MAPE-K4SEM –

Selbst-adaptive Software-Engineering-Methoden

Silke Geisen

sgeisen@s-lab.upb.de

Dissertation

zur Erlangung des Grades “Doktor der Naturwissenschaften” (Dr. rer. nat.)

Erster Gutachter: Prof. Dr. Gregor Engels
Zweiter Gutachter: Prof. Dr. Jürgen Ebert

Fakultät für Elektrotechnik, Informatik und Mathematik
Universität Paderborn
Zukunftsmeile 1
D-33102 Paderborn

Paderborn, November 2014

Danksagung

Auch wenn das Erarbeiten und Schreiben einer Dissertation eine eigenständige Leistung ist, wäre sie ohne die Unterstützung von anderen Personen kaum möglich. An dieser Stelle möchte ich mich bei all jenen bedanken, die mich über die ganzen Jahre unterstützt und immer wieder motiviert haben, diese Arbeit fertig zu stellen.

Mein größter Dank geht hier an meinen Doktorvater und Mentor Prof. Dr. Gregor Engels. Durch seine Unterstützung, konstruktive Kritik und sein Vertrauen in mich ist diese Arbeit erst möglich gewesen. Gerade an Tiefpunkten konntest Du mich immer wieder motivieren, so dass ich heute an dieser Stelle stehe. Danke!

Neben Gregor Engels geht mein Dank an Dr. Stefan Sauer. Nicht nur, dass er mir die Ehre erweist, ein Mitglied meiner Promotionskommission zu sein, sondern dass er mir zusammen mit Gregor die Möglichkeit gegeben hat, im s-lab anzufangen, obwohl ich in meinem Vorstellungsgespräch von einem meiner heutigen Spezialgebiete, den Agilen Methoden und insbesondere Scrum, noch keine Ahnung hatte. Ich konnte gerade auch durch die verschiedenen Projekte im s-lab viel Neues lernen und sie haben mir die Möglichkeit gegeben, mich immer weiter zu entwickeln.

Bedanken möchte ich mich auch bei meinem Zweitgutachter Prof. Dr. Jürgen Ebert, durch sein konstruktives Feedback schon während des Doktorandensymposiums auf der SE 2012 in Berlin. Ebenso möchte ich mich bei meinen weiteren Mitgliedern der Promotionskommission Prof. Dr. Wilhelm Schäfer und Prof. Dr. Franz J. Rammig bedanken, dass sie diesen „letzten Schritt“ gemeinsam mit mir gegangen sind.

Für die vielen fruchtbaren Diskussionen und die Unterstützung geht ein weiteres Dankesschön an meine Arbeitskollegen, dabei insbesondere meinen Bürokollegen Holger Fischer für das Lesen von Teilen meiner Arbeit und unsere regelmäßigen Diskussionen, Baris Güldali den ich immer wieder um Rat fragen konnte und wir gerade im Testbereich in Verbindung mit Agilen Methoden glaub ich einiges auf die Beine gestellt haben, sowie meinen ehemaligen Bürokollegen Frank Brüseke und Simon Oberthür, der mir gerade in der „heißen Endphase“ den Rücken im Projekt freigehalten hat. Nicht vergessen möchte ich an dieser Stelle die „guten Seelen“ der AG Engels Friedhelm Wegener, Beatrix Wiechers und Sonja Saage, die einem bei all den kleineren und größeren Problemchen mit Rat und Tat zur Seite standen.

Bedanken möchte ich mich auch bei meinem ehemaligen Projektkollegen Dietmar Koch, der sich bereit erklärt hat, meine Arbeit Korrektur zu lesen. Dabei geht ein weiteres großes Dankeschön an meine „persönliche Deutschlehrerin“, meine Schwester Stefanie Müser, die ihre Herbstferien geopfert hat, um meine Arbeit Korrektur zu lesen. Durch ihre teils witzigen Kommentare hat sie dafür gesorgt, dass ich „meine Fehler“ gern zur Kenntnis genommen habe, insbesondere wenn meine Grammatik mal wieder wie bei Yoda gewesen ist. Auch ist sie mit zuständig für mein Maskottchen, den „Scrum-Wusel“; ich hoffe, er streift nicht mehr allzu sehr durch deine Träume.

Als letztes möchte ich mich bei meinen Eltern Gisela und Hans Brandenburg bedanken. Ohne eure Unterstützung und Motivation würde ich heute nicht hier stehen, denn ihr habt mich immer wieder ermutigt, meinen Weg zu gehen und ihr habt mir erst mein Studium und meine wissenschaftliche Laufbahn ermöglicht.

Zusammenfassung

Damit die erfolgreiche Entwicklung einer Software und somit der Erfolg eines Projektes gewährleistet sind, wird häufig eine Software-Engineering-Methode (SEM) zu Beginn auf die Projektsituation abgestimmt. Dennoch scheitern teilweise bis zu 75% von Projekten im IT-Umfeld und überschreiten ihr Budget oder können Deadlines nicht einhalten. Gerade während der Durchführung der Software-Engineering-Methode können Änderungen an der Projektsituation oder mangelnde Qualität den Projekterfolg gefährden. Doch eine kontinuierliche Überwachung während der Nutzung der SEM und speziell die zeitliche Komponente sind kritische Faktoren. Sie machen es schwierig, eine Software-Engineering-Methode entsprechend zur Laufzeit anzupassen. Diese Situationen und insbesondere der Faktor Zeit machen eine dynamische und möglichst eigenständige Anpassung der SEM erforderlich.

Bekannte Verbesserungs- und Anpassungsverfahren wie das Situational Method Engineering, Six Sigma oder der PDCA-Zyklus, auch Deming-Zyklus genannt, sind aufgrund ihrer langen Durchführungsdauer kaum für eine solche Anpassung geeignet. Ferner finden diese entweder zu Beginn des Projektes oder erst wenn bereits Probleme aufgetreten sind und das Projekt bereits „vor die Wand gelaufen“ ist, statt. Das Projektcontrolling oder die Agilen Methode bieten zwar erste Ansätze für eine Anpassung zur Laufzeit, sind jedoch für eine eigenständige Anpassung nicht einsetzbar, da ihnen insbesondere Möglichkeiten zur Automatisierung fehlen.

Im Gegensatz zu Six Sigma, dem Deming-Cycle oder auch den Agilen Methoden beobachten Ansätze aus den selbst-adaptiven Systemen zur Laufzeit automatisch das gegebene System über sogenannte Feedbackschleifen und passen das System gegebenenfalls eigenständig an. Das bekannteste Modell ist dabei die Feedbackschleife MAPE-K. Diese Arbeit stellt mit MAPE-K4SEM einen Ansatz vor, wie sich diese MAPE-K-Feedbackschleife auf die Anpassung von einer SEM übertragen lässt und somit eine selbst-adaptive Software-Engineering-Methode ermöglicht.

Abstract

For the successful development of software and thus the success of a project, a software engineering methodod (SEM) is first tailored to the project situation. Despite that up to 75% of projects in the IT enviroment fail during their enactment. Deadlines are exceeded as well as the budget of the project. The implementation of the software engineering method, changes to the project situation or lack of quality endanger the success of the project. Especially time is a critical factor to detect problems and adapt a SEM in time. A continous monitoring of the software engineering method is needed as well as a dynamic and autonomous adaptation.

Approaches as Situational Method Engineering, Six Sigma or the PDCA-Cycle, also known as Deming Cycle, are well-known for the improvement and adaptation of software engineering methods. However, due to their long implementation duration they are hardly suitable for such an adjustment. Furthermore, these procedures are typically performed before or after problems are detected and the project has failed. Though agile methods like Scrum or project controlling use first approaches for inspection and adaptation of a running project, they are not usable for an autonomous adaptation due to lack of automating possibilities.

Unlike Six Sigma, the Deming Cycle or the Agile Methods, approaches from the self-adaptive systems domain observe systems automatically at runtime using feedback loops and adapt the system autonomously if necessary. The best-known model is called the MAPE-K loop. This PhD thesis presents with MAPE-K4SEM an approach that uses the MAPE-K loop for a continous monitoring and automatic adaption of software engineering method and thus makes a self-adaptive software engineering method possible.

Inhaltsverzeichnis

Kapitel 1 Einleitung

1.1 Motivation

Effiziente, effektive und qualitativ hochwertige Softwareentwicklung wird in der heutigen Zeit immer wichtiger. Dennoch scheitern nach Voller bis zu 75% der Projekte im IT-Umfeld [Vo13]. Als Gründe und mögliche Faktoren nennt Voller hierfür u.a. undefinierte Ziele, fehlende Unterstützung des Managements, unzureichend definierte Rollen und Verantwortlichkeiten, aber auch ungenügende Kommunikation, das Ignorieren von Warnzeichen im Projekt sowie nicht genügende Beachtung von Störeinflüssen.

Um ein Projekt zum Erfolg zu führen und eine Software erfolgreich zu entwickeln, gibt es die verschiedensten Vorgehensmodelle bzw. Softwareentwicklungsprozesse und die neuen Agilen Methoden. In dieser Arbeit wird der Begriff Software-Engineering-Methode [ES10] verwendet. Als eine Software-Engineering-Methode (SEM) wird die relevante Menge an Elementen verstanden, welche benötigt wird, um ein Software-Projekt in allen wichtigen Aspekten zu beschreiben. Die SEM beinhaltet also nicht nur den Softwareentwicklungsprozess an sich und seine Aktivitäten, sondern zusätzlich alle Artefakte, Rollen und Aufgaben die durchgeführt werden müssen um die Meilensteine zu erreichen, sowie die Werkzeuge und Techniken die für die Umsetzung der SEM benötigt werden [GLE12].

Doch auch wenn eine Software-Engineering-Methode im Projekt verwendet wird, so gibt es nicht **die** eine Software-Engineering-Methode, welche für jedes Projekt passt, die „*one-size-fits-all*“ Methode [Br96]. Typischerweise wird eine SEM vor Beginn des Projektes entweder entsprechend zugeschnitten (Tailoring [LL10]) oder eine Software-Engineering-Methode wird eigens für die Situation im Projekt entwickelt. Um eine solche situationsbezogene SEM für unternehmensinterne Zwecke zu entwickeln, wird das Situational Method Engineering [Br96, HSR10] angewendet.

Auch wenn eine Software-Engineering-Methode eigens für ein Projekt entwickelt worden ist, kann es dennoch in der Praxis zu Problemen kommen, so dass die Projekte am Ende scheitern und zum Misserfolg führen. Neben den oben genannten Faktoren können folgende weitere Beispiele aus Projekten des s-lab – Software Quality Labs[1] in Paderborn genannt werden:

- In dem Scrum-Projekt „Quasi-Scrum“ [EG09] wurden nicht-funktionale Anforderungen durch fehlende Einträge im Product-Backlog nicht betrachtet. Beim Kunden traten anschließend Performanz-Probleme auf. Um die Betrachtung der nicht-funktionalen An-

[1] http://s-lab.upb.de

forderungen in einem Entwicklungsprozess frühzeitig sicherzustellen, musste die SEM angepasst werden.

- Während eines großen und langlaufenden Projektes wächst die Teamgröße. Die Verwaltungsstrukturen zur Unterstützung für beispielsweise Dokumente und Tools müssen im laufenden Projekt eingeführt oder angepasst werden.

- Projektressourcen ändern sich beispielsweise durch eine Fusion oder eine Umstrukturierung. Das betrifft auch laufende Projekte, sofern diese nicht abgebrochen oder neu gestartet werden können.

- Gerade in agilen Projekten bekommt das Testen zwar eine wichtige Rolle zugeschrieben. Aber schaut man sich die Original-Literatur an, wie beispielsweise den Scrum Guide [SS13], wird das Testen nur sehr oberflächlich oder gar nicht beschrieben. Es werden häufig nur die Aufgaben wie beispielsweise Product Backlog Einträge in Scrum selbst getestet und nicht der große Zusammenhang. Ein Relase- oder End-To-End-Test ist insbesondere in der bekanntesten agilen Methode Scrum gar nicht erst vorgesehen [GG12]. Durch dieses eher unstrukturierte Testen bzw. Weglassen eines End-To-End- oder auch vollständigen System-Tests kann es passieren, dass nur ungenügend oder einige Funktionen gar nicht getestet werden.

- In Projekten sollen Standards und Normen eingehalten werden. Ändern sich diese kann es passieren, dass ebenfalls die Software-Engineering-Methode angepasst werden muss.

- Die Erfahrung in einem weiteren s-lab-Projekt für ein großes Telekommunikationsunternehmen mit verteilten Teams zeigte, dass ein unstrukturiertes Vorgehen gerade im Projektcontrolling und in der Ausführung zu großen Problemen führte. Deadlines konnten nicht eingehalten werden, die Software zeigte sowohl in den Tests als auch im produktiven Betrieb teilweise schwerwiegende Fehler, was zu hohen Kosten führte. Es war klar, dass die Software-Engineering-Methode angepasst werden musste, doch es war zum einen nicht deutlich, wie die genaue Anpassung aussehen sollte. Zum anderen war nicht ersichtlich, wie die Software-Engineering-Methode im laufenden Betrieb überwacht und analysiert werden kann. Das größte Problem war, wie der Projektleiter mitteilte, dass die Anpassung zwar notwendig sei, doch das weder wirklich Zeit noch Ressourcen für eine aufwendige Anpassung vorhanden waren. Somit blieb es am Ende bei kleinen und nicht strukturierten ad-hoc Anwendungen, die teils im Team, teils aber auch vom Projektleiter „aus dem Bauch

heraus“ entschieden wurden. Die Probleme wurden dadurch wenig oder gar nicht gelöst.

- In dem Projekt wurde ebenfalls deutlich, dass bei Änderungen gerade über ein verteiltes Team hinweg, (neue) Verantwortlichkeiten nicht klar gewesen sind. Die Teammitglieder haben Anpassungen bzw. Änderungen zwar befürwortet, doch hätten sie sich eine konkrete Zuteilung von Verantwortlichkeiten gewünscht und vor allem eine Benachrichtigung, dass sich etwas geändert hat. Dies wurde nur unzureichend oder gar nicht vorgenommen.

- Eine weitere Erfahrung aus zwei Scrum-Projekten war, dass die Teams zeitweise recht groß gewesen sind, teilweise bis zu 15 Personen. In Scrum ist vorgesehen, dass ab einer Größe von mehr als neun Personen das Team geteilt wird und ein Scrum of Scrums eingeführt werden sollte. Dies ist den Personen zwar bekannt gewesen, doch der Aufwand für die Anpassung, also das Aufteilen des Teams, Zuweisung der neuen Aufgaben, Einführen eines neuen Meetings, war den Mitgliedern zu aufwendig. Durch die höhere Anzahl von Personen gab es regelmäßig Kommunikationsschwierigkeiten, beispielsweise dass nicht jeder wusste, woran der andere gerade arbeitete. Es kam sowohl zu Überschneidungen, als auch zu Konflikten, z.B. dass eine Aufgabe, welche für die Weiterarbeit eines Teammitglieds wichtig gewesen ist, noch gar nicht angefangen war. Dadurch ist es vorgekommen, dass Aufgaben nicht pünktlich beendet werden konnten.

Diese Beispiele und die oben genannten Faktoren zeigen, dass es zwar notwendig ist, eine Software-Engineering-Methode im laufenden Projekt dynamisch anzupassen, doch in der Praxis ist dies – wenn überhaupt – nur unstrukturiert und ad hoc der Fall. Es ist wichtig, dass unmittelbar auf problematische Situationen reagiert werden kann, um den Erfolg sowie die Qualität des Projektes weiterhin gewährleisten zu können. Doch gerade die Überwachung während der Nutzung der SEM und speziell die zeitliche Komponente sind kritische Faktoren. Diese machen es schwierig, eine Software-Engineering-Methode entsprechend zur Laufzeit anzupassen.

1.2 Problemstellung und Forschungsfrage

Wie die genannten Beispiele im vorherigen Abschnitt zeigen, ist es notwendig, eine Software-Engineering-Methode im laufenden Projekt zu überwachen, aber insbesondere sie zeitnah anzupassen. Ein Projekt besitzt auf der einen Seite eine Software-Engineering-Methode für die Durchführung, auf der anderen Seite wird das Projekt wie in Abbildung 1 zu sehen vom Projektmanagement und hauptsächlich von der Projektkontrolle überwacht.

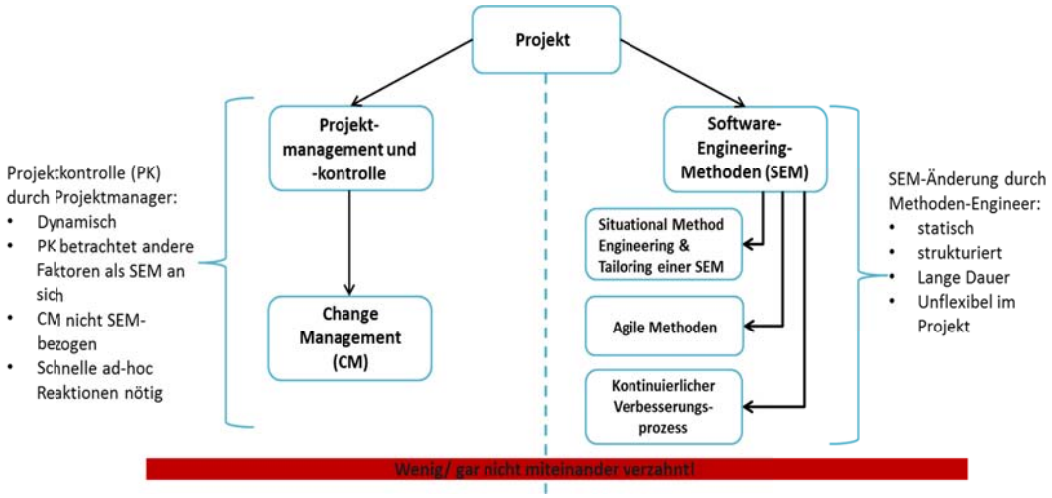

Abbildung 1 Projektmanagement, Projektkontrolle und SEM sind kaum miteinander verzahnt

Projekte sind allgemein eine komplexe Angelegenheit und können von langer Dauer sein. Dabei ist es wichtig, dass die Verantwortlichen zu jeder Zeit einen umfassenden Überblick über das gesamte Projekt haben. Im Detail umfasst das Projektmanagement somit alle Elemente eines Projektes, die wichtigsten sind dabei u.a. die Planung, die Kontrolle, die Organisation und die Personalführung [Sc13, Fi10]. Im Projektmanagement dreht es sich immer um eine bestimmte Klasse von Zielen, welche das „Magische Dreieck" bilden [Sn05]. Bei dem Dreieck handelt es sich auch um die Abhängigkeiten zwischen der Dauer des Projektes, den Kosten und dem Aufwand des Projektes, sowie der Qualität bzw. des gelieferten Leistungsumfangs.

Die Projektkontrolle unterstützt dabei hauptsächlich das Projektmanagement bei der Planung, Überwachung und Steuerung des Projektes. Wichtig ist dabei basierend auf [F10], [Sc13] und [BK12], dass die Rahmenbedingungen eingehalten und die definierten Projektziele erreicht werden. Allerdings heißt die Erreichung der Projektziele, dass es sich bei der Projektkontrolle um Ziele im Rahmen des Umfangs des Projektes, den Projektkosten, den Terminen, der Qualität und auch der Ressourcen handelt. Es geht weniger direkt um die eigentliche Software-Engineering-Methode. Der Fortschritt des Projektes wird allgemein anhand der Einhaltung der Kosten, der Einhaltung von Terminen sowie der Erfüllung der Qualitätsvorgaben kontrolliert. Die Kontrolle findet somit auf einer anderen Ebene als eine Überwachung der genutzten Software-Engineering-Methode statt. Zusammenfassend ist zu sagen, dass die Projektkontrolle zwar dynamisch ist, doch sie betrachtet andere Faktoren als für die Software-Engineering-Methode nötig wären.

Das Change Management, zu Deutsch Änderungs- oder auch Veränderungsmanagement, beschäftigt sich zwar mit Änderungen, aber typischerweise mit denen auf der Unternehmensebene. Das Hauptaugenmerk des

Change Managements liegt dabei mehr auf Personen und allgemeinem Wandel im Unternehmen, weniger auf der konkreten Software-Engineering-Methode. Die Veränderungen können sich zwar auf die SEM auswirken und somit eine Ursache sein, dass diese ebenfalls geändert werden muss. Allerdings ist dies höchstens ein Baustein im Gesamtkontext des Veränderungsprojektes, aber nicht primär. Doch die enge Verwandtschaft mit dem Projektmanagement und der zu Grunde liegende Prozess für die Durchführung eines Veränderungsprojektes können in eine Änderung an einer Software-Engineering-Methode mit einfließen.

Nach [SH13] findet Veränderungsmanagement in drei Phasen statt:

- Planung, wo die Vision erstellt, der aktuelle Status analysiert und Maßnahmen geplant werden;
- Umsetzung, wo die erstellten Maßnahmen und Lösungen umgesetzt werden;
- Evaluierung, wo die entsprechenden Maßnahmen überprüft werden, ob sie den gewünschten Effekt geliefert haben.

Schaut man sich nun in Abbildung 1 die rechte Seite die Software-Engineering-Methode an, so gibt diese dem Projekt eine strukturierte Vorgehensweise zur eigentlichen Durchführung, lässt aber Komponenten des Projektmanagements wie die Personalführung, die eigentliche Kontrolle selbst usw., außen vor. Es gibt auf dieser Seite verschiedene Möglichkeiten, die Software-Engineering-Methode anzupassen, was durch einen Methoden-Engineer erfolgt. Dieser kann eine Software-Engineering-Methode für das Projekt spezifisch mit Hilfe von Situational Method Engineering [HSR10] zusammensetzen oder sie für das Projekt „zuschneiden“ (Tailoring). Neben diesen beiden vorgelagerten Ansätzen gibt es noch weitere im Rahmen des Kontinuierlichen Verbesserungsprozesses wie Six Sigma [KA06] oder dem PDCA-Zyklus [De86]. Diese beschäftigen sich ebenfalls mit Veränderungen und Verbesserungen im Projekt.

Auch wenn die Vorgehensweisen alle Ansätze zur Verbesserung zeigen, so liegt, bis auf bei den vorgelagerten Ansätzen, der Fokus allgemein mehr auf dem gesamten Projekt, als auf der genutzten Software-Engineering-Methode. In den vorgelagerten Ansätzen wird zwar eine Anpassung vorgenommen, jedoch ist diese statisch, das heißt sie wird nur einmal vor Beginn des Projektes durchgeführt. Für die weitere Anpassung im Projekt sind diese Ansätze sehr unflexibel.

Bei den Ansätzen zum kontinuierlichen Verbesserungsprozess kommt hinzu, dass sie von langer Dauer sind, das heißt sie können unter Umständen mehrere Wochen und länger dauern. Wie sich in den Beispielen im vorheri-

gen Abschnitt zeigte, ist aber gerade die Zeit ein wichtiger und kritischer Faktor. Eine verzögerungsfreie und dynamische Änderung während des laufenden Projektes ist mit diesen Verfahren nur schwer oder gar nicht möglich.

Einen weiteren Ansatz zur Verbesserung und Anpassung einer Software-Engineering-Methode bieten die Agilen Methoden [Co02] und dabei insbesondere Scrum [SB02]. Diese benutzen verschiedene Inspektionspunkte um mögliche unerwünschte Abweichungen (Produkt- und Entwicklungsprozessabweichungen) zu entdecken [SS13].

Wenn eine Abweichung festgestellt wird, die außerhalb der akzeptablen Grenze liegt, dann sollen Arbeitsgegenstand oder Prozess so schnell wie möglich angepasst werden. Interessant ist dabei die Retrospektive. Sie ist eine Gelegenheit für das Scrum Team, *„sich selbst zu überprüfen, und einen Verbesserungsplan für den kommenden Sprint zu erstellen*“ [SS13, S. 12]. Damit ist die Retrospektive das wichtigste Ereignis innerhalb von Scrum, welches auf Inspektion und Adaption des gesamten Entwicklungsprozesses fokussiert ist.

Doch auch wenn die Agilen Methoden bereits Ansätze zur Verbesserung liefern, liegt ihr Fokus zunächst nur zum Teil auf der Software-Engineering-Methode, der Hauptfokus liegt auf dem zu entwickelnden Produkt. Durch bestimmte Techniken wie tägliche Meetings, eine Retrospektive, Fortschrittsanalysen usw. kann zwar eine regelmäßige Überwachung gegeben sein, diese ist aber typischerweise bezüglich der SEM eher informell und nicht strukturiert. Gerade in der Planung einer Anpassung hat der Ansatz hier Schwächen. Ferner sind die Überwachungen nur punktuell und weniger kontinuierlich. Ebenfalls wird bei den gesamten Ansätzen der Faktor Zeit nicht mit betrachtet. Um die Dauer der Überwachung und Durchführung der Anpassung zu verkürzen, wären Automatisierungsmöglichkeiten wichtig, doch diese sind alle in den Ansätzen nicht gegeben.

Aus der Betrachtung der verschiedenen Ansätze und der Notwendigkeit einer Überwachung und Anpassung einer Software-Engineering-Methode im laufenden Projekt stellt sich die Frage:

Wie kann eine Software-Engineering-Methode während der Nutzung automatisch und somit eigenständig überwacht, hinsichtlich Abweichungen analysiert sowie anschließend selbstständig und automatisch angepasst werden?

1.3 Zielsetzung und Lösungsansatz

Für die Beantwortung der im vorherigen Abschnitt gestellten Frage sollen die Vorteile aus den Ansätzen und aus der Projektkontrolle, die drei genannten Phasen aus dem Change Management und die Software-Engineering-Methode zusammengebracht und miteinander „verzahnt“ werden. Das Ziel ist es, einen Ansatz zu entwickeln, der eine Software-Engineering-Methode möglichst selbstständig, also adaptiv anpasst.

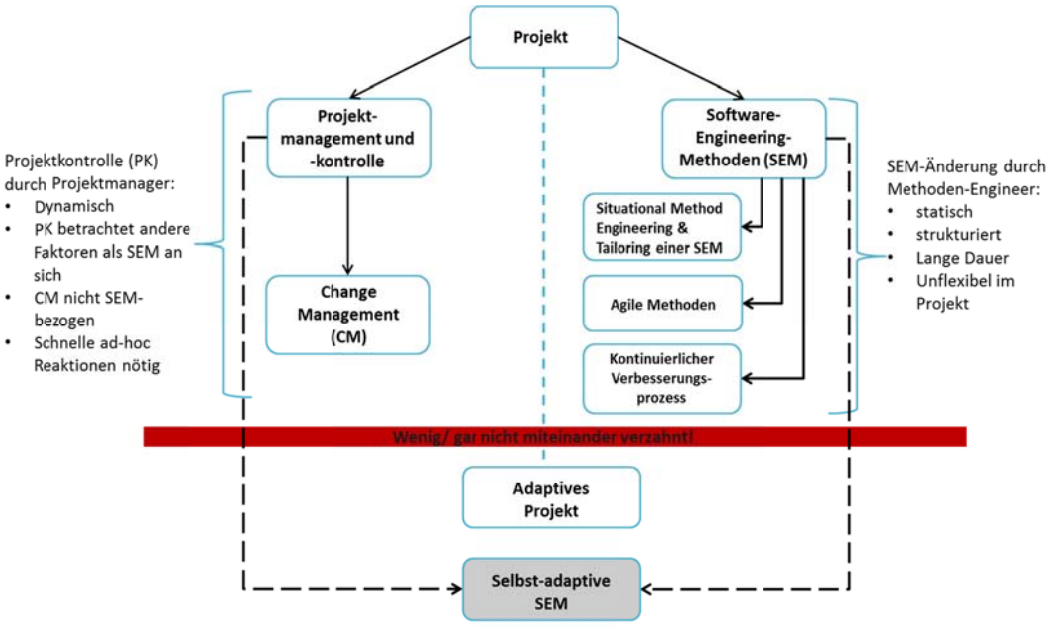

Abbildung 2 Verbindung von Projektmanagement/ Projektkontrolle und SEM zu einer selbst-adaptiven Software-Engineering-Methode

Die Software-Engineering-Methode soll während ihrer Nutzung auf der Instanz-Ebene kontinuierlich überwacht, der aktuelle Status hinsichtlich einer nötigen Anpassung ausgewertet, darauf aufbauend eine Anpassung auf Typ-Ebene geplant und diese soll anschließend dynamisch und möglichst automatisiert während der Laufzeit durchgeführt und in das Projekt zurück überführt werden.

