zu entwickeln sind. Der weitere Ablauf zur Lösungssuche und Auswahl entspricht dem Problemlösungszyklus aus dem BWI-Hall Modell. Das Ergebnis wird danach kommuniziert. Auch Ergebnisse von Planungsschritten können und sollen mit dem Kunden diskutiert werden. Nach Realisierungsschritten ist aber, aufgrund der dann materiell vorhandenen Zwischenergebnisse, ein besseres Feedback zu erwarten, weshalb bei Planungsschritten oft auch nur eine interne Kommunikation sinnvoll erscheinen wird.

In Realisierungsschritten werden die Testkriterien von den Anforderungen abgeleitet. Nach der Realisierung wird das (Teil-)System getestet. Neben den expliziten Entwicklungsschritten, kann es natürlich auch implizite geben, z.B. nach Fehlern oder für Abstimmungsmaßnahmen für die keine expliziten Schritte geplant werden.

Die Trennung zwischen Planungsschritten und Realisierungsschritten soll es speziell für Hardware leichter machen, den AEZ sinngemäß umzusetzen. Für einfach herzustellende Komponenten oder kleinere Änderungen kann auch auf eine zeitliche Trennung von Planungs- und Realisierungsschritt verzichtet werden. Diese werden dann parallel angesetzt und durchgeführt. Bei Software, die naturgemäß nicht materiell realisiert werden muss, ist dies immer der Fall. Die Formulierung der Testkriterien soll dann kombiniert mit der Anforderungsauswahl stattfinden und das Testen kann mit der Bewertung kombiniert werden.

Entwicklungsschritte von unterschiedlichen Entwicklungsabteilungen oder Systemkomponenten können auch parallel ablaufen und sind gegebenenfalls zu geeigneten Zeitpunkten zu synchronisieren (siehe 5.4.5).

Die Entwicklung wird, wie erwähnt, vom Feedback auf die Ergebniskommunikation vorangetrieben und gelenkt. Wie dargestellt, können Änderungen im Anforderungskatalog aufgenommen werden. Auch das Berücksichtigen von Veränderungen außerhalb der Entwicklung ist hier leichter möglich. Der nächste explizite Entwicklungsschritt startet dann unter Berücksichtigung des neuen Anforderungskatalogs. Sollte es keine Änderungen gegeben haben, kann dies auch direkt, ohne Bearbeitung des geänderten Katalogs, geschehen. Bei der Ergebniskommunikation mit dem Kunden besteht auch immer die Möglichkeit zum Abbruch.

Wie bereits beschrieben, gilt das Projekt als abgeschlossen, wenn der Kunde die Anforderungen als erfüllt betrachtet und das Ergebnis akzeptiert hat.

Dieses Vorgehensmodell soll es möglich machen, die wesentlichen Vorteile des AEZ zu realisieren und damit einen hohen Grad an Agilität zu erreichen. Durch unterschiedliche Ausprägungsformen der Entwicklung in kleinen Schritten bietet es auch zahlreiche Möglichkeiten eine Umsetzung zumindest teilweise zu erreichen (siehe 6.4). Trotzdem gibt es aber wesentliche Voraussetzungen bzw. Risiken für die Anwendung dieses Modells. Vor allem Flexibilität im Produkt ist ein wesentliches Kriterium dafür, dass Anforderungen zu Beginn unbestimmt bleiben und während der Entwicklung leicht verändert werden können. Sollen bei Einzelfertigung die realisierten Zwischenergebnisse auch Teil des fertigen

Systems sein, ist zu hinterfragen, ob die Art des Systems das erlaubt. Genauso, ob Zwischenergebnisse für den Kunden nutzbar sein können. Auch das Festlegen von Fixpreisen oder das nachverfolgbare Dokumentieren von sicherheitskritischen Anforderungen ist schwierig. Deshalb kann dieses Vorgehensmodell als jenes gesehen werden, das am meisten Potential zur Agilitätssteigerung im Systems Engineering bietet, aber vermutlich nur selten in der Systementwicklung wird angewandt werden können.

### 7.5.3 Auf Set-Based Design basierendes Vorgehensmodell

*Prämisse: Es soll ein Modell gestaltet werden, das möglichst den vollen Umfang aller Maßnahmen aus Abb. 30 unterstützt aber davon ausgeht, dass die Art des Systems keine Entwicklung in kleinen Schritten nach AEZ Ausprägungen zulässt.*

Wird mehr Agilität benötigt, als das BWI-Hall Modell bieten kann, aber der Einsatz des AEZ-SE Modells durch bestimmte Voraussetzungen verhindert, so kann eventuell ein auf Set-Based Design basierendes Vorgehen Abhilfe schaffen. Dabei wird weniger versucht, durch Flexibilität (welche z.B. aufgrund der Art des Systems nicht umsetzbar ist) den Veränderungen gegenüberzutreten, sondern durch ein möglichst langes Offenhalten von Entscheidungsmöglichkeiten.

Das Projekt startet hier nach der Produktvision mit einer Projektplanungsphase, in welcher ein Lösungsbereich identifiziert wird. Dieser beschreibt keine diskrete Lösung, sondern eine Menge an Lösungsvarianten, welche das Problem lösen bzw. die Produktvision umsetzen sollen. Nach dem Simultaneous Engineering Prinzip werden die Phasen der Entwicklung überlappend durchgeführt.

Der Lösungsbereich wird dann separat in den verschiedenen Fachbereichen der Entwicklung (Entwicklungsabteilungen, Komponenten) analysiert, um Lösungen hinsichtlich ihrer Machbarkeit zu überprüfen. Bei fortschreitender Entwicklung wird der mögliche Lösungsbereich immer kleiner und Varianten werden kontinuierlich eliminiert. Am Ende der Entwicklung soll eine machbare Variante aller Komponenten und Phasen (Produktdesign, Produktionsdesign …) fixiert sein. Die Realisierung des Systems erfolgt erst nach der planenden Entwicklungsphase. In dieser können und sollen aber schon Prototypen als Zwischenergebnisse realisiert werden, um möglichst viele Informationen zu gewinnen, bevor Entscheidungen getroffen werden müssen. So kann durch Kundeneinbindung auch die Ermittlung der tatsächlichen Kundenanforderungen forciert werden, die sonst von diesem Modell nicht direkt unterstützt wird.

Je Entwicklungsbereich kann das Vorgehen zur Eingrenzung des Lösungsbereichs wieder als zyklische Entwicklung in kleinen Schritten dargestellt werden, die implizit oder auch explizit durchgeführt werden kann. Explizite Entwicklungsschritte können natürlich auch wieder in Kombination mit impliziten durchgeführt werden. Die Entwicklung wird aber nicht über das Erhalten von Feedback auf Zwischenergebnisse der expliziten Entwicklungsschritte gelenkt, wie das bei einer Entwicklung in kleinen Schritten sonst zur Agilitätssteigerung vorgesehen ist. Hier ist es die Abstimmung der Entwicklungsabteilungen zur Machbarkeit, die den Lösungsbereich weiter einengen und die Entwicklung damit vorantreiben soll. Dieses „Set-Based-SE Vorgehensmodell“ ist in Abb. 35 zu sehen.

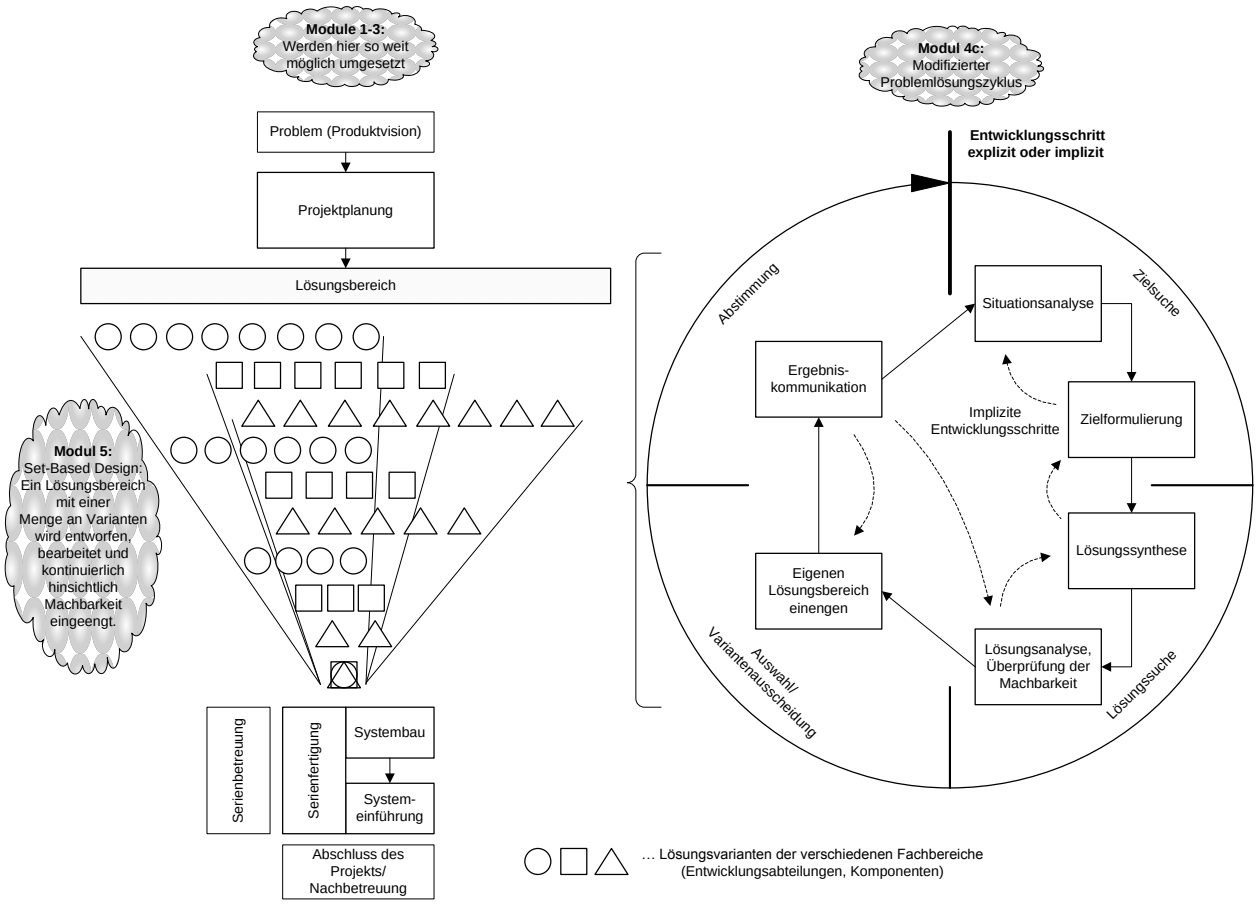


**Abb. 35: Set-Based-SE Vorgehensmodell**

In der Situationsanalyse wird hier verstärkt auf die Entwicklungsergebnisse der anderen Entwicklungsbereiche eingegangen und wie diese die Machbarkeit im eigenen Lösungsbereich beeinflussen. Die Lösungssuche ist ein Bearbeiten und Detaillieren innerhalb des Lösungsbereichs. Die Lösungsauswahl ist hier eher eine Variantenausscheidung. Die Lösung soll sich am Ende des Entwicklungsprozesses mehr oder weniger von selbst dadurch ergeben, dass sie sich als die am besten machbare herausstellt. In den Entwicklungszyklen findet nach der Lösungssuche deshalb eine Einengung des Lösungsbereichs statt, in der die nicht machbar erscheinenden Varianten ausgeschieden werden. Darauf werden die Ergebnisse der Abteilungen ausgetauscht, wodurch eine Abstimmung erfolgen soll.

Veränderungen können in diesem Vorgehensmodell leicht eingebracht werden, solange keine Entscheidungen getroffen wurden, die das verhindern und die Änderungen im noch verfügbaren Lösungsbereich liegen. Dadurch bietet dieses Modell Flexibilität durch die Variantenvielfalt, ohne dass diese im Produkt vorhanden sein muss.

Die Art des Produkts hat natürlich dennoch einen Einfluss darauf, wie groß der Lösungsbereich sein kann und wie lange Teile davon offengehalten werden können. Der Prozess selbst ist relativ flexibel, da er keine genaue Abfolge von Entwicklungstätigkeiten oder Phasen vorsieht. Es kann aber für die Mitarbeiter

sehr schwierig werden, mit einer großen Variantenvielfalt zu arbeiten. Das gleichzeitige Entwickeln mehrerer Varianten bedeutet auch einen hohen Aufwand, weshalb der Agilitätsgewinn der Wirtschaftlichkeit gegenübergestellt werden muss. Ebenso können Fixpreise und sicherheitskritische Anforderungen Schwierigkeiten bereiten.

Mit diesem Vorgehensmodell sollte also eine höhere Agilität, als mit dem BWI-Hall Vorgehensmodell (auch in seiner agilen Ausprägungsform), erreichbar sein. Es ist aber von einigen Voraussetzungen betroffen bzw. unterliegt einigen Risiken. Ein wichtiges Merkmal ist aber, dass die Art des Systems keine Entwicklung in kleinen Schritten unterstützen muss, was die Hauptvoraussetzung für das AEZ-SE Vorgehensmodell ist. Deshalb kann es ev. als Ersatz für das AEZ-SE Modell gelten, wenn dieses aufgrund der eben genannten Voraussetzung nicht machbar ist.

### 7.5.4 Vergleich der Vorgehensmodelle

Die Vorgehensmodelle werden in Abb. 36 anhand ihrer Charakteristika miteinander verglichen. Es wird dabei auf die Ziele, Vor- und Nachteile und die Voraussetzungen eingegangen. Ganz besonders wird aber Wert darauf gelegt, die unterschiedlichen Mechanismen zu beschreiben, mit denen versucht wird, die Agilität zu steigern.

Zum Vergleich wird auch ein traditionelles Vorgehensmodell dargestellt. Dieses soll auf Phasen und Top-Down Prinzip basieren, ähnlich wie das BWI-Hall Modell. Um darzustellen, was eine besonders niedrige Agilität bedeutet, soll es aber besonders starr ausgelegt werden. Die gezeigte starre Ausprägung wird vom BWI-Hall Modell nicht verlangt, kann in der Praxis aber vorkommen.

Diese Übersicht soll dazu dienen, ein geeignetes Vorgehensmodell für die praktische Anwendung zu finden. Die gezeigten Vorgehensmodelle können als Vorlagen für die Gestaltung von Entwicklungsprozessen verwendet werden.

Um die Mechanismen im Umgang mit Veränderungen bzw. die grundlegenden Funktionsprinzipien der agilen Vorgehensmodelle gegenüber der traditionellen Denkweise weiter zu verdeutlichen, wird dies danach in Abb. 37 auch noch grafisch getan. Damit wird gezeigt, wie die agilen Modelle bei Änderungen den Entwicklungsaufwand verringern und eventuelle Lieferverzögerungen verhindern können oder bei einem unveränderten Liefertermin eine bessere Zielerreichung gewährleisten sollen.

| Merkmal | Wenig agiles traditionelles Modell | Agiles BWI-Hall Modell | AEZ-SE Modell | Set-Based-SE Modell |
|---|---|---|---|---|
| **Grundsätzliche Charakteristik** | Lineare, in Phasen unterteilte Entwicklung aller zu Beginn definierten Anforderungen. | Lineare, in Phasen unterteilte Entwicklung aller zu Beginn definierten Anforderungen mit Bedachtnahme auf mögliche Veränderungen. | Zyklische Vorgehensweise in kleinen Schritten, durch Kundenfeedback auf Zwischenergebnisse in Richtung der tatsächlichen Kundenanforderungen gelenkt. | Auf späte Entscheidungspunkte bedachte Entwicklung eines Lösungsbereichs der je nach Machbarkeit kontinuierlich verkleinert wird bis sich eine diskrete Lösung ergibt. |
| **Anwendung des Top-Down Prinzips** | Ja | Ja | Nein | Möglich je Entwicklungsbereich, Phase, Komponente … |
| **Anwendung des Phasenmodells** | Ja, zuerst Planungsphasen, dann Realisierung. | Ja, zuerst Planungsphasen, dann Realisierung. | Nein | Planungsphasen parallel (Simultaneous Engineering), Realisierung danach. |
| **Variantenprinzip** | Wird empfohlen, Varianten werden stufenweise je Phase ausgeschieden. | Wird empfohlen, Varianten werden stufenweise je Phase ausgeschieden. | Eher nein, je Entwicklungs-schritt können aber Varianten bedacht werden. | Durch Lösungsbereich definierte Menge an Varianten wird kontinuierlich reduziert. |
| **Anforderungen** | Müssen zu Projektbeginn genau bekannt sein und werden fix definiert, Änderungen sind gewöhnlich mit hohem Aufwand verbunden. | Müssen zu Projektbeginn genau bekannt sein. Änderungen sind gewöhnlich mit hohem Aufwand verbunden. Es wird versucht diesen zu verringern. | Können zu Beginn ungenau oder teilweise unbekannt sein. Änderungen sind leichter möglich. | Können zu Beginn ungenau oder teilweise unbekannt sein. Änderungen sind leichter möglich, solange diese sich im noch aktuellen Lösungsbereich befinden. |
| **Situationskenntnis** | Soll zu Projektbeginn so gut wie möglich erlangt werden. | Soll während des Projekts aktualisiert werden. | Wird durch den Prozess laufend aktualisiert. | Soll soweit möglich während des Projekts verbessert werden. |
| **Fokus auf tatsächlichen Kundennutzen** | Nein, sondern auf formulierte Anforderungen. | Soweit möglich | Ja | Möglich |
| **Kunde in Entwicklung einbinden** | Nicht vorgesehen, Kunde wird als, durch die Anforderungen repräsentiert, gesehen. | Soweit möglich | Notwendig und forciert. | Möglich |
| **Grundsätzlicher Umgang mit Veränderungen außerhalb der Entwicklung** | Sollen durch Vertragsbestimmungen ausgeschlossen werden, im Bedarfsfall Nachverhandlungen. | Sollen durch Vertragsbestim-mungen ausgeschlossen werden, im Bedarfsfall Nachverhandlungen. Es wird aber versucht den Umgang damit effizienter zu gestalten. | Es wird versucht, diese auch noch spät implementierbar zu machen durch das Bereitstellen von Flexibilität. | Durch das Hinauszögern von Entscheidungen wird versucht, Veränderungen auch noch spät implementierbar zu machen. |
| **Grundsätzlicher Umgang mit Veränderungen innerhalb der Entwicklung** | Es wird davon ausgegangen, dass durch Vorausplanung solche Veränderungen ausgeschlossen werden können. Diese werden deshalb eher als Fehler angesehen. | Es wird davon ausgegangen, dass solche Veränderungen möglich sind und es wird versucht, diese so früh wie möglich zu erkennen und so gut wie möglich damit umzugehen. | Werden als normaler Bestand-teil der Entwicklungstätigkeit angesehen. Die Entwicklung wird danach ausgerichtet. Ein frühes Auftreten wird forciert, Flexibilität vorgesehen. | Werden als normaler Bestand-teil der Entwicklungstätigkeit angesehen. Durch Informationsbeschaffung vor Entscheidungen wird versucht Änderungen zu vermeiden. |
| **Flexibilität im System** | Nicht betrachtet wenn Kunde es nicht verlangt. | Wird betrachtet für den Fall, dass Veränderungen passieren. | Notwendig und forciert. | Nicht notwendig |
| **Effizienz** | Soll durch einen standardisierten Prozess erreicht werden. | Soll durch einen standardisierten Prozess erreicht werden, aber auch durch effizienten Umgang mit Veränderungen. | Soll durch Vorgehensweise erreicht werden, die es den Mitarbeitern erlaubt, so zu entwickeln, wie sie es am besten können. Außerdem durch einfaches Produkt und einfache Prozesse. | Durch anfängliche Betrachtung von Lösungsbereichen anstatt diskreter Lösungen soll die Diskussion über Details (die sich wieder ändern können) vermieden werden. |
| **Effektivität** | Durch perfektes Erfassen der Kundenanforderungen zu Beginn und effizientes Entwickeln. | Durch möglichst gutes Erfassen der Kundenanforderungen zu Beginn und effizientes Entwickeln aber auch Versuch sich dem tatsächlichen Kundennutzen zu nähern. | Durch Ausrichtung der Entwicklung am tatsächlichen Kundennutzen und effizienter Entwicklung trotz Änderungen. | Durch langes Offenhalten von Entscheidungsmöglichkeiten ist es möglich, sich später in der Entwicklung an die aktuellen Kundenwünsche anzupassen. |
| **Wesentliche Voraussetzungen** | Anforderungen zu Beginn bekannt und stabil. | Anforderungen zu Beginn möglichst bekannt und stabil. | System muss geeignet sein, Kunde muss zur Einbindung gewillt sein, Preisgestaltung muss kompatibel sein. | Wirtschaftlichkeit muss gegeben sein (ev. sehr hohe Anzahl an Varianten für welche Prototypen notwendig sind). |