Daraus und nach [Ge12] und [GLE12] ergeben sich folgende erste Anforderungen für einen eigenen Ansatz zur selbst-adaptiven Anpassung einer Software-Engineering-Methode:

A1. Der Fokus der Anpassung liegt auf der genutzten Software-Engineering-Methode.

A2. Der Anpassungszeitpunkt ist zur Laufzeit und die Dauer der gesamten Anpassung soll möglichst kurz sein.

A3. Die SEM soll kontinuierlich und möglichst eigenständig während ihrer Ausführung in Hinblick auf notwendige Abweichungen und Anpassungen auf der Instanz-Ebene beobachtet werden.

A4. Die beobachteten Werte und der aktuelle Status müssen zur Laufzeit analysiert und schnell beurteilt werden können.

A5. Eine unmittelbare Anpassung der Software-Engineering-Methode zur Laufzeit muss unter Betrachtung von vorher definierten Qualitätszielen geplant und durchgeführt werden können.

A6. Die Anpassung soll schnellstmöglich und zeitnah sowie möglichst automatisch erfolgen.

Um eine Software-Engineering-Methode zur Laufzeit eigenständig zu überwachen und hinsichtlich ihres aktuellen Status analysieren zu können, ist es wichtig, ein kontinuierliches Feedback von der SEM zu bekommen. Das Feedback muss ausgewertet, die SEM gegebenenfalls angepasst und anschließend muss die SEM wieder beobachtet werden, vor allem ob die Qualität weiterhin oder wieder gegeben ist [GLE12]. Dies entspricht dem Verhalten einer sogenannten Feedbackschleife. Aufgrund der Beurteilung soll eine weitere Planung für die Anpassung der SEM auf Typ-Ebene folgen, welche anschließend zurück in das aktuelle Projekt überführt werden muss. Diese Anpassung und Überführung sollen nicht nur während des Projektes selbst stattfinden, sondern so schnell wie es geht, wenn möglich sogar automatisiert.

Die Idee einer Feedbackschleife und der Auswertung des Feedbacks innerhalb eines laufenden Systems ist nicht neu. Diese Feedbackschleifen, engl. Feedback-Loops [Do06, Br09], kommen heute insbesondere im Bereich Autonomic Computing [KC03] zum Einsatz. Eine der bekanntesten Feedbackschleifen in diesem Bereich ist die MAPE-K-Feedbackschleife [IB06, KC03]. „MAPE“ steht dabei für die verschiedenen Phasen *Monitor – Analyse – Plan – Execute*. Das „K“ bezeichnet dabei eine Wissensbasis die *Knowledge Base,* auf der die vier MAPE-Phasen agieren.

Über einen sogenannten „Autonomic Manager“ wird ein „Managed Element“, welches ein laufendes System abbildet, kontinuierlich zur Laufzeit durch Sensoren überwacht. Diese Daten werden aufbereitet, anschließend analysiert und wenn diese nicht den definierten Werten entsprechen, wird eine Anpassung geplant und über Effektoren ausgeführt. Für die Durchführung der verschiedenen Phasen wird das „Wissen“ aus der Wissensbasis genutzt. Diese kann durch hinzugewonnenes Wissen fortlaufend erweitert werden.

Die Idee ist nun, den Ablauf des MAPE-K auf die Anpassung einer Software-Engineering-Methode zu übertragen. Eine der entscheidenden Fragen ist dabei, wie die Anpassung einer Software-Engineering-Methode mit dem MAPE-K kombiniert werden kann; ist es möglich alles auf die entsprechenden Phasen abzubilden und wie müssten diese dann aussehen?

Der Beitrag dieser Arbeit ist die Vorstellung des Ansatzes MAPE-K4SEM, welcher MAPE-K als Kern enthält und eine selbst-adaptive Software-Engineering-Methode ermöglicht. Mit dem analog zum Autonomic Manager entwickelten SE Method Manager ist es möglich, eine Software-Engineering-Methode automatisch und somit eigenständige sowie kontinuierlich zu überwachen, den aktuellen Status der SEM bzgl. möglicher Abweichungen zu analysieren, eine nötige Anpassung zu planen und diese anschließend schnellstmöglich auszuführen. Um die „richtigen“ Werte in der SEM zu messen und diese entsprechend anhand von Analyseregeln auszuwerten und eine Anpassung bzgl. vordefinierter Ziele zu planen, wurde der SE Method Manager um eine Pre-Work erweitert. Beide Teile zusammen, die Pre-Work und der SE Method Manger (MAPE-K) ergeben den MAPE-K4SEM-Ansatz, welcher durch einen Ablauf in 10 Schritten charakterisiert ist.

Neben der zeitnahen und eigenständigen Anpassung einer Software-Engineering-Methode zur Laufzeit ist ein weiterer Beitrag der Arbeit, dass der Ansatz MAPE-K4SEM verschiedene Themengebiete wie Projektmanagement und –kontrolle sowie das Software Engineering nun miteinander verbindet.

1.4 Aufbau der Arbeit

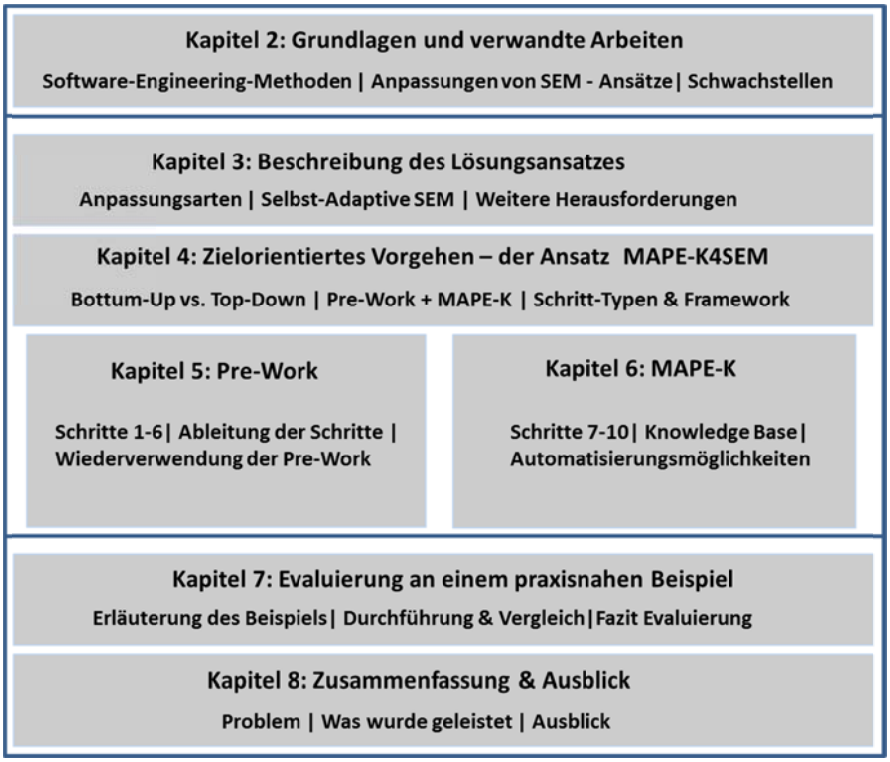

Abbildung 3 Aufbau der Arbeit

Nach dieser Einleitung besteht die Arbeit aus drei Blöcken mit insgesamt sieben weiteren Kapiteln. Der erste Block besteht aus dem zweiten Kapitel und behandelt die Grundlagen, welche für die Arbeit benötigt werden, sowie verwandte Arbeiten. Dabei wird vor allem auf die Software-Engineering-Methoden, die Projektkontrolle und das Change Management eingegangen. Bei den verwandten Arbeiten werden verschiedene Ansätze dargestellt, welche sich mit der Anpassung einer Software-Engineering-Methode beschäftigen, sowohl vor Beginn eines Projektes als auch Ansätze zur kontinuierlichen Verbesserung. Das Kapitel schließt mit den Schwachstellen der bekannten Ansätze und den ersten Anforderungen an den eigenen Ansatz.

Der Hauptblock besteht aus vier Kapiteln, beginnend mit Kapitel 3, dem Beschreiben des Lösungsansatzes. Das Kapitel startet aufbauend auf den Schwachstellen aus Kapitel 2 mit einer genauen Problem-Analyse und –Beschreibung. Nach einer Beschreibung verschiedener Anpassungs-Arten von Software-Engineering-Methoden und dem Konkretisieren der Anforderungen wird die Konzeption eines sogenannten SE Method Managers vorgestellt. Dieser ist an die bekannte Feedbackschleife MAPE-K aus den selbstadaptiven Systemen angelehnt. Nach einem kurzen Zwischenfazit wird der Ansatz tiefergehend analysiert und weitere Herausforderungen werden herausgearbeitet.

Das Kapitel 4 erweitert den Lösungsansatz um ein zielorientiertes Vorgehen, um den Herausforderungen zu begegnen. Nach der Diskussion, ob ein Bottom-Up oder Top-Down-Ansatz gewählt werden sollte, wird eine Kombination der beiden Ansätze, der MAPE-K4SEM, mit einem 10-Schritte-Ablauf entwickelt. Dieser Ansatz besteht aus einer „Pre-Work“ und dem späteren MAPE-K. Das Kapitel schließt mit der Beschreibung verschiedener Schritt-Typen, welche der MAPE-K4SEM-Ansatz beinhaltet und einem Framework, wie diese Schritte in Kurzform beschrieben werden können.

Die Kapitel 5 und 6 beschreiben die jeweiligen Schritte in der Tiefe, wobei Kapitel 5 die „Pre-Work“ erläutert und Kapitel 6 die spätere Ausführung, den MAPE-K. Neben den einzelnen Schritten wird in Kapitel 5 insbesondere darauf eingegangen, wie sich die einzelnen Schritte, Regeln und Werte in der Pre-Work herleiten lassen. Zusätzlich wird ein Ansatz vorgestellt, wie diese wiederverwendet werden können. Kapitel 6 beschreibt zusätzlich zu den Schritten zunächst den Baustein der Wissensbasis (Knowledge Base), welcher für das Gelingen des MAPE-K essentiell ist. Das Kapitel schließt mit Möglichkeiten zur Automatisierung der einzelnen Schritte.

Der dritte Block der Arbeit behandelt zunächst mit Kapitel 7 die Evaluierung anhand eines praxisnahen Beispiels, anhand dessen die Funktionsweise des Ansatzes exemplarisch durchgeführt und überprüft wird. Dieses Beispiel ist im Kern an das im ersten Abschnitt erwähnte Praxisprojekt Quasi-Scrum [EG09] angelehnt. Das Projekt wurde vom s-lab durchgeführt und wird im Laufe der Arbeit neben dem Beispiel der Teamgröße in Scrum, welche nicht mehr als 9 und nicht weniger als 3 Personen betragen darf, in dieser Arbeit mehrfach erwähnt werden. Im Projekt Quasi-Scrum wurde eine Kreditkalkulationssoftware entwickelt. Dafür wurde die eine angepasste Form der Agilen Methode Scrum eingesetzt. Dieses Beispiel wird um einige Komponenten aus der eigenen Erfahrung sowie aus einem anderen Praxisprojekt, welches in einem großen Telekommunikationsunternehmen durchgeführt wurde, erweitert. Die Dissertation schließt in Kapitel 8 mit einer Zusammenfassung der geleisteten Arbeit und gibt einen Ausblick über zukünftige Aufgaben.

Kapitel 2 Grundlagen und verwandte Arbeiten

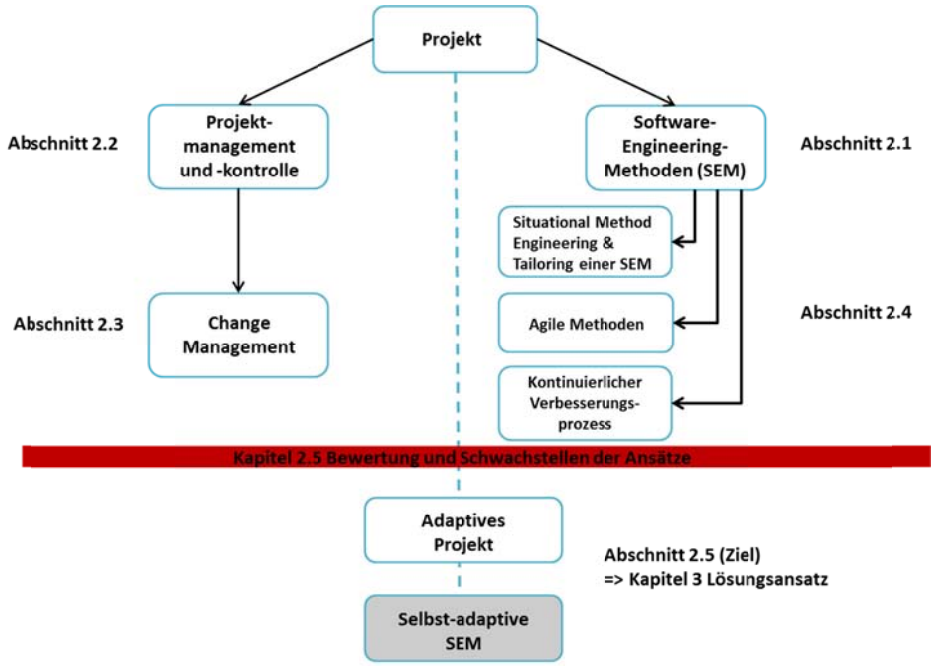

Abbildung 4 Aufbau Kapitel 2 und Übergang zu Kapitel 3

In diesem Kapitel werden beginnend mit der Beschreibung und Definition einer Software-Engineering-Methode die Grundlagen für die späteren Kapitel sowie verwandte Arbeiten beschrieben. Zunächst wird erläutert, was alles zu einem Software-Projekt und im Speziellen einer Software-Engineering-Methode gehört. Daran knüpfen die Modellierung und die Vorstellung verschiedener Arten von Software-Engineering-Methoden an. Im nächsten Abschnitt werden das Projektmanagement und dessen Inhalte erläutert. Insbesondere wird dabei auf die Projektkontrolle eingegangen, welche sich mit der Überwachung von Projekten und bei Bedarf regelnden Maßnahmen beschäftigt. Da sich diese Arbeit mit der Anpassung von Software-Engineering-Methoden beschäftigt, also der Änderung dieser Methoden, soll in Abschnitt 2.3 auf das Change Management eingegangen werden. Dies beschäftigt sich zwar mit Änderungen auf Unternehmensebene, aber sowohl die Änderungen auf dieser Ebene können die Software-Engineering-Methoden beeinflussen als auch die Änderung von Software-Engineering-Methoden können sich unter Umständen auf das Unternehmen auswirken. Im folgenden Abschnitt werden verwandte Ansätze und Arbeiten vorgestellt, welche sich mit der Anpassung von Software-Engineering-Methoden zu verschiedenen Zeitpunkten beschäftigen, zum Beispiel vor Beginn des Projektes. Das Kapitel wird mit einer Analyse der Schwachpunkte der Ansätze und Anforderungen an den eigenen Ansatz abgeschlossen.

2.1 Software-Engineering-Methoden

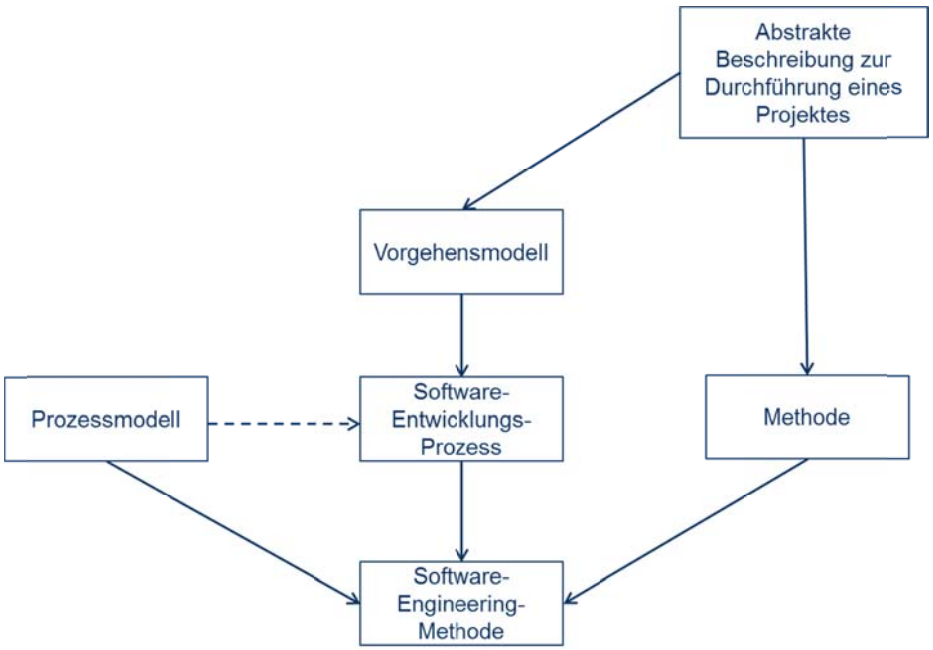

Abbildung 5 Begriffsentwicklung Software-Engineering-Methode (SEM)

Um Software heutzutage erfolgreich zu entwickeln, wird in einem sogenannten „Softwareprojekt“, kurz Projekt, eine konkrete Folge von Schritten ausgeführt, die ein konkretes Ergebnis haben. Diese Abfolge kann zunächst durch eine *abstrakte Beschreibung zur Durchführung des Projektes* erfolgen. Dabei kann diese abstrakte Beschreibung eine Reihenfolge der Schritte beschreiben, welche zur Durchführung des Projektes nötig sind. Alle Bestandteile, die zu einem Projekt und einer Software-Engineering-Methode gehören, werden in Abschnitt 2.1.1 erläutert.

Abstrakte Beschreibungen zur Durchführung eines Projektes sind zwar hilfreich, doch häufig in ihrer Ausführung nicht konkret genug, was in der Durchführung zu Missverständnissen führen kann. Im schlimmsten Fall führen diese Missverständnisse zum Misserfolg des Projektes. Die Vorgehensweise in einem Projekt kann konkret über ein *Vorgehensmodell* beschrieben werden. Es beschreibt auf abstrakte Weise, wie und welcher Reihenfolge die verschiedenen Aktivitäten durchgeführt werden und es werden die zu erzielenden Ergebnisse festgelegt [GIAK]. Das Vorgehensmodell beschreibt dabei ein optimales Vorgehen, welches typischerweise für jedes Softwareprojekt angepasst werden sollte, denn jedes Projekt ist anders.

Ein *Softwareentwicklungsprozess* beschreibt nach [GIAK, LL10] den Handlungsablauf zur Entwicklung einer Software, welcher durch das Vorgehensmodell geformt ist. Einem Softwareentwicklungsprozess liegt dabei ein

Prozessmodell zu Grunde, welches eine Menge von Tätigkeiten beschreibt, die in einer bestimmten Handlungsabfolge ausgeführt werden, damit ein Produkt entsteht oder weiterentwickelt wird [So07, GIAK]. Ludewig und Lichter gehen in ihrer Definition des Prozessmodells noch weiter: Für sie beinhaltet ein Prozessmodell nicht nur das Vorgehensmodell sondern zusätzlich Aussagen über die personelle Organisation, die Gliederung der Dokumentation sowie die Verantwortlichkeiten für Aktivitäten und Dokumente [LL10]. Da wie bereits beschrieben jedes Projekt anders ist, gibt es verschiedene Ausprägungen dieser Prozessmodelle, es können keine oder nur sehr schwer allgemeingültigen Aussagen getroffen werden.

Ein weiterer Begriff, der für die Beschreibung der Ausführung eines Projektes genutzt wird, ist der Begriff *Methode*. Eine Methode beinhaltet absolut alles, was für die Entwicklung gebraucht wird, das heißt beispielsweise sogenannte Artefakte, Rollen und ausgeführte Tätigkeiten, aber auch in welcher Reihenfolge und zu welchem Zeitpunkt was ausgeführt wird [HSR10].

Engels und Sauer [ES10] gehen in ihrer Beschreibung noch ein Stück weiter und führen den Begriff *Software-Engineering-Methode* ein. Dieser bringt den Softwareentwicklungsprozess bzw. das Prozessmodell und eine Methode zusammen. Eine Software- Engineering-Methode ist das vollständige Set an Elementen (z.B. Rollen, Artefakte und Tätigkeiten) welches benötigt wird, um ein Projekt in allen relevanten Aspekten zu beschreiben. Diese Beschreibung beinhaltet, ähnlich wie bei der Beschreibung des Prozessmodells nach [LL10], nicht nur ein Vorgehensmodell an sich und seine Aktivitäten, sondern zusätzlich alle Artefakte und Aktivitäten die durchgeführt werden müssen um gesetzte Ziele – die Meilensteine – zu erreichen. Zusätzlich beinhaltet eine Software-Engineering-Methode die Rollen, Werkzeuge und Techniken die gebraucht werden, ebenso wie alle Abhängigkeiten zwischen den Konzepten. Dies kann ebenfalls die Organisationsstrukturen, die Vorgaben für das Projektmanagement, sowie die Qualitätssicherung, die Dokumentation und die Konfigurationsverwaltung mit beinhalten, also alles was zu einem Software-Projekt mit dazugehört.

Aufgrund dieser ausführlichen Definition soll der Begriff Software-Engineering-Methode (SEM), in der vorliegenden Arbeit verwendet werden.

2.1.1 Bestandteile eines Projektes und einer Software-Engineering-Methode

Software wird typischerweise in *Unternehmen* entwickelt. Je nach Größe und Spezialisierung des Unternehmens kann es wie in Abbildung 6 zu sehen, ein oder mehrere Projekte zur Entwicklung oder Verbesserung von Software besitzen. Um die Software erfolgreich zu entwickeln, wird das Projekt mit Hilfe einer Software-Engineering-Methode durchgeführt. Doch

jedes Projekt besitzt nicht nur eine Software-Engineering-Methode, sondern hat noch weitere Merkmale, u.a. nach [LL10]:

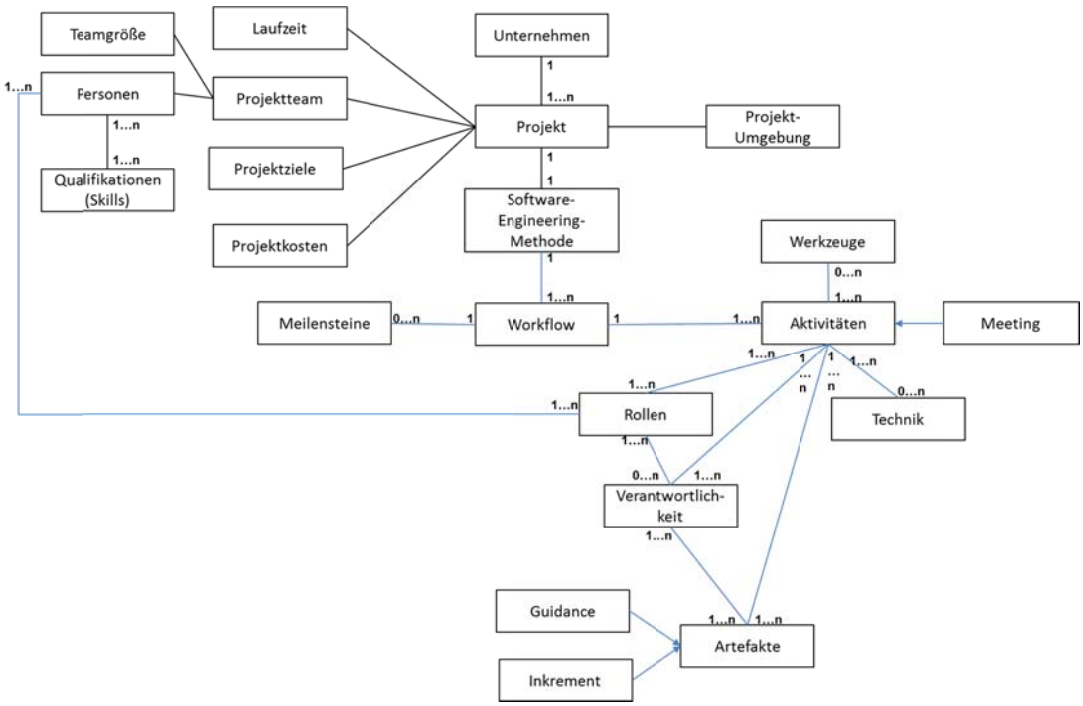

Abbildung 6 Begriffe basierend auf [BK13], [OMG08], [LL10]

- Jedes Projekt hat eine *Laufzeit*, das heißt es läuft über einen bestimmten, teilweise begrenzten, Zeitraum mit einem Start- und einem oder mehreren End- bzw. Zwischenendpunkten.
- Die Personen, welche das Projekt durchführen, sind das *Projektteam*, kurz das Team. Dabei kann dieses Team noch in weitere Teams unterteilt werden, z.B. Entwickler-Team, Test-Team etc. Jedes Team besitzt dabei eine (optimale) *Teamgröße*, welche mit Hilfe der Software-Engineering-Methode beschrieben wird.
- Ein Team besteht aus natürlichen *Personen*, welche bestimmte *Qualifikationen (Skills)* besitzen müssen, um die verschiedenen Aktivitäten zur Durchführung des Projektes meistern zu können.
- Die Personen im Team nehmen bestimmte *Rollen* ein, beispielsweise Projektleiter, Entwickler, Software-Architekt, Tester u.a. Dabei kann ein Teammitglied eine oder mehrere Rollen innehaben.
- Ein Projekt hat verschiedene *Projektziele*. Diese werden zu Beginn des Projektes zusammen mit allen Beteiligten definiert. Projektziele bewegen sich meistens im Rahmen der Qualität des Endproduktes, welcher Leistungsumfang für die Erstellung des Endproduktes gebraucht wird, aber auch welche Zeit (s. Laufzeit) das Projekt zur Fer-

tigstellung braucht und was das Projekt kostet (s. Projektkosten). Nähere Erläuterungen dazu stehen in Abschnitt 2.2.1. Projektmanagement. Ein Projekt ist erfolgreich, wenn die (Teil-) Ziele erreicht sind.

- Ein Projekt hat ein konkretes Budget, welches aus den errechneten *Projektkosten* resultiert. Die Projektkosten errechnen sich u.a. aus den Aufwänden, um die Software zu erstellen, aus der Laufzeit usw. Ein Ziel im Projekt ist es, die Projektkosten einzuhalten oder sie, wenn möglich, zu reduzieren und zu optimieren.
- Jedes Projekt befindet sich in einer *Projektumgebung* und hat ein eigenes *Projektumfeld*. Dazu zählen unter anderem die Rahmenbedingungen, die nicht direkt etwas mit dem Projekt selbst zu tun haben. Diese Rahmenbedingungen beinhalten beispielsweise das Unternehmen selbst, wie die Größe (z.B. Mittelstand oder Großunternehmen), wie das Unternehmen strukturiert und organisiert (z.B. flache Hierarchien) ist oder auch die Unternehmenspolitik. Hierzu zählt auch das kulturelle Umfeld, sei es die Unternehmenskultur oder die Kultur der einzelnen Personen im Zusammenspiel. Gibt es Änderungen im Projektumfeld, so können diese das durchzuführende Projekt und somit die Software-Engineering-Methode beeinflussen.

Die verschiedenen Bestandteile eines Projektes können Auswirkung auf die gewählte Software-Engineering-Methode haben. Im Folgenden sollen die Bestandteile einer Software-Engineering-Methode beschrieben werden, u.a. nach [BK13], [LL10] und [OMG08]:

- Eine Software-Engineering-Methode wird durch eine oder mehrere voneinander abhängigen Reihenfolge von zu erledigenden Schritten beschrieben. Diese Reihenfolge wird *Workflow* genannt. Dieser Workflow beschreibt, wie die einzelnen Elemente in der Software-Engineering-Methode miteinander zusammenhängen und zu welchem Zeitpunkt welches Element von wem ausgeführt wird.
- Einzelne Abschnitte im Workflow können sogenannte *Meilensteine* beinhalten. Meilensteine bezeichnen einen Zeitpunkt oder Abschluss im Workflow, wo beispielsweise Elemente fertiggestellt sein müssen. Diese Meilensteine und deren Inhalt sowie ihr Zeitpunkt werden im Vorfeld anhand von Kriterien definiert und festgelegt. Der Abschnitt des Workflows ist erfolgreich beendet, wenn die im Vorfeld definierten Kriterien des Meilensteins zum entsprechenden Zeitpunkt erfüllt sind.
- Ein Workflow setzt sich aus einzelnen *Aktivitäten* zusammen. Eine Aktivität bezeichnet eine bestimmte Aufgabe oder Tätigkeit, welche

im Rahmen der Software-Engineering-Methode durchgeführt wird. Viele Aktivitäten werden durchgeführt, um *Artefakte* zu entwickeln.

- *Artefakte* in einer Software-Engineering-Methode haben verschiedene Ausprägungen. Ein wichtiges Artefakt sind die sogenannten *Inkremente* der zu erstellenden Software. Je nach Software-Engineering-Methode sind dies die vollständige Software am Ende des Projektes oder einzelne Bestandteile auf dem Weg zum Endprodukt. Des Weiteren sind Artefakte einzelne Dokumente o.ä., welche für den Erfolg der Software-Engineering-Methode wichtig sind. Dies können Anforderungsspezifikationen, (Test-)Reporte oder andere Ergebnisse sein. Ein weiteres Artefakt sind sogenannte *Guidances*, Anleitungen oder Hilfestellungen für die Durchführung der einzelnen Aktivitäten.
- Die einzelnen Aktivitäten benötigten Artefakte sowohl als Input, damit die Aktivität durchgeführt werde kann, als auch dass sie Artefakte als Output für weitere Aktivitäten generieren. Der Zusammenhang zwischen Artefakten und Aktivitäten wird ebenfalls vom Workflow festgelegt.
- Die einzelnen Aktivitäten werden wie in Abbildung 6 zu sehen von einzelnen oder mehreren *Rollen* durchgeführt. Eine Rolle wird durch eine konkrete Person im Team ausgefüllt, welche die entsprechenden Qualifikationen für diese Rolle besitzt.
- Eine Rolle kann *Verantwortung* für ein Artefakt oder eine Aktivität beinhalten, muss dies aber nicht. Wichtig ist, dass jede Aktivität und jedes Artefakt mindestens einen Verantwortlichen besitzen.
- Um bestimmte Aktivitäten zu erfüllen oder um ein bestimmtes Artefakt zu erzeugen, können verschiedene *Techniken* und auch *Werkzeuge* (engl. Tools) eingesetzt werden. Diese unterstützen die durchführenden Personen, um die Aktivität erfolgreich durchzuführen.
- Damit ein Projekt funktioniert, ist es wichtig, dass eine gesunde Kommunikationskultur gepflegt wird. Das „Hauptinstrument“ dafür ist eine Spezialform einer Aktivität, das *Meeting*. Meetings können im unterschiedlichen Maße in Projekten eingesetzt werden. Einerseits können sie als kurze Informations-Meetings dienen, andererseits aber auch zur Entscheidungsfällung. Dabei muss der „richtige“ Mittelweg gefunden werden, damit der zeitliche Aufwand nicht für ergebnislose Meetings überhandnimmt und dass ausreichend Zeit für die eigentliche Durchführung bleibt.

Die verschiedenen Bestandteile einer Software-Engineering-Methode sind wichtig zu wissen, da anhand dieser Bestandteile eine Software-Engineering-Methode im Endeffekt angepasst wird, indem beispielsweise

eine Aktivität hinzugefügt oder gelöscht oder der Workflow geändert wird. Um Software-Engineering-Methoden auszuführen und später entsprechend anpassen zu können, sollten sie anhand eines Modells dargestellt werden. Durch ein Modell wird ersichtlich, wie die einzelnen Komponenten zusammenhängen und an welcher Stelle etwas entsprechend angepasst werden kann.

2.1.2 Modellierung von Software-Engineering-Methoden

Um eine Software-Engineering-Methode zu modellieren, werden die Bestandteile entsprechend definiert und alle Rollen, ihre Verantwortlichkeiten, Artefakte, Aktivitäten usw. miteinander verknüpft. Ebenso wird über die Abhängigkeiten die Reihenfolge der Aktivitäten festgelegt und dargestellt. Zusätzlich wird dadurch visualisiert, welche Artefakte als In- und Output für eine Aktivität nötig sind, welche Rolle daran beteiligt ist oder die Verantwortlichkeit dafür besitzt. Zusätzlich können Regeln definiert werden, welche bei der Ausführung einer Software-Engineering-Methode einzuhalten sind oder den einzelnen Tätigkeiten werden Techniken zugeordnet. Ein Modell dient nicht nur der Visualisierung zur Verständlichkeit, sondern ein automatisches Ausführen hilft den Personen bei der Umsetzung und dem Entwickeln eines Produktes.

Ein Modell ist ebenfalls wichtig, damit die Software-Engineering-Methode angepasst werden kann. Dadurch wird ersichtlich, an welcher Stelle sich ein Element der SEM befindet und welche Abhängigkeiten bestehen. Soll ein solches Element später ausgetauscht oder gelöscht werden, müssen diese Abhängigkeiten ebenfalls mit in Betracht gezogen werden, um die Konsistenz und Gesamtfunktionalität der Methode weiterhin zu gewährleisten. Wird später ein Element hinzugefügt, kann im Modell geprüft werden, an welcher Stelle dies eingefügt werden soll.

Für die Modellierung von Software-Engineering-Methoden gibt es bereits verschiedene Modelle und Methoden. Die bekannteste und am meisten eingesetzte ist das Software and Systems Process Engineering Meta-Modell, kurz SPEM, der OMG [OMG08]. Dieses ist ein Meta-Modell und ebenfalls ein UML-Profil zur Spezifikation von Software-Engineering-Methoden. Das Modell der konkreten Software-Engineering-Methode, welches später im Projekt genutzt wird, befindet sich auf der sogenannten „Typ-Ebene“. Auf der darunterliegenden „Instanz-Ebene“ befinden sich die konkreten und instanziierten Werte aus dem eigentlichen Projekt, beispielsweise welche konkrete Person eine Rolle ausfüllt.

SPEM unterscheidet in seinem Modell zwischen den Methodeninhalten wie Artefakte, Aktivitäten bzw. Aufgaben, Rollen usw. und den Prozessen. Dabei können verschiedene Prozesse über ein Set von Methodeninhalten spezi-

fiziert werden. Über Plugins ist es möglich, SPEM-Spezifikationen zu variieren, das heißt es wird eine Basismethodik für einen Prozess oder Teil-Prozess definiert und anschließend projektspezifisch angepasst.

Ein weiteres bekanntes Modell ist die ISO 24744 [ISO07], welches auf einem anderen Prinzip als SPEM beruht. Dieses Modell wird zusätzlich im Buch von Cesar Gonzales-Perez und Brian Henderson-Sellers[GPHS08] beschrieben, welche an der Entwicklung der ISO 24744 mitgewirkt haben. Dieses Model integriert Prozesse, Produkte und Rollen („Producer“) in einem einzigen Paket einer Methodenspezifikation.

Bei Sauer und Engels [ES10] wird mit MetaME ebenfalls eine Meta-Methode zur Modellierung von Software-Engineering-Methoden vorgestellt. Diese baut auf einem Meta-Model des Software Engineering auf und kombiniert Ideen des Meta-Modelling mit denen des Method-Engineering. MetaME beschreibt weniger, wie eine Methode angepasst werden könnte, sondern mehr, wie eine Methode entsprechend anhand eines Meta-Modells erstellt werden kann.

Für den späteren Ansatz dieser Arbeit ist es egal, ob das Modell mit Hilfe von SPEM, ISO 24744, MetaMe, einer Variante oder einer Vereinfachung davon erstellt wurde. Wichtig ist zum einen, dass alle Artefakte, Rollen, Aktivitäten, Techniken und der Workflow sowie die Abhängigkeiten untereinander modelliert sind. Zum anderen sollte konstant dieselbe Methode zur Modellierung bei der Nutzung des Ansatzes verwendet werden.

2.1.3 Arten von Software-Engineering-Methoden

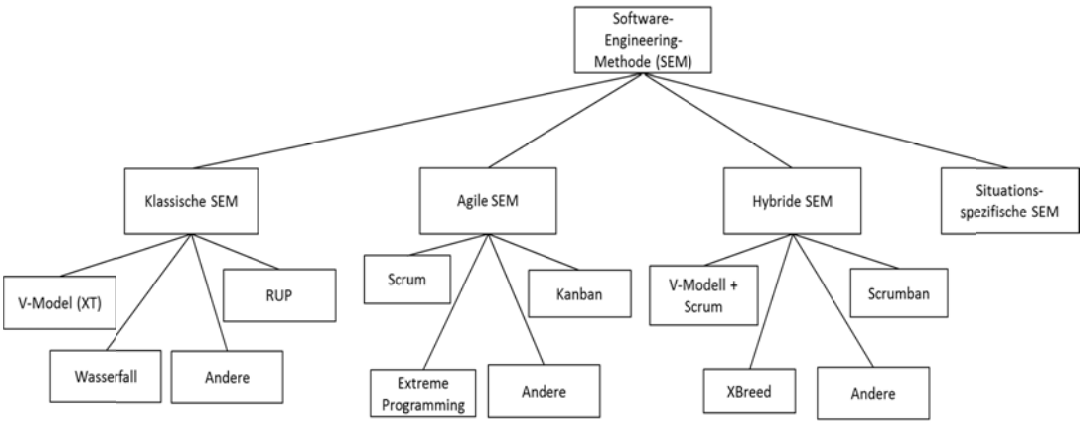

Abbildung 7 Verschiedene Arten von Software-Engineering-Methoden

Projekte unterscheiden sich gerade in den Feinheiten von anderen Projekten. Wenn ein bestimmtes Vorgehen für das eine Projekt erfolgreich ist, kann es beim nächsten schon zum Misserfolg führen. Es gibt nicht **die** Software-

Engineering-Methode, welche für alles und jedes Projekt passt, eine *one-size-fits-all*“ Methode (vgl. auch [HS06], [Co02]).

Von daher gibt es die verschiedensten Arten von Software-Engineering-Methoden, um ein Projekt zum Erfolg zu führen. In Abbildung 7 sind diese mit einigen Vertretern exemplarisch dargestellt. Dabei wird hauptsächlich zwischen den „klassischen“ Software-Engineering-Methoden und den sogenannten „Agilen Software-Engineering-Methoden“, auch Agile Methoden genannt, unterschieden. Zusätzlich zu diesen beiden Gruppen gibt es „hybride“ Software-Engineering-Methoden, welche sich sowohl aus klassischen und Agilen SEM zusammensetzen können als auch beispielsweise rein aus verschiedenen Agilen Methoden. Die letzte Gruppe bilden die situationsspezifischen Software-Engineering-Methoden. Diese Methoden zeichnen sich dadurch aus, dass sie aus verschiedenen Bausteinen, den „Method Elements“, eine Software-Engineering-Methode speziell für ein Projektsituation zusammenbauen. Die einzelnen Arten sollen im Folgenden näher erläutert werden.

2.1.3.1 Klassische Software-Engineering-Methoden

Die traditionellen oder klassischen Software-Engineering-Methoden sind im Gegensatz zu den Agilen Methoden besser strukturiert (formalisiert). Sie besitzen mehr Vorgaben und Dokumentation und sind deshalb *schwergewichtiger*. Hingegen werden Agile Methoden gern als *leichtgewichtig* bezeichnet. Ferner sind in traditionellen Methoden insbesondere die Anforderungen stabiler und ändern sich im Laufe der Zeit eher weniger. Bekannte Vertreter sind hier das V-Modell [DW99] bzw. V-Modell XT [RB07], das Wasserfallmodell [Bo81] oder der Rational Unified Process (RUP) [Kr03, Kr98], wobei der RUP ein besonderer Fall ist. Er kann sowohl schwergewichtig, als auch leichtgewichtig implementiert werden. Eine agile Variante ist beispielsweise der Agile Unified Process (AUP) von Scott Ambler [Am06].

2.1.3.2 Agile Methoden

In den letzten Jahren hat die Popularität von sogenannten „leichtgewichtigen“ Vorgehensmodellen zugenommen. Erstmals wurde ein solches Modell von Kent Beck mit Extreme Programming (XP) vorgestellt [Be00]. Heute sind diese Modelle als Agile Software-Engineering-Methoden, kurz Agile Methoden bekannt. Scrum [BS02, Gl08], Feature Driven Development (FDD) [DCL99] und Crystal [Co02] sind nur einige weitere Beispiele, die im Laufe der Zeit entstanden sind.

Die Bezeichnung „agil“ (lat. agilis: flink; beweglich) wurde auf einer Konferenz 2001 in Utah ausgewählt, wo ebenfalls das „Agile Manifest“ [AM01] entstand. Dies stellt das Fundament für die agile Softwareentwicklung dar.

Eine Hauptcharakteristik Agiler Methoden ist die Vorgehensweise in iterativen Zyklen. Das Ziel eines jeden iterativen Zyklus ist es, sowohl funktionierende Software als auch neue Funktionalitäten für eine (bestehende) Software zu liefern. In den verschiedenen Vorgehensmodellen gibt es bestimmte Rollen, die fest verteilt sind. Dabei liegt der Hauptfokus auf dem Team, welches am besten interdisziplinär zusammengesetzt ist. Agile Methoden sind dabei kommunikationsintensiv und legen Wert auf den Einzelnen selbst sowie auf Selbstverantwortung. Ein Team soll sich selbst organisieren und alles kommunizieren, was zu einer Verminderung von aufwändiger Dokumentation führen soll.

Ziel der Prozesse ist es, durch kurze iterative Zyklen schnellstmöglich Software mit neuer Funktionalität zu liefern [DNZ07]. Dabei ist es schwierig, wenn nicht unmöglich, in Projekten alle Anforderungen vorher festzulegen. Mit den Agilen Methoden soll die Möglichkeit gegeben werden, zeitnah auf neue Anforderungen und Änderungen reagieren zu können.

2.1.3.3 Hybride Software-Engineering-Methoden

Unter hybriden Software-Engineering-Methoden versteht man die Kombination von zwei Methoden miteinander. Die bekannteste Kombination im agilen Bereich ist die Kombination von Scrum und Extreme Programming. Eine typische Kombination ist die Verwendung von Scrum als Hülle mit der Anreicherung von XP-Techniken, beispielsweise dem Pair Programming, Test Driven Development usw. Ein bekannter Vertreter, welcher beide Agile Methoden miteinander verknüpft ist xp@Scrum [MS02, Vr03]. Ein neuerer Vertreter unter den agilen Hybriden ist „Scrumban“ welches 2008 von Ladas [LA09] vorgestellt wurde und Scrum mit Kanban kombiniert.

Der Agile Unified Process ist weniger ein Hybrid als die Kombination von RUP mit den agilen Prinzipien. Es ist eine agile Ausprägung des RUP. Dies ist mit ein Grund, warum teilweise darüber diskutiert wird, ob RUP nun zu den agilen oder zu den klassischen Methoden gehört.

Seit einiger Zeit gibt es zusätzlich zu den agil-hybriden Methoden, die ersten Kombinationen von klassischen mit agilen Methoden. Die bekannteste Kombination ist hier die des V-Modells mit Scrum. Da das V-Modell häufig im Dienstleistungssektor eingesetzt wird, wollten agile Anbieter möglichst ebenfalls an Ausschreibungen für Projekte teilnehmen, obwohl das V-Modell Voraussetzung war. Die Idee war nun, das V-Modell als äußere Hülle zu nutzen, mit den entsprechenden Meilensteinen usw. aber die innere Entwicklung wurde in Form von Scrum genauer in Sprints durchgeführt.

Diese Beispiele zeigen gut, dass es möglich ist, Software-Engineering-Methoden oder Methodenelemente miteinander zu kombinieren und somit SEMs für den Zweck des Projektes anzupassen. Wie es möglich ist, eine

Software-Engineering-Methode vollständig aus Methodenelementen zusammenzusetzen, wird in der Unterdisziplin des Method Engineering im Situational Method Engineering behandelt.

2.1.3.4 Situationsspezifische Software-Engineering-Methoden

Method Engineering (ME) beschäftigt sich mit dem Design und der Konstruktion von Methoden, insbesondere für „information systems development“ [HS06]. Sie wählen wiederverwendbare Methoden-Komponenten, auch Method Fragments oder Method Chunks genannt, aus einer Methoden-Basis (engl. Method Base) aus und setzen sie zu einer Methode zusammen. Nützlich und aussagekräftig ist hier die Definition von Sjaak Brinkkemper [Br96, S. 276]:*“Method engineering is the engineering discipline to design, construct and adapt methods, techniques and tools for the development of information systems“*.

Nach Sjaak Brinkkemper ist Method Engineering also eine (ingenieur-) wissenschaftliche Disziplin für das Designen, Konstruieren und Anpassen von Methoden, Techniken und Tools für die Entwicklung vor allem von Informationssystemen. Dabei wird der Begriff Methode im Sinne von Vorgehensmodell oder Softwareentwicklungs-Prozess genutzt. Nach Brinkkemper ist eine Methode ein Ansatz, um ein Software- oder System-Entwicklungsprojekt durchzuführen. Diese Methoden basieren auf einem Ansatz, welcher aus verschiedenen Regeln und möglichen Richtungen für die Entwicklung besteht. Die Methoden sind dabei systematisch strukturiert in ihren Entwicklungsaktivitäten und den entsprechenden Artefakten, die dabei entstehen. Brinkkemper spricht in seiner Definition allerdings nur von Informations-Systemen.

Diese Definition wurde von [HSR10] insofern angepasst, dass sie „information systems“ durch „system development“ zur Verallgemeinerung ausgetauscht haben.

Wenn die Methode nun genau auf ein Projekt oder eine Situation abgestimmt ist und sie wird für diesen Kontext aus den verschiedenen Methoden-Komponenten zusammengebaut, dann wird von *Situational Method Engineering* (SME) [RR01,HS06] gesprochen. Neue Methoden werden konstruiert, indem Methoden-Elemente von verschiedenen Methoden ausgewählt werden, welche für die Situation am besten geeignet sind.

[HSR10] beschreiben in ihrem State-of-the-Art Review des SME, dass die optimistischste Art, eine situationsspezifische Methode zu erstellen, die Methoden-Konstruktion unterstützt durch Tailoring/Customization einer Methode sei.

Diverse Ansätze und Arbeiten haben sich mit Situational Method Engineering beschäftigt; viele von ihnen werden gut im State-of-the-Art Review von Henderson-Sellers und Ralyté [HSR10] zusammengefasst. Zusätzlich wird abschließend ein Ausblick gegeben, welche Forschungsrichtungen auf diesem Gebiet in den nächsten Jahren interessant sein könnten und wo es noch Probleme gibt. Beispiele dafür sind, wie die Qualität von einer solchen Methode beurteilt werden kann oder wie es möglich ist sicherzustellen, dass die Konfiguration der ausgewählten Methoden-Elemente vollständig, korrekt und konsistent ist.

2.2 Projektmanagement und -kontrolle

Ein Projekt ist im Allgemeinen eine sehr komplexe Angelegenheit und ein Unternehmen ist von den Erfolgen seiner Projekte abhängig. Damit Projekte erfolgreich sind, müssen sie zum einen gut verwaltet und zum anderen mit entsprechenden Mechanismen kontrolliert und überwacht werden. Das Projektmanagement ist für die Verwaltung der Projekte zuständig, ein wichtiges Element ist dabei die Projektkontrolle bzw. das Projektcontrolling. In den folgenden Abschnitten soll das Projektmanagement und insbesondere die Projektkontrolle vorgestellt werden. Diese ist nicht nur mit der Überwachung eines Projektes betraut, sondern greift bei Bedarf aktiv ein und führt soweit möglich Änderungen durch.

2.2.1 Projektmanagement

Projekte sind komplex in der Durchführung und können von langer Dauer sein. Die Verantwortlichen müssen zu jeder Zeit einen umfassenden Überblick über das Projekt haben und sich nach [Fi10] bei der Durchführung von Projekten in regelmäßigen Abständen verschiedenen Fragen stellen, wie beispielsweise:

- Wie viele Mitarbeiter werden für das Projekt benötigt, welche Qualifikationen müssen sie dafür besitzen?
- Wie werden Projektleiter und die Mitarbeiter für das Projekt ausgewählt?
- Sind genug Ressourcen für das Projekt; was ist, wenn ein Engpass im Projekt auftritt?
- Wie teuer wird das Projekt, welche Kosten können anfallen?
- Wie lang soll das Projekt dauern, was ist der konkrete Zeitraum für die Durchführung?
- Was passiert bei Terminverzögerungen im Projekt, wie wirken sich diese aus?

Die Grundlage, um all diese Fragen zu beantworten, der Komplexität Herr zu werden und zu jeder Zeit den Überblick zu behalten, bildet das Projektmanagement. Nach der DIN 69901 beschreibt Projektmanagement die „*Ge-*

samtheit von Führungsaufgaben, -organisation, -Techniken und -mittel für die Abwicklung eines Projektes.“

Genauer umfasst Projektmanagement somit alle Elemente eines Projektes, die wichtigsten sind dabei nach der klassischen Managementlehre u.a. die Planung, die Kontrolle, die Organisation und die Personalführung [Sc13, Fi10]. Zusätzlich beschreibt das Projektmanagement, wer alles am Projekt beteiligt ist und wie mit welchen Instrumenten, Techniken und Vorgehensweisen ein Projekt durchgeführt wird [Fi10].

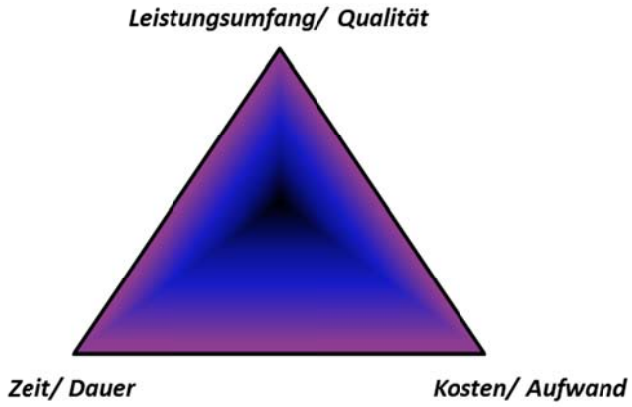

Abbildung 8 Das magische Dreieck im Projektmanagement nach [Sn05]

Im Projektmanagement dreht es sich typischerweise um eine bestimmte Klasse von Zielen, welche das „Magische Dreieck“ in Abbildung 8 nach Sneed [Sn05] verdeutlicht. Es stellt dabei ebenfalls die Abhängigkeiten zwischen Dauer des Projektes, Kosten und Aufwand des Projektes, sowie die Qualität und dem durch das Projekt gelieferte Leistungsumfang dar. Es wird ein bestimmter Leistungsumfang, sogenannte Sachziele definiert, welche die Qualität und gewünschte Leistung widerspiegeln. Es werden Termine als Ziel festgelegt, welche das Projektende, aber auch Liefertermine zwischendurch beschreiben. Ferner wird der Aufwand für das Projekt geschätzt und in welchem Kostenrahmen sich das Projekt bewegen soll [Fi10, BK13]. Alle drei Aspekte sind voneinander abhängig. Denn werden ein oder zwei im Laufe des Projektes angepasst und wird versucht, sie zu optimieren, geht dies immer zu Lasten des dritten Aspektes. Von daher muss zu Beginn festgelegt werden, wo die Prioritäten im Projekt liegen, welche Ziele am wichtigsten sind und zu Lasten welchen Aspektes eine Anpassung erfolgen darf.

Das Projektmanagement besteht aus verschiedenen Elementen, die relevantesten sind in Abbildung 9 zu sehen.

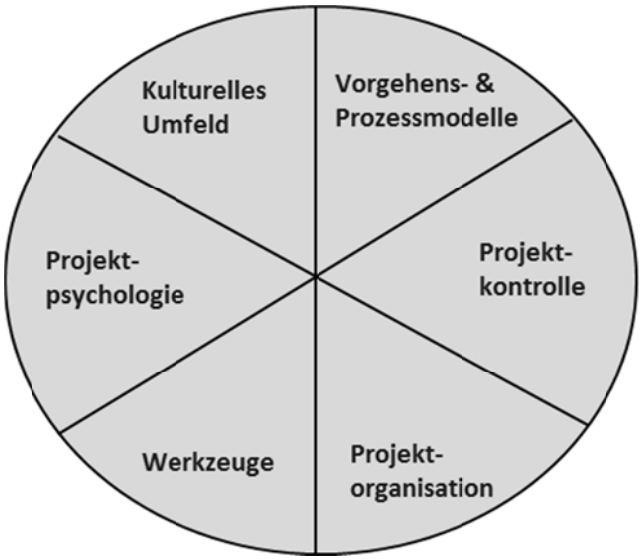

Abbildung 9 Elemente des Projektmanagements basierend auf [Fi110, AE13]

Basierend auf [AE13, Fi10] sind dies:

Vorgehens- und Prozessmodelle: Projekte werden typischerweise anhand eines definierten Prozesses durchgeführt. Vorgehensmodelle und im Softwarebereich Software-Engineering-Methoden dienen dabei als Vorlage, welche dem Unternehmen helfen, das Projekt optimal abzuwickeln. Dabei müssen diese Modelle für die Projektsituation entsprechend angepasst werden.

Projektkontrolle: Die Projektkontrolle bietet Mechanismen, um alle Phasen des Projektes im Blick zu behalten und bei Änderungen gegebenenfalls gegensteuern zu können.