**Abb. 36: Vergleich der Vorgehensmodelle**

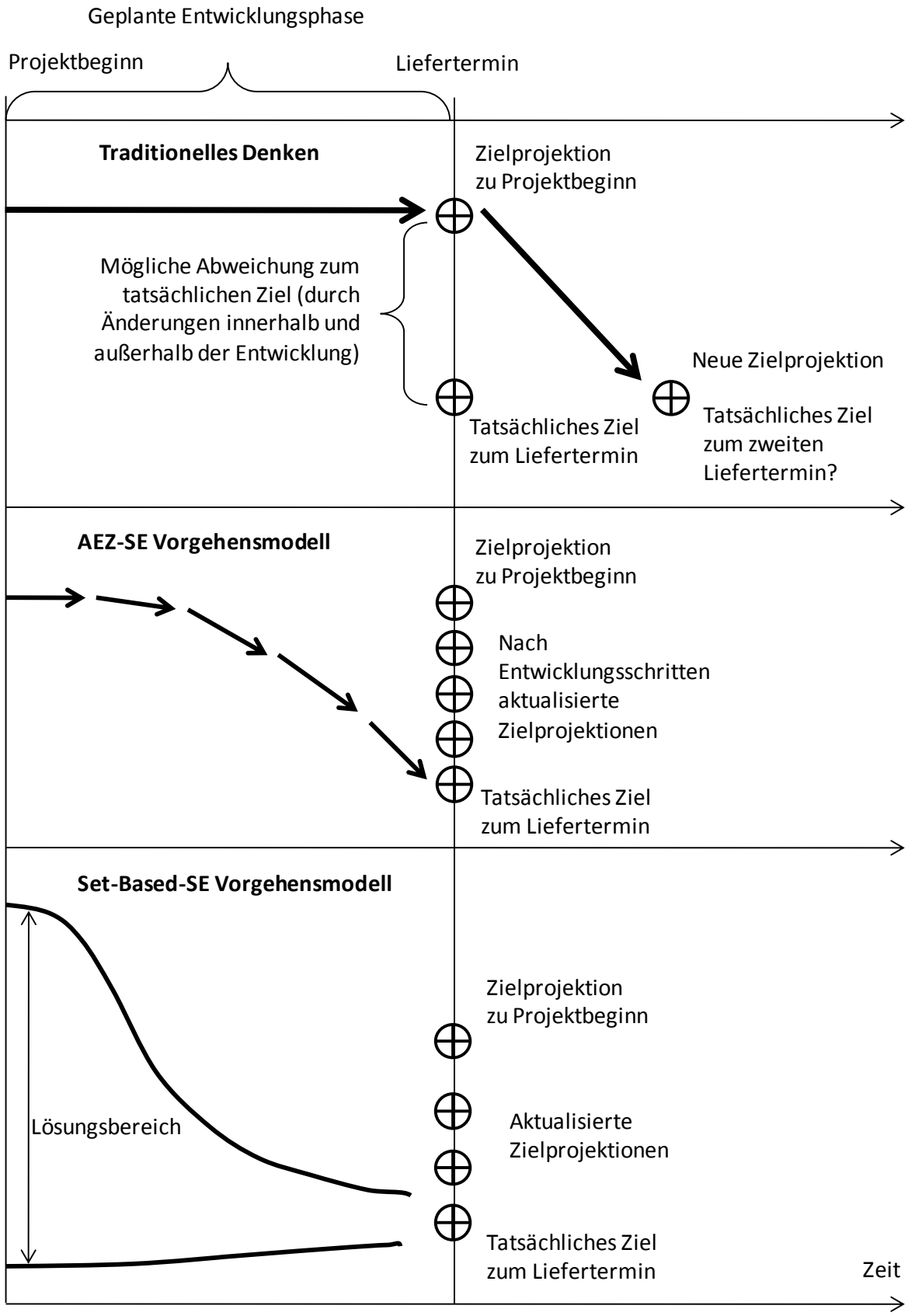


**Abb. 37: Grundlegende Funktionsprinzipien der agilen SE Vorgehensmodelle hinsichtlich Änderungen**

## 7.6 Zusammenfassung

In diesem Kapitel wurden, die zur Agilität im SE gewonnenen, Erkenntnisse zu einer Vorgehensmethodik zusammengefügt. Diese hat die BWI-Hall SE-Methodik (2.1.5) als Grundlage und erweitert sie um Lösungsmöglichkeiten hinsichtlich Veränderungen (siehe auch Abb. 39). Innerhalb des ASE Ansatzes dieser Arbeit soll die Vorgehensmethodik zeigen, wie die einzelnen Komponenten zusammenhängen und wie sie im SE angewandt werden können.

Als wichtigstes Ergebnis des ASE Ansatzes wird das Verständnis für die äußerst komplexe Thematik der Agilität im Systems Engineering betrachtet. Es erscheint genauso wenig möglich zu sein, eine komplette Liste an Praktiken für Agilität im SE zu erstellen, wie eine allgemeingültige Vorgehensweise dafür zu finden. Agilität kann auch nicht direkt durch Maßnahmen eingeführt oder installiert werden. Sie ist vielmehr eine Fähigkeit, die sich aufgrund zahlreicher Einflussfaktoren ergibt. In diesem Kapitel wurde deshalb gezeigt, in welchen Bereichen Maßnahmen (7.3) getroffen werden können und wie diese zusammenhängen. Diese Maßnahmen können als Beispiele gesehen werden, für den Umgang mit Veränderungen, welcher dem BWI-Hall Ansatz auf Ebene des Problemlösungsprozesses (Abb. 7) hinzugefügt werden soll. Es wurde auch gezeigt, wie diese Maßnahmen zur Steigerung der Teilfunktionen der Agilität führen können (7.2). Mit diesem Verständnis der Agilität soll die SE-Philosophie des BWI-Hall Ansatzes zu einer agilen SE-Philosophie erweitert werden. Nur wenn alle Zusammenhänge in der praktischen Anwendung sinnvoll hergestellt werden, sind eine Steigerung der Agilität und damit ein erfolgreicheres SE in veränderlichen Umgebungen zu erwarten.

Die ASE Vorgehensmethodik berücksichtigt die unterschiedlichen Arten von Veränderungen (im Wesentlich ob ihre Ursache außerhalb oder innerhalb der Entwicklung zu suchen ist) und Voraussetzungen, die für bestimmte Vorgehensweisen gelten. Deshalb empfiehlt sie keine konkrete Methode und kein konkretes Vorgehensmodell, da keines davon unter allen Bedingungen eine geeignete Lösung darstellen kann. Das ASE stellt aber zahlreiche Praktiken (7.4) zur Verfügung und gibt Beispiele für Vorgehensmodelle (7.5) anhand derer Entwicklungsprozesse grundlegend gestaltet werden können. Dabei wird auch auf die Anwendungsgebiete je nach Bedarf und Voraussetzungen eingegangen. Die Prinzipien und Praktiken können auf der untersten Ebene des BWI-Hall Ansatzes als „agile Techniken“ gesehen werden. Die Vorgehensmodelle stellen agile Alternativen zum SE-Vorgehensmodell (Abb. 8) aus dem BWI-Hall Ansatz dar. Als wesentliches Funktionsprinzip für die Gestaltung eines agilen Vorgehensmodells, wurde die Entwicklung in kleinen Schritten verwendet. Damit wurde einerseits der Problemlösungszyklus des BWI-Hall Vorgehensmodells modifiziert und ein Vorgehensmodell gestaltet, in dem die anderen Komponenten aber dennoch ihre Gültigkeit behalten (7.5.1). Andererseits wurde auch ein Vorgehensmodell gestaltet, dass die Richtung der Entwicklung durch Feedback auf diese kleinen Entwicklungsschritte lenkt (7.5.2), wie dies auch in den agilen SW-Entwicklungsmethoden geschieht. Zusätzlich wurde auch noch das Set-Based Design verwendet, um damit auch ein Vorgehensmodell zu gestalten, das auch eine höhere Agilität erwarten lässt (7.5.3).

Das ASE ist damit als umfassende Anleitung zur Beschäftigung mit dem Thema Agilität im Systems Engineering zu sehen. Es stellt theoretische Zusammenhänge her, die auch notwendig sind, um die praktische Umsetzung sinnvoll durchführen zu können und gibt Hilfestellungen und Empfehlungen für die Anwendung geeigneter Praktiken. Zum besseren Verständnis der praktischen Anwendung wird im Folgenden noch ein Fallbeispiel beschrieben.

# 8 Fallbeispiel

Anhand eines Fallbeispiels aus der Fahrzeugindustrie soll nun die praktische Anwendung des ASE Ansatzes näher erklärt werden. Dazu wird ein fiktives Fallbeispiel für ein Entwicklungsprojekt verwendet, das aber an ein reales technisches Problem (ABS-Einheit für ein Motorrad) angelehnt ist. Damit sollen die Vorgehensmodelle, das Zusammenspiel der Prinzipien und Praktiken und auch das Vorgehen bei der Auswahl dieser besser verständlich gemacht werden.

Der Kontext (Bedarf und Voraussetzungen) für die Anwendung des ASE Ansatzes wird in Kapitel 8.2 analysiert. Danach wird aber nicht nur eine geeignete Herangehensweise für den bestimmten Kontext beschrieben. Sondern es wird das Fallbeispiel für alle drei in Kapitel 7.5 beschriebenen Vorgehensmodelle diskutiert und es wird jeweils der Umgang mit Veränderungen innerhalb und außerhalb der Entwicklung aufgezeigt. Ebenso wird das Fallbeispiel für eine Entwicklung nach einem wenig agil ausgeprägten traditionellen Modell dargestellt.

Die zuerst beschriebene, wenig agile Umsetzung, nach einem auf Phasen und Top-Down Prinzip beruhenden Modell, dient dabei hauptsächlich dem Aufzeigen von Problemen im Zusammenhang mit Veränderungen. In den drei darauf folgenden Kapiteln soll dann dargestellt werden, wie diese Probleme mit den agilen Vorgehensweisen gelöst bzw. besser bewältigt werden können. Es wurde darauf geachtet, die Problemstellungen so praxisnahe wie möglich zu behandeln.

Es sei auch darauf hingewiesen, dass im Fallbeispiel nicht wieder alle Voraussetzungen bzw. Hindernisse für die Anwendung der einzelnen Praktiken angeführt werden, die schon in den Kapiteln 5 und 6 diskutiert wurden. Das Fallbeispiel soll die Anwendung der Praktiken beschreiben unter der Annahme, dass dies sinnvoll durchgeführt werden kann. Auf die wichtigsten Probleme wird aber dennoch eingegangen.

## 8.1 Gemeinsame Ausgangssituation

Das zu entwickelnde System soll eine ABS-Einheit für Motorräder sein, welche von einem Systementwickler im Auftrag eines Motorradherstellers entwickelt und in Serie produziert werden soll. Der Motorradhersteller will dabei möglichst alle herkömmlichen Komponenten für Vorder- und Hinterradbremse beibehalten. Der Systementwickler soll mit der zusätzlichen, ins Motorrad zu integrierenden, ABS-Einheit zwei Hauptfunktionen erfüllen, die den wesentlichen Kundennutzen darstellen. Einerseits soll bei Bremsmanövern ein Blockieren der einzelnen Räder verhindert werden. Andererseits soll eine Kombinationsfunktion der beiden Bremsen dafür sorgen, dass in Notsituationen - also wenn eine der beiden Bremsen so stark betätigt wird, dass die Antiblockier-Regelung aktiviert wird und eine Notsituation anzunehmen ist - beide Räder bis zur maximal möglichen Verzögerung abgebremst werden, auch wenn der Fahrer in seiner Schrecksituation nur eine der beiden Bremsen betätigt.

Nachdem der Motorradhersteller bisher nur einfache Bremsanlagen selbst entwickelt hat, aber keine Erfahrung mit komplexen Regelungsanlagen hat, die in ein hydraulisches System eingreifen können, wendet er sich an einen Fahrzeugkomponentenhersteller (den genannten Systementwickler), der bisher u.a. ABS Systeme für PKWs hergestellt hat und der auch Forschung im Bereich von Motorrad-ABS-Systemen betreibt. In diesem Beispiel soll angenommen werden, dass der Systementwickler, aus hier

nicht gezeigten Überlegungen, bereits feststeht und sich keine konkurrierenden Entwickler mehr um den Auftrag bemühen.

Die in Abb. 38 dargestellte, grobe Systemarchitektur steht zwar zu Beginn der Entwicklung noch nicht fest, sie soll aber für alle Fallbeispiele gleich sein, deshalb wird sie hier schon gezeigt. Es wird für jedes Fallbeispiel separat beschrieben, zu welchem Zeitpunkt diese Systemarchitektur entwickelt und fixiert wird.

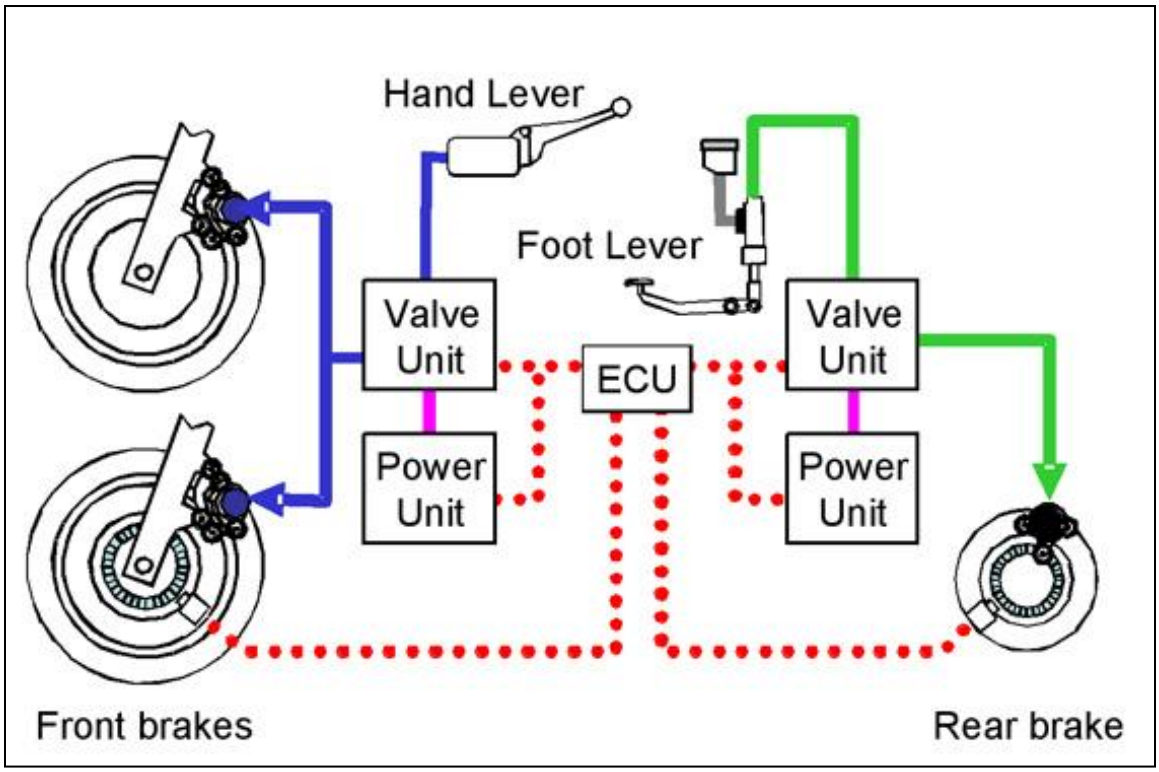


**Abb. 38: Vereinfachte Systemarchitektur des ABS Systems (Quelle: www.honda.at)**

Es sei darauf hingewiesen, dass die von Honda veröffentlichte Skizze lediglich zur Abbildung eines realistischen Systems verwendet wird. In keiner der folgenden Versionen des Fallbeispiels wird der Honda Entwicklungsprozess beschrieben oder eine weitere Anleihe bei Honda genommen.

Als Teil der Ausgangssituation kann angenommen werden, dass das in die Bremsfunktion eingreifende System als sicherheitskritisch betrachtet werden muss. Ebenso kann angenommen werden, dass der Kunde an Zwischenergebnissen interessiert sein wird, um die Entwicklung des Gesamtfahrzeugs durchführen zu können. Dazu benötigt er nutzbare Prototypen zur Integration in das Motorrad, mit denen Erkenntnisse zum Verhalten des Gesamtfahrzeugs gewonnen werden sollen. Sie dienen aber auch zur genaueren Definierung der Anforderungen an das ABS-System, da diese teilweise zu Beginn noch nicht genau bekannt sind.

Weiters soll angenommen werden, dass noch keine gesetzliche Verpflichtung für ABS bei neu eingeführten Motorrädern besteht, diese aber immer wieder in der Öffentlichkeit und vom Gesetzgeber diskutiert wird.

## 8.2 Kontextbestimmung und Auswahlverfahren

Bevor in den Kapiteln 8.3 bis 8.6 mögliche Vorgehensweisen für die Entwicklung des ABS Systems beschrieben werden, soll hier dargestellt werden, wie man die geeignetste dieser Vorgehensweisen bestimmt. Dazu muss der Kontext des Entwicklungsprojekts näher analysiert werden. Wesentliche Faktoren dafür wurden in 6.2.3 beschrieben und in 7.4 wurde damit ein Auswahlverfahren entworfen. Im dargestellten Fallbeispiel kann man damit zu folgenden Erkenntnissen gelangen:

***Bedarf allgemein***

Wie beschrieben wurde, ist der Bedarf an Agilität im SE besonders dann hoch, wenn die Systementwicklung von zahlreichen und schwerwiegenden Veränderungen betroffen ist. Als mögliche Gründe für Veränderungen innerhalb der Entwicklung wurden eine hohe Komplexität und Neuartigkeit des Systems genannt. Beides kann für das ABS-System als gegeben betrachtet werden. Als mögliche Veränderung außerhalb der Entwicklung wurde in der Ausgangssituation schon eine potentielle Gesetzesänderung betreffend einer ABS-Pflicht für Motorräder angedeutet. Die Motorradbranche soll im Allgemeinen zumindest moderat veränderlich angenommen werden. Ohne weitere Potentiale für Veränderungen beim Entwickler und in seiner Umgebung zu analysieren, kann also bereits angenommen werden, dass im Allgemeinen ein markanter Bedarf an Agilität besteht.

Das bisherige Vorgehen des Entwicklers soll als am BWI-Hall Vorgehensmodell orientiert angenommen werden, aber in einer eher starren Form, welche durch den Wunsch nach Standardisierung über Projekte hinweg, zustande kam. In der Vergangenheit hatte der Entwickler damit immer wieder Probleme, vor allem weil bei komplexen oder neuartigen Produkten Informationen erst spät zu Tage traten, welche Anforderungsänderungen bewirkten und Lieferverzögerungen und Budgetübertretungen zur Folge hatten.

Nachdem in dieser Situation die vorhandene Agilität geringer anzunehmen ist, als die erforderliche bzw. wünschenswerte Agilität, kann die Anwendung des ASE Ansatzes grundsätzlich als sinnvoll erachtet werden.

***Bedarf hinsichtlich einzelner Praktiken***

Von einem allgemeinen Bedarf an Agilität kann noch nicht direkt auf die Sinnhaftigkeit einer bestimmten Praktik geschlossen werden. Zuerst gilt es zu überprüfen, wo (in welcher Teilfunktion der Agilität im SE) ein Mangel besteht, bzw. wo eine Optimierung durchgeführt werden soll. Anhand der Teilfunktionen der Agilität sollen nun beispielhaft mögliche spezifische Bedarfsgründe genannt werden:

- Situationskenntnis: Hier kann ein Mangel bestehen, wenn die tatsächlichen Kundenanforderungen nicht oder zu spät erkannt werden (Konzentration nur auf niedergeschriebene Anforderungen und fehlerbehaftete Interpretation dieser). Auch ein unzulänglicher Informationsaustausch zwischen den Entwicklungsabteilungen kann eine Optimierung dieser Teilfunktion der Agilität erforderlich machen.
- Kontinuierliche Verbesserung: Hier kann Verbesserungspotential bestehen, wenn eine negative Grundhaltung gegenüber Änderungen und ein starrer Entwicklungsprozess die laufenden Verbesserungen von Produkt, Prozess und Organisation und das Lernen der Mitarbeiter behindern.

- Bewältigung von Veränderungen: Wird ein schnelles und einfaches Ändern von Produkt, Prozess oder Organisation durch starre organisatorische Regeln verhindert oder den Mitarbeitern fehlen die Fähigkeiten zur Umsetzung der Änderungen, können Bedarfsgründe zur Verbesserung dieser Teilfunktion bestehen.
- Flexibilität: Hier kann Handlungsbedarf bestehen, wenn die mangelnde Anpassbarkeit von Produkt bzw. Produktionslinie notwendige Änderungen nicht zulässt oder diese erheblich erschwert.
- Effizienz: Unzweckmäßige Prozesstätigkeiten oder unzureichend qualifizierte Mitarbeiter können hier Ansatzpunkte liefern.
- Effektivität: Ein Optimierungspotential kann bestehen, wenn das Produkt Funktionalitäten umfasst, die keinen Nutzen für den Kunden haben.