Projektorganisation: Am Projekt und auch am Projektmanagement selbst sind mehrere Personen beteiligt. Dies können verschiedene Rollen, Teams, aber auch Gremien oder andere Bereiche im Unternehmen sein. All diese Personen müssen verschiedene Aufgaben erledigen und haben entsprechende Verantwortlichkeiten, die definiert und zugewiesen werden müssen.

Werkzeuge: Softwaregestützte Werkzeuge unterstützen komplexe Projekte, da diese ab einer bestimmten Größe nur noch schwer von einzelnen Personen bewältigt werden können. Mit Hilfe dieser Werkzeuge werden Projekte übersichtlicher und können einfacher abgewickelt werden.

Projektpsychologie: Projekte und Projektmanagement werden von Personen durchgeführt, die sich je nach Individuum und Situation anders verhalten. Beispielsweise sind die Kommunikation, die Akzeptanz bestimmter Vorgehensweisen und Werkzeuge, aber auch ein Team gekonnt zu motivieren, von enormer Wichtigkeit.

Kulturelles Umfeld: Ebenso wie die Psychologie in einem Projekt ist auch das kulturelle Umfeld sowohl vom Unternehmen als auch von jeder einzelnen Person sehr wichtig zu erachten. Sie beeinflussen das Handeln und Denken einer Person.

Auch wenn alle Elemente für das Projektmanagement wichtig sind, ist für diese Arbeit insbesondere die Projektkontrolle wichtig, da diese sich mit der Überwachung und gegebenenfalls mit Änderungen im Projekt beschäftigt. Sie bezieht ebenso das Element der Modelle als auch die der Organisation mit ein. Gerade die letzten beiden Elemente werden in dieser Arbeit nicht betrachtet. Im folgenden Abschnitt soll von daher die Projektkontrolle nun ausführlicher beschrieben werden.

2.2.2 Projektkontrolle

Die Projektkontrolle, auch Projektcontrolling oder kurz Controlling genannt, unterstützt das Projektmanagement hauptsächlich bei der Planung, Überwachung und Steuerung des Projektes. Wichtig ist dabei auf der Basis von [F10], [Sc13] und [BK12], dass die Rahmenbedingungen eingehalten und die definierten Projektziele erreicht werden. Es muss somit darauf geachtet werden, dass geregelt ist, welche Pläne basierend auf den Zielen zu erstellen sind und wie diese kontrolliert werden. Zusätzlich ist zu beachten, wer wofür verantwortlich ist, wie Termine und Kosten kontrolliert werden und ob genug Ressourcen vorhanden sind usw.

Der Grundprozess im Controlling besteht nach [GA02] aus den vier verschiedenen Schritten Zielfindung, Planung, Überwachung und Steuerung, die in Abbildung 10 zu sehen sind.
Im ersten Schritt werden die Ziele definiert, welche durch das Projekt erreich werden sollen. Im zweiten Schritt wird geplant, wie diese Ziele im Projekt erreicht werden können. Im Projekt wird laufend überwacht, ob die Pläne eingehalten werden. Bei Änderungen wird wenn nötig dadurch gegengesteuert, dass die Pläne angepasst werden.

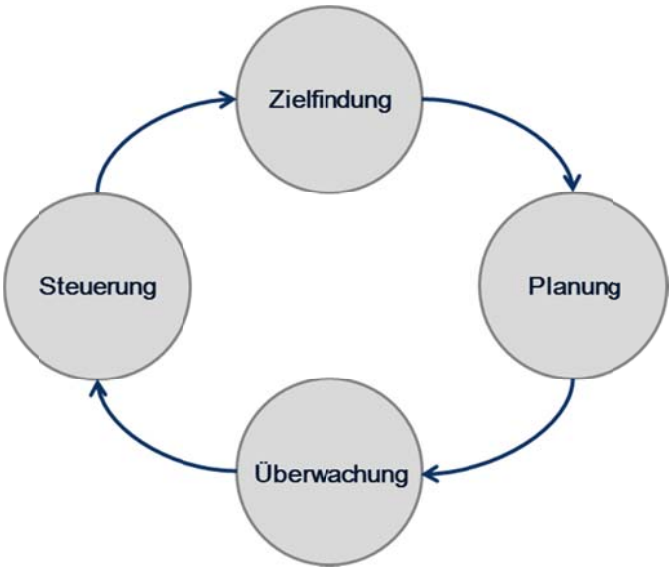

Abbildung 10 Allgemeiner Controlling-Ablauf nach [GA02]

In der DIN 69901 wird ein solcher Regelkreis als *„Sicherung des Erreichens der Projektziele durch:*
Soll-Ist-Vergleich, Feststellung der Abweichungen, Bewerten der Konsequenzen und Vorschlagen von Korrekturmaßnahmen, Mitwirkung bei der Maßnahmenplanung und Kontrolle der Durchführung“ beschrieben.

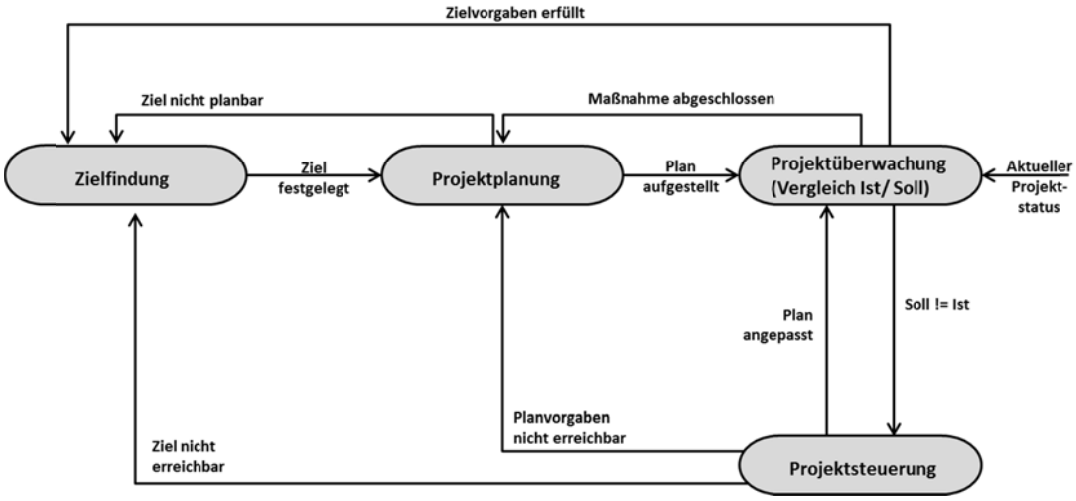

Abbildung 11 Der Regelkreis des Controlling basierend auf [GA02]

Zusammengefasst ergeben sich basierend auf der Definition und [GA02] der folgende Ablauf, welcher in Abbildung 11 zu sehen ist:

Nachdem die Ziele festgelegt sind, wird ein Plan zur Erreichung der Ziele aufgestellt. Während der Überwachung werden regelmäßig Statusmeldungen aufgenommen und der Soll-Zustand mit dem Ist-Zustand verglichen. Werden Abweichungen festgestellt, werden diese an die Projektsteuerung

übergeben, wo mögliche Konsequenzen bewertet und Korrekturmaßnahmen geplant werden. Der Plan wird wenn möglich angepasst und die Überwachung wird weiter durchgeführt. Kann entweder der Plan oder können sogar die Ziele nicht erfüllt werden, müssen diese jeweils entsprechend angepasst werden.

Um diesen Regelkreis entsprechend durchführen zu können, besteht ein Controlling-System neben dem Grundprozess aus weiteren Komponenten, welche beruhend auf [GA02] in Abbildung 12 dargestellt sind.

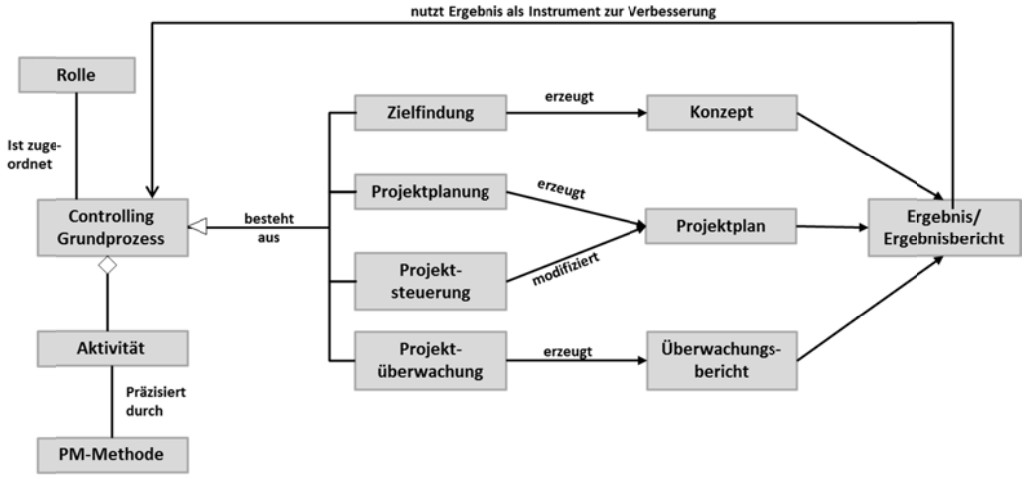

Abbildung 12 Komponenten Controlling-System beruhend auf [GA02]

Der Grundprozess wird von Personen durchgeführt, welche entsprechende Rollen ausfüllen, zum Beispiel die von einem Projektmanager. Der Grundprozess setzt sich aus verschiedenen Aktivitäten zusammen, denen eine Vorgehensweise zugrunde liegt. Das Vorgehen wird mit Hilfe von Projektmanagement-Methoden präzisiert.

Der bereits beschriebene Grundprozess erzeugt verschiedene Artefakte, die Zielfindung erzeugt zunächst das Konzept, in dem die Projektziele festgehalten sind. In der Projektplanung wird anschließend ein Projektplan erzeugt, welcher bei Bedarf von der Projektsteuerung modifiziert wird. Bei der Projektüberwachung wird ein Überwachungsbericht erzeugt. Diese drei Artefakte werden zu einem Ergebnisbericht zusammengefasst, welcher zur Verbesserung des Grundprozesses herangezogen wird. Durch diese Mechanismen wird der Prozess immer weiter optimiert.

Wichtig ist es an dieser Stelle zu erwähnen, dass die Erreichung der Projektziele bedeutet, dass es sich hier um Ziele im Rahmen von Umfang des Projektes, den Projektkosten, den Terminen, der Qualität und auch der Ressourcen handelt und es weniger um eine Software-Engineering-Methode geht.

Beispielsweise geht es bei dem Ziel der Aufrechthaltung der Rahmenbedingungen hauptsächlich darum, dass ein Projektteam arbeitsfähig bleibt, ohne überbelastet zu werden [BK13]. Der Fortschritt des Projektes wird anhand der Einhaltung der Kosten, der Einhaltung von Terminen sowie der Erfüllung der Qualitätsvorgaben kontrolliert. Kontrollmechanismen sind dafür u.a. regelmäßige Fortschrittsberichte, Projektbesprechungen oder die Überprüfung der Ergebnisse in der Qualitätskontrolle. Die Kontrolle findet somit auf einer anderen Ebene als einer Überwachung der eigentlichen Software-Engineering-Methode statt. Ferner sind die genannten Überwachungsmechanismen zwar wichtig, finden aber in großen Abständen statt und stützen sich, zumindest teilweise, auf Berichte.

Nach [BK13] ist es zwar wichtig, Kennzahlen für die Überwachung zu erfassen, wie beispielsweise den erbrachten Leistungsumfang, welche Ressourcen eingesetzt worden sind und der Vergleich des Ist- und Soll-Zustandes der Pläne. Aber es ist schwierig, dafür entsprechende Messdaten zu erfassen. Zwar können mit modernen Werkzeugen Arbeitsdaten und Erstellung von Berichten bezüglich der Software wie Build-, Fehler- und Test-Statistiken unterstützt werden. Doch die Festlegung dieser Messwerte, genannt Metriken sowie deren Auswertung und Präsentation stellen eine große Herausforderung dar.

Auch für das Zusammenstellen aller wichtigen Projektinformationen ist es essentiell, wie im Rahmen des Projektmanagements beispielsweise Termin-, Arbeits- und Ressource-Pläne festgelegt werden. Ebenso wichtig ist die Identifikation von Problemen und das Festlegen von Maßnahmen usw. Dabei die Übersicht zu behalten und adäquat zu reagieren ist auch mit Softwareunterstützung nicht einfach. Aber gerade auftretende Abweichungen von den Plänen müssen möglichst früh erkannt werden, damit rechtzeitig gegengesteuert werden kann. Der Plan muss dafür ständig aktualisiert werden.

Die Frage ist also, wie sich diese Bereiche der Projektkontrolle auf die Software-Engineering-Methode auswirken. Wie können Maßnahmen verbessert und besser mit dem Überwachen einer Software-Engineering-Methode verzahnt und somit optimiert werden? Wird eine Software-Engineering-Methode im Projekt optimiert, so können auch die Ziele des Projektmanagements besser eingehalten werden.

2.2.3 Resilienz in Projekten – Adaptive Projekte

Um die sich ständig ändernden Bedingungen, Umgebungen und Faktoren in einem Projekt und somit einer Software-Engineering-Methode begegnen zu können, wurden bei [Bo13] die Resilienz in Projekten und adaptive Projekte thematisiert.

Resilienz bezieht sich zunächst auf den Menschen und umfasst seine Fähigkeit, dynamisch mit widrigen Umständen und sich ändernden Situationen umgehen zu können. Dafür nutzt er verschiedene Faktoren wie Optimismus, Lösungsorientierung etc. Diese ermöglichen es dem Menschen mit Krisen umzugehen. Betrachtet man nun Resilienz im Sinne der Systemtheorie, so sagt diese aus, dass Systeme ihren Zustand auch bei Störungen und Einfluss sowohl von innen als auch von außen ausgleichen können. Somit behalten sie ihre Systemintegrität [Bo13]. Ein System versucht also immer wieder einen stabilen Zustand zu erreichen. Alles, was diese Stabilität gefährden kann, wird nach der Definition der statischen Resilienz ausgeblendet oder gleich verhindert.

Doch ein Projekt ist immer im Wandel und kann diese Zustände auch nicht ausblenden. Es sollte darauf entsprechend reagieren können. Von daher ist für Projekte und somit auch Software-Engineering-Methoden die Definition der ökologischen Resilienz sinnvoll. Danach existieren mehrere Systemzustände, welche tolerierbar sind. Die Einflüsse von innen und außen sorgen dafür, dass bei bestimmten Abweichungen in einen anderen Zustand gewechselt werden kann [Bo13]. Somit wird die Systemintegrität gewahrt.

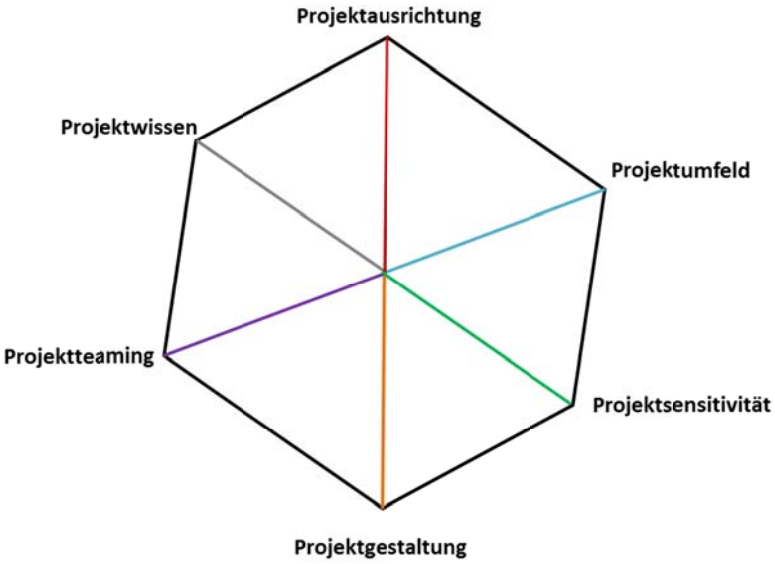

Abbildung 13 Das H.A.P.-Modell nach Bogert [Bo13] mit seinen sechs Dimensionen

Mit Hilfe des H.A.P.-Modells (Hoch Adaptives Projekt) in Abbildung 13 nach Bogert [Bo13] werden mit den sechs Dimensionen die Eigenschaften und Fähigkeiten eines adaptiven Projektes beschrieben. Ein Projekt befindet sich nach Bogert immer innerhalb dieser Dimensionen. Es gibt kein Projekt, welches nicht adaptiv ist, einige sind nur adaptiver als andere. Mit Hilfe des Modells und den verschiedenen Dimensionen lassen sich die Potentiale von einem System und somit auch von einem Projekt aufzeigen, welche zur An-

passung und Veränderungen besonders geeignet sind. Neben den Personen ist hier auch der Kontext zu betrachten.

Die sechs Dimensionen des H.A.P.-Modells sind:

1. **Projektausrichtung (rot):** Hier geht es sowohl um den Faktor Zeit, als auch wie die Personen die Zukunft des Projektes betrachten und wie sie es angehen wollen. Wichtig ist dabei die Vorausschau. Das heißt es wird betrachtet, welche Szenarien eintreten könnten und wie darauf entsprechend reagiert werden kann, damit das System bzw. das Projekt seine Integrität wahrt. Dabei sollte sich immer an den Zielen orientiert werden, denn nach Bogert sind adaptive Projekte zielorientiert.
2. **Projektumfeld (blau):** Das Projektumfeld bei der Adaptivität von Projekten wird hier im Sinne von Beziehungen, insbesondere in Hinsicht auf Personen betrachtet, welche sie untereinander, aber auch mit der Umwelt haben. Um tragfähige Beziehungen zu erreichen, ist ganz besonders das Feedback wichtig. Dynamische Systeme und in diesem Fall Projekte können ohne Feedback nicht erfolgreich sein. Sie brauchen die Rückkopplung, um Änderungen und Korrekturen vornehmen zu können.
3. **Projektsensitivität (grün):** Bei der Projektsensitivität geht es hauptsächlich um die Achtsamkeit der Projektmitglieder in Bezug auf das Projekt. Damit sind hier unter anderem die Aufmerksam des Einzelnen, aber auch die eigenen Erwartungen, Denkweisen, Interpretationsmöglichkeiten und Bewertungen gemeint. Dabei ist es wichtig, Details aus dem Projektumfeld wahrzunehmen und herauszufinden, ob es Turbulenzen und Probleme geben könnte, um gegebenenfalls gegensteuern zu können.
4. **Projektgestaltung (orange):** Bei der Projektgestaltung geht es in erster Linie um die kulturellen Aspekte. Dabei kommt die Projektgestaltung im Wesentlichen vom Management und soll von ihm entsprechend ausgefüllt werden. Wesentliche Aspekte sind dabei eine gute Fehlerkultur, Redundanzen, Entscheidungskompetenzen und Flexibilität der einzelnen Teammitglieder.
5. **Projektteaming (violett):** Ein zentraler Aspekt von einem Projekt ist das Projektteam mit seinen Personen, ihren Fähigkeiten und verschiedenen Persönlichkeiten. Beim Teaming geht es darum, ein optimales und flexibles Team zu finden.
6. **Projektwissen (grau):** Die letzte Dimension beschäftigt sich mit dem Wissen und den Erfahrungen der einzelnen Projektmitglieder. Wichtig ist dabei vor allem, wie alle Mitglieder (voneinander) lernen können, um die Erfahrungen zu nutzen, umzusetzen und sich zu verbessern.

Das adaptive Projekt und H.A.P.-Modell von Bogert ist relativ abstrakt. Es bezieht sich größtenteils auf die Personen in einem Projekt und wie diese einerseits adaptiv mit dem Projekt umgehen, andererseits aber auch auf Krisensituationen reagieren. Personen sind ein wichtiger Faktor im Projekt und nötig, um eine Software-Engineering-Methode durchzuführen. Auch wenn der Fokus nicht auf der Software-Engineering-Methode liegt, ist zu überlegen, wie diese Aspekte von einem adaptiven Projekt genutzt werden können, um eine adaptive Software-Engineering-Methode zu entwickeln.

Ein Bereich, welcher sich ebenfalls mit Veränderungen beschäftigt und sich auf Projekte und gegebenenfalls auf die Software-Engineering-Methode auswirken kann, ist das Change Management. Dieses wird im nächsten Abschnitt näher erläutert wird.

2.3 Change Management auf Unternehmensebene

Change Management, zu Deutsch Änderungs- oder auch Veränderungsmanagement, beschäftigt sich mit den Änderungen hauptsächlich auf den Unternehmensebenen. Es beinhaltet dabei insbesondere „*die speziellen Managementtechniken, die zur Steuerung der Prozesse im Rahmen von Wandel selbst erforderlich sind*“ [La10, S. 3]. Wandel oder Veränderungen in einem Unternehmen können die unterschiedlichsten Ursachen haben, ein wichtiger Einflussfaktor ist dabei die Umwelt. Häufig ist es nötig, dass Unternehmen auf die unterschiedlichen Anforderungen im Markt reagieren und sich anpassen müssen. Die Anforderungen können ganz verschieden sein und desto unterschiedlicher sind auch die Änderungsprozesse, die im Unternehmen angestoßen werden können. Denn jeder Veränderungsprozess kann auf einer anderen Ebene wirken und bildet einen anderen Schwerpunkt [SH13].

Das Change Management setzt dabei auf den Prozess selbst, um vom aktuellen Ausgangspunkt ein definiertes Ziel zu erreichen. Dabei liegt nach [La10] der Fokus weniger auf dem Ziel bzw. dessen Definition selbst, sondern auf der Gestaltung des Weges, also des Änderungsprozesses, um das Ziel zu erreichen. Change Management hat von daher nach Abbildung 14 verschiedene Ansatzpunkte, welche sich nach „innen richten“. Diese gehen dabei primär auf die Personen im Unternehmen [La10] ein.

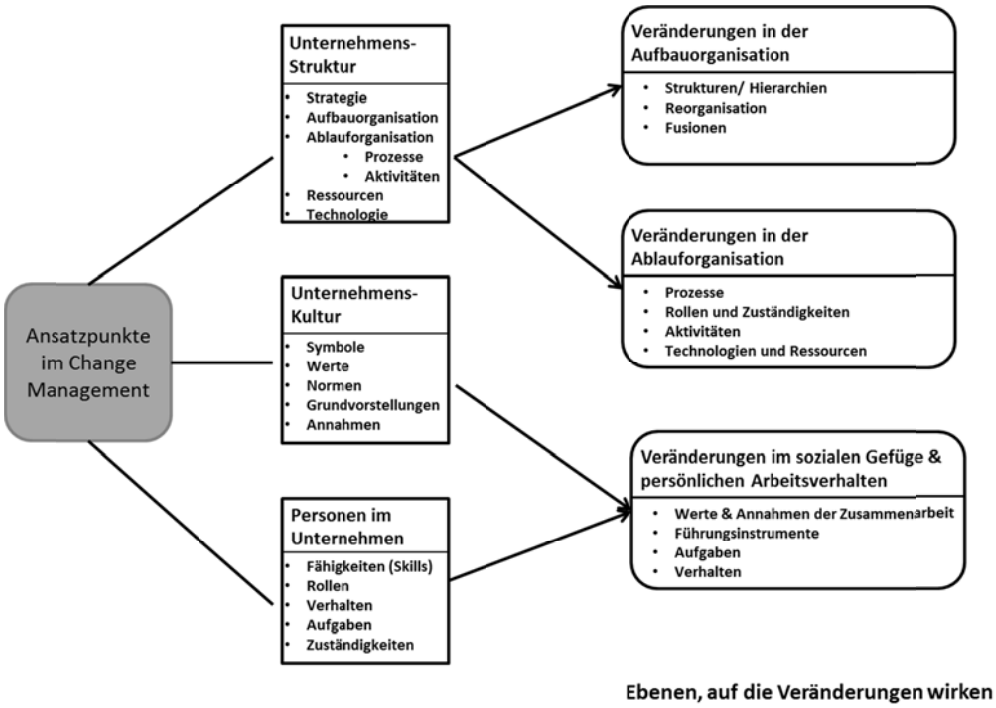

Abbildung 14 Ansatzpunkte des Change Managements und Ebenen, auf die Veränderungen wirken, basierend auf [SH13, La10]

Ohne Personen wäre in einem Unternehmen nichts möglich, denn ohne ihre Mitarbeit würde weder das Unternehmen überleben, noch wäre eine Veränderung im Unternehmen möglich. Dabei geht es sowohl um die Fähigkeiten der Personen selbst, den Wandel durchzuführen und sich anzupassen, als auch um ihre Rollen im Gesamtgefüge. Beispiele sind wie sich Personen anderen gegenüber und im Projekt verhalten oder auch welche Aufgabe mit entsprechenden Zuständigkeiten sie besitzen und ausführen.

Ein weiterer Ansatzpunkt sind die Strukturen im Unternehmen. Dabei geht es sowohl um die Hierarchie im Unternehmen, als auch darum, wie die Organisation an sich aufgebaut ist. Des Weiteren ist es wichtig, wie die Abläufe organisiert sind, welche Strategien im Unternehmen gefahren werden, welche Ressourcen und Technologien vorhanden sind usw. Gerade die Projekte im Unternehmen leben von den, teilweise informellen, Prozessen und Abläufen. Diese entstehen manchmal evolutionär und auch über Jahre hinweg.

Ebenso wie an den Strukturen im Unternehmen kann eine Änderung der Unternehmenskultur wichtig sein. Diese lebt dabei zum einen von der Kultur der Mitarbeiter selbst. Zum anderen lebt sie aber auch davon, welche Werte, Annahmen und Grundvorstellungen im Unternehmen selbst herrschen. Auch wenn es verschiedene Ansatzpunkte für Veränderung im Unternehmen gibt, so sollten doch immer die anderen Punkte mit in den Änderungsprozess einbezogen werden. Eine Änderung auf der Strukturebene

ohne die Personen oder auch die ganze Kultur im Unternehmen mit zu berücksichtigen ist meistens nur schwer möglich oder wird komplett scheitern [La10].

Diese verschiedenen Ansatzpunkte im Change Management führen zu Veränderungen auf drei verschiedenen Ebenen: Veränderungen in der Aufbauorganisation, in der Ablauforganisation und Veränderungen im sozialen Gefüge sowie im persönlichen Arbeitsverhalten [SH13]. Auch wenn die Änderung an den Werten, Führungsinstrumenten und Verhalten der einzelnen Personen im Unternehmen sehr wichtig sein kann, ist für diese Arbeit der Ansatzpunkt der Unternehmens-Struktur und von daher Veränderungen an der Aufbau- und Ablauforganisation von größerem Interesse.

Änderungen auf der Ebene der Aufbauorganisation können sich auf Projekte und damit auch auf deren Software-Engineering-Methode auswirken. Vor allem Reorganisationen und Fusionen können es dringend nötig machen, dass die Abläufe und somit die Software-Engineering-Methode angepasst werden muss. Gerade aber auch die Ebene der Ablauforganisation kann sich direkt auf die Änderung eines Projektes und somit der Software-Engineering-Methode auswirken. Im Change Management wird aber bei der Ablauforganisation nicht die genutzte Software-Engineering-Methode selbst angepasst, sondern die Prozesse oder Geschäftsprozesse auf Unternehmensebene.

Für den Veränderungsprozess wird typischerweise ein eigenes Veränderungsprojekt ins Leben gerufen. An diesem können verschiedene Personen beteiligt sein, beispielsweise Personen aus der Führungsebene, aus dem Projektmanagement und Qualitätsmanagement, aus dem Betriebsrat und dem Projektcontrolling usw. Für die Durchführung des Veränderungsprojektes wird ein Kernteam zusammengestellt. Dieses plant sowohl das Projekt als auch den Veränderungsprozess und führt sie anschließend durch [No14].

Der eigentliche Veränderungsprozess wird vorher genau geplant und entworfen. Dabei werden die Veränderungen sowohl „Top-Down“ als auch „Bottom-Up“ durchgeführt. Das bedeutet zum einen, die Rahmenbedingungen und die Vorgehensweise werden von oben vorgegeben (top-down for targets). Zum anderen bedeutet es, dass die spätere konkrete und inhaltliche Umsetzung „von unten“ mit Hilfe der Betroffenen umgesetzt wird (bottom-up for how to do it). [No14]

Für die Vorgehensweise heißt dies konkret, dass das Team die Veränderungen plant, in dem es sich zunächst auf allen Ebenen den aktuellen Stand, die aktuellen Abläufe sowie Prozesse anschaut und diese bezüglich Verbesse-

rungspotentials analysiert. Daraus können Zielsetzungen und Lösungen abgeleitet und erstellt werden. Diese werden in konkrete Maßnahmen übersetzt und anschließend im eigentlichen Veränderungsprozess durchgeführt [SH13]. Die Ergebnisse des Veränderungsprozesses werden abschließend evaluiert.

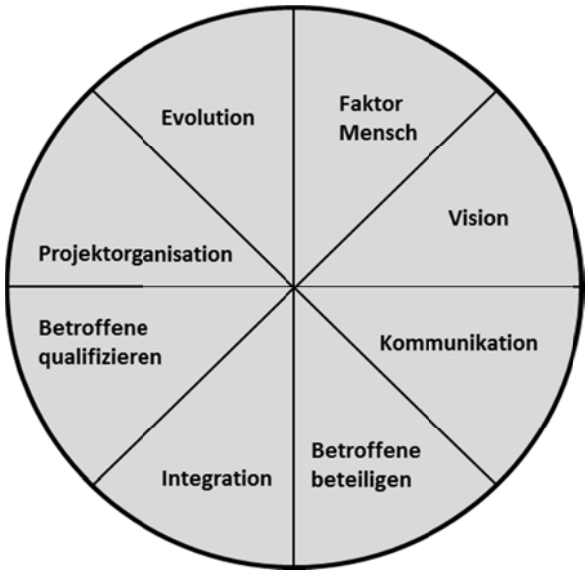

Abbildung 15 Kernthemen und Faktoren des Change Management basierend auf [La10, SH13]

Damit das Veränderungsprojekt und somit das Veränderungsmanagement zum Erfolg führen, gibt es wie in Abbildung 15 zu sehen verschiedene Kernthemen und Erfolgsfaktoren [La10, SH13].

Ein Faktor sind dabei zum einen die Personen die im Unternehmen arbeiten und am Wandel mitwirken müssen. Zum anderen kann dies eine (Führungs-) Person sein, welche den Wandel anstößt und mit begleitet. Ohne diesen Faktor Mensch wäre eine Veränderung überhaupt nicht möglich.

Um nun eine Änderung im Unternehmen durchzuführen, muss zunächst eine Vision entwickelt werden, welche das Unternehmen verbessern soll. Diese kann auch als verschiedene Ziele definiert werden, welche die Beteiligten motiviert, eine Veränderung umzusetzen. Wichtig ist es, während des Veränderungsprozesses alle davon Betroffenen mit am Prozess aktiv zu beteiligen sowie die Kommunikation zwischen den Betroffenen selbst und den Durchführenden zu stärken. So werden alle Personen sofort mitgenommen und es kommt im Laufe des Prozesses nicht zu Missverständnissen. Ferner ist damit eine höhere Akzeptanz an der Veränderung bei den Betroffenen gewährleistet. Durch eine gute Kommunikation wird zusätzlich die Integration gestärkt. Unterschiede, zum Beispiel in der Kultur, können somit einfacher überwunden werden.

Wichtig ist nach [La10, SH13] die anschließende (Weiter-)Qualifizierung aller Beteiligten. Durch die Veränderungen werden die Personen weiter entwickelt, dabei speziell in ihrem Wissen und Können. Aber auch ihr Verhalten und ihre Einstellungen können sich dadurch wandeln und zum Erfolg der Veränderungen beitragen.

Abschließend sind noch die Themen der Projektorganisation und der späteren Evolution wichtig. Veränderungsprojekte können durch die vielen verschiedenen Themen und Faktoren sehr komplex werden. Sie erfordern von daher eine strukturierte und geplante Projektvorbereitung, Projektdurchführung und Projektüberwachung. Deshalb sind das Projektmanagement und das Change Management stark verwandt, nur das sich Change Management auf Unternehmensebene bezieht und Projektmanagement auf die Projektebene. Es ist aber nicht nur der aktuelle Veränderungsprozess wichtig sondern es sollte ein fortlaufender Prozess sein. Veränderung und Wandel soll kontinuierlich initiiert werden, damit sich ein Unternehmen stetig weiter entwickeln kann [La10].

Wie an den verschiedenen Themen und Erfolgsfaktoren zu erkennen ist, liegt ein Hauptaugenmerk des Change Managements auf Personen und allgemeinem Wandel im Unternehmen, weniger auf der konkreten Software-Engineering-Methode. Wie bereits erwähnt, können sich Veränderungen auf die SEM auswirken und somit eine Ursache sein, dass diese ebenfalls geändert werden muss. Dies ist höchstens ein Baustein im Gesamtkontext bzw. im Veränderungsprojekt, aber nicht primär. Doch die enge Verwandtschaft mit dem Projektmanagement und der zu Grunde liegende Prozess für die Durchführung eines Veränderungsprojektes können in eine Änderung an einer Software-Engineering-Methode mit einfließen.

Dabei findet nach [SH13] Veränderungsmanagement in drei Phasen statt:

- Planung, wo die Vision erstellt, der aktuelle Status analysiert und Maßnahmen geplant werden;
- Umsetzung, wo die erstellten Maßnahmen und Lösungen umgesetzt werden;
- Evaluierung, wo die entsprechenden Maßnahmen überprüft werden, ob sie den gewünschten Effekt geliefert haben.

Diese „Phasen“ können für die Änderung bzw. Anpassung einer Software-Engineering-Methode ebenfalls betrachtet werden, ähnlich wie die Mechanismen der Projektkontrolle. Es muss überprüft werden, wie sehr einzelne Mechanismen aus dem Change Management und der Projektkontrolle bereits für die Anpassung von Software-Engineering-Methode zur Laufzeit bei eingesetzt werden und wie effektiv diese sind. Ferner kann auch betrachtet

werden, ob bestehende Ansätze eher „Bottom-Up“, „Top-Down“ oder wie im Change Management beides sind.
Bestehende Ansätze für Anpassungen von einer Software-Engineering-Methode werden im nächsten Abschnitt vorgestellt und diskutiert.

2.4 Anpassungen von SEM

Für die Verbesserung von Projekten, allgemeinen Prozessen oder Software-Engineering-Methoden gibt es verschiedene Modelle und Vorgehensweisen. Doch zunächst werden erste Anforderungen, welche an die möglichst eigenständige Anpassung einer genutzten Software-Engineering-Methode zur Laufzeit gestellt werden, definiert und erläutert.

2.4.1 Erste grobe Anforderungen an die Anpassung einer Software-Engineering-Methode

Damit eine Software-Engineering-Methode während eines laufenden Projektes erfolgreich angepasst werden kann, sollen dafür die Vorteile und Konzepte vom Projektcontrolling, aber auch des Change Managements integriert sein. Gerade die drei im vorherigen Abschnitt erwähnten Phasen nach [SH13] sollen sich bei der Anpassung einer Software-Engineering-Methode wiederfinden. Ferner ist zu betrachten, inwiefern die Konzepte des adaptiven Projektes genutzt werden können. Aus den bereits beschriebenen Abschnitten und aus der eigenen Erfahrung mit Projekten ergeben sich somit folgende erste grobe Anforderungen an die Anpassung einer Software-Engineering-Methode zur Laufzeit:

1. Der Fokus der Anpassung soll auf der genutzten Software-Engineering-Methode liegen und nicht auf dem gesamten Projekt, auch wenn dieses nicht außer Acht gelassen werden darf.
2. Die Anpassung der Software-Engineering-Methode soll zur Laufzeit des Projektes erfolgen.
3. Die Anpassung der Software-Engineering-Methode soll schnell und möglichst eigenständig erfolgen sowie nicht von langer Dauer sein und beispielsweise mehrere Tage oder länger in Anspruch nehmen. Gerade die Erfahrung auch in eigenen Projekten hat gezeigt, dass Zeit im Projekt ein kritischer Faktor ist und nicht viel davon für die Anpassung in Anspruch genommen werden kann.
4. Die Anpassungen sollen nicht einmalig zu bestimmten Zeitpunkten, sondern kontinuierlich und regelmäßig erfolgen. Eine Anpassung soll insbesondere dann erfolgen, wenn es nötigt ist und die Gefahr besteht, dass die Software-Engineering-Methode und somit der Erfolg des Projektes in Gefahr sind.
5. Sowohl im Change Management als auch im adaptiven Projekt und im Projektcontrolling wird erwähnt, dass die Vordefinition

von Zielen und Rahmenwerten und damit die Vorausschau wichtig sind, um mögliche Abweichungen zu erkennen und darauf angemessen reagieren zu können.

6. Die genutzte Software-Engineering-Methode und das Umfeld sollen kontinuierlich während der Laufzeit und Durchführung überwacht werden. Das heißt wie Abschnitt 2.2.3 erwähnt ist regelmäßig Feedback bzw. Rückkopplung zum aktuellen Status sehr wichtig.
7. Wie bereits beim Change Management erwähnt, soll dieser aktuelle Status regelmäßig analysiert und ausgewertet werden, ob Abweichungen vorhanden sind und Maßnahmen geplant und eingeleitet werden müssen.
8. Die nötigen Maßnahmen oder Anpassungen müssen entsprechend geplant und zeitnah umgesetzt werden.
9. Wie in der dritten Phase des Change Management soll die erfolgte Anpassung in der Software-Engineering-Methode evaluiert werden. Dies kann über eine weitere Überwachung der Nutzung der SEM und entsprechendes Feedback im laufenden Projekt erfolgen.
10. Ergebnisse von erfolgten Anpassungen und/ oder Planungen sollen für spätere Zeitpunkte und andere Projekte wiederverwendet werden können.
11. Da die Zeit und somit die Ausführung der Anpassung ein kritischer Faktor ist, soll alles möglichst automatisiert erfolgen.

Nachdem nun die ersten Anforderungen an eine Anpassung einer Software-Engineering-Methode definiert wurden, werden verschiedene bereits vorhandene Ansätze vorgestellt. Diese sind nach dem Kriterium des Zeitpunkts der Anpassung sortiert. Dabei wird zuerst auf die Ansätze eingegangen, durch welche die Software-Engineering-Methode vor der Durchführung angepasst wird. Anschließend wird der „Spezialfall“ der Agilen Methoden erläutert, welche erste Ansätze zur Anpassung während der Durchführung aufzeigen. Abgeschlossen wird mit Ansätzen, welche im Rahmen des Kontinuierlichen Verbesserungsprozesses anzusiedeln sind.

2.4.2 Anpassungen von SEM – Vorgelagerte Ansätze

In einem ersten Ansatz, um die passende Software-Engineering-Methode für ein Projekt anzupassen, kann zum einen das Zuschneiden einer SE-Methode sein. Dies nennt sich Tailoring. Zum anderen kann eine Software-Engineering-Methode an sich aus verschiedenen Bausteinen mit Hilfe von Situational Method Engineering neu erstellt werden.

2.4.2.1 Tailoring von Software-Engineering-Methoden

Beim Tailoring, zu Deutsch zuschneiden, wird eine gegebene Software-Engineering-Methode an das Projekt angepasst, indem sie dafür entsprechend zugeschnitten wird. Das Zuschneiden beinhaltet dabei Anpassungen bzw. Änderungen im Projekt, welche aber mit dem Grundsatz der Software-Engineering-Methode konform sind. Da es keine „one-size-fits-all"-Methode gibt und nicht jede gleich gut für ein Projekt geeignet ist, ist dies eine häufige Art der Anpassung. Zum Beispiel hat ein Projekt weniger Budget und ist kleiner angelegt als ein anderes Projekt in demselben Unternehmen. Das Projekt soll aber dieselbe Software-Engineering-Methode benutzen. Es kann dann nötig sein, dass nicht alle Aktivitäten durchgeführt und nicht alle Artefakte, welche in der generellen Software-Engineering-Methode definiert sind, erzeugt werden [Sa11]. Doch auch beim Tailoring muss beachtet werden, dass ein Zuschneiden einer Methode immer Konsequenzen für andere Artefakte oder Aktivitäten etc. beinhaltet. Es muss darauf geachtet werden, dass die Konsistenz der Abhängigkeiten gewährleistet ist. Beispielsweise entscheidet ein Projektleiter, dass eine neue Software nicht mit sogenannten „Use Cases" spezifiziert werden soll. Anstatt der Use Cases soll eine Kombination von Businessprozessen, Business-Regeln usw. genutzt werden. Diese Änderung hat einen großen Einfluss auf eben solche Artefakte, welche typischerweise in Relation zu Use Cases stehen [Sa11].

Obwohl das Tailoring eine gute Methode ist, um Software-Engineering-Methoden auf die Projektsituation anzupassen, muss zum einen bereits eine Methode gegeben sein und zum anderen geschieht die Anpassung nur vor Projektbeginn. Eine weitere Betrachtung der erfolgten Anpassung sowie weitere mögliche Anpassung während der Nutzung der SEM im laufenden Projekt ist hier nicht weiter vorgesehen

2.4.2.2 Anpassung durch Situational Method Engineering

Situational Method Engineering (SME) beschäftigt sich wie in Abschnitt 2.1.3.4 beschrieben mit dem Design und der Konstruktion von situationsspezifischen Software-Engineering-Methoden. Normalerweise wird die SEM allerdings zu Beginn, also vor dem Start des Projektes, erstellt und während der Laufzeit nicht weiter betrachtet. Im Situational Method Engineering muss zunächst eine Methoden-Basis mit Methoden-Elementen aufgebaut werden. Zusätzlich müssen die Projektumgebung sowie das Projekt selbst charakterisiert werden. Eines der wichtigsten Elemente im SME ist dabei die Auswahl geeigneter Methoden-Elemente, auch Method Fragments oder Method Chunks genannt, sowie das spätere Zusammensetzen dieser Methoden-Elemente zu einer entsprechenden Software-Engineering-Methode.

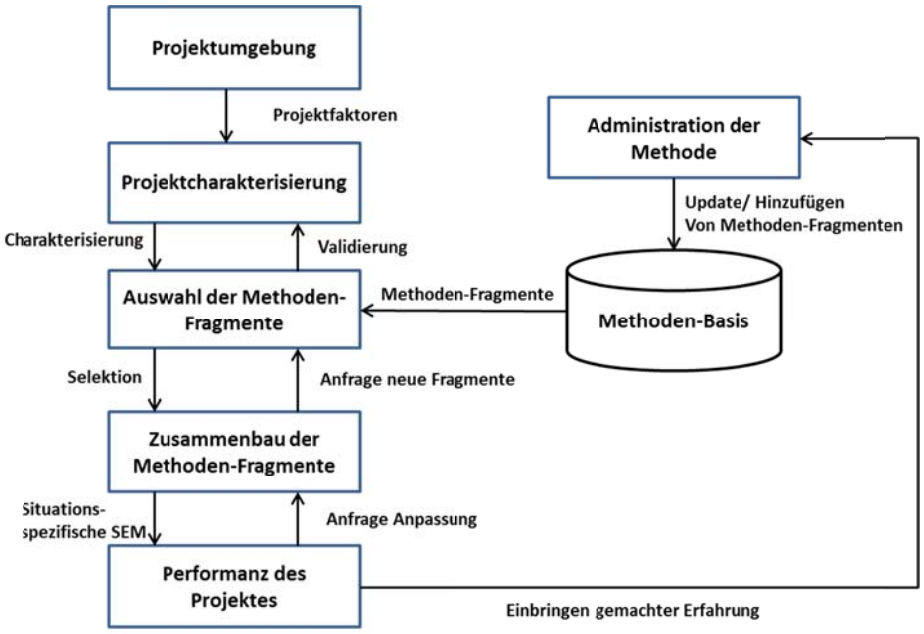

Abbildung 16 Konfigurationsprozess für situationsspezifische Methoden nach [Br96]

Sjaak Brinkkemper [Br96] hat 1996 einen Konfigurationsprozess für situationsspezifische Software-Engineering-Methoden vorgestellt, welcher in Abbildung 16 zu sehen ist. Dabei werden zunächst das Projekt und dessen Umgebung charakterisiert. Aufgrund dieser Spezifikation werden anschließend Methoden-Elemente aus der Methoden-Basis ausgewählt und ebenfalls gegen die Charakterisierung validiert. Diese Methoden-Elemente werden anschließend zu einer situationsspezifischen SEM zusammengesetzt. Abschließend soll diese Software-Engineering-Methode im Projekt durchgeführt werden.

Brinkkemper erläutert, dass es während der Ausführung einer Software-Engineering-Methode zu drastischen Änderungen im Projektumfeld kommen kann, was zu einer Anpassung SEM durch z.B. das Austauschen von Methoden-Elementen führen kann. Er erwähnt allerdings nur, **dass** dieses möglich sein soll, aber nicht **wie** dieses möglich ist. Ferner wird bei Brinkkemper nicht erläutert, wie die auszutauschenden Elemente bestimmt werden. Ferner wird im Prozess zwar erwähnt, dass das Projekt und dessen Umgebung charakterisiert werden (müssen). Es wird aber nicht erwähnt, wo diese Informationen abgelegt werden, um sie später zu nutzen, beispielsweise um die Methoden-Elemente gegen diese validieren zu können.

Im State-of-the-Art Review von Jolita Ralyté und Brian Henderson-Sellers von 2010 [HSR10] wird u.a. darauf eingegangen, wie Methoden-Elemente für eine Methoden-Basis identifiziert und wie aus diesen eine Software-Engineering-Methode konstruiert werden kann. Inwiefern sich nun dies und das Situational Method Engineering für die selbst-adaptive Anpassung einer

Software-Engineering-Methode eignet, wird in Abschnitt 2.5 anhand einer genauen Analyse gegenüber den definierten Anforderungen beschrieben.

2.4.3 Anpassungen von SEM im laufenden Projekt durch Agile Methoden

Die Agilen Methoden beschäftigen sich mit der Entwicklung von Software in Iterationen und einer näheren Kundenanbindung, um schnell qualitative hochwertige Software zu liefern. Die Iterationen in den Agilen Methoden folgen meist dem Schema in Abbildung 17. Es wird zunächst eine Vision erstellt und in Anforderungen herunter gebrochen. Diese müssen zunächst nur grob sein und werden im Laufe der Zeit verfeinert. Das geschieht vor jeder Iteration im zweiten Schritt der Planung. Die Planungsschritte oder eine Teilmenge davon, werden im dritten Schritt ausgeführt. Nach der Ausführung wird am funktionsfähigen Inkrement festgestellt, was angepasst werden muss. An dieser Stelle kann zwar auch überprüft werden, ob die Software-Engineering-Methode angepasst werden müsste, aber im Allgemeinen bezieht sich das Anpassen an dieser Stelle auf die Software und auf die Anforderungen an die Software.

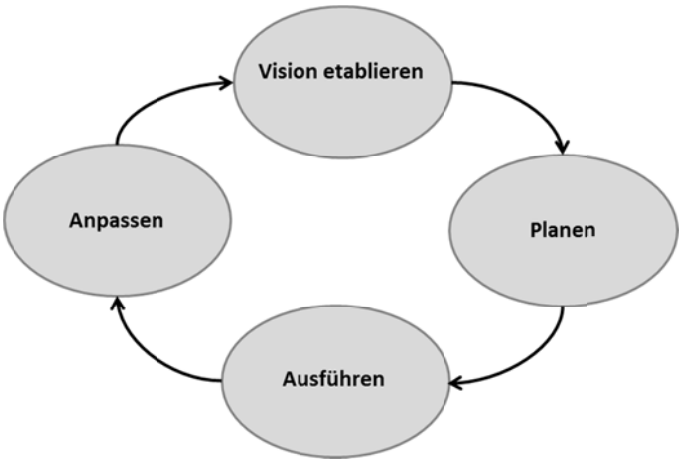

Abbildung 17 Vorgehen von Agilen Methoden angelehnt an [ITA134]

Nähere Ansatzpunkte für die Anpassung einer Software-Engineering-Methode zur Laufzeit mit Hilfe von Agilen Methoden, liefert jedoch Scrum [SS10],[Gl08]. Scrum ist ein (Management-) Framework, das sowohl Kommunikation in Form von Meetings einbindet als auch mit der Retrospektiven einen Mechanismus liefert, um das eigene Vorgehen und die eigene SEM zu reflektieren und gegebenenfalls anzupassen.

Die Retrospektive ist dabei ein internes Meeting der Teammitglieder. In diesem Meeting wird die vorherige Iteration, der sogenannte Sprint, genau besprochen, zum Beispiel was gut und was schlecht gelaufen ist. Anschließend wird beraten, wie das Vorgehen im nächsten Sprint verbessert werden kann.

Diese Meetings sind sehr gut als „Check-Points“, welche innerhalb der SEM durchgeführt werden. Dadurch kann gegebenenfalls innerhalb der Software-

Engineering-Methode gegengesteuert werden, falls etwas schlecht läuft. Nach jedem Sprint ist es ebenfalls möglich, dass neue Teammitglieder, also auch neue Rollen, hinzukommen oder auch wegfallen.

Allerdings ist Scrum, wie allgemein Agile Methoden, recht generisch und ungenau gehalten. Es wird nicht beschrieben wie festgestellt wird, was schief läuft, außer der subjektive Eindruck oder ein evtl. nicht eingehaltenes Ziel des Sprints (Sprint-Ziel). Des Weiteren wird auch nur im Team besprochen, wie das eigene Vorgehen anzupassen ist. Die genutzte SEM an sich, Aufgaben des sogenannte Product Owners oder andere Rahmenbedingungen im Unternehmen sind nicht von Interesse.

Scrum-Teams sind meist recht klein und sollten nicht mehr als 9 Personen betragen. Gibt es mehrere Scrum-Teams die an einem Projekt arbeiten, müsste es eine allgemeine Retrospektive geben und Abhängigkeiten unter einander müssten gerade im Falle von Anpassungen betrachtet werden. Dies wird aber genau beschrieben noch betrachtet.

2.4.4 Anpassungen von SEM – Kontinuierliche Verbesserungsprozesse

Gerade im Bereich des Qualitätsmanagements gibt es verschiedene Methodiken und Prozesse, um die Qualität zu verbessern. Dabei soll sich hier nicht auf die Verbesserung von Software-Engineering-Methoden beschränkt, sondern allgemein vorgestellt werden, welche Möglichkeiten es gibt und auch für das Software Engineering genutzt werden können. Typischerweise werden diese Prozesse zwar kontinuierlich durchgeführt, jedoch nachdem ein Problem bereits aufgetreten ist, sind diese Prozesse sehr langwierig und können als eigene Projekte angesehen werden. Durch ihren hohen Arbeitsaufwand mit meist mehreren Beteiligten ist damit ein hoher Zeit- und Kostenaufwand verbunden. Zwei bekannte Vertreter des Kontinuierlichen Verbesserungsprozesses sind das CMMI (CapabilityMaturityModel Integration) [Wa07], eine Familie von Referenz-Modellen, und SPICE (Software Process Improvement and Capability Determination/ ISO 15504) [Wa07]. Diese beiden sollen allerdings hier nicht weiter betrachtet werden, da sie mehr den Reifegrad von Software-Engineering-Methoden bestimmen und weniger ihre eigentliche Anpassung.

2.4.4.1 PDCA-Zyklus

Als Grundlage für einen Kontinuierlichen Verbesserungsprozess (KVP) dient der bekannte PDCA-Zyklus. Dieser Zyklus ist nicht eingeschränkt auf die Verbesserung von Prozessen, sondern auch von Strukturen, Produkten, Leistungen usw. Er hat seine Ursprünge in der Qualitätssicherung bzw. im Qualitätsmanagement. Der PDCA gehört mittlerweile in Industrieunternehmen zum Standard und er ist Bestandteil der Norm ISO9001 [ISO08].

Die Systematik der kontinuierlichen Verbesserung und dem späteren PDCA-Zyklus geht auf den Qualitätsexperten Deming zurück. Er wird deshalb manchmal auch Deming-Zyklus (Deming-Cycle) oder „Deming-Wheel“ genannt [MN06]. Der Deming-Zyklus wurde in den 50er-Jahren von den Japanern übernommen und zum PDCA-Zyklus weiter entwickelt [MN06]. Der PDCA-Zyklus definiert vier Grundschritte, welche zyklisch wiederholt werden sollen, um eine kontinuierliche Verbesserung zu erreichen. Der Zyklus wird mittlerweile in verschiedensten Situationen und Projekten eingesetzt. Jeder der Buchstaben im PDCA-Zyklus bezeichnet dabei eine Phase bzw. einen der Grundschritte [De86, ISO08,MN06], welche in Abbildung 18 zu sehen sind:

- **P – Plan (Planen):** In der Planungsphase werden zunächst die Ziele, Prozesse sowie die Maßnahmen entwickelt und festgelegt, welche zur Qualitätsverbesserung dienen sollen. Diese werden zusätzlich mit allen Beteiligten, z.B. Kunden und ihren Anforderungen, der Kultur im Unternehmen, dem Management etc. abgestimmt.
- **D – Do (Durchführen):** Die geplanten Maßnahmen und entwickelten Prozesse werden durchgeführt und im gesamten Unternehmen umgesetzt.
- **C – Check (Prüfen):** Die umgesetzten Maßnahmen und Prozesse sowie Produkte werden überwacht und anhand der festgelegten Ziele kontrolliert.
- **A – Act (Handeln):** Das Ergebnis des Checks wird genutzt um eventuell nötige Anpassungen einzuleiten.

Die Korrekturmaßnahmen der letzten Phase bilden wiederum den Ausgangspunkt für ein erneutes Durchlaufen des Zyklus.