Für das gegebene Fallbeispiel sollen vor allem die unter „Situationskenntnis“ und „Flexibilität“ genannten Punkte als bereits identifizierte, wesentliche Mängel angenommen werden. Wie man passende Praktiken dazu findet, wird nach der Analyse der Voraussetzungen unter „Wahl einzelner Praktiken“ beschrieben. Eine mögliche allgemeine Steigerung der Agilität (durch die Wahl des Vorgehensmodells) soll dann aber unter Berücksichtigung aller Teilfunktionen überprüft werden.

***Voraussetzungen***

Als nächstes soll analysiert werden, ob die Voraussetzungen für die Anwendung von bestimmten Praktiken gegeben sind, bzw. ob Risiken für deren Einsatz bestehen. Dazu soll das Entwicklungsprojekt hinsichtlich der Voraussetzungs-Faktoren aus 6.2.3 überprüft und wesentliche Punkte aufgezeigt werden:

- Art des Systems: Das ABS-System besteht aus mechanischen, elektronischen und hydraulischen Hardwarekomponenten und Software. Es soll nach der Entwicklung in Serie gefertigt werden. Das Herstellen von Prototypen ist in frühen Phasen der Entwicklung aufwendig und teuer, da hier noch keine Serienwerkzeuge zur Verfügung stehen. In späteren Phasen, wenn teilweise schon Serienwerkzeuge angeschafft wurden, wird die Herstellung von Prototypen zwar billiger. Führen dadurch gewonnene Erkenntnisse aber zu Änderungen an den schon angeschafften Serienwerkzeugen, so ist deren Implementierung dann aber umso aufwendiger und teurer. Zum Testen der Hardware sind Prüfstände notwendig, welche das Testen auch teuer und oft langwierig machen. Das Testen von einzelnen Komponenten kann z.B. zur Überprüfung der Festigkeit Sinn machen. Die Überprüfung der Funktionalität ist in vielen Fällen aber nur für das Gesamtsystem sinnvoll.
- Kundeneinbindung: Der Kunde ist hier am Projektfortschritt zwar interessiert, er will sich aber nicht mit allen Zwischenergebnissen auseinandersetzen. Ein häufiges Testen von Teilen des Systems würde ihm nur Kosten verursachen, hätte aber keinen großen Nutzen für ihn. Deshalb interessieren den Kunden vorrangig nutzbare Prototypen, welche er in Prototypen des Motorrads integrieren kann um die Gesamtfahrzeugentwicklung voranzutreiben.
- Kritikalität: Nachdem eine Fehlfunktion des ABS-Systems zu einem Totalausfall der Bremsanlage, aber auch zu einem gefährlichen, ungewollten Bremsvorgang (z.B. Blockieren der Räder in Kurvenlage) führen kann, ist dieses System als sicherheitskritisch zu betrachten. Auf die dazu zu

verwendenden Sicherheitsstandards soll nicht im Detail eingegangen werden. Es kann aber angenommen werden, dass nicht nur technische Maßnahmen (Rückfallebene um normale Bremsfunktion sicherzustellen wenn ABS-System versagt) umzusetzen sind, sondern auch Maßnahmen im Prozess (Nachverfolgbare Dokumentation der Entwicklung aller sicherheitskritischen Anforderungen).

- Kultur: Die Entwickler sollen als eher aufgeschlossen und experimentierfreudig angenommen werden und sich von den bisherigen starren Prozessvorschriften eher beengt gefühlt haben.
- Mitarbeiterqualifikation: Über alle Entwicklungsabteilungen und Projekte hinweg sollen die Mitarbeiter zu je einem Drittel als ausgezeichnet, durchschnittlich und unerfahren angenommen werden.
- Teamgröße und Verteilung: Beim Systementwickler wird eine Matrix-Organisation mit nach Fachbereichen gegliederten Abteilungen angenommen. Für Entwicklungsprojekte werden aus jedem Fachbereich Mitarbeiter entsandt und in die Projektorganisation eingegliedert. Örtlich bleiben sie aber in den jeweiligen Fachabteilungen und im Normalfall arbeiten sie auch an mehreren Projekten gleichzeitig. Für das betrachtete Entwicklungsprojekt werden 25 beteiligte Entwickler angenommen.
- Preisgestaltung: Die Beauftragung des Systementwicklers erfolgte auf Basis eines Fixpreises für die Entwicklungskosten und die in Serie herzustellenden Systeme.
- Wirtschaftlichkeit: Die Wirtschaftlichkeit muss als Voraussetzung für jeden Einzelfall natürlich separat überprüft werden. Im Allgemeinen kann aber angenommen werden, dass aufgrund hoher Prototyping-, Änderungs- und Testkosten des Produkts ein häufiges Durchführen dieser Tätigkeiten die Wirtschaftlichkeit des Projekts stark negativ beeinflusst. Ebenso kann dies aufgrund hoher Änderungskosten der Produktionslinie resultieren. Aber auch das Einplanen von Flexibilität um Möglichkeiten für Änderungen im Vorhinein zu schaffen oder das Einplanen von Entwicklungszeitreserven oder Reserven im Potential des Produkts sind hinsichtlich Wirtschaftlichkeit zu überprüfen.

***Wahl einzelner Praktiken***

Für die genannten Bedarfsfälle hinsichtlich einzelner Praktiken soll nun eine Auswahl durchgeführt bzw. sollen Empfehlungen ausgesprochen werden. Dazu sollen die in den Kapiteln 5 und 6 dargestellten Zusammenhänge, die überblicksartig auch in Abb. 32 aufgezeigt wurden, zur Verwendung kommen. Zur Behebung des bei der Situationskenntnis genannten Mangels, sind also primär die dafür in der Auswahlmatrix gekennzeichneten Prinzipien und Praktiken zu überprüfen.

Um die tatsächlichen Kundenanforderungen früher zu erkennen, könnte der AEZ (4.3.4) eingesetzt werden. Das scheitert aber an zahlreichen der vorher genannten Voraussetzungen. Den Kunden auf andere Art in die Entwicklung einzubinden (5.1.3) ist vermutlich auch schwierig. Wie beschrieben, ist er daran nicht besonders interessiert. Außerdem ist es kostenaufwändig in frühen Phasen Prototypen (5.4.7) herzustellen, um damit abzusichern, dass die Kundenanforderungen richtig verstanden wurden. Eventuell kann aber der bisherige Entwicklungsprozess des Entwicklers dahingehend optimiert werden, dass bei der Analyse der Anforderungen geeignete Ansprechpartner beim Kunden kontaktiert werden, um die Interpretation der Anforderungen zu erläutern, u.U. auch mit der Hilfe von einfachen Simulationsmodellen (5.4.7). Ebenso kann der Einsatz eines Product Owners (5.1.6) helfen, die

Sichtweise des Kunden in der Entwicklung besser zu verstehen, wenn der dazu notwendige Personaleinsatz sich wirtschaftlich rechtfertigen lässt. Auf jeden Fall lassen sich die Agilität eines Projekts im Allgemeinen und auch das Verständnis der tatsächlichen Kundenanforderungen im Speziellen durch den Einsatz der bestqualifizierten Mitarbeiter (5.3.2, 5.2.3) verbessern. In der Unternehmung kann dies z.B. so durchgeführt werden, dass unerfahrene Mitarbeiter zuerst in Standardprojekten eingesetzt werden und bei Projekten mit hoher Komplexität und hohem Neuigkeitsgrad vorrangig hochqualifizierte und erfahrene Mitarbeiter eingesetzt werden. Bei diesen ist zu erwarten, dass sie die niedergeschriebenen Anforderungen besser interpretieren und damit spätere Änderungen in Richtung der tatsächlichen Anforderungen vermeiden können. Damit wäre ein besserer Fokus auf den Kundennutzen erreicht, dieser muss aber vor der Preisfixierung geschehen, damit im Preis auch die tatsächlichen Anforderungen abgebildet werden können.

Als zweiter Problemfall soll nach Praktiken für eine höhere Flexibilität von Produkt und Produktionslinie gesucht werden. Eine Möglichkeit wäre hier die gleichzeitige Entwicklung mehrerer Varianten (5.4.17). Dadurch besitzt man die Flexibilität zwischen mehreren Variante entscheiden zu können, solange man sich bezüglich der Anforderungen im Unklaren ist. Präzisieren sich diese, wird nur mehr die Variante weiterentwickelt, mit welcher die Anforderungen erreicht werden können. Diese Praktik wird in vielen Fällen aber nicht wirtschaftlich umsetzbar sein, was auch bei den folgenden Praktiken zu untersuchen ist. Die Anpassbarkeit (5.5.2) von Produkt und Produktionslinie kann z.B. durch Modularität, Unabhängigkeit von Designparametern und Skalierbarkeit erhöht werden. Auf die konkreten technischen Lösungen zu diesen Prinzipien soll hier nicht eingegangen werden. Auch indem man versucht HW-Designparameter, bei denen dies möglich ist, in SW abzubilden, lässt sich die Flexibilität aufgrund der leichter zu ändernden SW erhöhen. Eine weitere Möglichkeit wäre das Vorhalten von Leistungsreserven (5.5.4) im Produkt. Eine Möglichkeit, welche die Flexibilität erhöht, ohne die Problematik der Wirtschaftlichkeit, ist eine möglichst einfache Gestaltung (5.5.1) des Produkts. Nachdem bei Einfachheit weniger Designparameter voneinander abhängig sind, müssen bei Änderungen auch nur weniger Abhängigkeiten berücksichtigt werden, was die Änderungen einfacher und kostengünstiger macht.

Mit diesen beiden Beispielen sollte die Vorgehensweise bei der Suche nach Praktiken für konkrete Probleme beschrieben werden. Kapitel 7.4 stellt hierfür einen Überblick über die theoretischen Grundlagen dar. Es ist aber anzumerken, dass für die sinnhafte Auswahl und Anwendung eines Prinzips oder einer Praktik auf die gesamten Erkenntnisse der Kapitel 5 - 7 zurückgegriffen und vor allem auch das Fachwissen der jeweiligen Branche angewandt werden muss.

***Wahl des Vorgehensmodells***

Wie erwähnt, benutzte der Systementwickler bisher das BWI-Hall Vorgehensmodell, allerdings in einer recht starren und wenig agilen Art und Weise. Für das aktuelle Projekt ist ein höherer Grad an Agilität von Vorteil. Ein Vorgehensmodell stellt die grundlegende Struktur des Entwicklungsprozesses dar, welchen der Systementwickler normalerweise über Projekte hinweg in einer Standardform beibehalten möchte. Es kann deshalb angenommen werden, dass es bei einem Vorgehensmodell-Wechsel Widerstände in der Unternehmung geben wird, was aber hier nicht weiter diskutiert werden soll. Die Wahl des Vorgehensmodells soll also nur für das aktuelle Projekt hinsichtlich der damit verbundenen Risiken diskutiert werden.

Als erstes soll die Frage gestellt werden, ob der AEZ in seiner vollen Ausprägung angewandt werden kann. Als Sammlung der Regeln und Prinzipien zur agilen Entwicklung von Software kann er auch mit einem Vorgehensmodell verglichen werden. Sollten die Voraussetzungen es erlauben, wäre mit seiner Anwendung im Fallbeispiel das Erreichen derselben Vorteile wie bei den agilen SW-Entwicklungsmethoden möglich. Dies ist aber selbst in reinen Softwareentwicklungsprojekten nur in einem bestimmten Kontext möglich, wie in 6.1 gezeigt wurde. Und auch die hier im Fallbeispiel beschriebenen Voraussetzungen lassen eine vollinhaltliche Umsetzung des AEZ nicht zu, wie z.B. die Art des Systems, welche keine Entwicklung von vom Anwender nutzbaren Zwischenergebnissen im Takt von wenigen Wochen erlaubt.

Das AEZ-SE Vorgehensmodell (7.5.2) versucht mit den Maßnahmen aus 6.4, speziell durch eine unkonventionelle (von den AEZ Ausprägungen abweichende) Anwendung der Entwicklung in kleinen Schritten, die wesentlichen Vorteile des AEZ umzusetzen und dennoch allgemeiner anwendbar zu sein. Trotzdem wird seine Umsetzung im Fallbeispiel schwierig sein. Zur Bewertung der Risiken sind hier die Praktiken, aus denen sich das AEZ-SE Modell zusammensetzt, zu betrachten, speziell die Entwicklung in kleinen Schritten. Vor allem die Aufteilung des Entwicklungsumfangs, die Kundeneinbindung, die Preisgestaltung und die Sicherheitskritikalität sind als problematisch zu betrachten. Die Zyklusdauer müsste zumindest bei den HW-Komponenten sehr lange angesetzt werden. Viele Zwischenergebnisse wären vermutlich nicht vom Kunden nutzbar. Es wären nur wenige Änderungen mit dem vereinbarten Fixpreis darzustellen und sicherheitsrelevante Dokumentation würde den Prozess vieles seiner Agilität kosten. Berücksichtigt man all diese Dinge so würde der damit gestaltete Entwicklungsprozess kaum mehr dem AEZ-SE Vorgehensmodell entsprechen bzw. nur wenige Vorteile hinsichtlich Agilität bieten.

Als Alternative kann das Set-Based-SE Vorgehensmodell (7.5.3) gesehen werden. Vor allem das Problem der Unterteilung des Entwicklungsumfangs in kleine Schritte besteht hier nicht, weil keine nutzbaren Zwischenergebnisse erstellt werden müssen. Dafür kann dann aber auch nicht durch Demonstration dieser Zwischenergebnisse und Feedback des Kunden auf seine tatsächlichen Anforderungen geschlossen werden. Die Entwicklung eines Lösungsbereichs statt einer einzelnen Lösung ist im Fallbeispiel auch kritisch zu sehen. Dazu müssten nämlich auch Prototypen zu mehreren Varianten erstellt werden, was hinsichtlich Wirtschaftlichkeit zu hinterfragen ist.

Der Systementwickler wird aufgrund der Voraussetzungen im Fallbeispiel also vermutlich beim BWI-Hall Modell (2.1.5) bleiben. Man könnte ihm aber die zur Agilitätssteigerung modifizierte Version (7.5.1) empfehlen, da diese ohne weitere Risiken anwendbar ist und die Agilität zumindest in geringem Ausmaß steigern könnte.

Im Folgenden wird nun beschrieben, wie die Systementwicklung mit dem in der Entwicklungsfirma bisher verwendeten Vorgehensmodell abgelaufen wäre. Danach wird der Agilitätsvorteil des agil ausgelegten BWI-Hall Modells aufgezeigt. Es wird aber auch aufgezeigt, wie die Entwicklung nach dem AEZ-SE oder dem Set-Based-SE Vorgehensmodell ablaufen könnte, wenn dies möglich wäre, bzw. wenn die Risiken dabei beherrschbar wären.

## 8.3 Wenig agil umgesetztes traditionelles Vorgehensmodell

Um aufzuzeigen welche Probleme mit Veränderungen bei traditioneller Entwicklung auftreten können und den Unterschied zu agilitätsbetonten Varianten deutlich zu machen, wird in der ersten Version des Fallbeispiels ein besonders stark strukturiertes und starres Vorgehensmodell angenommen. Im BWI-Hall Modell wird eine solch starke Strukturierung nicht verlangt, sie kann aber in der Praxis vorgeschrieben sein, z.B. begründet durch einen Wunsch, eine möglichst umfassende Standardisierung des Entwicklungsprozesses zu erreichen und alles im Voraus planen zu wollen, ohne Berücksichtigung von Veränderungen.

Die Nomenklatur für die Entwicklungstätigkeiten soll aus Abb. 33 übernommen werden. Das hier angewandte Vorgehensmodell soll weiters dem in Abb. 36 beschriebenen „wenig agilen traditionellen Modell“ entsprechen. Die Verteilung der Entwicklungsumfänge auf die Phasen ist Ansichtssache und kann unterschiedlich geschehen. Hier sollen alle Phasen durchgespielt und mit dem Anstoß des Kunden begonnen werden, der mit einer Produktvision an den Entwickler herantritt. Die Vorstudie entspricht dann einer Angebotsphase, in der die Projektplanung durchgeführt, der Anforderungskatalog formuliert und die in Abb. 38 gezeigte grobe Systemarchitektur entwickelt wird. Nach Unterzeichnung des Vertrages wird in der Hauptstudie das Gesamtkonzept des Produkts entwickelt. In der Detailstudie findet die Detail- und Serienentwicklung des Produkts statt und es wird die Fertigung geplant.

***Produktvision***

Die Bedeutung der Produktvision ist hier nicht besonders groß, da aus ihr noch ein detaillierter Anforderungskatalog abgeleitet werden muss. Die Entwicklung orientiert sich danach an der Erfüllung dieses Anforderungskataloges. Die Produktvision bzw. das Vorhandensein einer Problemstellung ist aber nötig um das Projekt bzw. die Vorstudie zu initiieren.

***Projektplanung in der Vorstudie***

In der Vorstudie erfolgt eine ausführliche Planung des gesamten Projekts. In diese Phase fällt auch die Erstellung eines detaillierten Anforderungskatalogs. Man versucht alles im Vorhinein festzulegen, was einen Einfluss auf den Preis haben kann, nachdem dieser fix vereinbart werden soll. Auch der Liefertermin bzw. der Start der Serienproduktion wird fest vereinbart. Ebenso werden Meilensteine vereinbart, zu denen Prototypen geliefert werden sollen, die der Kunde zur Entwicklung des Gesamtfahrzeugs benötigt. Es wird auch festgelegt, dass sicherheitskritische Aspekte des Systems nach dem Stand der Technik zu entwickeln und zu dokumentieren sind und dieses in der Verantwortung des Entwicklers liegt. Der Systementwickler stellt ein Entwicklungsteam zusammen und die Art der Kommunikation mit dem Kunden wird vereinbart.

***Anforderungskatalog***

Der Anforderungskatalog ist hier ab der Unterzeichnung des Vertrages am Ende der Vorstudie als fix zu betrachten. Spätere Änderungen müssen bewertet und nachverhandelt werden. Der Kunde versucht deshalb auf Nummer sicher zu gehen und die Anforderungen an das Produkt möglichst hoch zu stecken, auch wenn er sie am Schluss gar nicht benötigt. Der Entwickler berücksichtigt vermeintlich zu hoch gesteckte Anforderungen im Preis oft nicht, um nicht gegenüber Konkurrenten im Kampf um das Entwicklungsprojekt mit einem zu hohen Preis den Kürzeren zu ziehen (was normalerweise gilt, in

diesem Fallbeispiel aber ausgeschlossen wurde). Beiden Vertragsparteien fehlen zu diesem Zeitpunkt noch wichtige Informationen zu den tatsächlichen Anforderungen. Deshalb ist es vorhersehbar, dass es zu Änderungen und Nachverhandlungen kommen wird, wenn diese Informationen während der Entwicklung verfügbar werden.

Beispielsweise werden technische Spezifikationen, wie genaue Regelzeiten des Systems bereits fixiert. Diese können zum jetzigen Zeitpunkt aber noch nicht sinnvoll bestimmt werden, da der Kunde aufgrund mangelnder Erfahrung mit diesem komplexen und neuartigen System diese noch gar nicht kennen kann. Wichtiger wären die für den Anwender zur Verfügung zu stellenden Funktionen und der Sicherheitsgewinn. Die Spezifikationen, die dazu notwendigerweise einzuhalten sind, können aber erst im Laufe der Entwicklung - vorläufig durch Simulationen und dann endgültig erst durch Tests - bestimmt werden. Es ist demnach sinnlos, solche Spezifikationen schon im Vorhinein genau festzulegen. Geschieht es trotzdem, so ist dies eine potentielle Ursache für spätere Änderungen.

Weiters ist es auch möglich, dass Anforderungen an das Produkt gestellt werden, die den Entwickler unnötigerweise beschränken, z.B. wenn Materialien oder die Verwendung bestimmter Bauteile in den Anforderungskatalog aufgenommen werden, auch wenn diese nicht unbedingt notwendig sind, weder für die Erfüllung der Funktion noch hinsichtlich anderer Anforderungen, wie Integrierbarkeit ins Fahrzeug oder gesetzlicher Vorschriften.

***Entwicklungstätigkeit in der Vorstudie***

Die Entwicklung der groben Systemarchitektur erfolgt parallel zur Generierung des Anforderungskatalogs, da viele Anforderungen in direktem Zusammenhang mit der Architektur stehen. Die in Abb. 38 dargestellte Systemarchitektur soll als Lösungsprinzip gesehen werden, das am Ende der Vorstudie gewählt wird. Nachdem die vertraglich vereinbarten Anforderungen darauf basieren, soll dieses aber auch ausreichend entwickelt und abgesichert sein. Dazu hat es der Systementwickler schon mit einem Simulationsmodell getestet und ist sich sicher, dass es erfolgversprechend ist. Hierauf wird es dem Kunden präsentiert. Sollte die Systemarchitektur doch nicht dem Kundenwunsch entsprechen, wäre schon viel Aufwand investiert worden. Und es wäre fraglich, ob der Entwickler das Falsche entwickelt oder der Kunde die Anforderungen unzureichend beschrieben hatte. Eventuell muss dann die Vorstudie verlängert oder erneut ausgeführt, der Liefertermin verschoben oder der Vertrag nachverhandelt werden. Im noch ungünstigeren Fall analysiert der Kunde die vorgelegte Systemarchitektur aber gar nicht genau. Dadurch wird das Risiko einer vom tatsächlichen Kundenwunsch abweichenden Entwicklung bis zum geplanten Projektende deutlich größer.