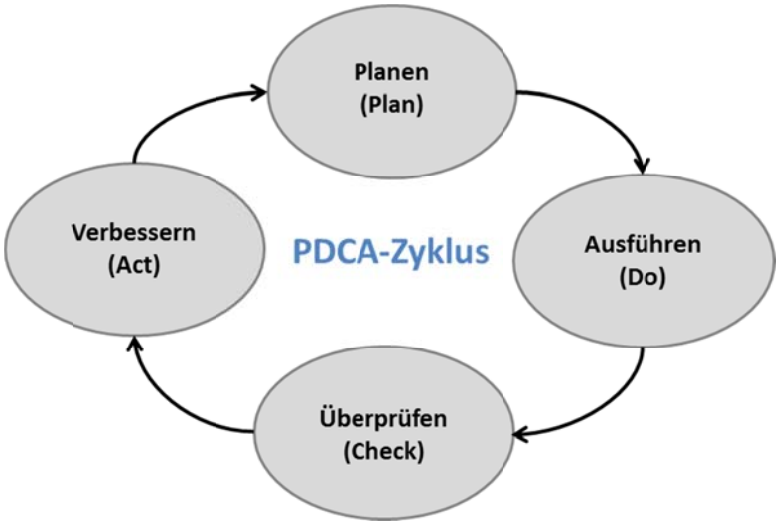

Abbildung 18 Der PDCA-Zyklus nach [De86, MN06]

Der PDCA-Zyklus wirkt auf das gesamte Unternehmen und ist daher vom Management anzustoßen. Er dient zum einen zur Problemlösung, aber auch um Prozesse, Produkte etc. im Allgemeinen zu überwachen und zu verbessern. Der Zyklus wird dabei typischerweise nicht auf ein Projekt angewendet, höchstens um Probleme im Nachhinein zu erörtern. Ferner ist der PDCA-Zyklus als allgemeiner angelegter Qualitätsprozess im Unternehmen anzusehen, welcher z.T. einen hohen Aufwand und hohe Kosten beinhaltet. Gerade die vierte Phase, in der die Maßnahmen als Standard definiert und regelmäßig auf Einhaltung überprüft werden sollen, kann zu einem hohen Organisations- und Arbeitsaufwand führen.

2.4.4.2 Design for Six Sigma und DMAIC-Zyklus

Bei Design for Six Sigma, kurz Six Sigma genannt, handelt es sich um eine Methode zum Qualitätsmanagement, Diese wird häufig in der Wirtschaft angewandt [KA06]. Besonders der DMAIC-Zyklus, welcher im Prinzip der Kernprozess von Six Sigma ist, dient zur Messbarkeit von bereits bestehenden Prozessen und um sie nachhaltig zu verbessern.

Dabei handelt es sich um einen Projekt- und Regelkreis-Ansatz. Wie in Abbildung 19 zu sehen, besteht dieser aus den fünf Phasen Define – Measure – Analyze – Improve – Control (= Definieren – Messen – Analysieren – Verbessern – Steuern) woraus sich der Name „DMAIC“ ableitet.

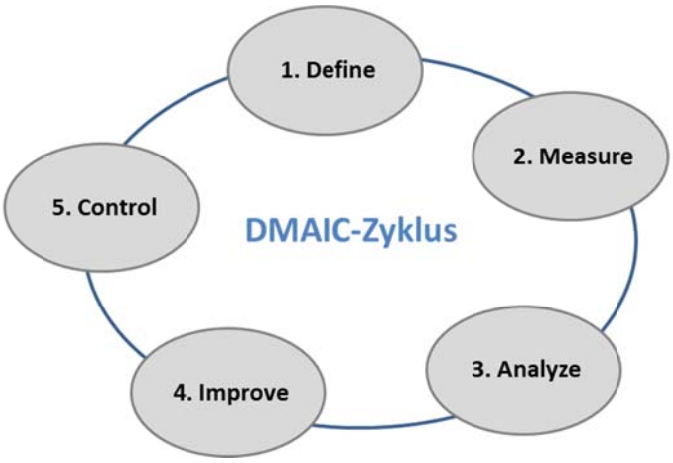

Abbildung 19 Der DMAIC-Zyklus, das Phasenmodell von Six Sigma basierend auf [KA06]

1. Define (D) – In der Define-Phase werden die Anforderungen und Erwartungen des Kunden definiert, ebenso wie Projektgrenzen. Zusätzlich wird das Problem im identifizierten Prozess beschrieben, häufig in Form einer Projektcharta.

2. Measure (M) – In dieser Phase geht es darum festzustellen, wie gut der Prozess wirklich die bestehenden Kundenanforderungen erfüllt. Dies beinhaltet eine Prozessfähigkeitsuntersuchung für jedes relevante Qualitätsmerkmal.

3. Analyse (A) – Ziel der Analysephase ist es, die Ursachen dafür herauszufinden, warum der Prozess die Kundenanforderungen heute noch nicht im gewünschten Umfang erfüllt. Dazu werden Prozessanalysen durchgeführt.

4. Improve (I) – Nachdem verstanden wurde, wie der Prozess funktioniert, wird nun die Verbesserung geplant, getestet und schließlich eingeführt. Ein zusätzliches Ziel ist, dass der Prozess Variationen beseitigt und kreative Alternativen beispielsweise zur Kosteneinsparung entwickelt.

5. Control (C) – Der neue Prozess wird mit statistischen Methoden überwacht.

Der Aufwand bei der Durchführung eines solchen DMAIC-Zyklus ist in Six Sigma hoch. Die Durchführung eines solchen Prozesses dauert ebenfalls sehr lang, teilweise 90-120 Tage [KA06]. Ferner geht es bei Six Sigma mehr darum, bestehende Prozesse im Allgemeinen zu verbessern und diese neu zu implementieren. Es geht bei dem Ansatz nicht um die Anpassung einer laufenden Software-Engineering-Methode, dafür wären die Durchführungszeiten zu lang.

2.4.4.3 8D-Methodik

Ähnlich Six Sigma und auch dem PDCA-Zyklus ist die 8D-Methodik [JSW11] zur Problemlösung, welche auch im Reklamations- und Beschwerdemanagement eingesetzt wird. Es handelt sich um eine nachgelagerte Methode, da Fehler zu diesem Zeitpunkt bereits aufgetreten sind und behoben werden müssen. Dadurch enthält die Methode zum einen Sofortmaßnahmen, um schnell handeln zu können. Zum anderen enthält sie Schritte für eine nachhaltige Entwicklung, um diesen Problemen durch anhaltende Korrekturmaßnahmen im weiteren Verlauf vorzubeugen. Die 8D-Methodik beinhaltet folgende acht Schritte, auch Disziplinen genannt [JSW11]:

Schritt 1- Team bilden: Sobald ein Problem erkannt wurde, wird ein Teamleiter festgelegt. Dieser ist sowohl für die Zusammenstellung des Teams als auch für die korrekte Einhaltung der acht Schritte verantwortlich.

Schritt 2 – Problembeschreibung: Die Aufgabe des Teams ist es, dass aufgetretene Problem vollständig zu beschreiben und genau abzugrenzen. Es müssen die Abweichungen, also der Unterschied vom Ist- zum Soll-Zustand genau definiert werden. Als Hilfe dazu dienen beispielsweise Spezifikationen, getroffene Vereinbarungen mit dem Kunden, Anforderungen usw. Ferner muss das Team die Auswirkungen der Abweichung abschätzen, ob beispielsweise Maßnahmen nötig sind oder nicht.

Schritt 3 – Sofortmaßnahmen: In diesem Schritt muss das Team Entscheidungen für Sofortmaßnahmen treffen, um den Kunden vor schwerwiegenden Auswirkungen zu schützen. Im produzierenden Gewerbe heißt dies, dass fehlerhafte Teile aus dem Umlauf genommen werden müssen, aber die Lieferfähigkeit sichergestellt sein muss, beispielsweise durch Nacharbeiten bei fehlerhaften Stücken.

Schritt 4 – Fehlerursachen: Nachdem die Sofortmaßnahmen eingeleitet wurden, wird im Team untersucht, welche Ursachen es für den Fehler gibt. Dies kann mit Hilfe von Tests, Experimenten oder auch durch einen Vergleich erfolgen. Zur Hilfe können dabei beispielsweise die 5-Why-Methode oder ein Ishikawa-Diagramm herangezogen werden. Am Ende ist die Grundursache gefunden und genau beschrieben.

Schritt 5 – Planen von Korrekturmaßnahmen: Ist die Grundursache gefunden, werden anschließend die Korrekturmaßnahmen vom Team geplant, welche vor allem die Grundursache nachhaltig beseitigen sollen. Die besten Maßnahmen werden am Ende ausgewählt, beispielsweise mit Hilfe von Versuchen und Tests, welche überprüfen, dass das Problem effektiv beseitigt wurde.

Schritt 6 – Maßnahmenumsetzung und Prüfung der Wirksamkeit: Sind die Maßnahmen vom Team definiert, werden diese anschließend in den Prozess eingeführt und entsprechend mit den Produktionsdaten umgesetzt. Zusätzlich wird überprüft, ob die Maßnahmen greifen und das Problem komplett behoben wurde. Ist dies der Fall, können die Sofortmaßnahmen aufgehoben werden.

Schritt 7 – Vorbeugungsmaßnahmen: Ist das Problem beseitigt, muss das Team Vorbeugungsmaßnahmen treffen, damit dieser oder ähnliche Fehler nicht wieder auftreten.

Schritt 8 – Abschluss und Würdigung der Teamleistung: Sind alle Aufgaben des Teams erledigt, wird die Leistung des Teams zum Abschluss gewürdigt und Erfahrungen, die im Laufe des Prozesses gesammelt wurden, werden ausgetauscht und für die Zukunft festgehalten.

Wie an den acht Schritten zu erkennen ist, sind sich die 8D-Methodik und Six Sigma sehr ähnlich und überschneiden sich teilweise in den Kernprozessen. Ebenso wie Six Sigma ist die 8D-Methodik aufwendig und von längerer Dauer. Deshalb sollte zu Beginn immer überlegt werden, ob die Anwendung der Methodik nötig ist oder ob andere Vorgehensweisen zur Behebung kleinerer Fehler angewendet werden können.

2.5 Bewertung und Schwachstellen der Ansätze

Auch wenn in den einzelnen Abschnitten der Ansätze jeweils eine kurze Einschätzung bezüglich der gestellten Anforderungen vorgenommen wurde, sollen in diesem Abschnitt die verschiedenen Ansätze nochmals im Detail überprüft und bewertet werden. Den beschriebenen Ansätzen wird zusätzlich das Projektcontrolling gegenübergestellt, da dieses sich mit der Überwachung von Projekten beschäftigt.

2.5.1 Probleme und Bewertung der verschiedenen Ansätze

Für die Bewertung der einzelnen Ansätze wurden verschiedene Bewertungskriterien in Anlehnung an die bereits definierten Anforderungen erstellt. Die Kriterien werden im Folgenden aufgelistet. Anschließend werden die einzelnen Kriterien genauer erklärt und die Ansätze sowie das Projektcontrolling werden entsprechend in Bezug auf das Kriterium bewertet. Die Bewertung entspricht dabei einer Zahl zwischen 0 und 5, wobei 0 bedeutet, dass das Kriterium gar nicht erfüllt und 5 bedeutet, dass das Kriterium im vollsten Maße erfüllt ist. Die Gesamtbewertung ist der gemittelte Wert aus allen Kriterien. Eine Übersicht der Bewertung ist in der untenstehenden Tabelle dargestellt.

Die einzelnen Bewertungskriterien, die in die Bewertung der Ansätze eingingen sind:

1. Fokus der Anpassung (Fokus)
2. Anpassungszeitpunkt (AZP)
3. Dauer der Anpassung (Dauer)
4. Kontinuität und Häufigkeit der Anpassung (KH)
5. Überwachung der genutzten SEM zur Laufzeit (ÜW)
6. Analyse und Auswertung des aktuellen Status (AA)
7. Planung von Maßnahmen (PM)
8. Evaluierung der Ergebnisse (EE)
9. Vorausschau (VS)
10. Wiederverwendung von Ergebnissen (WVE)
11. Automatisierungen (Auto)

Die Abkürzungen in den Klammern entsprechen denen in der Tabelle. Die einzelnen Ansätze werden folgendermaßen abgekürzt:

- Situational Method Engineering (SME)
- Tailoring von Software-Engineering-Methoden (Tailoring)
- Agile Methoden (AM)
- PDCA-Zyklus (PDCA)
- Six Sigma (SiSi)
- 8D-Methodik (8D)
- Projektcontrolling (PC)

	SME	**Tailoring**	**AM**	**PDCA**	**SiSi**	**8D**	**PC**
Fokus	5	5	4	2	2	0	2
AZP	2	2	4	3	3	2	4
Dauer	3	3	4	3	1	3	3
KH	1	1	4	4	2	3	5
ÜW	0	0	4	4	2	2	5
AA	1	1	4	4	4	4	5
PM	0	0	2	3	5	5	3
EE	1	0	4	3	4	3	4
VS	3	3	2	5	3	1	5
Auto	1	0	0	0	0	0	1
WVE	3	3	3	3	3	3	3
Gesamt	Ø 1.8	Ø 1.6	Ø 3.2	Ø 3.1	Ø 2.45	Ø 2.4	Ø 3.6

Tabelle 1 Übersicht der Bewertung der einzelnen Ansätze und des Projektcontrollings

Fokus der Anpassung

Bei diesem Kriterium wurde bewertet, welchen Fokus die einzelnen Ansätze insbesondere bezüglich einer Anpassung oder einer Verbesserung besitzen. Dabei ist festzustellen, dass nur bei den beiden vorgelagerten Ansätzen der

Fokus direkt auf der Software-Engineering-Methode liegt. Deswegen haben beide Ansätze hier die höchste Bewertung erhalten.

Bei den Agilen Methoden liegt der Fokus nur indirekt auf der Software-Engineering-Methode selbst, der Hauptfokus liegt zunächst auf der zu entwickelnden Software. Doch wird beispielsweise durch die Technik der Retrospektive auch die Software-Engineering-Methode selbst hinterfragt. Des halb erhalten die Agilen Methoden die zweithöchste Bewertung

Bei den weiteren Ansätzen liegt der Fokus mehr auf den Projekten, als auf der genutzten Software-Engineering-Methode. Die 8D-Methodik wird eher im produzierenden Gewerbe als in der Softwareentwicklung eingesetzt, weswegen sie bei diesem Kriterium mit 0 bewertet wurde.

Anpassungszeitpunkt (AZP)

Das Kriterium des Anpassungszeitpunktes beschreibt hier, wann eine Software-Engineering-Methode oder ein Projekt angepasst wird. Das heißt, ob sie vor Beginn der Durchführung angepasst werden, während der Laufzeit oder erst nach der Durchführung. Der gewünschte Zeitpunkt war hier laut den Anforderungen eine Anpassung zur Laufzeit.

Die beiden ersten Ansätze passen zwar eine Software-Engineering-Methode an, aber dies geschieht vor der eigentlichen Durchführung. Deswegen haben sie zusammen mit der 8D-Methode die geringste Punktzahl erhalten. Die 8D-Methodik passt den Prozess erst an, wenn schon ein Fehler passiert ist. Der PDCA-Zyklus und Six Sigma werden zwar dem kontinuierlichen Verbesserungsprozess zugeordnet, sie überprüfen aber ein Projekt erst gegen Ende der Laufzeit oder wenn das Projekt schon abgeschlossen ist, beispielsweise wenn ein Projekt bereits „vor die Wand gelaufen“ ist. Die Untersuchungen fließen gerade bei Six Sigma mehr in die späteren Projekte als in das aktuelle Projekt mit ein.

Die Agilen Methoden betrachten regelmäßig über Feedback, beispielsweise durch eine Retrospektive oder tägliche Meetings, den Fortschritt des Projektes und somit der Software-Engineering-Methode. Somit kann bei Bedarf gegengesteuert werden, allerdings typischerweise ohne die genutzte Agile Methode an sich zu ändern. Das Projektcontrolling geht ähnlich vor, in dem es das Projekt mit Hilfe von definierten Zielen während der Durchführung überwacht. Deswegen haben diese beiden die Bewertung von 4 erhalten.

Dauer der Anpassungen

Mit dem Kriterium Dauer wird überprüft, wie lange eine Anpassung beim jeweiligen Ansatz dauert. Das heißt, wieviel Zeit vergeht, bis die Anpassung in die aktuelle Software-Engineering-Methode, entsprechend in das Projekt

oder in den Prozess überführt wird. Da Zeit in vielen Projekten ein kritischer Faktor ist, soll wie in der entsprechenden Anforderung beschrieben die Anpassung schnellstmöglich erfolgen.

Design for Six Sigma ist ein Ansatz, der wie beschrieben sehr lange dauern kann, teilweise 90-120 Tage [KA06], weswegen dieser Ansatz eine niedrige Bewertung von 1 erhalten hat. Auch die vorgelagerten Ansätze können eine längere Zeit in Anspruch nehmen, bis eine Software-Engineering-Methode erstellt oder entsprechend zugeschnitten ist. Da dies aber vor dem Projekt geschieht, wird dies nicht ganz so zeitkritisch angesehen, wie während der Laufzeit, weswegen sie die Bewertung 3 erhalten haben.

Der PDCA-Zyklus, die 8D-Methode und das Projektcontrolling haben ebenfalls eine Bewertung von 3 erhalten. Diese nehmen auch eine gewisse Zeit bis zur Anpassung und Durchführung der geplanten Maßnahmen in Anspruch. Diese können aber normalerweise innerhalb von einigen Tagen bis wenigen Wochen durchgeführt werden.

Die Agilen Methoden schneiden in Bezug auf dieses Kriterium am besten ab, da sie mit Hilfe der Techniken von regelmäßigen Meetings relativ schnell reagieren können. Meistens greifen die Änderungen zur nächsten Iterationsschleife. Je nachdem wie der Rhythmus gewählt ist, kann dies innerhalb von 2-4 Wochen erfolgen. Kleine Änderungen sind innerhalb von einem Tag mit Hilfe von täglichen Meetings möglich.

Kontinuität und Häufigkeit der Anpassungen (KH)

Mit diesem Kriterium wird untersucht, wie kontinuierlich und häufig der Ansatz eine entsprechende Anpassung überprüft und durchführt, beispielsweise ob dies zu jedem Zeitpunkt untersucht und durchgeführt wird, nur punktuell oder sogar nur zu einem einzigen Zeitpunkt. Entsprechend der Anforderung ist gewünscht, dass der Ansatz die Software-Engineering-Methode kontinuierlich während der Nutzung und nicht nur punktuell auf eine Anpassung hin überprüft und diese bei Bedarf durchführt.

Die beiden vorgelagerten Ansätze passen die Software-Engineering-Methode nur zu genau einem Zeitpunkt an, vor Beginn der Durchführung. Deswegen haben sie die niedrigste Bewertung erhalten. Six Sigma führt zwar eine umfassende Untersuchung und Anpassung durch, welche auch an mehreren Punkte greifen kann, doch auch diese findet typischerweise nur am Ende des Projektes statt. Es wird gegebenenfalls noch einmal justiert, falls die Maßnahmen die Vorgaben nicht optimal erfüllen. Deswegen hat dieser Ansatz eine Bewertung von 2 erhalten

Zwar ist die 8D-Methodik dem Six Sigma recht ähnlich, jedoch überprüft und passt sie in der Hinsicht den Prozess häufiger an, je nachdem wie häufig entsprechende Probleme und Fehler auftauchen. Aufgrund der etwas häufigeren Untersuchung erhält dieser Ansatz die Bewertung 3.

Durch die Regelmäßigkeit der Überprüfung anhand eines kontinuierlichen Prozesses (PDCA-Zyklus) bzw. mit Hilfe von regelmäßigem Feedback und Meetings (Agile Methoden) erhalten der PDCA-Zyklus und die Agilen Methoden die Bewertung 4. Die höchste Bewertung erhält das Projektcontrolling aus dem Grund, da es ein Projekt wesentlich differenzierter und beispielsweise nicht punktuell durch Meetings betrachtet.

Überwachung der genutzten SEM zur Laufzeit (ÜW)

Dieses Kriterium ist eng mit dem vorherigen der kontinuierlichen Anpassung verknüpft. Denn ohne eine fortwährende Überwachung einer Software-Engineering-Methode während der Laufzeit wäre eine entsprechende Anpassung nicht möglich. Hier wird untersucht, ob der Ansatz überhaupt die genutzte SEM zur Laufzeit überwacht, wie kontinuierlich er das Projekt oder die SEM überwacht und wie sehr dabei das Feedback bzw. die Rückkopplung aus der Umgebung mit betrachtet wird. Wichtig wäre, wie in der Anforderung beschrieben, dass der Ansatz die SEM oder das Projekt während der Laufzeit kontinuierlich überwacht und dabei die Umgebung und deren Feedback mit einbezieht.

Die beiden vorgelagerten Ansätze werden mit 0 bewertet, da sie die Software-Engineering-Methode nicht zur Laufzeit überwachen. Wie schon im vorherigen Kriterium erwähnt führt Six Sigma zwar große Untersuchungen und Analysen durch, überwacht dabei aber das Projekt nicht kontinuierlich. Da es aber in der Analyse das Feedback der Umgebung mit einbezieht, erhält der Ansatz wie im vorherigen Kriterium die Bewertung 2. Dieselbe Bewertung erhält die 8D-Methode, da diese ebenfalls den Prozess nicht kontinuierlich überwacht, aber die Umgebung in das Feedback mit einbezieht.

Die Agilen Methoden erhalten wiederum eine Bewertung von 4, da sie rengelmäßig den Fortschritt des Projektes und somit der SEM kontrolliere. Zusätzlich ist bei ihnen das Einbeziehen der Umgebung besonders wichtig, denn das Feedback hat hier einen hohen Stellenwert. Dennoch bekommt der Ansatz nicht die Höchstbewertung in diesem Kriterium, da die Überwachung noch zu punktuell ist.

Der PDCA-Zyklus erhält ebenfalls wieder die Bewertung 4. Dieser besitzt zwar ebenfalls einen kontinuierlichen Zyklus, jedoch dauert entweder die Durchführung der Überwachung aber sehr lange oder es werden nur die

eingesetzten Maßnahmen beobachtet. Allerdings wird hier ebenfalls die Umgebung in die Überwachung mit einbezogen.

Das Projektcontrolling erhält hier die Höchstbewertung. Es wird mit verschiedenen Mitteln versucht zu überwachen, ob die vordefinierten Ziele während der Projektlaufzeit eingehalten werden. Falls Abweichungen bei der Überwachung auffallen, wird versucht entsprechend gegenzusteuern. Dabei wird auch die Umgebung in die Betrachtungen mit einbezogen.

Analyse und Auswertung des aktuellen Status (AA)

Bei diesem Kriterium wird untersucht, ob der aktuelle Status der Software-Engineering-Methode oder des Projektes regelmäßig analysiert und bezüglich möglicher Abweichungen ausgewertet wird. Für das Erkennen von Abweichungen werden entsprechende Kennzahlen und Messdaten benötigt, die vorher ermittelt werden müssen.

Die beiden vorgelagerten Ansätze bekommen beide die geringste Bewertung, da sie den aktuellen Status nicht während der Laufzeit analysieren. Sie bekommen dennoch eine Bewertung mit dem Wert 1, da beide Ansätze zumindest vorher die Situation des Projektes analysieren und entsprechend die Software-Engineering-Methode erstellen oder zuschneiden.

Alle weiteren Ansätze analysieren den aktuellen Status in der einen oder anderen Ausprägung und werten diesen anschließend aus. Die Agilen Methoden betrachten in Meetings regelmäßig den aktuellen Fortschritt. Es gibt allerdings außer den Anforderungen an die Software, welche im Projekt zu entwickeln ist, keine Kennzahlen oder Messdaten. Der PDCA-Zyklus überprüft in der Phase „Check“ den aktuellen Status und kontrolliert die umgesetzten Maßnahmen. Als Kennzahlen dienen hier wiederum nur die Ziele und Anforderungen.

Six Sigma und die 8D-Methodik überprüfen ebenfalls in längeren und detaillierten Untersuchungen den Status des Projektes bzw. Prozesses. Dafür werden Kennzahlen definiert, um die entsprechenden Werte zu ermitteln. Allerdings findet die Überprüfung des aktuellen Status nur zu genau diesem Zeitpunkt statt und noch ein weiteres Mal, um die umgesetzten Maßnahmen zu überprüfen.

Das Projektcontrolling erhält hier die höchste Bewertung, da der aktuelle Status des Projektes regelmäßig kontrolliert, analysiert und auf mögliche Abweichungen ausgewertet wird. Als Kennzahlen dienen dabei unter anderem die definierten Ziele. Es können aber auch weitere Metriken und Kennzahlen ermittelt werden.

Planung von Maßnahmen (Anpassung) (PM)

Das Kriterium zur Planung von Maßnahmen soll die eigentliche Anpassung oder ihre Planung im zur Laufzeit beinhalten. Dafür müssen die Ist- und Soll-Werte bekannt sein und ob eine Anpassung überhaupt nötig ist.

Die beiden vorgelagerten Ansätze erhalten wiederum die Bewertung von 0, da sie in dem Sinne keine Planung von Maßnahmen enthalten und diese vor allem nicht während des Projektes durchführen.

Die Agilen Methoden planen zwar jede Iteration und was dort umgesetzt werden kann, dies bezieht sich aber typischerweise auf die Anforderungen und m.E. im seltensten Fall auf die Software-Engineering-Methode. Innerhalb einer Retrospektive können Maßnahmen angesprochen und auch geplant werden. Dies ist aber kein fester Bestandteil der Retrospektive und die Planungen sind nicht strukturiert. Von daher bekommen sie die Bewertung 2.

Der PDCA-Zyklus und das Projektcontrolling beinhalten beide eine Plan-Phase. Diese Planungen, welche auch Maßnahmen beinhalten können, beziehen sich allerdings auf vorher definierte Ziele, welche später kontrolliert werden sollen. Falls diese nicht eingehalten werden, muss der Plan entsprechend geändert werden. Sie beziehen sich deswegen maximal indirekt auf die Planung einer eigentlichen Anpassung und bekommen von daher eine mittlere Bewertung.

Six Sigma und die 8D-Methodik planen konkret die Maßnahmen bzw. die Anpassung, nachdem die Ist- und Soll-Werte (Six Sigma) verglichen oder die Fehlerursachen identifiziert und Korrekturmaßnahmen (8D-Methode) nötig sind. Deshalb bekommen beide die höchste Bewertung.

Evaluierung der Ergebnisse (EE)

Hier wird bewertet, ob wie im Change Management angeregt, die Ergebnisse der Anpassungen bei der weiteren Durchführung der Software-Engineering-Methode überprüft werden. Dies ist nötig um festzustellen, dass die korrigierte Software-Engineering-Methode oder die Projekte auch den gewünschten Erfolg bringen und ein gewünschtes Ziel erfüllen.

Das Tailoring erhält wiederum die Bewertung von 0, da die zugeschnittene Software-Engineering-Methode nicht überprüft wird, ob sie während der Durchführung ihren Zweck erfüllt. Das Situational Method Engineering bekommt aus dem Grund die Bewertung 1, da es zumindest in Ansätzen überprüft, ob die Software-Engineering-Methode im späteren Gebrauch per-

formant ist. Falls nicht werden neue Bausteine ausgewählt, damit sie auf die Projektsituation passt.

Der PDCA-Zyklus bekommt eine mittlere Bewertung. Er überprüft indirekt, ob die Ergebnisse den gewünschten Erfolg haben. Im PDCA-Zyklus erfolgt dies nur, wenn dieser von neuem beginnt. Die 8D-Methode erhält ebenfalls die mittlere Bewertung. In Schritt 6 werden zwar die Maßnahmen theoretisch auf ihre Wirksamkeit überprüft, aber nicht in der eigentlichen Durchführung.

Das Projektcontrolling beinhaltet die Überwachung von möglichen Ergebnissen mit in der allgemeinen Überwachung und bekommt von daher die Bewertung von 4. Dieselbe Bewertung bekommen ebenfalls Six Sigma und die Agilen Methoden. Six Sigma überprüft die Maßnahmen nicht in der eigentlichen Durchführung, sondern später mit statistischen Methoden. Die Agilen Methoden überprüfen insofern regelmäßig die Maßnahmen, in dem sie regelmäßig Feedback von den Beteiligten sammeln, beispielsweise durch Meetings.