***Hauptstudie***

Der Anforderungskatalog wird als Repräsentation des Kunden gesehen und erst am Ende der Phase gibt es wieder einen Austausch mit dem Kunden. Zwischenzeitlich findet ein Kundenkontakt nur statt, wenn dies von einer Seite ausdrücklich gewünscht wird.

Die zuvor ausgewählte, grobe Systemarchitektur wird nun zu einem Konzept weiterentwickelt, in dem wesentliche Parameter schon feststehen müssen. Es wird auch mit der Suche nach mögliche Lieferanten für Einzelteile oder Komponenten des Systems begonnen. In der Weiterentwicklung der Anforderungen ist jetzt besonderes Geschick gefragt. Im Konzept müssen wesentliche Produkteigenschaften spezifiziert werden, um danach mit der Serienentwicklung starten zu können. Nachdem aber auch Informationen

aus der Serienentwicklung (Produktionsdesign …) einen Einfluss auf das Gesamtkonzept haben können, ist hier mit Schwierigkeiten zu rechnen bzw. mit Änderungen, die in der Detailstudie bekannt werden, aber Rückwirkungen auf das Gesamtkonzept haben können.

Die Erstellung von Prototypen ist in dieser Phase teuer, da noch keine Serienwerkzeuge zur Verfügung stehen. Da die Prototypen sich noch stark vom Serienstand unterscheiden können, sind sie für den Kunden nur bedingt nützlich.

***Detailstudien***

In dieser Phase werden nicht nur die Details zu allen Komponenten fixiert, es wird auch die Produktion geplant und vorbereitet. Je weiter sich die Entwicklung dem Serienstand nähert und einzelne Komponenten schon von den Serienlieferanten verfügbar sind oder bereits mit Serienwerkzeugen gefertigt werden können, umso einfacher ist es auch Prototypen herzustellen. Das wird deshalb auch erst in dieser Phase häufiger getan, was dazu führt, dass Probleme mit dem Systemverhalten häufig erst zu diesem Zeitpunkt erkennbar werden. Als Beispiel soll ein stark vibrierender Bremshebel genommen werden, der von einem Testfahrer bei einem Testlauf mit einem Prototyp des Motorrads gemeldet wird. Ein weiteres beispielhaftes Problem soll ein Nichterreichen der Regelungszeiten im Hydrauliksystem sein. Dies kann auf die Ventileinheit zurückgeführt werden, welche die erforderlichen Stellzeiten nicht erreicht. Solche und ähnliche Probleme können oft nicht durch Berechnungen und Simulationen vorhergesehen werden, sondern werden erst beim Testen mit seriennahen Prototypen zu Tage gefördert.

Beides sind aber Probleme die eine Änderung des Lösungsprinzips oder des Lösungskonzepts nach sich ziehen können. Zu diesem Zeitpunkt sollten aber eigentlich nur mehr Details entwickelt und die Produktion vorbereitet werden. Nachdem der Zeitpunkt für den Serienstart nicht überschritten werden darf (Konkurrenten des Motorradherstellers würden mit einem früheren Marktstart einen deutlichen Wettbewerbsvorteil haben), muss der Entwickler kostspielige Aktionen setzen. Z.B. könnte die Entwicklungskapazität erhöht werden. Dies ist in einer so späten Phase aber oft nicht mehr sinnvoll, da die neuen Entwickler sich im Projekt und mit dem Produkt zurechtfinden müssten. Dementsprechend wurde auch Brooks‘s Law „Adding more manpower to a late project makes it even later“ formuliert (Brooks, 1995). Andererseits sind auch provisorische Maßnahmen am Produkt möglich, welche aber die Stückkosten erhöhen könnten, bis das neue Konzept fertiggestellt und in die Serienfertigung implementiert ist. Solche Änderungen würden vom Kunden kaum bezahlt werden, sondern von diesem eher als Entwicklungsfehler angesehen werden, für die der Entwickler geradezustehen hat.

Sollte sich ein technisches Problem überhaupt nicht lösen lassen, hätte man den ganzen Entwicklungsaufwand bis zur Erkennung des Problems umsonst geleistet.

Ein erfolgreicher Abschluss der Detailstudie bzw. Serienentwicklung durch Erfüllung aller Anforderungen und erfolgreiches Bestehen aller Tests, stellt auch das Ende des Entwicklungsprojekts dar und es wird mit der Serienproduktion begonnen. Sehr häufig unterscheidet sich der Entwicklungsstand des Systems zum SOP aber von den ursprünglich definierten Anforderungen und es kommt zu Nachverhandlungen der Vertragsdetails.

***Unerwartete Änderungen***

In der Detailstudie wurden schon Änderungen innerhalb der Entwicklung besprochen, die sich aufgrund besserer Kenntnisse zum Systemverhalten ergeben hatten, welche aber erst in dieser späten Phase erlangt werden konnten.

Ein weiteres Beispiel wäre ein Wunsch des Kunden nicht mehr nur einen Motorradtyp (Reisemotorrad), sondern mehrere andere (Supersport bis Chopper) auch mit ABS auszustatten. Dieser Wunsch kann aufkommen, nachdem der Kunde die Systemarchitektur gesehen und erkannt hat, dass die Steuerung durch die ECU eine Anpassbarkeit des Systems für unterschiedliche Motorradtypen leicht umsetzbar macht. Er gibt dazu in Auftrag die Brauchbarkeit des Systems bei den fahrdynamischen Verhältnissen der zusätzlichen Typen zu überprüfen, ist aber nicht bereit dafür zu bezahlen, mit dem Hinweis, dass der Entwickler durch die Erhöhung der Stückzahl in der Lieferphase ohnehin mehr verdienen würde. Außerdem darf der Produktionsstart für das ABS des Reisemotorrads nicht verschoben werden. Bei entsprechender Gewinnaussicht wird die Geschäftsführung des Systementwicklers damit einverstanden sein und es wird an der Entwicklungsabteilung liegen, mit dieser Veränderung umzugehen. Erfolgt diese am Ende der Vorstudie bei Analyse der Systemarchitektur, ist vermutlich die Zeit bis zum Serienstart auch noch ausreichend für eine erfolgreiche Umsetzung.

Derselbe Kundenwunsch kann aber auch zu einem anderen Zeitpunkt durch eine Änderung von außen entstehen. Eine Gesetzesänderung kann nach ABS Systemen für alle neu zugelassenen Motorräder verlangen. Die Machbarkeit der Änderung hängt dann von der noch vorhandenen Flexibilität zum jeweiligen Zeitpunkt ab. Diese ist auch gefragt, wenn als nächstes Beispiel für eine Veränderung von außen, die Entwicklung einer neuen Technologie angenommen werden soll. Diese könnte z.B. einen Sensor oder einen Aktuator möglich machen, der einen ABS Einsatz auch bei Kurvenfahrt (Schräglage) ermöglicht. Dann würde sich die Frage stellen, ob diese Funktion im System noch integrierbar wäre.

Änderungen, die unerwartet und unabhängig vom Entwicklungsstand auftreten, können aber auch Probleme darstellen, wenn die Flexibilität zur Implementierung dieser Änderung im Produkt sogar vorhanden wäre. Oft ist es nämlich im Prozess nicht möglich, kosten- und terminneutral zu reagieren. Zu detailliert sind diese Dinge oft geplant und optimiert und zu wenig auf mögliche Änderungen vorbereitet, um dann die nötige Reaktionsfähigkeit bieten zu können.

***Nach Projektende***

Auch nach Projektende können noch Veränderungen auftreten. Umsetzbarkeit und Kosten von Änderungen sind im Wesentlichen von der Flexibilität in Produkt und Produktionslinie abhängig. Wenn vom Systementwickler im Vorhinein kein Wert darauf gelegt wurde, sind relativ hohe Kosten zu erwarten. Solche Änderungen sind normalerweise aber auch in einem neuen Vertrag auszuhandeln.

## 8.4 Agiles BWI-Hall Vorgehensmodell

Es soll hier auch weiterhin angenommen werden, dass frühe Prototypen des Systems ressourcenaufwändig herzustellen sind und die daraus zu gewinnenden Erkenntnisse in keiner Relation zum Aufwand stehen. Außerdem ist der Kunde nicht daran interessiert, jedes Zwischenergebnis präsentiert zu bekommen, er will nur zu vereinbarten Terminen Prototypen zur Integration ins Fahrzeug

und zum Ende jeder Phase einen Fortschrittsbericht. Der agile Entwicklungszyklus der darauf beruht, die Zwischenergebnisse dem Kunden oder Anwender zu präsentieren, um darauf Feedback zu erhalten, ist damit nach wie vor nicht umsetzbar und es soll nach dem BWI-Hall Modell entwickelt werden. Dieses soll aber so agil wie möglich ausgelegt werden, was dem in Abb. 33 gezeigten agilen BWI-Hall Vorgehensmodell entspricht. Es sollen dabei auch möglichst viele der, in Abb. 30 gezeigten, Maßnahmen umgesetzt werden.

***Produktvision***

Es folgt auch hier noch eine detaillierte Beschreibung der Anforderungen in einem Katalog, nach welchem sich die weitere Entwicklung richtet. Die vom Kunden beschriebene Produktvision kann vom Entwickler aber genutzt werden, um die tatsächlichen Kundenwünsche besser zu verstehen.

***Projektplanung***

Nachdem auch hier die Festlegung eines Fixpreises angestrebt wird, muss relativ detailliert im Vorhinein geplant werden. Es wird aber versucht, nur den Aufwand genau zu planen. Details zum Ablauf können ev. erst später geplant werden, nämlich dann wenn sie unmittelbar bevorstehen und keine Änderungen mehr zu erwarten sind. Auch sonst wird mit dem Wissen geplant, dass Änderungen in komplexen Systementwicklungsprojekten unvermeidlich sind. Der Entwicklungsprozess darf dazu nicht zu starr gestaltet sein, z.B. soll der laufende Dokumentationsaufwand niedrig gehalten werden. Jene Entwicklungsschritte welche viele Informationen liefern, sind möglichst früh einzuplanen. Das Anforderungsmanagement ist für einen effizienten Umgang mit Änderungen zu gestalten. Veränderungen außerhalb der Entwicklung können eventuell antizipiert werden (Diskussionen über die Einführung einer ABS Pflicht für Motorräder).

***Anforderungskatalog***

Zu Beginn formulierte Anforderungen gelten auch hier als fixiert. In einer guten Abstimmung der Anforderungen mit dem Kunden kann der Umgang mit möglichen Änderungen schon im Vorhinein wesentlich erleichtert werden. Z.B. können einzelne Anforderungen zu Beginn unbestimmt bleiben und der Kunde erhält einen Termin, bis zu dem die Anforderungen endgültig fixiert werden müssen, unter Berücksichtigung der Kosten. Aber auch ein möglichst gutes Verstehen der tatsächlichen Anforderungen kann spätere, dahingehende Änderungen vermeiden.

***Vorstudie***

Die Erstellung einer groben Systemarchitektur ist auch hier der Inhalt der Vorstudie. In dieser Phase macht es vermutlich noch keinen Sinn eine explizit zyklische Entwicklung einzuführen. Durch häufiges Abstimmen (auch mit dem Kunden) versucht man aber Informationen zu Fehlern, tatsächlichen Anforderungen und Änderungen möglichst früh zu erhalten. Es wird auch versucht, die Systemarchitektur möglichst flexibel zu gestalten.

Durch die flexible Gestaltung der Systemarchitektur kann eventuell auch besser mit der vorher beschriebenen Ausweitung des Einsatzgebietes für das ABS-System umgegangen werden, wenn der Kunde danach verlangt. Wird häufiger der Kontakt gesucht, so besteht früher die Chance, diesen Wunsch zu erkennen. Außerdem ist es möglich, schneller und kostengünstiger darauf zu reagieren, wenn die vorhandene Flexibilität dies ermöglicht. So kann es unter Umständen geschafft werden, die

Untersuchungen zum Einsatz auch bei anderen Motorradtypen schon innerhalb der Vorstudie auf den gleichen Stand zu bringen, wie beim ersten Motorradtyp.

Um agil zu bleiben, empfiehlt es sich unter Berücksichtigung der wirtschaftlichen, technischen und organisatorischen Sinnhaftigkeit so viel Flexibilität wie möglich im Produkt oder den Zwischenergebnissen (Systemarchitektur, Simulationsmodell) offen zu halten. Es soll auch wenn möglich die Nähe zum Kunden gesucht werden, egal ob durch die Entwickler oder auf anderen Ebenen.

***Hauptstudie***

Eventuell kann hier schon in kleinen expliziten Schritten entwickelt werden. Nachdem noch keine Realisierung durchgeführt wird, können die Schrittdauern für die Entwicklung der Hardwarekomponenten auch relativ kurz sein. Die kurzen Zyklen können das Lernen der Entwickler und die Wissensverbreitung im Team oder in der Unternehmung durch eine geregelte Abstimmung fördern. Auch wenn Prozessmodelle oft linear gezeichnet werden, ist das implizite Durchführen von kleinen Entwicklungsschritten ohnehin gängige Praxis. Es ist durchaus denkbar, dass diese Schritte auch explizit angesetzt werden (mit impliziten Zwischenschritten) und dadurch die Agilität erhöhen, auch wenn keine für den Kunden nutzbaren Zwischenergebnisse entstehen, sondern nur interne Abstimmung passiert. Durch die häufigeren Deadlines dieser Zyklen, aber auch durch die Vermeidung von unnötigen Prozessvorschriften kann die Effizienz der Entwicklung verbessert werden.

***Detailstudien***

In dieser Phase werden nicht nur die Details zu allen Komponenten fixiert, es wird auch die Serienproduktion geplant und vorbereitet. Aufgrund der immer umfangreicher zur Verfügung stehenden Serienwerkzeuge und der damit immer einfacher herzustellenden Prototypen, können hier die expliziten Entwicklungsschritte auch leichter (zumindest teilweise) nutzbare Zwischenergebnisse liefern. Auch wenn hier die Agilität höher ist, als in der ersten Version des Fallbeispiels, ist dieser Zeitpunkt für Änderungen als sehr spät zu betrachten. Änderungen innerhalb dieser Phase sind zwar leichter möglich, Rückgriffe auf frühere Phasen trotzdem mit großem Aufwand verbunden. Ein Vibrieren des Bremshebels und auch das Problem mit den Stellzeiten der Ventileinheit würden vermutlich auch erst jetzt zu Tage treten. Eventuell kann damit aber leichter umgegangen werden, weil mehr auf Flexibilität geachtet wurde. Außerdem werden diese Änderungen weniger als Fehler betrachtet, sondern als normale Entwicklungstätigkeit und deshalb von den Mitarbeitern mit höherer Motivation behandelt.

***Unerwartete Änderungen***

Unerwartete Änderungen von außen sind aufgrund der getroffenen Maßnahmen (Beachtung der Flexibilität, Änderungsmanagement) hier eventuell etwas einfacher handzuhaben. Je nach Zeitpunkt des Auftretens können sie aber doch große Herausforderungen bedeuten, speziell wenn auf frühere Phasen zurückgegriffen werden muss. Auf die erwähnte Gesetzesänderung kann zu einem späten Zeitpunkt also kaum mehr durch eine Anpassung des, schon fast fertig entwickelten, Systems reagiert werden. Die Software könnte eventuell noch geändert werden. Aber Hardwarekomponenten, welche nicht mit den zusätzlichen Motorradtypen kompatibel sind, wären sehr schwierig oder aufwändig anzupassen, speziell wenn die Produktionswerkzeuge dafür schon hergestellt wurden.

***Nach Projektende***

Eventuell können Änderungen auch zu diesem Zeitpunkt etwas leichter umgesetzt werden, weil auf Flexibilität im Produkt geachtet wurde. Zu Lernzwecken wird man hier auch versuchen, möglichst viele Informationen aus dem in Nutzung befindlichen System zu gewinnen.

## 8.5 AEZ-SE Vorgehensmodell

In dieser Version des Fallbeispiels soll angenommen werden, dass keine Hindernisse für die Anwendung des AEZ-SE Vorgehensmodells bestehen. Es soll die in Abb. 34 dargestellte Vorgehensweise als Grundlage für den Entwicklungsprozess verwendet und möglichst alle der in Abb. 30 dargestellten Maßnahmen sollen umgesetzt werden.

***Produktvision***

Im ersten Schritt tritt der Kunde mit der in 8.1 beschriebenen Vorstellung des Produktes an den Systementwickler heran. Entsprechend dem hier vertretenen agilen Ansatz wird danach zwar auch ein Anforderungskatalog ausgearbeitet, dieser ist aber flexibel und kann geändert werden. Die Produktvision soll das grundlegende Verständnis über die Funktion des Produkts herstellen.

***Projektplanung***

In diesem Schritt wird die grobe Planung für das Projekt durchgeführt. Dazu gehören der Termin für den Anlauf der Produktion (Projektabschluss), aber auch Termine für Meilensteine (z.B. wann der Motorradhersteller Prototypen braucht, um mit der Integration ins Fahrzeug beginnen zu können). Anhand der beschriebenen Produktvision stellt der Systementwickler ein Entwicklungsteam zusammen, das allen Entwicklungsaufgaben, die sich aus der Produktvision erkennen lassen, gewachsen sein soll. Dabei ist darauf zu achten, dass nicht nur Wissen zu allen technischen Belangen vorhanden ist, sondern auch Prozesswissen für die Entwicklung sicherheitskritischer Funktionen. Beim hier verwendeten flexiblen Anforderungsmanagement müssen die sicherheitskritischen Funktionen speziell behandelt und dokumentiert werden.

Unter Zugrundelegung der erkannten Entwicklungsumfänge und unter Berücksichtigung der Meilensteine des Kunden definiert das Team den Entwicklungsprozess und legt zusätzliche interne Meilensteine fest. Ebenso soll von Anfang an abgeklärt werden, welche Entwicklungsumfänge bereits am Markt verfügbar sind und mit potentiellen Lieferanten Kontakt aufgenommen werden.

In diesem Schritt der Projektplanung wird noch kein (wesentlicher) Entwicklungsaufwand getrieben. Alle Festlegungen (Team, Prozess …) sind nur als vorläufig zu betrachten. Überall, wo Flexibilität nach technischen, wirtschaftlichen und organisatorischen Gesichtspunkten sinnvoll erscheint, wird diese offen gehalten, um auf mögliche Änderungen besser reagieren zu können.

***Anforderungskatalog***

Im Rahmen der Projektplanung soll auch ein erster Anforderungskatalog erstellt werden, im Sinne eines Product-Backlogs (siehe Kapitel 5.1.4). Die Produktvision wird also in eine erste Anforderungsliste übergeleitet, welche konkrete technische Spezifikationen beinhaltet. Diese sind aber noch nicht als fixiert anzusehen, sondern können bei Bedarf noch verändert, ergänzt oder gelöscht werden. Zusammen

mit dem Kunden werden die Anforderungen auch priorisiert. Sicherheitskritische Anforderungen werden dabei als unveränderlich gekennzeichnet (Höchste Priorität). Technische Spezifikationen, wie z.B. genaue Regelungszeiten des Systems werden noch nicht fixiert. Diese können zum jetzigen Zeitpunkt auch noch nicht sinnvoll bestimmt werden, da der Kunde noch überhaupt nicht wissen kann, wie hoch diese genau sein dürfen. Wichtig sind die, dem Anwender zur Verfügung zu stellenden Funktionen und der Sicherheitsgewinn.

***Entwicklungsschritt 1***

Wie in Abb. 34 dargestellt, beginnt jeder explizite Entwicklungsschritt mit einer Situationsanalyse. Bei dieser Tätigkeit wird erkannt, dass zuerst eine grobe Systemarchitektur benötigt wird, um die Gesamtfunktion und auch einzelne Komponenten entwickeln zu können. Deshalb wird dieser Entwicklungsschritt mit dem Inhalt der Erstellung der Systemarchitektur geplant und stellt damit gemäß Abb. 34 einen Planungsschritt dar. In der Zielformulierung werden die Anforderungen aus dem Anforderungskatalog ausgewählt, die durch die grobe Systemarchitektur abgedeckt werden sollen. In der Lösungssynthese werden durch das Team verschiedene Lösungsvarianten erstellt, die dann in der Lösungsanalyse auf Machbarkeit überprüft und einer Risikobewertung zugeführt werden. Prinzipiell taugliche Varianten werden in der Bewertung nach den in der Zielformulierung definierten Kriterien bewertet. Zum Abschluss soll eine Lösung ausgewählt werden. Der Problemlösungszyklus darf dazu implizit auch so oft wie nötig durchlaufen werden, bis eine zufriedenstellende Lösung gefunden ist. Implizite Entwicklungsschritte können dazu auch nur Teile des Problemlösungszyklus betreffen z.B. wenn bei der Lösungsanalyse keine Lösung als machbar betrachtet wird, geht man zurück zur Synthese um neue Lösungsmöglichkeiten zu generieren. Die für den expliziten Entwicklungsschritt geplante Zeitdauer soll aber im Sinne des Timeboxings (5.4.8) nicht überschritten werden. Am Ende dieser Zeitdauer sollen die vorhandenen Ergebnisse bewertet werden, um den weiteren Entwicklungsverlauf abzustimmen.

Während eines expliziten Entwicklungsschritts kann permanent ein Vertreter des Kunden anwesend sein oder es kann zwischenzeitlich mit dem Kunden kommuniziert werden. Um einen agilen Entwicklungszyklus abzubilden, soll dem Kunden zumindest das Ergebnis eines expliziten Entwicklungsschritts präsentiert werden, um sein Feedback abfragen zu können. Im angeführten Fall soll das Ergebnis des ersten expliziten Entwicklungsschritts die in Abb. 38 dargestellte grobe Systemarchitektur sein.

Bei der Erklärung der Systemarchitektur erkennt der Kunde das große Potential dieser Lösung. Die Steuerung durch die ECU gewährleistet die Anpassbarkeit des Systems für unterschiedliche Motorradtypen. Es wird wieder angenommen, dass der Kunde nach diesem Kommunikationsschritt auch mehrere weitere Motorradtypen mit ABS ausstatten will, was ursprünglich nicht geplant war. Er gibt dazu den Auftrag, die Brauchbarkeit des Systems bei den fahrdynamischen Verhältnissen dieser Typen zu überprüfen.

Es kann angenommen werden, dass der explizite Entwicklungsschritt kürzer gedauert hat, als die Vorstudien in 8.3 oder 8.4. Schließlich mussten auch keine Entscheidungen getroffen werden, wie am Ende einer Phase und es konnte deshalb schneller entwickelt werden, da weniger Arbeit in die Absicherung des Prinzips investiert werden musste. Durch das schnellere Vorhandensein der

Systemarchitektur, kann der Änderungswunsch des Kunden früher erkannt und schneller darauf reagiert werden.

***Entwicklungsschritt 2***

Ursprünglich hatte der Entwickler in seinem Prozess als zweiten expliziten Entwicklungsschritt einen einfachen Prototyp geplant, um das Lösungsprinzip materiell darstellen zu können. Die neue Situation lässt aber die Erstellung eines Simulationsmodells sinnvoller erscheinen. Dieses soll so gestaltet werden, dass die unterschiedlichen Motorradtypen durch die Veränderung von Parametern, auf Integrierbarkeit des ABS Systems überprüft werden können. Das Testen von mehreren Motorradtypen mit materiellen Prototypen ist natürlich möglich, aber es wird angenommen, dass dies höhere Kosten verursacht hätte. Diese am Anfang des zweiten expliziten Entwicklungsschritts gewonnene Erkenntnis, führt nun zuerst zur Planung und dann zur Umsetzung des Simulationsmodells mit allen Schritten des Problemlösungszyklus.

Für die Erstellung des Simulationsmodelles werden Software- und Funktionsentwickler benötigt. Die Daten zur Fahrdynamik des Motorrads liefert der Kunde. Die Hardware-Entwickler unterstützen die Modellerstellung soweit nötig und möglich, kümmern sich sonst aber gemeinsam mit dem Einkauf um die Bestimmung von möglichen Zukaufteilen und eruieren die für die Simulation benötigten Spezifikationen.

Bei den Anforderungen für diesen Entwicklungsschritt und den Testkriterien wird sinngemäß auf die Funktion des Simulationsmodells Wert gelegt. Das Modell wird in kleinen impliziten Entwicklungsschritten solange weiterentwickelt, bis es zufriedenstellende Ergebnisse aus Sicht des Systementwicklers liefert. Danach wird es dem Kunden präsentiert. Bei der implizit schrittweisen Weiterentwicklung des Modells ist darauf zu achten, dass dem Kunden zum vereinbarten Zeitpunkt eine lauffähige Version vorgeführt werden kann, auch wenn diese noch nicht perfekt ist.

Das Simulationsmodell stellt auch ein Zwischenergebnis auf dem Weg zum fertigen System dar, anhand dessen die Funktion überprüft werden kann. Durch das Kundenfeedback können die Anforderungen weiter detailliert und die nächsten Schritte geplant werden. Es würde hier auch schon vorzeitig erkannt werden, sollte der Kunde grundsätzlich mit dem Konzept nicht zufrieden sein und es könnte die Erstellung eines neuen Konzeptes als nächster Entwicklungsschritt durchgeführt werden. Hier soll aber angenommen werden, dass das Gesamtkonzept als tauglich - auch für den erweiterten Anwendungsbereich des Kunden - erscheint. Mithilfe des Simulationsmodells wurde auch eine Parametrisierung des Systemdesigns identifiziert, die lediglich zu verschiedenen Softwareparametern bei Verwendung in unterschiedlichen Motorradtypen führt.

Als weiterer Output der Simulation wird angenommen, dass die Ventileinheit als kritische Komponente für die Regelungsgenauigkeit und -geschwindigkeit des ABS-Systems erkannt wurde. Eine erhöhte Priorisierung der entsprechenden Anforderungen führt daher zur Auswahl der Ventileinheit als nächsten Entwicklungsschritt.

***Entwicklungsschritt 3***

Wie eben beschrieben, werden nun alle möglichen Ressourcen dazu verwendet, um die Ventileinheit zu entwickeln und einen materiellen Prototypen herzustellen. Dieser explizite Entwicklungsschritt kann also als kombinierter Planungs- und Realisierungsschritt angesehen werden. Nachdem die Ventileinheit als

kritisch für die Gesamtfunktion angesehen wird, dient dieser Entwicklungsschritt auch der Risikobegrenzung. Sollte sich herausstellen, dass das Gesamtkonzept an der Ventileinheit scheitert, so würde man das direkt nach diesem neu eingeplanten Entwicklungsschritt herausfinden. Änderungen am Gesamtkonzept könnten dann noch relativ leicht durchgeführt werden und es wäre nur der Entwicklungsaufwand für die Ventileinheit riskiert worden. Würde man nach herkömmlicher Weise zuerst das ganze System fertig entwickeln, hätte man viel mehr Entwicklungsaufwand riskiert, der dann an der Ventileinheit scheitern könnte.

Die Testkriterien wurden anhand der, durch die Simulation im vorigen Schritt genauer bestimmten, Anforderungen erstellt. Die Ventileinheit wird dann in impliziten, kleinen Entwicklungsschritten so lange weiterentwickelt, bis sie die Tests erfüllt. Sollte das nicht innerhalb der Zeitdauer eines expliziten Entwicklungsschritts erreicht werden, wird der Zwischenstand dennoch präsentiert und es wird ein weiterer expliziter Entwicklungsschritt dafür eingeplant. Auch unterschiedlich lange explizite Entwicklungsschritte sind denkbar. Dabei spielt es keine Rolle, ob die Ventileinheit eine Eigenentwickelung darstellt. Sollte diese Einheit am Markt verfügbar sein, ist natürlich auch ein Zukauf möglich und statt im Problemlösungszyklus eine Eigenentwicklung durchzuführen, würde darin dann die Auswahl aus mehreren am Markt vorhandenen Einheiten durchgeführt werden. Der Test der Ventileinheit erfolgt nach dem Prinzip „Hardware in the Loop“, wobei die anderen Bauteile mit dem im vorigen Entwicklungsschritt erstellten Modell simuliert werden.

In diesem Beispiel soll nun davon ausgegangen werden, dass mit einer bestimmten Ventileinheit die Gesamtfunktion darstellbar ist, aber andere Bauteile speziell darauf abgestimmt werden müssen. Diese Anforderungen waren zu Projektbeginn nicht klar und fließen jetzt in den Anforderungskatalog (Product-Backlog) ein. Ein Nachverhandeln des Vertrages ist nicht notwendig, es wurde bisher auch noch kein unnötiger Entwicklungsaufwand getrieben. In dieser Version des Fallbeispiels wurden damit im Vergleich zur ersten Version Zeit und Kosten gespart, da bessere Kenntnisse zum Systemverhalten zu einem früheren Zeitpunkt generiert werden konnten und keine Änderung bzw. kein Fehler zu einem späten Zeitpunkt behandelt werden musste (der Prototyp der Ventileinheit wird dabei kostengünstiger als die späte Änderung angenommen).

***Weitere Entwicklungsschritte***

Die weiteren, expliziten Entwicklungsschritte werden nach Bedarf festgelegt. Kriterien zur Prioritätenbildung wurden in Kapitel 5.1.5 beschrieben. Es kann durchaus vorkommen, dass die expliziten Entwicklungsschritte einzelnen Komponenten entsprechen. Nachdem Funktionen oft nur durch ein Zusammenspiel mehrerer Komponenten zustande kommen, ist eine parallele Weiterentwicklung mehrerer Komponenten sinnvoll. Dabei wird es zu unterschiedlichen Zykluszeiten in der Entwicklung der einzelnen Komponenten kommen, z.B. kann Software viel schneller produziert und auch geändert werden, während Hardware einem physikalisch aufwändigeren Herstellungsprozess unterliegt. Der Test der Gesamtfunktion kann dann nur zu Synchronisationszeitpunkten, sogenannten Anchoring Point Milestones passieren (siehe Kapitel 5.4.5).

Entwicklungsschritte, bei denen eine Abstimmung zwischen Hardware und Software notwendig ist, sind sehr wahrscheinlich, da die Darstellung der Funktion eine Integration dieser beiden Komponenten benötigt. Diese wird zu vorher bestimmten Meilensteinen durchgeführt. Es ist möglich, diese Anchoring

Point Milestones zu frei definierten Zeiten abzuhalten oder zyklisch (z.B. nach jeweils 3 Softwarezyklen wird ein Hardwarezyklus beendet und auch ein Anchoring Point Milestone abgehalten). Die Zeitabstände müssen natürlich an die Zykluszeiten der einzelnen Komponenten angepasst sein.

Es kann aber auch Entwicklungsschritte geben, die nur Funktionen behandeln, die entweder nur Hardware oder nur Software betreffen. Diese können dann von unterschiedlichen Entwicklungsteams getrennt entwickelt und dem Kunden vorgeführt werden. Andere Anordnungen der expliziten Entwicklungsschritte sind ebenso möglich, solange sie sinnvoll und dem Entwicklungsfortschritt dienlich sind. Die Entwickler tragen selbst die Verantwortung dafür, die Entwicklung bestmöglich durchzuführen. Es ist auch denkbar, dass für die Softwareentwicklung eine der in 2.2 beschriebenen agilen Methoden verwendet und in den Systementwicklungsprozess integriert wird.

***Projektende***

Gegen Ende des Projekts benötigt der Kunde eine Reihe von seriennahen Prototypen für Vorserienfahrzeuge. Die Entwicklungsschritte sollten sich dann hauptsächlich auf die Realisierung beschränken und kaum noch Designänderungen beinhalten. Natürlich wird das Feedback von Kunden und Anwendern aber weiterhin verwendet um Verbesserungen durchzuführen.

Wenn die finalen Prototypen den tatsächlichen (und möglicherweise geänderten) Kundenanforderungen entsprechen, der Kunde also am Ende eines Entwicklungsschritts, wie in Abb. 34 gezeigt, alle Anforderungen als erfüllt betrachtet und das Ergebnis akzeptiert, ist die Entwicklung abgeschlossen.

Die schrittweise Entwicklung soll sicherstellen, dass dem Kunden zumindest ein eingeschränkt lauffähiges System geliefert werden kann, wenn zu einem Liefertermin die benötigten Anforderungen noch nicht vollinhaltlich umgesetzt werden konnten. Damit können z.B. Terminverschiebungen in der Entwicklung des Motorrades verhindert werden, wenn z.B. nur die Software des ABS Systems noch nicht fertig entwickelt wurde. Diese kann dann per Software-Update nachgereicht werden. Wichtig ist hier nur die rechtzeitige Prioritätenbildung für Funktionen, die der Kunde unbedingt zur Weiterentwicklung des Motorrades braucht. Auch auf sicherheitskritische Anforderungen ist besonders zu achten.

***Unerwartete Änderungen***

Gesetzliche Änderungen, neue oder aktualisierte Kundenwünsche oder neu verfügbar werdende Technologien können unerwartet zu Anforderungsänderungen während der Entwicklung führen. Der in diesem Fallbeispiel beschriebene Entwicklungsprozess lässt solche Änderungen zu. Für eine technische und wirtschaftliche Realisierbarkeit dieser Änderungen ist aber auch die Flexibilität im System wichtig. Um diese Flexibilität zu gewährleisten bzw. zu vergrößern, können z.B. die in Kapitel 5.5 beschriebenen Prinzipien angewandt werden. Damit sollen Spielräume für spätere Änderungen zur Verfügung gestellt werden. Je größer die Produktflexibilität ist, umso größer ist das Spektrum unerwarteter Anforderungsänderungen, welche noch im Produkt implementiert werden können.

Der Prozess ist ebenfalls wichtig für den Umgang mit unerwarteten Veränderungen außerhalb der Entwicklung. Hier geht es aber mehr darum, wie schnell die Veränderungen und ihre Relevanz erkannt und welche Verfahren dann angewandt werden, um auf erkannte Veränderungen zu reagieren. Der Prozess ist aber wichtiger für die Implementierbarkeit von Anforderungsänderungen innerhalb der Entwicklung, die sich aufgrund eines höheren Wissensstandes über das Produkt ergeben. Hier gilt es, den

Kunden möglichst früh mit Zwischenergebnissen der Entwicklung zu versorgen, um von ihm Feedback für die weitere Entwicklung erhalten zu können.

Als beispielhafte Änderung innerhalb der Entwicklung soll hier wieder der stark vibrierende Bremshebel verwendet werden. Das Entscheidende am Prozess ist, dass dieses Problem so früh wie möglich erkannt wird und sofort Maßnahmen eingeleitet werden. Je früher es erkannt wird, umso wahrscheinlicher kann auch noch bei einer wenig flexiblen Systemarchitektur eine technisch und wirtschaftlich sinnvolle Lösung gefunden werden. Wird ein Problem sehr spät erkannt, z.B. erst wenn schon wesentliche Vorbereitungen für die Serienproduktion getroffen wurden und auf Flexibilität nicht speziell Wert gelegt wurde, so sind Änderungen nur mit großem Aufwand möglich. Bei Verwendung des AEZ-SE Modells sind eine frühere Erkennung dieses Problems und dadurch eine einfachere Behandlung zu erwarten, als in den ersten beiden Versionen dieses Fallbeispiels.

Als Beispiel für eine Änderung von außen sei wieder die Gesetzesänderung genannt, die nach ABS Systemen für alle neu zugelassenen Motorräder verlangt. Es stellt sich hier die Frage, wie weit solche Informationen schon als schwache Signale wahrnehmbar sind und vom Projektteam oder anderen Stellen in der Firma beobachtet werden können, bevor sie wirklich zu Tatsachen werden. Erkennt der Entwickler rechtzeitig, dass die Möglichkeit einer baldigen Einführung einer allgemeinen ABS Pflicht besteht, kann er eine dahingehende Anpassbarkeit einplanen. Ist die Veränderung bereits eingetreten, kann der Prozess nur noch die Reaktion regeln. Die Machbarkeit der Änderung hängt dann aber von der Flexibilität im System ab. Diese ist auch gefragt, wenn als zweites Beispiel für eine Veränderung von außen die Entwicklung einer neuen Technologie angenommen werden soll. Diese könnte z.B. einen Sensor oder einen Aktuator möglich machen, der einen ABS Einsatz auch bei Kurvenfahrt (Schräglage) ermöglicht. Auch dann würde sich die Frage stellen, ob diese Funktion im System noch integrierbar wäre.

***Nach Projektende***

Je nach Flexibilität im System sind Änderungen mehr oder weniger leicht zu implementieren. Software-Updates bieten eine relativ kostengünstige Methode um Änderungen der Funktionalität im Feld durchzuführen. Änderungen die an der Produktionslinie für zukünftig zu produzierende Einheiten durchgeführt werden sollen, sind in ihrer technischen und wirtschaftlichen Machbarkeit wieder stark von der Flexibilität abhängig, sowohl jener des Produktes als auch jener der Produktionslinie. Nachdem die Flexibilität des Systems hier eine Voraussetzung für das Vorgehensmodell war, ist natürlich auch eine gute Anpassbarkeit nach Projektende zu erwarten.

***Offene Fragen und Probleme***

Die wichtigste offene Frage betrifft wohl die Preisfestsetzung für das Entwicklungsprojekt. Traditionellerweise wird ein Preis für einen genau definierten Entwicklungsumfang zu Beginn bestimmt und jede Änderung muss neu verhandelt werden. Komplikationen sind vorprogrammiert, da bei komplexen und neuartigen Systemen immer wieder Anforderungsänderungen oder unerwartete Entwicklungstätigkeiten auftauchen, von denen der Kunde und der Entwickler unterschiedliche Sichtweisen über die Zugehörigkeit zum bestehenden Vertrag haben. Die Entwicklung nach dem AEZ-SE Vorgehensmodell beinhaltet diesbezüglich Chancen und Gefahren. Der Entwickler hat normalerweise das Problem, dass er den Preis nicht hoch (um genug Reserven für Änderungen zu haben) ansetzen darf, weil er dann im Vergleich mit konkurrierenden Anbietern (welche potentielle Änderungen nicht im Preis

berücksichtigen) das Kostenargument gegen sich hat. Andererseits müssen die Änderungen, die sich aufgrund der Komplexität und Neuartigkeit des Systems erst im Laufe der Entwicklung herausstellen können, natürlich wirtschaftlich berücksichtigt werden.

Eine Möglichkeit dem gegenüber zu treten, könnte auch bei der Preisfestsetzung eine agile Vorgehensweise sein. Es könnte z.B. jeder explizite Entwicklungsschritt einzeln bewertet werden. Damit wäre es einerseits viel leichter möglich, sich einem adäquaten Preis anzunähern, andererseits wäre auch das Risiko begrenzt, da jederzeit abgebrochen werden könnte und die Kosten für die bisherige Entwicklungstätigkeit transparent wären.

Es wäre auch möglich, auf eine gute Zusammenarbeit zwischen Kunden und Entwickler zu setzen. Der anfänglich vereinbarte Preis für Entwicklung und Serienproduktion kann dann während der Entwicklung angepasst werden, um den tatsächlichen Entwicklungsaufwand, bzw. die tatsächlichen Produktionskosten abzubilden. Dabei wäre natürlich ein faires Verhalten beider Partner Voraussetzung.

Nicht nur für die Preisfestlegung, sondern auch für die Anwendbarkeit dieses Vorgehensmodells im Allgemeinen, sind die Gewohnheiten der Industrie und der Willen sich damit auseinanderzusetzen ein wesentliches Kriterium.

## 8.6 Set-Based-SE Vorgehensmodell

Für die letzte Version des Falles soll das Set-Based-SE Vorgehensmodell aus Abb. 35 verwendet werden. Dieses kann eventuell einen guten Kompromiss darstellen, da es agiler als das BWI-Hall Modell sein sollte, aber auch bei mehr Arten von Systemen anwendbar ist, als das AEZ-SE Modell. Vor allem ist es auch bei wenig flexiblen Systemen anwendbar, die keine Entwicklung in kleinen Schritten nach den Ausprägungsformen des AEZ zulassen.

Bei diesem Vorgehensmodell wird der Entwicklungsfortschritt durch eine spezielle Betrachtung der Machbarkeit der Systemteile gelenkt. Die Einbindung des Kunden und die Fokussierung auf seine tatsächlichen Anforderungen sind nicht direkt vorgesehen, sie können aber implementiert werden.

***Produktvision***

Auch dieses Modell startet mit einer Produktvision des Kunden. Diese wird für die Ermittlung des Lösungsbereichs benötigt. Einen Anforderungskatalog benötigt man trotzdem, schon alleine um die Zielerfüllung zu überprüfen. Die Entwicklung wird aber primär nicht über die Erreichung der Anforderungen gelenkt, sondern über die Machbarkeitsüberprüfung der Varianten im Lösungsbereich. Dazu muss sichergestellt werden, dass der Lösungsbereich auch der Produktvision entspricht.