Vordefinition von Zielen und Rahmenwerte (Vorausschau) (VZR)

Bei diesem Kriterium wird überprüft, ob im Vorfeld Ziele und Rahmenwerte definiert werden, die es einzuhalten gilt und ob diese hinsichtlich Anpassungen oder Maßnahmen ausgewertet und überprüft werden können. Wie in Anforderung 5 erwähnt ist dies wichtig, weil die Vordefinition von Zielen und Rahmenwerten, die Vorausschau erst ermöglicht, um eventuelle Abweichungen zu erkennen und darauf angemessen reagieren zu können.

Die 8D-Methodik erhält hier die Bewertung 1. Es werden an keiner Stelle direkt die Ziele oder Rahmenbedingungen definiert. Sie müssen zumindest indirekt bekannt sein, damit den Problemen entsprechend begegnet werden kann.

Die Agilen Methoden erhalten eine Bewertung von 2, da Ziele typischerweise die Anforderungen sind, welche erfüllt werden müssen. Doch wird zu Beginn des Projektes eine Vision festgelegt, welche die ersten Ziele enthält und damit eine Richtung vorgibt, wie das Endprodukt aussehen sollte. Dies bezieht sich aber in den seltensten Fällen auf die Agile Methode selbst.

Die beiden vorgelagerten Ansätze bekommen hier eine mittlere Bewertung. Sie analysieren beide die Projektumgebung und sehen im Prinzip voraus, welches die besten Anpassungen in der Software-Engineering-Methode sind, damit die Rahmenwerte, welche durch die Projektumgebung gegeben sind, eingehalten werden. Six Sigma erhält ebenfalls eine Bewertung von 3, da in der ersten Phase zwar Anforderungen und Erwartungen definiert wer-

den, was auch Zielen und Rahmenbedingungen entsprechen kann, dies aber nur indirekt etwas mit der gewünschten Vorausschau zu tun hat.

Der PDCA-Zyklus und das Projektcontrolling erhalten die höchste Bewertung. In beiden Ansätzen werden zu Beginn Ziele definiert, auf denen die späteren und weiteren Planungen von Maßnahmen und Anpassungen beruhen.

Automatisierungen (Auto)

Bei diesem Kriterium geht es darum, ob die Ansätze Automatisierung bei der Anpassung einsetzen oder zumindest Möglichkeiten dafür bieten. Automatisierung bietet gerade beim wichtigen Faktor Zeit einen entscheidenden Vorteil, da sie Vorgänge, die sonst manuell durchgeführt werden, wesentlich beschleunigen können.

Betrachtet man allerdings die gegebenen Ansätze so setzt keiner der Ansätze bisher Automatisierung zur Anpassung ein. Das Projektcontrolling kann an einigen Stellen zwar durch Software unterstützt werden, es wird aber weniger oder gar nicht zur Anpassung eingesetzt. Auch das Situational Method Engineering bietet zwar Ansätze zur Automatisierung, um beispielsweise Methoden-Elemente miteinander zu verbinden etc., aber diese befinden sich größtenteils noch in der Forschung.

Wiederverwendung von Ergebnissen (WVE)

Das Kriterium untersucht, inwiefern die Ergebnisse, wie beispielsweise Planungen von und durchgeführte Maßnahmen oder Anpassungen etc., für weitere Software-Engineering-Methoden und/ oder Projekte wiederverwendet werden können.

Alle Ansätze erhalten hier die mittlere Bewertung. Es wird nicht direkt erwähnt, wie bzw. dass die Ergebnisse wiederverwendet werden. Es ist jedoch von jedem Ansatz vorstellbar, dass Teile wiederverwendet werden können. Dafür müssen die Ergebnisse allerdings entsprechend dokumentiert und unter Umständen aufbereitet werden. Beispielsweise muss die Situation, in der es eine Abweichung gegeben hat, notiert und mit der entsprechenden Anpassung versehen werden. Die vorgelagerten Ansätze müssten die Projektsituation und die dazu entsprechend zusammengebaute oder zugeschnittene Software-Engineering-Methode mit einer entsprechenden Begründung dokumentieren. Teile könnten dann in ähnlichen Situationen verwendet werden.

2.5.2 Gesamtbewertung der Ansätze

Wie in der Tabelle an den jeweiligen Gesamtergebnissen zu sehen ist, erfüllt keiner der Ansätze und auch das Projektcontrolling die Anforderungen an

die adaptive Anpassung einer Software-Engineering-Methode zur Laufzeit vollständig.

Gerade die vorgelagerten Ansätze Situational Method Engineering und Tailoring haben ein niedriges Ergebnis. Dies liegt daran, dass die Anpassung der Software-Engineering-Methode nur vor dem Start des Projektes erfolgt. Im Gegensatz dazu sind sie allerdings die beiden Ansätze, deren Fokus auf der Software-Engineering-Methode und deren Anpassung selbst liegt. Dies ist bei den anderen Ansätzen und auch beim Projektcontrolling nicht der Fall. Der Fokus liegt hier auf dem Projekt.

Die Ansätze Six Sigma und die 8D-Methode haben eine durchschnittliche Bewertung von 2.45 und 2.4 erhalten. Ein großes Manko ist hier zum einen der Fokus der Anpassungen, welche nicht auf der Software-Engineering-Methode liegen. Des Weiteren ist der wichtige Faktor Zeit insbesondere bei Six Sigma sehr gering ausgeprägt, da die Anpassungszeiten teilweise bis zu 90 Tage und länger dauern. Auch der Anpassungszeitpunkt liegt hier eher nach dem Projekt bzw. wenn das Projekt, oder bei der 8D-Methodik der Prozess, bereits „vor die Wand gefahren“ ist. Die Ergebnisse fließen bei Six Sigma eher in das gesamte Unternehmen und zukünftige Projekte als in das aktuelle Projekt. Daraus folgt, dass es keine kontinuierliche und häufigere Anpassung gibt sowie keine oder nur wenig Überwachung zur Laufzeit. Im Analysieren und Auswerten des Status, der eigentlichen Probleme und der Planung von entsprechenden Maßnahmen sind beide Ansätze hingegen stark.

Der PDCA-Zyklus und die Agilen Methoden haben eine gute mittlere Bewertung erhalten. Sie besitzen bereits viele Ansätze, welche die Anforderungen zum Teil erfüllen. Doch gerade beim PDCA-Zyklus liegt der Fokus wiederum nicht auf der Software-Engineering-Methode und auch die Dauer ist hier wesentlich höher. Auch die Planungen von Maßnahmen beziehen sich hier nicht auf den analysierten Status, sondern werden im Voraus geplant und sind an die Ziele angelehnt. Dadurch ist der Ansatz in der Vorausschau wiederum sehr stark.

Bei den Agilen Methoden liegt der Fokus zum Teil auf der Software-Engineering-Methode, aber der Hauptfokus liegt auf dem zu entwickelnden Produkt. Doch durch bestimmte Techniken wie tägliche Meetings, eine Retrospektive, Fortschrittsanalysen usw. kann eine regelmäßige Überwachung zur Laufzeit gegeben sein. Diese ist aber typischerweise bezüglich der SEM eher informell und nicht strukturiert. Gerade in der Planung von Maßnahmen und der Vorausschau hat der Ansatz Schwächen.

Das Projektcontrolling hat die höchste Bewertung der Ansätze erzielt, doch auch hier gibt es Schwächen. Der Fokus liegt bei diesem Ansatz auf dem

Projekt selbst und betrachtet die Software-Engineering-Methode dadurch höchstens indirekt. Auch die Dauer der Anpassung und der Anpassungszeitpunkt selbst haben hier Abstriche bekommen. Das Projekt selbst wird zwar kontinuierlich überwacht, aber beispielsweise die Planungen von Maßnahmen bzw. Anpassungen beziehen sich dabei ähnlich wie im PDCA-Zyklus auf solche Planungen, die vor dem analysierten Status erfolgt sind. Auch wenn das Projektcontrolling in der Vorausschau ebenfalls stark ist, beziehen sich die Ziele und Rahmenbedingungen auf „eine andere Ebene“, wie beispielsweise die Zeit oder das Budget des Projektes und weniger auf die Software-Engineering-Methode.

Ein weiteres Manko aller Ansätze bezüglich der Anforderungen ist, dass keiner der Ansätze Möglichkeiten zur Automatisierung bietet. Gerade im Projektcontrolling gibt es zwar Werkzeuge als technische Unterstützung, diese haben aber weniger mit einer automatischen Anpassung zu tun. Gerade um den wichtigen Faktor Zeit mit einzubeziehen und die Dauer der Überwachung sowie Durchführung der Anpassung zu verkürzen, wären Automatisierungsmöglichkeiten wichtig.

Aus der Betrachtung der verschiedenen Ansätze und unter Einbezug der vorher genannten Anforderungen sowie den daraus abgeleiteten Bewertungskriterien hat sich ergeben, dass es wichtig ist, einen Kreislauf zu nutzen. Dieser soll die genutzte Software-Engineering-Methode kontinuierlich und möglichst eigenständig während der Ausführung überwachen, auf Abweichungen hin analysieren, eine Anpassung entsprechend planen und die Anpassung soweit möglich automatisiert durchführen. Aus den Ergebnissen der Bewertung und nach [Ge12] und [GLE12] lassen sich die erstellten Anforderungen in folgende konkrete Anforderungen für einen Ansatz zur selbst-adaptiven Anpassung einer Software-Engineering-Methode zusammenfassen:

A1. Der Fokus der Anpassung liegt auf der genutzten Software-Engineering-Methode.

A2. Der Anpassungszeitpunkt ist zur Laufzeit, und die Dauer der gesamten Anpassung soll möglichst kurz sein.

A3. Die SEM soll kontinuierlich und möglichst eigenständig während ihrer Ausführung in Hinblick auf notwendige Abweichungen und Anpassungen auf der Instanz-Ebene beobachtet werden.

A4. Die beobachteten Werte sowie der aktuelle Status müssen analysiert und schnell beurteilt werden können.

A5. Eine unmittelbare Anpassung der Software-Engineering-Methode zur Laufzeit muss unter Betrachtung von vorher definierten Qualitätszielen auf der Typ-Ebene geplant und durchgeführt werden können.

A6. Die Anpassung soll schnellstmöglich, zeitnah sowie möglichst automatisch erfolgen.

Das Ziel ist es nun, einen Ansatz zu entwickeln, der es einer Software-Engineering-Methode ermöglicht, sich während ihrer Durchführung größtenteils selbstständig zu überwachen und selbst-adaptiv anzupassen. Das heißt konkreter, ein solcher Ansatz soll eine Software-Engineering-Methode kontinuierlich und möglichst eigenständig zur Laufzeit überwachen und ihren aktuellen Status hinsichtlich einer nötigen Anpassung auswerten. Aufbauend auf dieser Auswertung wird wenn nötig eine Anpassung der SEM auf Typ-Ebene geplant und diese wird anschließend möglichst automatisiert während der Laufzeit durchführt und zurück in die SEM auf Instanz-Ebene übertragen.

Anhand einer kontinuierlichen Überwachung und der Anpassung zur Laufzeit können die Ergebnisse der Anpassung außerdem direkt evaluiert werden. Ferner soll mit in Betracht gezogen werden, wie die Ergebnisse, z.B. Anpassungs- und Auswertungsmöglichkeiten etc. in Zukunft wieder verwendet werden können. Für die Erarbeitung des Ansatzes werden die Vorteile der bereits bestehenden Ansätze genutzt. Dabei sollen hauptsächlich deren Kreisläufe verwendet und weiter entwickelt werden. Die genauere Analyse dieser Kreisläufe erfolgt nun im nächsten Kapitel „Beschreibung des Lösungsansatzes“

Kapitel 3 – Beschreibung des Lösungsansatzes

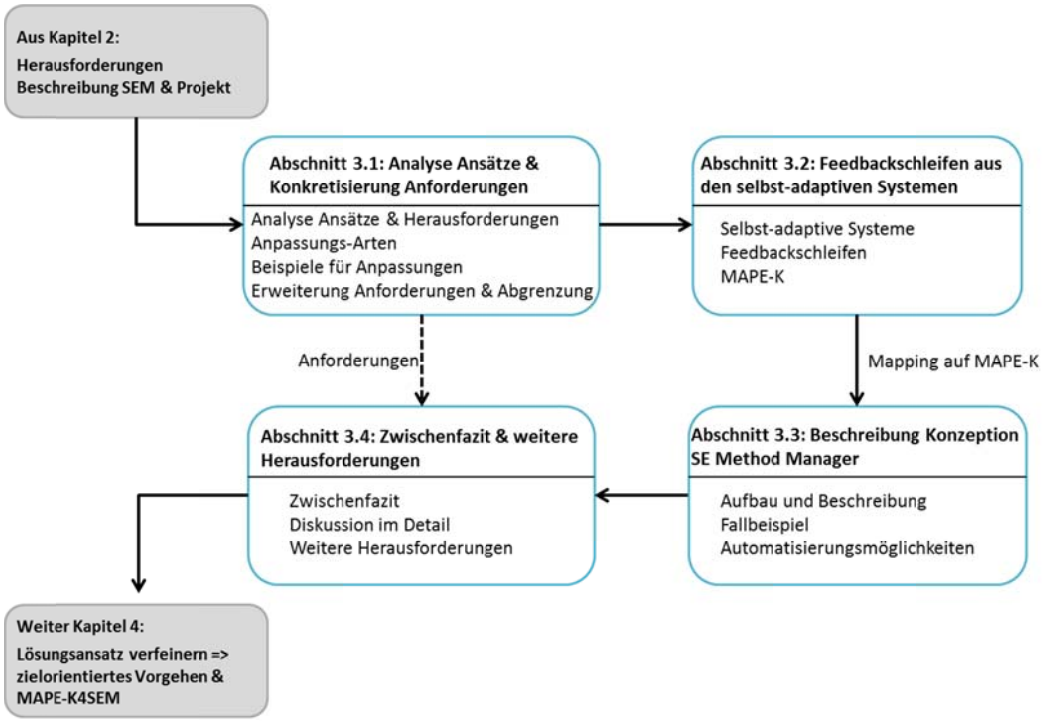

Abbildung 20 Aufbau Kapitel 3

Um einen möglichen Lösungsansatz zu beschreiben, soll in diesem Kapitel zunächst sowohl die Gemeinsamkeiten als auch die Schwachstellen der bisherigen Ansätze analysiert werden. Nach dieser Analyse von Gemeinsamkeiten werden zusätzlich weitere Herausforderungen bei der selbst-adaptiven Anpassung einer Software-Engineering-Methode untersucht. Dafür werden zunächst die Probleme aus dem vorherigen Kapitel aufgegriffen und anschließend wird geklärt, welche Möglichkeiten zur Anpassung in einer Software-Engineering-Methode alle existieren können, u.a. mit Hilfe von zwei komplexen Beispielen. Nach der Konkretisierung der Anforderungen werden allgemein die Feedbackschleifen der selbst-adaptiven Systeme und die MAPE-K-Schleife im Besonderen vorgestellt, da diese die Grundlage für den Lösungsansatz bilden.

Darauf basierend wird die Konzeption eines SE Method Managers vorgestellt, welcher als Framework für die selbst-adaptive Anpassung von Software-Engineering-Methoden dient. Nach einem kurzen Zwischenfazit werden weitere Details und Herausforderungen vorgestellt und diskutiert.

3.1 Analyse der Ansätze und Erweiterung der Anforderungen

In Abschnitt 2.4 wurden verschiedene Vorgehensweisen zur Anpassung von Software-Engineering-Methoden vorgestellt und in Abschnitt 2.5 wurden diese Ansätze in Bezug auf die gestellten Ansätze analysiert und bewertet. Es hat sich gezeigt, dass keiner der Ansätze alle Anforderungen ausreichend erfüllt. Doch um einen eigenen Ansatz zu entwickeln, sollen die Vorteile und Gemeinsamkeiten, welche die Ansätze besitzen, herausgefunden und genutzt werden. Des Weiteren soll genauer herausgefunden werden, welche weiteren Probleme und Herausforderung bei der adaptiven Anpassung einer Software-Engineering-Methode bestehen und mit einem eigenen Ansatz gelöst werden müssen. Im Anschluss werden die verschiedenen Möglichkeiten von Anpassungen erläutert, um im späteren Verlauf die sechs aufgestellten Anforderungen an einen Ansatz zur Anpassung von Software-Engineering-Methoden im laufenden Projekt zu erweitern und zu präzisieren sowie gegenüber dem Projektcontrolling abzugrenzen.

3.1.1 Analyse der Gemeinsamkeiten der Ansätze und weitere Herausforderungen

Wie sich in Abschnitt 2.4 und 2.5 gezeigt hat, ist eine der Schwächen der Ansätze, dass der Fokus zum einen nicht auf der Software-Engineering-Methode liegt. Zum anderen ist eine weitere Schwäche, dass die Dauer der Anpassung viel zu lang und sie somit ungeeignet ist. Ferner bieten die Ansätze keine oder nur wenige Möglichkeiten zur Automatisierung, um die Dauer der Anpassung zu verkürzen und eine selbst-adaptive Anpassung zu ermöglichen. Ferner passen zwei Ansätze eine Software-Engineering-Methode nur vor der Laufzeit an. Ansätze wie Six Sigma und die 8D-Methodik werden erst eingesetzt, wenn bereits Probleme aufgetreten sind. Gerade bei Six Sigma greifen Anpassungen häufig erst in anderen Projekten.

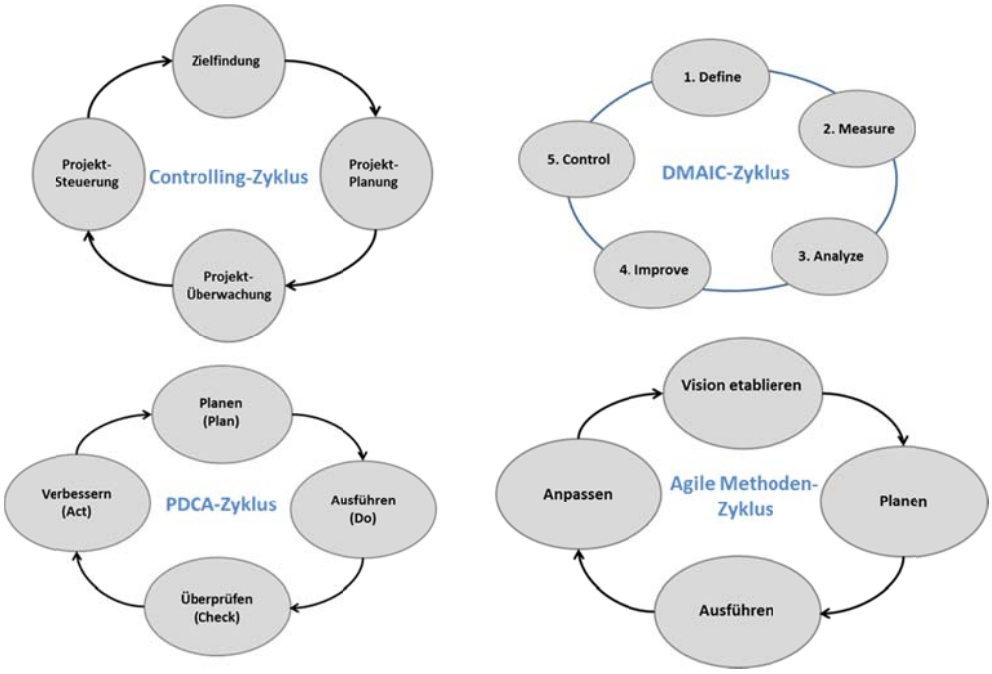

Abbildung 21 Die verschiedene Zyklen im Überblick

Doch wenn man sich gerade die Agilen Methoden, den ursprünglichen Controlling-Prozess und den kontinuierlichen Verbesserungsprozess anschaut, so fällt auf, dass diese Ansätze und auch Six Sigma größtenteils nach demselben Prinzip vorgehen und ein ähnliches Kernvorgehen besitzen. Diese vier Ansätze folgen alle wie in Abbildung 21 zu sehen einem Zyklus, der sich wiederholt oder bei Bedarf wiederholen kann (Six Sigma). Diese Zyklen folgen einerseits einem ähnlichen Schema, andererseits haben sie aber auch Unterschiede.

Ihnen ist gemeinsam, dass sie Ziele oder eine Vision entwickeln, um daran angelehnt die möglichen Änderungen zu planen. Im PDCA-Zyklus ist dies in der Phase „Planen“ und in Six Sigma in der Phase „Define“ enthalten. Es macht Sinn, Ziele und Rahmenbedingungen im Vorfeld zu definieren, um später eine Software-Engineering-Methode auf mögliche Abweichungen bezüglich dieser hin zu analysieren. Doch in den beschriebenen Ansätzen bezieht sich die Zielfindung auf ein Projekt, die Anforderungen an die Software (Agile Methoden) oder Erwartungen und Anforderungen des Kunden an das Projekt (Six Sigma). Es muss also herausgefunden werden, wie Ziele, Eigenschaften oder Erwartungen an eine Software-Engineering-Methode herausgefunden und beschrieben werden können.

Eine weitere Gemeinsamkeit der Ansätze ist, dass jeder Zyklus die Planung von Maßnahmen bzw. einer Anpassung besitzt, welche sich bis auf Six Sigma auf die Ziele bezieht. Zum einen ist es gut, dass mögliche Situationen

vorausgeplant werden. Doch zum anderen ist das Problem, dass die eigentliche Planung von Maßnahmen sich nicht auf den aktuellen Status bezieht. Es wäre somit sinnvoll, dass zunächst der aktuelle Status der Software-Engineering-Methode zur Laufzeit hinsichtlich einer möglichen Anpassung geprüft wird und anschließend eine Planung erfolgt, welche genau dieses Problem löst.

Bis auf den Agilen Methoden ist den Ansätzen ebenfalls allen gemeinsam, dass das Projekt oder Ergebnisse überprüft werden müssen. Im PDCA-Zyklus allerdings erfolgt dies zu einem späteren Zeitpunkt als in den anderen beiden Zyklen. Es ist also sinnvoll, dass es im neuen Ansatz ebenfalls möglich ist, die erfolgte Anpassung entsprechend überprüfen zu können.

Allen Zyklen ist ebenfalls eine Anpassungs- oder Verbesserungs-Phase gemeinsam, im Controlling-Zyklus befindet sich dies in der Steuerungs-Phase. Ebenso gibt es eine Ausführungs-Phase, beim PDCA-Zyklus im „Improve“ enthalten, im Controlling-Zyklus allerdings nur implizit. Die Ausführung befindet sich meistens nach der Planung und vor der Überwachung. Das heißt, dass die Planung ausgeführt und dann überprüft wird, bevor Maßnahmen zur Änderungen ergriffen werden.

Die Gemeinsamkeiten der Ansätze zeigen, dass es Sinn macht, ebenfalls einen Zyklus einzusetzen. Dieser überwacht kontinuierlich die Software-Engineering-Methode während der Durchführung, analysiert den aktuellen Status, plant Anpassungen und führt diese entsprechend aus. Die Frage ist hier, was die optimale Anordnung der verschiedenen Phasen in einem solchen Feedback-Zyklus wäre, denn bisher unterscheiden sich die verschiedenen Ansätze.

Ferner muss neben der Software-Engineering-Methode selbst auch die Projektumgebung mit betrachtet werden. Diese kann sich ebenfalls auf die Software-Engineering-Umgebung auswirken und Abweichungen erzeugen. Es ist allgemein zu entscheiden, ob und wann welche (Teil-) Prozesse der SEM geändert werden müssen.

Auch wenn Abweichungen einer Software-Engineering-Methode bekannt sind und eine Anpassung nötig ist, ist zu entscheiden, ob Methoden-Elemente ausgetauscht, gelöscht oder neue hinzugefügt werden müssen. Wie beispielsweise Method Engineering für existierende Projekte eingesetzt werden könnte, um u.a. diese Fragestellungen zu beantworten, steht als ein offenes Feld im State-of-the-Art Review [HSR10, S. 465]: „*How to use method engineering in the context of existing (legacy) methods (most of the SME literature assumes greenfield projects)*“

Wenn herausgearbeitet wurde, was und an welcher Stelle der Software-Engineering-Methode geändert werden muss, ist zu klären, wie und welche Methoden-Elemente ausgewählt werden müssen. Eine weitere Frage, die geklärt werden muss ist, wo diese Elemente herkommen und/oder ob diese im Ansatz bereits bekannt sind. Hier kann eine Technik aus dem Ansatz des Situational Method Engineering eingesetzt werden, die Methoden-Basis. Diese kann im Ansatz enthalten sein und entsprechende Elemente für die SEM bereithalten.

Damit eine Software-Engineering-Methode überhaupt untersucht werden kann, soll diese in einem Modell vorliegen (Typ-Ebene). Dabei ist es egal, ob das Modell mit Hilfe von SPEM, ISO 24744, MetaMe, einer Variante oder einer Vereinfachung davon erstellt wurde. Wichtig ist, dass konstant dieselbe Methode zur Modellierung bei der Nutzung des Ansatzes verwendet wird. Jedoch ist es möglich, das zwei verschiedene Unternehmen jeweils eine andere Möglichkeit zur Modellierung nutzen, diese anschließend aber bei der Durchführung konsequent nutzen.

Ebenso ist es nötig, dass neben dem ursprünglichen Modell der SEM eine aktuelle Instanz des Modells vorliegt (Instanz-Ebene). Dieses bildet den aktuellen Status der Software-Engineering-Methode während der Durchführung ab. Dies ist wichtig, um den aktuellen Status analysieren und später die Anpassung zur Laufzeit durchführen zu können. Zusätzlich müssen die Projekteigenschaften und das Umfeld bekannt sein, sowie die Änderungen, die geschehen sind. Eine Frage, die dabei zu klären wäre, ist, wo diese Elemente abgelegt werden.

Sind Methoden-Elemente ausgewählt, müssen sie noch mit der alten Software-Engineering-Methode zu einer neuen verschmolzen werden. Dies spielt sich sowohl auf der Typ-Ebene (neues Modell) als auf der Instanz-Ebene (weitere Durchführung der neuen SEM) ab. Es gibt verschiedene Techniken im (Situational) Method Engineering, wie eine Software-Engineering-Methode zusammengesetzt wird. Doch bevor dies erfolgt, sollten folgende Fragen betrachtet und beantwortet werden: Wie wirkt sich diese Änderung auf die gesamte Software-Engineering-Methode (Gesamt-SEM) aus? Wie sind die Abhängigkeiten zu anderen Methoden-Elementen während der Änderung, kommt es durch die Änderung an einer anderen Stelle zu Problemen? In Bezug auf die Methoden-Elemente selbst, sehen [HSR10, S.465] dies ebenfalls als eine noch offene Forschungsfrage an: „*How to ensure that the configuration of these selected method fragments is consistent and complete?*“

Um einen Ansatz entwickeln und die genannten Punkte und Anforderungen aus Abschnitt 2.5 erfüllen zu können, ist es wichtig zu wissen, wie eine

Software-Engineering-Methode überhaupt angepasst werden kann. Es muss bekannt sein, welche Elemente angepasst werden können und wie diese miteinander zusammenhängen. Diese verschiedenen Möglichkeiten zur Anpassung sollen im nächsten Abschnitt näher untersucht und erläutert werden.