***Projektplanung***

Es werden hier wieder Termine, die Teamzusammensetzung, Kommunikationsregeln u.Ä. geplant. Der Anforderungskatalog wird auch formuliert, der hier als definitiv betrachtet oder auch flexibel sein kann. Außerdem wird der Lösungsbereich entworfen und die Fachbereiche werden eingeteilt, die innerhalb des Lösungsbereichs die Machbarkeit überprüfen sollen. Diese sollen hier folgende sein:

- Mechanische Hardware
- Elektronische Hardware

- Hydraulik
- Software
- Produktionsplanung

Es ist hier auch möglich, den Kunden als Fachbereich in die Entwicklung einzubeziehen. Er kann dann genauso an Abstimmungsrunden teilnehmen und den Lösungsbereich einengen, indem er hilft Varianten auszuscheiden, mit denen er seine tatsächlichen Anforderungen als nicht machbar erachtet. Würde es sich im Fallbeispiel nicht um eine Auftragsentwicklung handeln, so hätte das Marketing die Aufgabe, die Einengung des Lösungsbereichs in Richtung der Marktwünsche voranzutreiben.

***Lösungsbereich***

Der Lösungsbereich stellt jene Menge an Varianten dar, mit denen - nach dem Informationsstand zu Projektbeginn - die Produktvision umsetzbar erscheint. Im Gegensatz zu einem Anforderungskatalog enthält er also keine Ziele sondern Lösungen. Es ist aber bei seiner Erstellung noch nicht klar, ob diese Lösungen machbar sind oder welche die beste Variante davon ist. Die Lösungsvarianten sind auch noch nicht konkret ausformuliert, sondern stellen die Summe aller machbaren Lösungen innerhalb eines Bereichs dar. Im Fallbeispiel wird der Lösungsbereich als erstes durch die Möglichkeiten zur Umsetzung der gewünschten Funktionalität begrenzt. Dazu sind aber durchaus mehrere Lösungsprinzipien denkbar, die sich von der Systemarchitektur in Abb. 38 unterscheiden können. Weitere Grenzen können durch die Schnittstellen zu den vorhandenen Motorradkomponenten und die sonstigen Kundenwünsche definiert werden. Die Anschlussmaße zu den vorhandenen Betätigungshebeln können den Lösungsbereich genauso begrenzen, wie der Druckbereich in dem Motorradbremssysteme normalerweise funktionieren. Weitere Grenzen sind das zulässige Gewicht, die verfügbare Energieversorgung, Sicherheitsrichtlinien und auch Produktionskosten. Innerhalb des Lösungsbereichs sind aber noch eine ganze Reihe von Kombinationen aus hydraulischen und elektrischen Schaltbildern, Ventileinheiten, Pumpen, mechanischen und elektronischen Reglern und Sensoren und Softwaredesigns möglich.

***Entwicklung***

Die Entwicklung startet gleichzeitig in allen Bereichen. Eventuell wird zuerst etwas mehr Fokus auf die Funktionsentwicklung gelegt (Hydraulik, Software) und etwas weniger auf das Produktionsdesign oder dieses auch erst etwas später gestartet. Es macht durchaus wieder Sinn, in kleinen Schritten zu entwickeln (diese können auch explizit geplant sein), auch wenn diese nicht dem AEZ entsprechen. So gibt es eine geregelte Abstimmung und durch die häufigen Deadlines werden die Mitarbeiter zur Leistung motiviert.

Die einzelnen Bereiche entwickeln ihre Lösungen separat, stimmen diese aber auch immer wieder ab. Die Einengung des Lösungsbereichs geschieht, wenn Varianten in einem Entwicklungsbereich nicht machbar sind oder wenn durch die Abstimmung mehrerer Bereiche Varianten wegfallen. Z.B. wird die Verwendung eines bestimmten Sensors ausgeschlossen, weil dieser zu teuer ist oder die notwendige Sicherheitszertifizierung fehlt. Dadurch kann ein bestimmtes Hydraulikkonzept undurchführbar werden, wenn es die Verwendung dieses Sensors voraussetzt. Ein anderes Lösungskonzept kann undurchführbar werden, wenn die Energieversorgung des Motorrads nicht für die Kombination an Bauteilen ausreicht.

Ein plötzlich auftretender Wunsch des Kunden, die ABS-Einheit für mehrere Motorradtypen verwendbar zu machen, würde hier auch (je nach Zeitpunkt) einer Einengung des Lösungsbereichs entsprechen. Es würden hier alle Varianten wegfallen, die dafür nicht geeignet wären. Somit stellen solche Änderungen kein großes Problem dar, solange sie noch im Lösungsbereich abgebildet werden können. Ist das nicht der Fall, kann der Aufwand für die Änderung sehr groß werden und ist mit einer Änderung über die Phasen des BWI-Hall Modells hinweg vergleichbar.

Im Vergleich zum AEZ-SE Modell funktionieren die Erkennung von Änderungen und der Umgang damit grundverschieden. Wird beim AEZ-SE Modell auf eine schnelle Informationsgenerierung gesetzt und die Möglichkeit sehr flexibel reagieren zu können, so wird hier versucht, Entscheidungen möglichst lange hinauszuzögern. Das Erkennen der Änderungen muss aber zusätzlich forciert werden. Es ist anzunehmen, dass die grobe Systemarchitektur erst später zur Verfügung steht und dem Kunden präsentiert werden kann. Dadurch besteht mehr Zeit, um Änderungen einzubringen. Man muss aber durch die entsprechenden Entwicklungstätigkeiten, Kommunikation, Abstimmung mit dem Kunden, Analysieren von Simulationsergebnissen und Prototypen erst zum notwendigen Wissen dafür gelangen.

Der Lösungsbereich kann auch durch fällig werdende Termine eingeengt werden, z.B. wenn für Gussformen oder Steuergeräte lange Lieferzeiten bestehen, ist es notwendig, diese rechtzeitig auszulösen, um den Liefertermin einhalten zu können. Danach sind diese Teile als fixiert zu betrachten und die noch möglichen Varianten müssen damit kompatibel sein. Im Fallbeispiel würden Vertreter des Produktionsdesigns diese Informationen in die Abstimmungsrunden einbringen. Dann würde versucht werden, eine Entscheidung für das Steuergerät oder für die Gestaltung des Gussteils zu treffen, welche den Lösungsbereich möglichst wenig einschränkt. Dabei würde natürlich versucht werden, den Lösungsbereich dort offen zu lassen, wo die bestmögliche Lösung hinsichtlich der Anforderungen vermutet wird. Das Steuergerät und der Gussteil wären aber fixiert, mit der Konsequenz einer reduzierten Flexibilität in der Entwicklung.

Wichtig sind auch Termine, die sich durch den Entwicklungsablauf ergeben. Es ist beim Set-Based-SE Modell auch nicht anzunehmen, dass das Lösungsprinzip sehr lange offengehalten werden kann. Zu groß wäre der Aufwand für die Entwicklung und zu teuer wäre die große Anzahl an Prototypen, die für jede mögliche Variante realisiert werden müssten. Man würde also auch hier die grobe Systemarchitektur, das Gesamtkonzept oder andere richtungsweisende Dinge nach einer gewissen Zeit festlegen müssen, auch wenn versucht wird, das möglichst lange hinauszuzögern.

Probleme wie der vibrierenden Bremshebel oder die unzureichenden Schaltzeiten der Ventileinheit könnten kleiner sein, wenn das Gesamtkonzept noch zu einem späteren Zeitpunkt änderbar ist, weil noch keine Fixierung notwendig war. Es hätten dafür aber vermutlich mehr und teurere Prototypen erstellt werden müssen. Die größere Flexibilität und das Hinauszögern der Entscheidung wäre also vermutlich durch höhere Kosten erkauft worden.

***Projektende***

Die Einengung des Lösungsbereichs endet prinzipiell, wenn nur mehr eine machbare Variante zur Verfügung steht. Natürlich müssen bis zum Projektende auch alle Anforderungen erfüllt sein. So kann angenommen werden, dass nach der Festlegung einer Variante, diese noch im Detail fertigentwickelt werden muss.

***Unerwartete Änderungen***

Bei unerwarteten Änderungen stellt sich immer die Frage, ob diese im Lösungsbereich noch umgesetzt werden können. Ist dies noch möglich, sind die Auswirkungen normalerweise gering und können ohne großen Aufwand implementiert werden. Natürlich können sich aber die Produktionskosten des Systems verändern, wenn dieses andere Anforderungen erfüllen soll oder auch die Entwicklungskosten, wenn andere Entwicklungstätigkeiten notwendig werden.

Das genannte Beispiel einer neu verfügbar werdenden Sensor-Technologie, die einen ABS Einsatz auch bei Kurvenfahrt ermöglicht, wäre auf jeden Fall mit einem höheren Aufwand verbunden. Diese Technologie konnte aufgrund der zu Beginn nicht vorhandenen Verfügbarkeit natürlich nicht im Lösungsbereich sein. Es müsste also der Lösungsbereich erweitert werden. Je nach Zeitpunkt kann auch das noch möglich sein. Dadurch, dass wichtige Entscheidungen hinausgezögert wurden, ist das eventuell auch länger als beim BWI-Hall Modell möglich. Sind aber schon Entscheidungen getroffen worden, die dagegen sprechen, ist die Situation wieder mit einer phasenübergreifenden Änderung im BWI-Hall Model vergleichbar.

***Nach Projektende***

Nachdem dieses Modell am ehesten bei schwer anpassbaren Systemen angewendet werden wird, ist vermutlich nach Projektende keine große Flexibilität mehr zu erwarten.

***Offene Fragen und Probleme***

Das System ist bei diesem Vorgehensmodell zwar keinen Voraussetzungen hinsichtlich seiner Entwickelbarkeit in kleinen Schritten unterworfen. Trotzdem hat die Art des Systems einen indirekten Einfluss auf die Machbarkeit dieser Methode. Je nach System wird der Lösungsbereich nur sehr klein sein können, bzw. sich innerhalb sehr kurzer Zeit stark verkleinern müssen. Das heißt dann wieder, dass die Anforderungen zu Beginn doch sehr genau bekannt sein müssen, damit der Lösungsbereich auch gut darauf abgestimmt werden kann.

Auch die Preisfestsetzung ist davon betroffen. Ein Fixpreis kann nur vereinbart werden, wenn der Lösungsbereich so begrenzt wird, dass darin für alle Varianten ähnliche Kosten zu erwarten sind. Die bessere Alternative wäre vermutlich, den Preis nicht zu Beginn für die Begrenzung des Lösungsbereichs heranzuziehen, sondern in der Machbarkeitsanalyse der einzelnen Varianten einfließen zu lassen. Varianten würden dann ausgeschieden werden, wenn feststeht, dass deren Kosten zu hoch für den vorgegebenen Preis sind.

Ebenso wie beim AEZ-SE Modell sind auch hier wieder kulturelle Aspekte bzw. Gewohnheiten in der Industrie ein wesentliches Kriterium für die Akzeptanz dieses Modells.

## 8.7 Zusammenfassung

In diesem Kapitel wurde die praktische Anwendung des ASE Ansatzes durch ein fiktives Fallbeispiel näher erläutert. In Kapitel 8.2 wurde die Vorgehensmethodik zur Auswahl der geeigneten Praktiken bzw. eines geeigneten Vorgehensmodells für die Entwicklung erklärt.

Danach wurde die Anwendung der verschiedenen Vorgehensmodelle gezeigt. Es wurden jeweils die Möglichkeiten im Umgang mit Änderungen innerhalb und außerhalb der Entwicklung beschrieben. Dabei wurde gezeigt, wie das AEZ-SE Modell den frühen Erkenntnisgewinn zu möglichen Änderungen innerhalb der Entwicklung forciert. Für Reaktionen darauf ist es auf Flexibilität im Produkt angewiesen, genauso wie bei Änderungen außerhalb der Entwicklung. Im Gegensatz dazu versucht man beim Set-Based-SE Modell Festlegungen länger hinauszuzögern und ist deshalb weniger auf Flexibilität im Produkt angewiesen. Dafür muss aber der Erkenntnisgewinn zusätzlich forciert werden und das Hinauszögern kann auch nur bis zu einem gewissen Zeitpunkt erfolgen.

In traditionellen Modellen ist kein expliziter Funktionsmechanismus für den Umgang mit Veränderungen inkludiert. Sie versuchen durch möglichst gute Abklärungen im Vorhinein spätere Änderungen zu vermeiden. Das BWI-Hall Modell als ein Vertreter der traditionellen Vorgehensmodelle bietet aber auch zahlreiche Freiheiten, die Agilität zu erhöhen.

Im Fallbeispiel wurden die grundlegenden Mechanismen zur Steigerung der Agilität und einige zusätzliche Praktiken aufgezeigt. Wenig Erwähnung fanden dabei die personellen Aspekte, die neben den methodischen Ansätzen aber einen ganz wesentlichen Einfluss auf die Agilität haben und deshalb hiermit noch erwähnt werden sollten.

# 9 Fazit und Ausblick

Dieses Kapitel soll nicht nur die wichtigsten Ergebnisse zusammenfassen, sondern auch die persönliche Meinung des Autors einfließen lassen. Es wird dargestellt, welche Ziele mit dieser Arbeit erreicht werden konnten, wo Einschränkungen bestehen und Ansatzpunkte für weitere Forschungstätigkeit zu finden sind. Dies soll anhand der Forschungsfragen untergliedert getan werden. Zuerst sollen aber noch allgemeine Dinge zu dieser Arbeit gesagt und Besonderheiten hervorgehoben werden.

Sowohl für Agilität als auch für Systems Engineering konnten unterschiedliche Verständnisse gefunden werden. Deshalb wurde auch viel Aufwand für die Definition dieser beiden Begriffe und für „Agile Systems Engineering“ als Kombination daraus betrieben. Der Autor meint, aufgrund umfassender Recherche und zahlreicher Diskussionen mit Interviewpartnern, Expertenpanel und Forschern bei Konferenzen eine allgemeingültige Definition gefunden zu haben, die von einem Großteil der Gesprächspartner akzeptiert werden kann. Damit sollte vor allem aber auch eine Abgrenzung gegenüber nahestehenden Problemstellungen gemacht werden, die ebenso unter Verwendung des Schlagwortes „Agile Systems Engineering“ beforscht werden können.

Wichtig für die Arbeit war die genaue Beschreibung des damit zu lösenden Problems und die Festlegung der Ziele eines ASE Ansatzes. Erst mit der Fragestellung „*Wie kann man in Umgebungen mit starken und häufigen, vorhersehbaren und unvorhersehbaren Veränderungen, in effizienter und effektiver Weise, plan- und vorhersehbar, erfolgreiche Systeme entwickeln und herstellen?“* konnte die Forschungsarbeit wesentlich vorangetrieben werden.

In den Interviews und vor allem bei den Diskussionen mit der Expertengruppe S²QI hat sich gezeigt, dass es ein großes Interesse für Agilität in vielen Bereichen der Unternehmung gibt. Dies ist aber nicht immer im eigentlichen Wesen der Agilität begründet, sondern liegt auch vielfach in der momentanen Popularität der agilen Softwareentwicklungsmethoden. Leider werden hier auch oft unrealistische Versprechungen gemacht, die sich in einem Großteil der Anwendungsfälle nicht verwirklichen lassen.

In der vorliegenden Arbeit wurde versucht, das Wesen und die Vorteile aber auch mögliche Nachteile oder Schwierigkeiten im Zusammenhang mit Agilität darzustellen. Es wurde auch erklärt, was möglicher Zusatznutzen mancher agiler Vorgehensweisen sein kann, für welchen diese aber eigentlich nicht entwickelt wurden. Die Anwendungsgebiete wurden aufgezeigt und mit der ASE Vorgehensmethodik eine Anleitung zur praktischen Umsetzung gegeben. Damit soll eine Hilfestellung geboten werden, für die - trotz Veränderungen - erfolgreiche Durchführung von Systems Engineering Projekten.

Das Ergebnis dieser Arbeit stellt aber nur eine Momentaufnahme dar. In Zukunft sind noch wesentliche Verbesserungen hinsichtlich Agilität zu erwarten. Je nach Größe der Unsicherheiten und Häufigkeit der Veränderungen wird der Druck auf Systementwickler immer größer werden, sich damit zu beschäftigen. Einerseits kann dies im Bereich der Technik geschehen. Sowohl bei Produkttechnologien als auch bei Produktions- und Prototyping-Technologien kann die Agilität durch eine Verbesserung der Flexibilität unterstützt werden. Andererseits sind es kulturelle Verhaltensweisen und Gewohnheiten, sowohl innerhalb als auch zwischen Unternehmungen, die Agilität momentan noch stark behindern (Wunsch alles planen zu können, Fixpreise …). Hier kann nur eine veränderte Geisteshaltung der handelnden Personen weitere Agilität ermöglichen. Bei Fortschritten in diesen Belangen sind methodische

Fortschritte in der Entwicklung zur weiteren Steigerung der Agilität durchaus zu erwarten. Die vorliegende Dissertation soll hierfür eine Basis bilden, indem sie ein allgemeingültiges Verständnis der Agilität aufbaut, Begriffe definiert, bereits vorhandene Konzepte und deren Zusammenhänge erklärt und in einem Rahmenwerk besser anwendbar macht, das auch die Grundlage für weitere Forschungstätigkeit in diesem Bereich bilden soll. Die im Rahmen dieser Arbeit gefundenen Antworten auf die fünf Forschungsfragen werden nachstehend zusammenfassend dargestellt.

## 9.1 Was ist das Wesen der Agilität?

Diese Frage wurde durch eine ausführlichen Recherche über den Gebrauch des Wortes Agilität und den damit bezeichneten Prinzipien und unter Berücksichtigung der Problemstellung dieser Arbeit beantwortet. Es kann als gesichert gelten, dass Agilität eine Fähigkeit darstellt. Sie ist z.B. keine Methode oder Sammlung von Praktiken, auch wenn Agilität als Bezeichnung für sehr unterschiedliche Dinge verwendet wird. Im Groben kann man die Fähigkeit, welche sich hinter dem Wort Agilität verbirgt, mit „Effizienz im Umgang mit neuen Situationen und Veränderungen“ beschreiben. Im Detail kann die Fähigkeit aber verschiedenen Prinzipien folgen und auch unterschiedlich definiert werden. Deshalb erscheint es wichtig, jeweils das Ziel zu nennen, zu welchem man durch die Fähigkeit der Agilität gelangen will. Angewandt auf das SE wurde das Ziel in dieser Arbeit als „effizientes, effektives, plan- und vorhersehbares Herstellen erfolgreicher Systeme, in Umgebungen mit starken und häufigen, vorhersehbaren und unvorhersehbaren Veränderungen“ benannt. Die Agilität ist dadurch einerseits als Fähigkeit definiert, welche die Erreichung dieses Ziels ermöglichen soll („top-down Definition“). Andererseits wurden aber auch Teilfunktionen identifiziert, welche dazu notwendig sind und als Charakteristika die Agilität im SE ebenso beschreiben können („bottom-up Definition“):

- Situationskenntnis
- Kontinuierliche Verbesserung
- Bewältigung von Veränderungen
- Flexibilität
- Effizienz
- Effektivität

Für den komplexen Zusammenhang der Agilität im SE stellen diese Punkte aber nur eine Lösungsmöglichkeit dar und es sind weitere denkbar, die ebenso sinnvoll sein mögen. Z.B. könnte man Effizienz und Effektivität durch Kosteneffizienz, Termintreue und Qualität ersetzen. Die weiteren Ergebnisse dieser Arbeit würden sich dadurch nicht ändern. Der Autor meint aber, dass die gewählte Liste alle Charakteristika umfassen kann, die wesentlich zur Erreichung des genannten Ziels sind.

## 9.2 Welche Vorgehensweisen können die Agilität steigern?

In der Softwareentwicklung wird dies von den sogenannten „agilen Methoden“ (2.2) angenommen. Der Kern dieser Methoden wird häufig als „iterative“, „inkrementelle“ oder „zyklische Entwicklung“ bezeichnet. In dieser Arbeit wurden diese Vorgehensweisen auf ihr gemeinsames, zugrundeliegendes Wirkprinzip hinsichtlich Agilität zurückgeführt. Dazu wurden diese Methoden hinsichtlich Gemeinsamkeiten in ihren Mechanismen zum Umgang mit Veränderungen analysiert. Es wurden aber

auch alle damit in Verbindung stehenden Regeln zusammengetragen, welche in den verschiedenen agilen Softwareentwicklungsmethoden genannt werden. Aus dem grundlegenden Wirkprinzip in Kombination mit sämtlichen Regeln wurde für diese Arbeit der „Agile Entwicklungszyklus (AEZ)“ gestaltet. Kurz dargestellt ist dieser ein expliziter Prozess, der in kleinen, zyklischen Schritten das System sukzessive herstellt, in jedem Schritt die Phasen Situationsanalyse, Planung, Realisierung und Test durchläuft und von Schritt zu Schritt das Feedback des Kunden zur Richtungsbestimmung für die weitere Entwicklung nutzt. Die Entwicklungsschritte sollen dabei nach Priorität und in einer gleichbleibend, kurzen Zykluszeit durchgeführt werden und jeweils nutzbare Zwischenergebnisse liefern. Eine detaillierte Beschreibung befindet sich in Kapitel 4.3.4.

Mit dem AEZ wurde eine Vorgehensweise dargestellt, die eine höhere Agilität in der Entwicklung erreichen soll als traditionelle Entwicklungsmethoden, welche meist nach dem Top-Down Prinzip und in langen, sequentiell durchgeführten Phasen vorgehen. Im Gegensatz dazu stellt der AEZ einen sich laufend wiederholenden Entwicklungsprozess für jeweils kleine Teile des Entwicklungsumfangs dar, auf welche der Kunde Feedback gibt und dadurch die Entwicklungsrichtung beeinflussen kann. Der AEZ stellt damit auch das am weitesten verbreitete Verständnis der konkreten Ausführung einer „agilen Entwicklung“ dar. Er ist aber nicht ident mit ihr, sondern steht nur im Verdacht eine Methode zu sein, mit welcher sich die Agilität in der Entwicklung erhöhen lässt. Davon abgeleitet, aber auch nach anderen Grundprinzipien, wurden in dieser Arbeit Vorgehensmodelle entworfen, die für eine höhere Agilität im SE sorgen sollen (siehe 7.5).

Zur Beantwortung der zweiten Forschungsfrage wurde aber speziell auch versucht, möglichst detailliert und auf niedriger Ebene Prinzipien und Praktiken als Vorgehensweisen herauszuarbeiten. Insgesamt wurden 50 Prinzipien und Praktiken identifiziert, analysiert und in Kapitel 5 kategorisiert nach ihren primären Anwendungsbereichen:

- Kunde
- Organisation
- Mitarbeiter
- Entwicklungsprozess
- System

In ihrer konkreten Anwendung sollen diese Prinzipien und Praktiken positiv auf eine oder mehrere Teilfunktionen der Agilität einwirken. Sie können aber die Agilität nicht direkt und nicht mit Sicherheit erhöhen, sondern nur Hilfestellungen bieten und bei sinngemäßer Anwendung im richtigen Kontext die Wahrscheinlichkeit der Zielerreichung erhöhen.

Die in dieser Arbeit vorgestellten Vorgehensweisen wurden ausgewählt, da sie in der Literatur oder in den empirischen Erhebungen als Lösungsmöglichkeiten genannt wurden. In Zukunft ist hier die Entwicklung weiterer Lösungsmöglichkeiten zu erwarten bzw. besteht hier auch Potential für Forschung zur Identifizierung weiterer Prinzipien und Praktiken. Ganz besonders interessant erscheint eine weitere Beschäftigung mit der „Entwicklung in kleinen Schritten“ (5.4.3), welche dem AEZ seine Grundstruktur gibt. Dieses Prinzip ist in der Softwareentwicklung relativ gut etabliert und dort in zahlreichen Methoden enthalten. Für die konkrete Anwendung in der Hardwareentwicklung bzw. für allgemeine Systeme ist aber weiterer Forschungsbedarf zur Gestaltung von dafür geeigneten Methoden notwendig. Auf einer

abstrakten Ebene stellen das Vorgehensmodell in 7.5.2 und die Maßnahmen in 6.4.9 allgemeine Lösungsansätze dafür dar. Eine konkrete Methode hat aber Rücksicht auf die Eigenheiten von Produkt- und Produktionsprozess und die Erfordernisse der jeweiligen Branche zu nehmen. Ebenso lassen unterschiedliche Veränderungen und die dafür notwendige Agilität verschiedene Methoden im jeweiligen, konkreten Anwendungsfall als die wirtschaftlichsten erscheinen. Deshalb ist nicht zu erwarten, dass eine konkrete Methode in allen Anwendungsfälle die geeignetste sein kann. Es ist viel mehr anzunehmen, dass viele spezifische Methoden entwickelt werden müssen. Das liegt zwar außerhalb des Rahmens der vorliegenden Arbeit, diese soll aber nützliche Grundlagen dafür bieten.

## 9.3 Wann besteht Bedarf an Agilität?

Im Allgemeinen hat jedes handlungswillige Wesen einen Bedarf an Agilität, da das Nicht-Vorhandensein (absoluter Nullpunkt) der Agilität eine absolute Unfähigkeit zu agieren bedeuten würde.

In Systementwicklungsprojekten sind neue Situationen und Veränderungen allgegenwärtig, es kann aber auch ein gewisser Grad an Agilität der entwickelnden Personen vorausgesetzt werden. Es stellt sich nun die Frage, wann ein Bedarf zur Steigerung der Agilität besteht, welcher mit dem ASE Ansatz dieser Arbeit abgedeckt werden kann. Dieser Bedarf kann als gegeben betrachtet werden, wenn die folgenden Faktoren einen markanten negativen Einfluss auf den Erfolg eines Systementwicklers erkennen lassen:

- Häufige Veränderungen (im Markt, bei Technologien, bei Kunden/Anwendern/Partnern …)
- Unzulängliche Produktspezifikationen zu Projektbeginn
- Hohe Komplexität und/oder hoher Neuigkeitsgrad des Produkts

Der erste Punkt betrifft Änderungen, die außerhalb und unabhängig der Entwicklung geschehen. Die beiden weiteren Punkte stellen Änderungen dar, die während der Entwicklung und ausgelöst durch diese auftreten, weil darin neue Erkenntnisse erzielt wurden. Die unzulänglichen Produktspezifikationen können dabei als vom Kunden verschuldet betrachtet werden, es trägt aber auch der Entwickler Mitschuld, da er diese akzeptiert hat. Hohe Komplexität und hoher Neuigkeitsgrad sind ebenso Ursachen für unzulängliche Produktspezifikationen zu Projektbeginn. Dafür trägt aber niemand Schuld, weil die Unvorhersehbarkeit des Systemverhaltens und der genauen Anforderungen in den beiden genannten Faktoren begründet liegt. Für den zweiten und dritten Punkt sei auch noch darauf hingewiesen, dass diese nur hinsichtlich der dadurch verursachten Änderungen während der Entwicklung als Bedarfsgrund für den ASE Ansatz gesehen werden. Es ist z.B. die Komplexität des Systems aber kein Bedarfsgrund für Agilität im Allgemeinen.

Strebt eine Unternehmung an, ihren Umgang mit Veränderungen zu verbessern, speziell hinsichtlich der drei genannten Punkte, so kann ihr ein Bedarf an Agilität unterstellt werden, für welchen der ASE Ansatz dieser Arbeit Hilfestellungen bietet.

## 9.4 Was sind Voraussetzungen für Agilität?

Mit der Forschungsfrage nach den Voraussetzungen sollte ergründet werden, ob es auch Umstände gibt, die gegen Agilität oder das Anwenden von agilen Prinzipien und Praktiken sprechen.

Grundsätzliche Voraussetzungen für Agilität als Fähigkeit gibt es nicht. In einigen Situationen erscheint es aber nicht angebracht, sich mit ihrer Steigerung zu beschäftigen (z.B. wenn es sich um Routineentwicklungen mit bekannten und bewährten Prozessabläufen handelt, die Umwelt stabil ist und deshalb kein Bedarf besteht). Es gibt aber zahlreiche Voraussetzungen für die praktische Anwendbarkeit der verschiedenen agilen Praktiken und Prinzipien. Die wichtigsten Faktoren dafür sind:

- Art des Systems
- Kundeneinbindung
- Kritikalität
- Kultur
- Mitarbeiterqualifikation
- Teamgröße und Verteilung
- Preisgestaltung
- Wirtschaftlichkeit

Es wurde in dieser Arbeit aufgezeigt, welche dieser Faktoren für die verschiedenen Prinzipien und Praktiken als Voraussetzung oder zumindest als Risiko gelten. Die Faktoren stammen aus der Literaturrecherche und den empirischen Erhebungen. Es ist nicht auszuschließen, dass weitere, vor allem unternehmungsspezifische Umstände, Hindernisse für den Einsatz agiler Praktiken darstellen. Es gibt aber auch Möglichkeiten, diese Voraussetzungen zu umgehen oder die Risiken zu reduzieren, wovon einige gezeigt wurden (6.4). Mit diesen kann z.B. die Entwicklung in kleinen Schritten in einer Ausprägungsform ausgestaltet werden, welche die Agilität des AEZ teilweise erreichen und trotzdem in einem größeren Anwendungsbereich tauglich sein kann, als der AEZ (z.B. durch Verzicht auf die Regel, dass nach jedem Zyklus ein Zwischenergebnis vorliegen muss, welches vom Kunden in der vorgesehenen Endanwendung genutzt werden kann oder durch entsprechend lange Zyklusdauern).

Die Anwendung des AEZ als Ganzem unterliegt aber zahlreichen Voraussetzungen, nicht nur in der Anwendung für allgemeine Systeme, wie das in 6.2.1 aufgezeigt wurde, sondern bereits auch schon innerhalb der Softwareentwicklung, wie in 6.1 dargestellt. Mögliche Probleme sind z.B. eine Sicherheitskritikalität des Systems, welche nach einem nachverfolgbaren Entwicklungsprozess verlangt, in dem Anforderungen nicht so leicht geändert werden dürfen. Ebenso können der Wunsch des Kunden, zu Beginn des Projekts einen Fixpreis für Entwicklung und Produkt zu vereinbaren und die fehlende Bereitschaft des Kunden, während der Entwicklung Feedback zu geben, Probleme für den AEZ darstellen. Ganz besonders abhängig ist der AEZ aber davon, dass die Art des Systems auch eine Entwicklung und Herstellung in kleinen, zyklischen Schritten erlaubt. Für Systeme wie z.B. Kraftwerke, Industrieanlagen oder Fahrzeuge ist kaum vorstellbar Zwischenergebnisse in Zyklusdauern von wenigen Wochen zu entwickeln und herzustellen, welche dann auch vom Kunden in der Endanwendung getestet werden können.

## 9.5 Wie kann ein „Agile Systems Engineering“ Ansatz aussehen?

Die soeben zusammengefassten Antworten auf die Forschungsfragen 1 bis 4, die genauer in den Kapiteln 4 bis 6 beschrieben werden, sind bereits als Bausteine des ASE Ansatzes zu betrachten. Die Vorgehensmethodik in Kapitel 7 soll die Zusammenhänge noch einmal verdeutlichen und eine Anleitung

für die Anwendung des ASE Ansatzes geben. Insgesamt soll damit eine umfassende Methodik dargestellt werden, die zur theoretischen Betrachtung der Agilität im SE, aber auch zu ihrer praktischen Umsetzung verwendet werden kann. Aufbauend auf der BWI-Hall SE Methodik (Haberfellner, et al., 2002) stellt der ASE Ansatz den nächsten Schritt für eine SE Methodik dar, die auch in dynamischen Umfeldern zu erfolgreicher Systementwicklung führen soll. Abgeleitet von der SE Philosophie der BWI-Hall Methodik (Haberfellner, et al., 2002 S. XIX) wurde die Philosophie des ASE Ansatzes in Abb. 39 grafisch dargestellt.

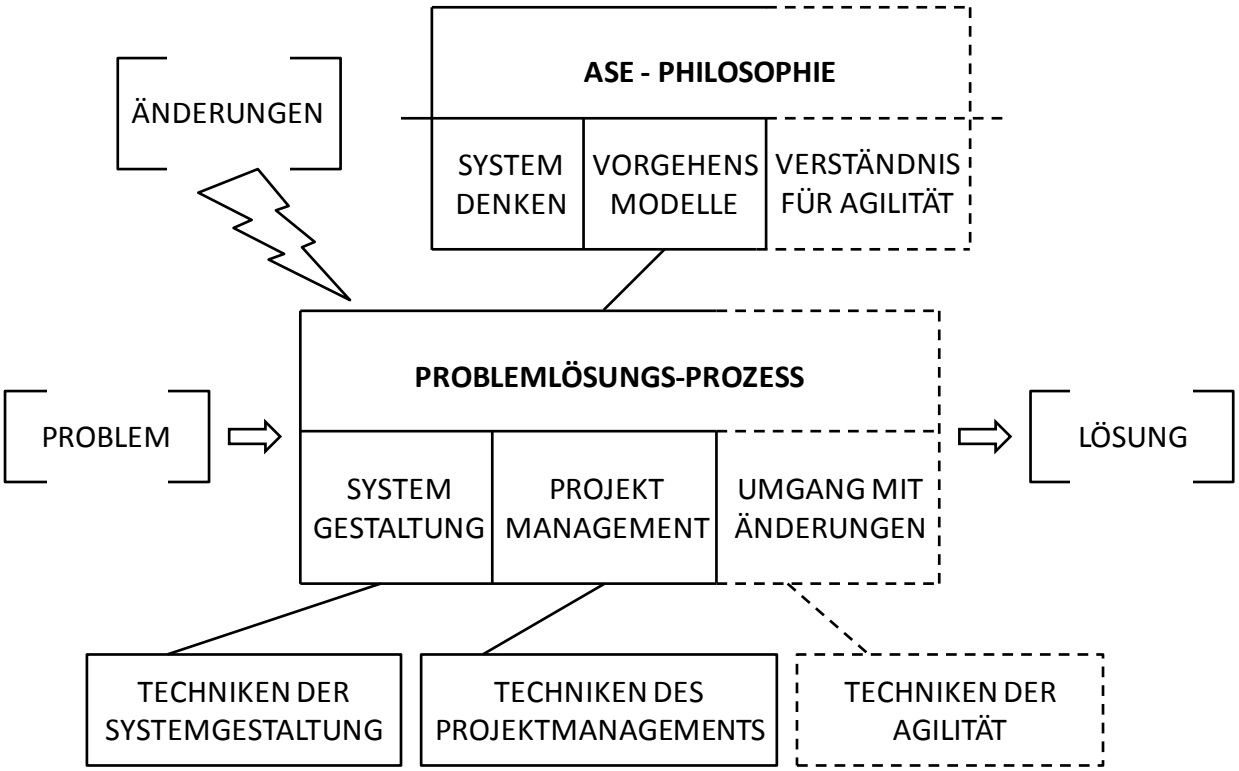


**Abb. 39: Agile Systems Engineering Philosophie**

Bei dieser Abbildung wurde versucht, die Bewegung darzustellen, die in einem dynamischen Umfeld auch in der SE Philosophie notwendig ist. Neben einem Problem gilt es nämlich auch noch Änderungen zu bewältigen. Im Zentrum der Methodik, dem Problemlösungsprozess, wurde deshalb noch der Umgang mit Änderungen hinzugefügt. Für den besseren Umgang mit Änderungen stehen Techniken der Agilität zur Verfügung. Als geistiger Überbau wird nun die ASE-Philosophie gesehen, die im Unterschied zur SE-Philosophie mehrere Vorgehensmodelle anbietet und zusätzlich noch um das Verständnis der Agilität erweitert wurde.

Dieses Verständnis der Agilität sollte durch die gesamte Arbeit und die Beantwortung aller Forschungsfragen geschaffen werden. Speziell durch die Erklärung der Zusammenhänge zwischen den Zielen und Mitteln über mehrere Ebenen hinweg (Abb. 29), mit denen gezeigt wurde, wie die agilen Praktiken in verschiedenen Eingriffsbereichen die Teilfunktionen der Agilität verbessern können. Aber auch durch das Aufzeigen der Zusammenhänge zwischen den Praktiken (7.3) und der grundlegenden Funktionsprinzipien der Vorgehensmodelle (Abb. 36 und Abb. 37).

Nach diesen Funktionsprinzipien wurden Vorgehensmodelle gestaltet, welche die Agilität im SE fördern sollen. Neben dem AEZ-Ansatz und seiner zyklischen Entwicklung in kleinen Schritten wurde ein weiteres auf Grundlage des Set-Based Designs gestaltet. Ebenso wurde eine Version des BWI-Hall Vorgehensmodells entworfen, welche zur Agilitätssteigerung modifiziert wurde.

Der Umgang mit Änderungen im Problemlösungsprozess ist in jedem Systementwicklungsprojekt individuell durchzuführen. Man kann aber durchaus Abb. 30 verwenden, um Möglichkeiten für den besseren Umgang mit Änderungen überblicksartig aufzuzeigen.

Die Techniken für die Agilität werden von den Prinzipien und Praktiken in Kapitel 5 dargestellt, welche in dieser Arbeit hinsichtlich ihres Anwendungsgebiets analysiert wurden (Abb. 32).

Damit stellt der ASE Ansatz nicht nur bloß eine Methode zur agilen Entwicklung von Systemen zur Verfügung, sondern er berücksichtigt unterschiedlichen Bedarf und vor allem die gegebenen Voraussetzungen. Er verknüpft die Teilfunktionen der Agilität mit den konkreten Praktiken und Prinzipien und erklärt die Zusammenhänge. Man kann damit Unternehmungen oder Projekte hinsichtlich Agilität analysieren, Handlungsbedarf erkennen und Empfehlungen dazu abgeben. Ebenso wurden in ihm auch eine große Anzahl an praktischen Lösungsansätzen (3 Vorgehensmodelle, 50 Prinzipien und Praktiken) aufbereitet.

Einige der Empfehlungen aus dem Ansatz mögen ungewöhnlich oder auch unrealistisch wirken, vor allem für den Einsatz in traditionellen Industrien. Dort hat sich in den empirischen Erhebungen gezeigt, dass viele Ideen aus der agilen Softwareentwicklung wohl kaum anwendbar sind. Dem begegnet der ASE Ansatz mit seinem flexiblen Aufbau (Verwendung unterschiedlicher Praktiken möglich) und der Anwendbarkeit auf verschiedene (auch traditionelle) Vorgehensmodelle. Die nicht durchführbaren Praktiken sollen dann zumindest als inspirierende Ideen betrachtet werden. Sie sind aber auf jeden Fall ernst zu nehmen, da im ASE Ansatz keine Vorgehensweisen vorgeschlagen werden, die nicht in irgendeiner Branche schon verwendet wurden. Es soll aber auch nicht verschwiegen werden, dass viele Entwicklungsprozesse in der Praxis notgedrungen viel „agiler“ und „iterativer“ ablaufen und ablaufen müssen, als es den Vorstellungen der Planer dieser Prozesse bzw. des Managements entspricht.

Wie in zahlreichen Fällen zu erkennen war, existiert häufig einen Zielkonflikt zwischen Flexibilität und Wirtschaftlichkeit. Auch bei Anwendung des ASE Ansatzes bleibt dieser Zielkonflikt möglicherweise noch bestehen, er sollte damit aber auf jeden Fall gemildert werden können. Sollte aufgrund von Voraussetzungen keine einzige methodische Möglichkeit zur Agilitätssteigerung bestehen, so soll noch einmal auf die enorme Wichtigkeit der Mitarbeiter verwiesen werden. Diese können dann als einziges Kriterium für die Agilität gelten, welche zum erfolgreichen Abschluss einer Entwicklung notwendig ist.

Der Autor hofft mit seiner Arbeit ein nützliches Konzept vorgelegt zu haben, welches in schnelllebigen Zeiten einen Beitrag zur erfolgreichen Entwicklung von Systemen leisten kann.

# 10 Abkürzungsverzeichnis

**AEZ**.....Agiler Entwicklungszyklus

**ASE**.....Agile Systems Engineering

**CMMI**.....Capability Maturity Model Integration

**HW**.....Hardware

**IID**.....Iterative and Incremental Development

**INCOSE**.....International Council on Systems Engineering

**OODA**.....Observe-Orient-Decide-Act (Loop)

**PLZ**.....Problemlösungszyklus

**SE**.....Systems Engineering

**SPICE**.....Software Process Improvement and Capability Determination (ISO/IEC 15504)

**SOP**.....Start of Production (Start der Serienproduktion)

**SW**.....Software

**S²QI**.....Systems and Software Quality Initiative (Ein Expertenpanel)

**XP**.....Extreme Programming

# 11 Abbildungsverzeichnis

# 12 Literaturverzeichnis


**Adolph Steve. 2006.** What lessons can the agile community learn from a maverick fighter pilot? G. Melnik [Hrsg.]. *Agile 2006 conference proceedings.*

**Aoyama Mikio. 1998.** Agile Software Process and Its Experience. *20th International Conference on Software Engineering Proceedings, Kyoto, Japan.*

**Aschbacher Helmut, Stelzmann Ernst, Kreuzer Ernst, Brenner Eugen. 2010.** Using Agile Systems Engineering for Improving a Company's Handling of Change. *Conference on Systems Engineering Research (CSER), Hoboken, NJ*.

**Beck Kent, Andres Cynthia. 2005.** *Extreme Programming Explained: Embrace Change.* 2. Edition. Addison-Wesley, Boston, MA.

**Beck Kent, Andres Cynthia. 2005.** Getting Started with XP: Toe Dipping, Racing Dives and Cannonballs. *Three River Institute.* www.threeriverinstitute.org [Abgerufen: 20. 3. 2009].

**Beck Kent, Beedle Mike, van Bennekum Arie, Cockburn Alistair, Cunningham Ward, Fowler Martin, Grenning James, Highsmith Jim, Hunt Andrew, Jeffries Ron, Kern Jon, Marick Brian, Martin Robert, Mellor Steve, Schwaber Ken, Sutherland Jeff, Thomas Dave. 2001.** Manifesto for Agile Software Development. http://agilemanifesto.org/ [Abgerufen: 5. 11. 2009].

**Bedoll Robert. 2003.** A Tail of Two Projects: How 'Agile' Methods Succeeded after 'Traditional' Methods Had Failed in a Critical System-Development Project. *XP/Agile Universe 2003. Springer,* pp. 25-34.

**Benjamin Perakath, Erraguntla Madhav, Mayer Richard, Marshall Charles. 1999.** Toolkit for Enabling Analysis and Modeling of Adaptive Workflow (TEAMWORK). *SIGGroup Bulletin* 1999, Vol. 20, 3, p. 9.

**Bernstein Joshua. 1998.** *Design Methods in Aerospace Industry: Looking for Evidence of Set-Based Practices (Master's Thesis).* Massachusetts Institute of Technology, Cambridge, MA.

**Blanchard Benjamin, Fabrycky Wolter. 2006.** *Systems Engineering and Analysis.* 4. Edition. Pearson Prentice Hall. Upper Saddle River, NJ.

**Boehm Barry. 1988.** A Spiral Modell of Software Development and Enhancement. *IEEE Computer.* May 1988, pp. 61-72.

**Boehm Barry. 1996.** Anchoring the Software Process. *IEEE Software.* 1996, July.

**Boehm Barry, Lane Jo Ann. 2006.** 21st Century Processes for Acquiring 21st Century Software-Intensive Systems of Systems. *Journal of Defence Software Engineering.* May 2006.

**Boehm Barry, Lane Jo Ann. 2008.** A Process Decision Table for Integrated Systems and Software Engineering. *Conference on Systems Engineering Research (CSER) Proceedings, Los Angeles, CA.*

**Boehm Barry, Turner Richard. 2006.** *Balancing Agility and Discipline.* 5th Printing. Addison-Wesley, Boston, MA.

**Boehm Barry, Turner Richard. 2005.** Management Challanges to Implementing Agile Processes in Traditional Development Organizations. *IEEE Software.* 2005, September/October, pp. 30-39.

**Boehm Barry. 2006.** Some Future Trends and Implications for Systems and Software Engineering Processes. *Journal of Systems Engineering.* 2006, Vol. 9, 1.

**Boehm Barry. 2008.** Unifying Software Engineering and Systems Engineering. *IEEE Computer.* 2008, 3.

**Boyd John. 1976.** Destruction and Creation. *The d-n-i echo.* http://www.danford.net/boyd/destruction.htm [Abgerufen: 17. 12. 2010].

**Boyd John. 1986.** Patterns of Conflict. www.slideshare.net/noobgank/patterns-of-conflict [Abgerufen: 15. 12. 2010].

**Boyd John. 1996.** The Essence of Winning and Losing,. *The d-n-i echo.* http://www.danford.net/boyd/essence.htm [Abgerufen: 17. 12. 2010].

**Brooks Fred. 1995.** *The Mythical Man-Month: Essays on Software Engineering.* 2. Edition. Addison-Wesley, Boston, MA.

**Büchel Alfred. 1969.** Systems Engineering - Eine Einführung. *Industrielle Organisation.* Vol. 38, 9.

**Capell Peter, Madni Azad. 2008.** Complexity and Architecture. *Conference on Systems Engineering Research (CSER) Proceedings, Los Angeles, CA*.

**Checkland Peter. 1981.** *Systems Thinking, Systems Practice.* Reprinted. John Wiley & Sons, Chichester.

**Clements Paul, Northrop Linda. 2002.** *Software Product Lines: Practices and Patterns.* Addison-Wesley, Boston, MA.

**Cockburn Alistair, Highsmith Jim. 2001.** Agile Software Development: The People Factor. *Computer.* 2001, 11, pp. 131-133.

**Cockburn Alistair. 2005.** *Crystal Clear: Agile Software-Entwicklung für kleine Teams.* Verlag moderne Industrie, Landsberg.

**Cockburn Alistair. 2007.** Iterative vs. Incremental Development. *Alistair Cockburn Homepage.* http://alistair.cockburn.us/Incremental+versus+iterative+development [Abgerufen: 05. 05. 2011].

**Cockburn Alistair. 2002.** Learning From Agile Software Development - Part One. *Cross Talk, The Journal of Defense Software Engineering.* 2002, 10.

**Cockburn Alistair. 2002.** Learning From Agile Software Development - Part Two. *Cross Talk, The Journal of Defense Software Engineering.* 2002, 11.

**Derby Ester. 2007.** Collaboration Skills for Agile Teams. *Cross Talk, The Journal of Defense Software Engineering.* 2007, 4.

**Dove Rick. 2005.** Agile Enterprise Cornerstones: Knowledge, Values & Response Ability. *Keynote at International Federation for Information Processing (IFIP) 8.6 Conference, Atlanta.*

**Dove Rick. 1996.** Agile Supply-Chain Management. *Automotive Production.* 1996, 4.

**Dove Rick, Turkington Garry. 2008.** Relating Agile Development to Agile Operations. *Conference on Systems Engineering Research (CSER) Proceedings, Los Angeles, CA.*

**Dove Rick. 2006.** Engineering Agile Systems: Creative-Guidance Frameworks for Requirements and Design. *Conference on Systems Engineering Research (CSER) Proceedings, Los Angeles, CA.*

**Dove Rick. 2005.** Fundamental Principles for Agile Systems Engineering. *Conference on Systems Engineering Research (CSER) Proceedings, Hoboken,NJ.*

**Dove Rick. 2001.** *Response Ability.* John Wiley & Sons, New York, NY

**Dvir Dov, Lechler Thomas. 2004.** Plans are nothing, changing plans is everything: the impact of changes on project success. *Research Policy.* 2004, 33, pp. 1-15.

**Ebster Claus, Stalzer Lieselotte. 2008.** *Wissenschaftliches Arbeiten für Wirtschafts- und Sozialwissenschaftler.* 3. Auflage. Facultas, Wien.

**Ehrlenspiel Klaus, Kiewert Alfons, Lindemann Udo. 2007.** *Kostengünstig Entwickeln und Konstruieren: Kostenmanagement bei der integrierten Produktentwicklung.* 6. Auflage. Springer, Heidelberg.

**Elkins Debra, Huang Ningjian, Alden Jeffrey. 2004.** Agile manufacturing systems in the automotive industry. *International Journal of Production Economics.* 2004, Vol. 91, pp. 201-214.

**Flick Uwe, Kardoff Ernst, Steinke Ines. 2000.** Was ist qualitative Forschung. *Qualitative Forschung: ein Handbuch*. Reinbek bei Hamburg. pp. 13-29.

**Fricke Ernst, Schulz Armin. 2005.** Design for Changeability (DfC): Principles To Enable Changes in Systems Throughout Their Entire Lifecycle. *Systems Engineering.* 2005, Vol. 8, 4. Wiley Periodicals.

**Fritzsche Martin, Keil Patrick. 2007.** Agile Methods and CMMI: Compatibility or Conflict? *e-Informatics Software Engineering Journal.* 2007, Vol. 1, 1, pp. 9-26.

**Gilb Tom. 2005.** *Competitive Engineering: A Handbook for Systems Engineering, Requirements Engineering and Software Engineering Using Planguage.* Reprinted 2007. Elsevier. Oxford.

**Gilb Tom. 2006.** Competitive Product Engineering: 10 Powerful Principles for winning product leadership, through advanced systems engineering. *Tom and Kai Gilb*. www.gilb.com [Abgerufen: 12. 4. 2008].

**Gilb Tom. 2008.** Decomposition of Projects: How to Design Small Incremental Steps. *Tom and Kai Gilb.* www.gilb.com.

**Gilb Tom. 2005.** Fundamental Principles of Evolutionary Project Management. *Tom Gilb and Kai Gilb.* www.gilb.com.

**Glaser Barney, Strauss Anselm. 2006.** *The Discovery of Grounded Theory: Strategies for Qualitative Research.* Renewed 1995. Aldine, New Brunswick.

**Gunasekaran Angappa. 1998.** Agile manufacturing: enablers and an implementation framework. *International Journal on Production Research.* 1998, Vol. 36, 5, pp. 1223-1247.

**Haberfellner Reinhard, de Weck Olivier. 2005.** Agile SYSTEMS ENGINEERING versus AGILE SYSTEMS engineering. *Fifteenth Annual International Symposium of INCOSE, Rochester, NY.*

**Haberfellner Reinhard, Stelzmann Ernst. 2008.** Systems Engineering: neu überdacht. *WINGbusiness.* 2008, 3.

**Haberfellner Reinhard. 1973.** *Dissertation: Die Unternehmung als dynamisches System: Der Prozesscharakter der Unternehmensaktivitäten.* Eidgenössische Technische Hochschule, Zürich.

**Haberfellner Reinhard, Nagel Peter, Becker Mario, Büchel Alfred, von Massow Heinrich. 2002.** *Systems Engineering, Methodik und Praxis.* Daenzer W. F., Huber F. [Hrsg.]. 11. Auflage. Verlag Industrielle Organisation, Zürich.

**Hall Arthur. 1962.** *A Methodology for Systems Engineering.* D. van Nostrand Company Inc., Princeton.

**Hansson Christina, Dittrich Yvonne, Gustafsson Björn, Zarnak Stefan. 2006.** How agile are industrial software development practices? *Journal of Systems and Software.* 2006, Vol. 79, pp. 1295-1311.

**Highsmith Jim. 2010.** *Agile Project Management.* 2. Edition. Addison-Wesley, Boston, MA.

**Highsmith Jim. 2002.** *Agile Software Development Ecosystems.* Boston : Addison-Wesley, Boston, MA.

**Highsmith Jim. 2002.** What Is Agile Software Development. *Cross Talk, The Journal of Defense Software Engineering.* 2002, 10.

**Hinkelmann Knut, Probst Fabian, Thönssen Barbara. 2006.** Agile Process Management: Framework and Methodology. *AAAI Spring Symposium.*

**Hruschka Peter, Rupp Chris, Starke Gernot. 2009.** *Agility kompakt: Tipps für erfolgreiche Systementwicklung.* 2. Auflage. Spektrum, Heidelberg.

**IEEE. 2005.** *ISO/IEC 15288:2004 Systems Engineering - System Life Cycle Processes.*: IEEE Computer Society, New York, NY .

**INCOSE. 2007.** *Systems Engineering Handbook.* v. 3.1. International Council on Systems Engineering.

**INCOSE. 2004.** *Systems Engineering Handbook.* v. 2a. International Council on Systems Engineering.

**Kähkönen Tuomo, Abrahamsson Pekka. 2004.** Achieving CMMI Level 2 with Enhanced Extreme Programming Approach. Bomarius F., Iida H. [Hrsg.]. *PROFES 2004, LNCS 3009.* Springer. pp. 378-392.

**Kaindl Hermann, Svetinovic Davor. 2008.** Distinction between Requirements and Their Representations. *CSER 2008 Proceedings.* April 4-5, 2008.

**Kaindl Hermann, Svetinovic Davor. 2010.** On confusion between requirements and their representations. *Requirements Engineering.* 2010, 15, pp. 307-311.

**Kaindl Hermann. 2005.** System and Software Co-Architecting. *Proceedings CSER, Hoboken, NJ.* 2005.

**Kaindl Hermann, Falb Jürgen, Arnautovic Edin, Ertl Dominik. 2010.** Increments in An Iterative Systems Engineering Life Cycle. *Proceedings of the 7th biannual European Systems Engineering Conference (EuSEC).* 2010.

**Karlström Daniel, Runeson Per. 2006.** Integrating agile software development into stage-gate managed product development. *Empirical Software Engineering.* 2006, 11. Springer. pp. 203-225.

**Kettunen Petri, Laanti Maarit. 2005.** How to steer an embedded software project: tactics for selecting the software process model. *Information and Software Technology.* 2005, Vol. 47, pp. 587-608.

**Kruchten Philippe. 2010.** Contextualizing Agile Software Development. Messnarz R., Riel A., Ekert D., Christiansen M., Johansen J., Koinig S. [Hrsg.], *17th EuroSPI² Conference Industrial Proceedings*.

**Lamnek Siegfried. 1993.** *Qualitative Sozialforschung: Methoden und Techniken.* Vol. II. BeltzPVU, Weinheim.

**Larman Craig, Basili Victor. 2003.** Iterative and Incremental Development: A Brief History. *IEEE Computer.* June 2003, pp. 47-56.

**Leitner Andrea, Kreiner Christian. 2011.** Software Product Lines - An agile success factor? *CCIS, 18th European Conference Proceedings, EuroSPI 2011.* Springer.

**Levardy Viktor, Tyson Browning. 2009.** An Adaptive Process Model to Support Product Development Project Management. *IEEE Transactions on Engineering Management.* November 2009, Vol. 56, 4.

**Liker Jeffrey, Sobek Durward, Ward Allen, Cristiano John. 1996.** Involving Suppliers in Product Development in the United States and Japan: Evidence for Set-Based Concurrent Engineering. *IEEE Transactions on Engineering Management.* May 1996, Vol. 43, 2, pp. 165-178.

**Lock Dennis. 2003.** *Project Management.* 8. Edition. Gower Publishing Company, UK.

**Madni Azad. 2008.** Agile Systems Architecting (ASA): Placing Agility Where it Counts. *Proceedings CSER, Los Angeles*.

**Maierhofer Sabine. 2009.** *Masterarbeit: Einsatz von agilen Methoden in der Produkt-/Systementwicklung.* Technische Universität Graz.

**Maierhofer Sabine, Stelzmann Ernst, Kohlbacher Markus, Fellner Björn. 2010.** Requirement Changes and Project Success: The Moderating Effects of Agile Approaches in Systems Engineering Projects. Riel A. et al. [Hrsg.] *EuroSPI 2010, CCIS 99.* Springer. pp. 60-70.

**Malotaux Niels. 2006.** Evolutionary Project Management Methods. 1.4a. *Malotaux Consultancy.* www.malotaux.nl [Abgerufen: 08. 05. 2009].

**Maskell Brian. 2006.** The age of agile manufacturing. *Journal of Supply Chain Management.* 2006, Vol. 6, 1, pp. 5-11.

**Mason-Jones Rachel, Towill Denis. 1999.** Total cycle time compression. *International Journal of Production Economics.* 1999, Vol. 62, 1-2, pp. 61-73.

**McMahon Paul. 2006.** Lessons Learned Using Agile Methods on Large Defense Contracts. *The Journal of Defense Software Engineering.* 2006, May.

**Oesterreich Bernd. 2006.** Timeboxing - Das Rückgrat agiler Projekte. *Agiles Projektmanagement.* dpunkt, Heidelberg.

**Pal Nirmal and Pantaleo Daniel [Hrsg.]. 2005.** *The Agile Enterprise: Reinventing Your Organization for Success in an on Demand World.* Springer, New York, NY.

**Palmer Stephen, Felsing John. 2001.** *A Practical Guide to Feature-Driven Development.* Prentice Hall PTR, Upper Saddle River, NJ.

**Paris Rupert, Hürzeler Peter. 2008.** Was versteht man unter Grounded Theory? *ETH Zürich - Technologie und Innovationsmanagement.* http://www.tim.ethz.ch/education/courses/courses_fs_2008/course_docsem_fs_2008/papers/17.pdf [Abgerufen: 23. 4. 2011].

**Paulk Mark. 2001.** Extreme Programming from a CMM Perspective. *IEEE Software.* 2001, November/December.

**Pich Michael, Loch Christoph, De Meyer Arnoud. 2002.** On Uncertainty, Ambiguity and Complexity in Project Management. *Management Science.* August 2002, Vol. 48, 8, pp. 1008-1023.

**Poppendieck Mary, Poppendieck Tom. 2009.** *Implementing Lean Software Development.* 7th Printing. Pearson Education, Boston, MA.

**Qumer Asif, Henderson-Sellers Brian. 2007.** An evaluation of the degree of agility in six agile methods and its applicability for method engineering. *Information and Software Technology.* 2007, 2.

**Qumer Asif, Henderson-Sellers Brian. 2006.** Measuring Agility and Adoptability of Agile Methods: A 4-dimensional Analytical Tool. *IADIS International Conference of Applied Computing.* pp. 503-507.

**Rico David. 2009.** What is the Return on Investment (ROI) of Agile Methods? *Tom & Kai Gilb.* www.gilb.com [Abgerufen: 11. 03. 2009].

**Sage Andrew, Armstrong James Jr. 2000.** *Introduction to Systems Engineering.*: John Wiley & Sons, Hoboken, NJ.

**Schatten Alexander, Schiefer Josef. 2007.** Agile Business Process Management. *IEEE International Conference on e-Business Engineering Proceedings.* pp. 319-322.

**Schwaber Ken. 2004.** *Agile Project Management with Scrum.* Microsoft Press, Redmond, WA.

**Schwaber Ken. 2007.** *Scrum im Unternehmen.* Microsoft Press Deutschland, Unterschleißheim.

**Seibert Siegfried. 2007.** Agiles Projektmanagement. *Projektmanagement aktuell.* 2007, 1, pp. 41-48.

**Singer David, Doerry Norbert, Buckley Michael. 2009.** What is Set-Based Design? *ASNE Naval Engineers Journal.* 2009, Vol. 121, 4, pp. 31-43.

**Sobek Durward, Ward Allen, Liker Jeffrey. 1999.** Toyota's Principles of Set-Based Concurrent Engineering. *Sloan Management Review.* Winter 1999, Vol. 40, 2, pp. 67-83.

**Song Xiping, Rudorfer Arnold, Hwong Beatrice, Matos Gilberto, Nelson Christopher. 2005.** S-RaP: A Concurrent, Evolutionary Software Prototyping Process. Li M., Boehm B., Osterweil L. [Hrsg.]. *Lecture Notes in Computer Science.* 2005, Vol. 3840.

**Stelzmann Ernst. 2011.** Contextualizing Agile Systems Engineering. *IEEE International Systems Conference (SysCon) Proceedings, Montreal, Kanada.* April 4-7, 2011.

**Stelzmann Ernst, Kreiner Christian, Spork Gunther, Messnarz Richard, Koenig Frank. 2010.** Agility meets Systems Engineering: A Catalogue of Success Factors from Industry Practice. Andreas Riel et al [Hrsg.]. *EuroSPI 2010, CCIS 99.Springer.*

**Stelzmann Ernst, Spork Gunther, Kreiner Christian, Messnarz Richard, Koenig Frank. 2010.** Practical Use of Agile Methods within Systems Engineering. *International Spice Days, Stuttgart.* June 21-23, 2010.

**Tsoukas Haridimos. 2003.** Do we really understand tacit knowledge? Easterby-Smith M., Lyles M.A. [Hrsg.]. *The Blackwell Handbook of Organizational Learning and Knowledge Management.* Blackwell Publishing, Cambridge.

**Turkington Garry. 2007.** *Architecting Agile Systems and Enterprises, Agile Development of Agile Systems - A Lifecycle Model.* working paper. Stevens Institute of Technology SDOE Program, Hoboken, NJ.

**Turner Richard. 2002.** Agile Development: Good Process or Bad Attitude? Oivo M., Komi-Sirviö S. [Hrsg.]. *Profes 2002, Lecture Notes on Computer Science.* 2002, Vol. 2559, pp. 134-144.

**Turner Richard, Jain Apurva. 2002.** Agile Meets CMMI: Culture Clash or Common Cause? Wells D., Williams L.[Hrsg]. *XP/Agile Universe 2002, Lecture Notes on Computer Science.* 2002, Vol. 2418, pp. 153-165.

**Turner Richard. 2007.** Toward Agile Systems Engineering Processes. U.S. Airforce Software Technology Support Center. *Cross Talk, The Journal of Defense Software Engineering.* 2007, April.

**US-Marine-Corps. 1997.** Warfighting. www.dtic.mil/doctrine/jel/service_pubs/mcdp1.pdf. [Abgerufen: 15. 12. 2010].

**Ward Allen, Liker Jeffrey, Cristiano John, Sobek Durward. 1995.** The Second Toyota Paradox: How Delaying Decisions Can Make Better Cars Faster. *Sloan Management Review.* Spring 1995, Vol. 36, 3, pp. 43-61.

**Williams Laurie, Kessler Robert, Cunningham Ward, Jeffries Ron. 2000.** Strengthening the Case for Pair Programming. *IEEE Software.* 2000, July/August.

**Wilson Mark, Mooz Harold. 2003.** Agile Systems Engineering for Rapid Project Solution Development. *13th Annual International INCOSE Symposium Proceedings*.

**Womack James, Jones Daniel, Roos Daniel. 1991.** *The Machine That Changed the World: The Story of Lean Production.* Harper Perennial, New York, NY.

**Yourdon Ed. 1999.** *Death March: The Complete Software Developer's Guide to Surviving "Mission Impossible" Projects.* Prentice Hall PTR, Upper Saddle River, NJ.

**Zobel Alexander. 2005.** *Agilität im dynamischen Wettbewerb: Basisfähigkeit zur Bewältigung ökonomischer Turbulenzen.* Dt. Univ. Verlag, Wiesbaden.