3.1.2 Anpassungs-Arten

Um eine Software-Engineering-Methode anzupassen, gibt es verschiedene Möglichkeiten. Die Anpassungen finden typischerweise auf der Typ-Ebene statt, also am Modell der konkreten Software-Engineering-Methode selbst. Um mögliche Arten einer Anpassung herauszufinden ist es wichtig zu wissen, was die Bestandteile von einer Software-Engineering-Methode sind. Die möglichen Bestandteile sowohl von einem Projekt als auch von einer Software-Engineering-Methode wurden bereits in Abschnitt 2.1.1 beschrieben und in Abbildung 6 dargestellt. Dazu gehören u.a. Artefakte (Dokumente, Inkremente, Guidances), Aktivitäten, Rollen und ihre Verantwortlichkeiten, Techniken, Werkzeuge sowie die Reihenfolge von Aktivitäten, der Workflow. Dies sind alles Elemente, welche in einer Software-Engineering-Methode angepasst werden können. Nun wird im Folgenden beschrieben, wie und in welcher Art diese angepasst werden kann.

3.1.2.1 Einer Software-Engineering-Methode ein Element hinzufügen

Eine Möglichkeit, eine Software-Engineering-Methode anzupassen, ist es, ihr etwas hinzuzufügen. Dies können verschiedene Elemente sein, z.B.:

a. eine neue Aktivität bzw. Aufgabe,
b. eine neue Rolle,
c. ein neues Artefakt,
d. eine neue Technik,
e. ein neues Werkzeug,
f. eine neue oder weitere Verantwortlichkeit für eine der oben genannten Elemente,
g. eine zusätzliche Reihenfolge.

Des Weiteren kann es möglich sein, dass nicht nur ein neues oder ein weiteres Element der Software-Engineering-Methode hinzugefügt wird, sondern mehrere. Dies heißt genauer, dass eine Kombination von Elementen hinzukommt, zum Beispiel eine neue Aktivität, welche von einer neuen Rolle durchgeführt wird und/oder mit einer neuen Technik. Beispiele, wie die Anpassung durch das Hinzufügen ein oder mehrerer neuer Elemente aussehen kann, werden in Abschnitt 3.1.3 beschrieben.

3.1.2.2 In einer Software-Engineering-Methode ein Element löschen

Das Gegenteil der Anpassung einer Software-Engineering-Methode durch Hinzufügen eines Elements ist das Löschen eines Elements in der SEM. Beim möglichen Löschen handelt es sich um dieselben Elemente wie im vorherigen Abschnitt 3.1.2.1

Es kann ebenfalls eine Kombination von Elementen gelöscht werden, beispielsweise eine Aktivität mit den zugehörigen Rollen und Techniken. Sowohl beim Löschen von Elementen, als auch beim Hinzufügen von neuen Elementen ist es wichtig, die Abhängigkeiten zwischen den einzelnen Elementen zu kennen. Dadurch muss sichergestellt werden, dass die Anpassung einer Software-Engineering-Methode nicht den Gesamtfluss der vollständigen SEM stört. Ferner kann so herausgefunden werden, ob nur ein einzelnes Element oder eine Kombination von Elementen gelöscht bzw. hinzugefügt werden muss.

3.1.2.3 In einer Software-Engineering-Methode ein Element ersetzen oder ändern

Im Gegensatz zum Hinzufügen oder Löschen von Elementen in einer Software-Engineering-Methode kann es sein, dass ein Element ersetzt oder geändert werden muss. Dies kann primär bei folgenden Elementen der Fall sein:

a. Eine Rolle wird durch eine andere ersetzt oder die Rolle wird weiter spezialisiert. Dabei ist speziell zu beachten, dass in diesem Fall nicht nur die Person auf der Instanz-Ebene wechselt, z.B. aufgrund von Krankheit, sondern dass die Rolle selbst geändert wird. Beispiele können sein, dass die Rolle des Kunden durch einen „Kunden-Tester“ spezialisiert, oder dass die reine Rolle „Softwareentwickler“ durch die Rolle „Tester“ für eine Aktivität ersetzt wird.
b. Eine Aktivität ändert sich zum Beispiel vom Entwickeln eines Artefakts hin zum Testen eines Artefakts.
c. Eine Technik wird ausgetauscht: Hat z.B. das Schätzen von Aufwand nach der MuSCoW-Methode nicht den gewünschten Erfolg gebracht, dann wird sie durch eine andere Schätzmethode ersetzt. Oder einfaches Testen wird durch die Technik „Pair-Testing“ ersetzt oder erweitert.
d. Ein Artefakt kann durch ein anderes ersetzt werden.

Typischerweise bleiben beim Ersetzen oder Ändern eines Elementes die Abhängigkeiten und vor allem die Reihenfolge der Elemente bestehen. Trotzdem sollte überprüft werden, wenn beispielsweise eine Aktivität geän-

dert wird, ob die zugeordneten Rollen noch in der Lage sind, diese Aktivität auszuführen. Das heißt es gilt zu prüfen, ob die Rollen weiterhin die richtigen für die Aktivität sind oder ob sie ebenfalls geändert werden müssen. Ein Beispiel könnte sein, dass sich die Aktivität von „Entwickeln“ zu „Testen“ ändert. Dann müssten die Rolle „Entwickler“ eventuell zu „Tester“ geändert werden.

3.1.2.4 Die Reihenfolge – der Workflow – von Elementen wird verändert

Des Weiteren kann es möglich sein, dass während einer Anpassung einer Software-Engineering-Methode die Reihenfolge von Elementen geändert werden muss. Dies kann insbesondere der Fall sein bei:

a. der Erstellung von Artefakten,
b. der Durchführung von Aktivitäten,
c. der Kombination von a. und b.

Auch in diesen Fällen muss darauf geachtet werden, dass die Abhängigkeiten betrachtet und der Gesamtfluss der Software-Engineering-Methode durch die Änderung der Reihenfolge nicht gestört wird.

Im nächsten Abschnitt werden zwei komplexe Beispiele für Anpassungs-Arten genannt und kurz erläutert.

3.1.3 Beispiele für Anpassungen

Um in den beiden folgenden Beispiele besser verdeutlichen zu können, warum eine solche Anpassung nötig ist und wie diese erfolgen könnte, müssen einige Punkte und Fragen bei den Beispielen beachtet und beantwortet werden. Folgende Punkte und Fragen sind wichtig:

1. Damit eine Software-Engineering-Methode angepasst werden soll, muss ein Problem auftreten, welches bei Nicht-Behandlung (fatale) Folgen für den Erfolg des Projektes haben kann. Die Frage, die bei den Beispielen nun dahinter steht ist: *Was könnte ein Problem sein, damit z.B. etwas einer SEM hinzugefügt werden muss und wie könnte sich das Problem äußern?*
2. Der nächste Punkt, welcher sich direkt an den ersten anschließt ist die Frage, warum das Problem überhaupt ein Problem für die SEM, die Teammitglieder, den Erfolg des Projektes usw. ist. *Was hat es für Auswirkungen, wenn dieses Problem NICHT gelöst wird?* Auswirkungen könnten unter anderem hohe Kosten sein, da das Einhalten von Deadlines nicht mehr möglich ist oder im schlimmsten Fall läuft das Projekt „vor die Wand“.
3. Eine weitere zu beantwortende Frage kann sein, welche Eigenschaften der SEM erfüllt sein müssen oder welche Ziele von ihr erfüllt sein müs-

sen, die mit dem Problem verletzt sein würden. Genauer würde die zweigeteilte Frage lauten: *Welche Eigenschaft bzw. welches Ziel der SEM soll sichergestellt werden und würde mit dem auftretenden Problem verletzt?*

4. Nachdem das Problem genau geklärt ist im Beispiel, muss aber das Problem auch für die Anpassung möglichst früh (im Vorfeld) gefunden werden. Die ebenfalls zweigeteilte Frage lautet dann: *Was müsste an Regeln oder Grenzen eingehalten werden, damit das Ziel oder die Eigenschaft nicht verletzt wird? Wie kann dies im Vorfeld herausgefunden werden, beispielsweise anhand einer Messung (was müsste gemessen werden)?*
5. Sind die vorhergehenden Fragen beantwortet und durchgespielt, muss im Beispiel geklärt werden, wie eine Lösung für das Problem aussieht, das heißt: *Wie müsste eine Anpassung im Beispiel aussehen?*
6. Abschließend muss die fertige Anpassung in das Projekt übertragen und vor allem müssen alle Beteiligten benachrichtigt werden. Die Frage lautet also: *Wer muss bei der Anpassung alles benachrichtig werden?*

3.1.3.1 Beispiel „Quasi-Scrum“

Ein erstes Beispiel für die Anpassung einer Software-Engineering-Methode ist das Projekt „Quasi-Scrum“ [EG09]. In diesem Projekt aus dem s-lab – Software Quality Lab wurde eine Kreditkalkulationssoftware entwickelt mit Hilfe der Agilen Methode Scrum. Scrum wurde für das Projekt im Vorfeld bereits angepasst. Es gab beispielsweise keine täglichen Stand-Up-Meetings, sondern nur zwei- bis dreimal die Woche halbstündige Teammeetings. Ferner gab es keinen einzelnen Product Owner sondern einen Lenkungskreis. Die Review-Meetings waren keine Informationsmeetings, sondern Abnahmemeetings durch den Lenkungskreis und es war nie das ganze Team anwesend.

Das Problem in diesem Projekt war, dass die nicht-funktionalen Anforderungen und hier besonders die Performanz große Schwierigkeiten bereiteten. Dies äußerte sich zum einen dadurch, dass die Software unter hoher Last lange Antwortzeiten besaß. Zum anderen war der Kunde sehr unzufrieden und es gab viele Fehlermeldungen (Frage 1). Die Auswirkungen des Problems (Frage 2) waren einerseits, dass viele und schwerwiegenden Fehlern sowie ein Ausfall der Software auftraten. Andererseits wurde das Produkt durch den Kunden nicht abgenommen und ein „Reparatur-Sprint“ musste durchgeführt werden, dessen Kosten beim Anbieter lagen. Zwar wurden die Deadlines immer gehalten, aber unter Umständen war das Inkrement nicht vollständig und wie vorher beschrieben performancetechnisch fehlerhaft.

Es war somit wichtig, dass die Optimierung der Performanz der Software sichergestellt wird. Auf Scrum genau übertragen hieß das, dass die nicht-

funktionalen Eigenschaften der Software im Artefakt Product Backlog stehen mussten (Frage 3), was hier im Projekt zunächst nicht der Fall war. Die Betrachtung der nicht-funktionalen Eigenschaften ist häufig ein Problem in den Agilen Methoden, gerade auch bei Scrum. Der Fokus liegt typischerweise auf den Funktionalitäten einer Software.

Damit dies vorher hätte auffallen können, hätte zum einen das Product Backlog überwacht werden müssen, z.B. anhand einer Überprüfung auf Performanz- oder Optimierungs-Einträge. Zum anderen hätte der Kunde stärker im Vorfeld mit eingebunden werden müssen, beispielsweise anhand von durch ihn durchgeführter Tests. Ferner hätten zum Erkennen des Problems die Fehlereinträge des Kunden und somit seine Zufriedenheit (anhand der Fehler oder eingetragener Kommentare) überprüft oder gemessen werden können (Frage 4).

Die Lösung des Problems war in diesem Projekt die Einführung einer neuen Aktivität und neuer Rollen (Frage 5). Es wurden 1-3 sogenannte „Kunden-Test-Tage“ eingeführt, welche nach den Entwicklungstagen im Sprint und vor dem Review-Meeting lagen. Zusätzlich wurde als neue Rolle das „Kunden-Test-Team“ eingeführt, welches aus „Kunden-Tester“ bestand. Diese führten spezielle Tests durch, wodurch der Kunde zum einen mehr mit eingebunden wurde. Zum anderen wurden ihm die nicht-funktionalen Eigenschaften wesentlich bewusster und diese wurden regelmäßig ins Product Backlog eingetragen.

Die neue Aktivität wurde in einem neuen Sprint das erste Mal durchgeführt. Während des vorherigen Sprints wurden sowohl die neuen Kunden-Tester als auch das aktuelle Team über die Änderung informiert (Frage 6).

Dieses Beispiel kann auch analog für die nicht-funktionale Eigenschaft Usability übernommen werden. An den Kunden-Test-Tagen können hier Usability-Tests mit eingeplant werden.

3.1.3.2 Beispiel „Test-Sprint in Scrum“

In den Agilen Methoden bekommt das Testen zwar eine wichtige Rolle zugeschrieben, wird aber in der Originalliteratur wie beispielsweise dem ScrumGuide [SS13] nur sehr oberflächlich oder gar nicht beschrieben. Wird die Aktivität Testen beschrieben, ist das Vorgehen nur vage und nicht unbedingt strukturiert. Ferner soll das Testen von den Entwicklern selbst durchgeführt werden; die Rolle eines expliziten Testers in Scrum-Teams ist im Original nicht vorgesehen, streng genommen ist dies sogar verboten [SS13].

Auch wenn viel Wert auf Unit-Tests, kontinuierliche Integration und sogenannte User-Acceptance Tests [Glo11] gelegt wird, kann es durch das unstrukturierte Testen passieren, dass nicht ausreichend getestet wird oder ei-

nige Funktionen gar nicht getestet werden. Die Product Backlog Einträge haben evtl. einen Eintrag für einen Testfall oder einen Unit-Test, aber durch das Fehlen eines systematischen Vorgehens kann es zu einer fehlerhaften Software kommen, welche erst im Betrieb durch den Kunden entdeckt wird. Zwar sollen gerade die Agilen Methoden dies verhindern, aber so etwas wie Release- oder gar End-to-End-Tests sind in Scrum nicht vorgesehen (Frage 1).

Unsystematische Tests oder nicht durchgeführte, fehlende Tests können somit zu mehr Kosten aufgrund einer höheren Fehlerrate führen. Falls Fehler erst im Betrieb auftauchen, kann es passieren, dass durch die Bereinigung der Fehler aufgrund der Zeit Deadlines nicht eingehalten werden können. Zusätzlich kann die Bereinigung der Fehler weitere Kosten verursachen (Frage 2).

Es müsste somit überprüft werden, ob entweder zu jedem Backlog Eintrag verschiedene Testarten (nicht nur der Unit-Test) vorhanden sind, oder ob es eigene Test Backlog Einträge, wie in [GG12] vorgeschlagen, gibt. Ferner kann die Liste mit den Mengen an Fehlermeldungen und deren Kritikalität als Indikator genutzt werden, ob ausreichend Tests vorhanden sind (Frage 3 + 4).

Um dieses Problem nun zu lösen, kann wie in [GG12] vorgeschlagen, eine neue Aktivität, insbesondere für ein vollständiges Release, ein sogenannter Release-Sprint eingeführt werden. Zusätzlich zu diesem neuen Sprint, welcher sich gut in die Scrum-Methodik einbetten lässt, müssten entsprechende neue Artefakt definiert und der Software-Engineering-Methode hinzugefügt werden, zum Beispiel ein Test Backlog. Die sogenannten Daily Tasks im normalen Sprint, also die täglichen Aufgaben, werden durch entsprechende Test Tasks ersetzt und die täglichen Ergebnisse (Daily Results) werden durch einen täglichen Test Report in diesem Sprint ersetzt. Dieser Sprint wird von denselben Personen durchgeführt, allerdings diesmal in der Rolle der „Tester“. Unter Umständen ist es nötig, für diesen Sprint einen Testexperten/Test-Designer o.ä. mit einzubringen (Frage 5).

Bevor dieser neue Sprint durchgeführt wird, werden alle Teammitglieder und gegebenenfalls neue Tester oder Testexperten benachrichtigt (Frage 6).

3.1.4 Konkretisierung der Anforderungen und Abgrenzung

Die beschriebenen Anpassungsarten zeigen, wie komplex eine Anpassung im laufenden Projekt sein kann und welche verschiedenen Möglichkeiten es gibt. Die beiden komplexen Beispiele und besonders die vorher diskutierten Gemeinsamkeiten und Schwächen der Ansätze sowie zusätzlich definierte Herausforderungen zeigen, was alles bei der Anpassung selbst und ebenfalls im Vorfeld zu beachten ist. Zusätzlich mit den Bewertungskriterien aus Ka-

pitel 2.4 und diesen neuen Erkenntnissen wurden die vorherigen sechs Anforderungen an den eigenen Ansatz um sieben weitere Anforderungen erweitert. Die konkreten Anforderungen lauten somit:

A1. Der Fokus der Anpassung liegt auf der genutzten Software-Engineering-Methode.

A2. Der Ansatz soll ähnlich wie die bereits beschriebenen Ansätze einem kontinuierlichen Zyklus folgen und das Feedback der Umgebung mit einbinden.

A3. Der Anpassungszeitpunkt ist zur Laufzeit und die Dauer der gesamten Anpassung soll möglichst kurz sein.

A4. Die SEM soll kontinuierlich und möglichst eigenständig in Hinblick auf notwendige Abweichungen und Anpassungen auf der Instanz-Ebene beobachtet werden.

A5. Für diese kontinuierliche Beobachtung müssen sowohl die Umgebung als auch die aktuelle Software-Engineering-Methode selbst überwacht werden können. Das heißt, es müssten konkrete Daten, welche für die Überwachung notwendig sind, gemessen werden.

A6. Die beobachteten Werte, also der aktuelle Status, muss analysiert und schnell beurteilt werden können.

A7. Die Werte, aber auch das Modell der Software-Engineering-Methode, müssen an einem zentralen Ort abgelegt und es muss darauf zugegriffen werden können.

A8. Eine unmittelbare Anpassung der Software-Engineering-Methode zur Laufzeit muss unter Betrachtung von vorher definierten Qualitätszielen geplant werden können.

A9. Es muss entschieden werden, wie die Anpassung erfolgt: ob ein Element ausgetauscht wird, ein Element gelöscht wird usw. Es muss ferner mit betrachtet werden, ob die Anpassung eine Auswirkung auf die Gesamtmethode hat. Die Anpassung muss anschließend am Modell auf der Typ-Ebene entsprechend durchgeführt werden.

A10. Die Anpassung muss von der Typ-Ebene in das laufende Projekt, also auf die Instanz-Ebene, zurückübertragen werden können.

A11. Die Anpassung soll schnellstmöglich und zeitnah, sowie möglichst automatisch und somit eigenständig erfolgen.

A12. Die Anpassung muss im laufenden Projekt evaluiert werden können.

A13. Die Ergebnisse der Anpassung sollen für weitere Projekte wiederverwendet werden können.

Der zu entwickelnde Ansatz zur Anpassung einer Software-Engineering-Methode weist Ähnlichkeiten zu den bestehenden Ansätzen und vorwiegend zum Projektcontrolling auf. Doch das Projektcontrolling selbst hat, wie schon in Abschnitt 2.4 und 2.5 beschrieben, einen ganz anderen Fokus – das Projekt selbst.

Im Projektcontrolling werden im Gegensatz zum entwickelnden Ansatz hauptsächlich die Facetten Einhaltung des Budgets, Einhaltung des Zeitplans und Einhaltung der (Sach-)Ziele betrachtet und überwacht. Dabei wird weniger genau auf die Software-Engineering-Methode selbst geschaut, was hier besonders wichtig ist. Es kann unter Umständen vorkommen, dass etwas an der Software-Engineering-Methode geändert wird, doch dies ist eher ad hoc und unstrukturiert.

Im zu entwickelnden Ansatz sollen zwar die im Projektcontrolling überwachten Ziele enthalten sein, aber es sollen vor allem die eigentlichen Ziele, Eigenschaften und Elemente der Software-Engineering-Methode betrachtet werden. Die Anpassung erfolgt zeitnah, strukturiert und wird im Modell der Software-Engineering-Methode abgebildet. Ferner ist die Anpassung durch den zu entwickelnden Ansatz von kürzerer Dauer und soll wenn möglich automatisiert und selbst-adaptiv durchgeführt werden.

Es ist sowohl dem Team als insbesondere auch dem Projektmanager mit Hilfe des Ansatzes möglich, Probleme innerhalb der Software-Engineering-Methode rechtzeitig zu erkennen und diesen rechtzeitig entgegen zu steuern. Mit dieser Unterstützung können die Ziele des Projektmanagements und somit des Projektcontrollings eingehalten werden.

Für die Entwicklung des Ansatzes wurde nach einem weiteren, ähnlichen Zyklus gesucht, welcher die Anforderungen abbildet. Der Zyklus bzw. die Zyklen, welche die Anforderungen und die Gemeinsamkeiten der bestehenden Ansätze ebenfalls abbilden, sind die Feedbackschleifen aus den selbstadaptiven Systemen, welche im nächsten Abschnitt näher vorgestellt werden.

3.2 Feedbackschleifen aus den selbst-adaptiven Systemen

Wie in Abschnitt 3.1 beschrieben, wird für Anpassungen, Veränderungen oder Verbesserungen immer wieder ein Zyklus in verschiedenen Ausprägungen verwendet. Am Ende des Abschnitts wurden Anforderungen definiert, welche ein Zyklus zur Anpassung im Projekt erfüllen muss. Durch diese Anforderungen wurde eine weitere Reihenfolge für den Ablauf des Zyklus vorgeschlagen.

In den selbst-adaptiven Systemen werden zur Überwachung und Anpassung von Systemen ebenfalls Zyklen verwendet, die sogenannten Feedbackschleifen. Diese überwachen automatisch ein gegebenes System und geben entsprechend Feedback, ob ein System möglicherweise angepasst werden muss oder nicht. Diese Feedbackschleifen sollen neben selbst-adaptiven Systemen im Folgenden vorgestellt werden, dabei speziell die bekannte Feedbackschleife MAPE-K, welche den gestellten Anforderungen sehr nahe kommt.

3.2.1 Selbst-adaptive Systeme und selbst-adaptive Software

Um mit der immer größer werdenden Komplexität von Software-Systemen umgehen zu können und Herr über die sich ständig ändernde Umgebung zu werden, haben Softwareentwickler und -techniker angefangen, sich mit Selbst-Adaptivität von Systemen zu beschäftigen [Br09]. Dabei haben sie sich in anderen verwandten Gebieten umgeschaut, beispielsweise in der Robotik oder der Künstlicher Intelligenz, um neue Wege für das Design von Systemen und Software zu finden. Selbst-adaptive Systeme können auf eine sich ändernde Umgebung und neue Anforderungen reagieren, welche zum Zeitpunkt des Designs noch nicht bekannt waren. „Selbst“ (engl. self) heißt in diesem Kontext, dass das System oder die Software selbstständig, also autonom und ohne oder nur mit minimalen (menschlichen) Einfluss von außen entscheiden kann, ob und wie auf Änderungen in der Umgebung reagieren und sich gegebenenfalls anpassen kann [Ch09].

Aus einem ähnlichen Grund sind die Agilen Methoden in Bezug auf Software-Engineering-Methoden bzw. die Entwicklung von Software(-Produkten) entstanden. Sie sollen ebenfalls auf sich ändernde Anforderungen an die Software zeitnah reagieren können, da nicht alle Anforderungen zu Beginn bekannt sind oder der Kunde plötzlich eine Änderung haben möchte.

Wissenschaftler und Entwickler haben sich viele Gedanken darüber gemacht, wie selbst-adaptive Systeme aussehen sollten, um eine große Bandbreite bei der Anpassung abdecken zu können, unter anderem die Performanz oder Sicherheit von Systemen. Gerade die Sicherheit ist bei sicher-

heitskritischen Systemen besonders wichtig. Aber auch das Fehlermanagement kann mit selbst-adaptiven Systemen gut adressiert werden [KC03].

Normalerweise befinden sich Software-Systeme wie beschrieben in einer sich andauernd ändernden Umgebung. Sie benötigen Personen, die sich mit möglichst allen Faktoren auseinandersetzen und alle Möglichkeiten „vorhersehen“, die passieren können. Ferner müssen diese Personen Änderungen bemerken und rechtzeitig agieren, damit die Systeme einwandfrei weiterlaufen. Doch gerade diese Aufgaben sind nicht nur kostspielig und sehr zeitaufwendig, sondern können kaum adäquat von Personen abgedeckt werden [ST09].

Gerade ein immer größeres Verlangen nach weiteren Automatisierungen, nach Robustheit oder Qualitätssicherung innerhalb des Budgets und innerhalb einer bestimmten Zeit konnte bis dato kaum erfüllt werden. Die selbstadaptiven Systeme aber auch selbst-adaptive Software, welche nach demselben Prinzip funktioniert, waren die Antwort auf diese Probleme.

Die Lösungen sind sogenannte „Closed-Loop-Systeme“, die eine Feedbackschleife enthalten mit dem Ziel, Veränderungen rechtzeitig zu erkennen und sich selbstständig anzupassen. Diese Veränderungen können durch die Software oder das System selbst intern ausgelöst werden oder aber auch durch den Kontext, in dem sie sich befinden, z.B. durch externe Events wie mehr User Reqests [ST09]. Um das Ziel zu erfüllen, müssen sich die Systeme oder die Software und ihre Umgebung selbstständig überwachen, signifikante Änderungen entdecken, entscheiden können, wie sie reagieren sollen und diese Entscheidungen entsprechend ausführen. Dies erfolgt alles automatisch zur Laufzeit.

Was selbst-adaptive Systeme und Software somit alle gemeinsam haben ist, dass die Änderungen und die Entscheidungen, wie reagiert werden soll alle automatisch und zur Laufzeit durchgeführt werden müssen. Dafür muss ihnen ihr aktueller Zustand, aber auch der Zustand ihrer Umgebung bekannt sein. Dies führt zu den vier Schlüssel-Aktivitäten im Feedback-Prozess [KC03, BR09, CH09,ST09]:

1. Sammeln (Collect)
2. Analysieren (Analyze)
3. Entscheiden (Decide) und
4. Handeln bzw. Ausführen (Act)

Diese Schlüssel-Elemente finden sich typischerweise in den Feedbackschleifen wieder, welche im folgenden Abschnitt näher beschrieben werden.

3.2.2 Allgemeine Beschreibung von Feedbackschleifen

Um das dynamische, sich immer ändernde Verhalten von selbst-adaptiven Systemen ansatzweise kontrollieren zu können, müssen sie ständig eine Rückmeldung vom aktuellen Stand des Systems und seiner Umgebung bekommen. Die Systeme brauchen dies Wissen um herauszufinden, ob und welche Entscheidungen zu einem bestimmten Zeitpunkt getroffen werden müssen.

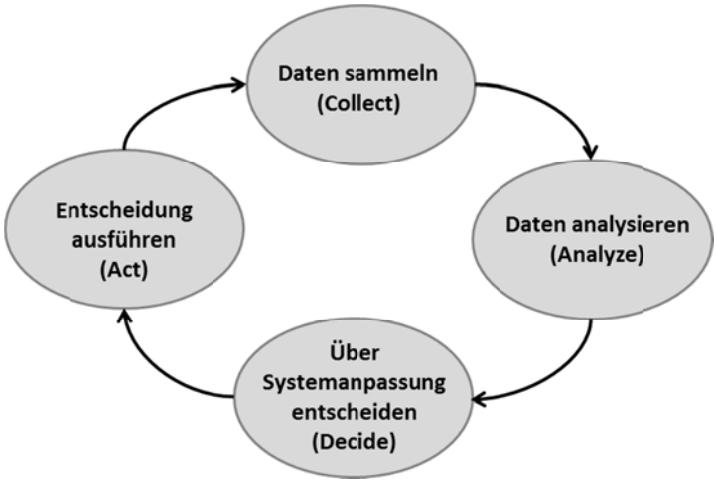

Abbildung 22 Schlüsselfaktoren der allgemeinen Feedbackschleife basierend auf [Do06, Ch09, Br09]]

Eine bewährte Möglichkeit für ein solches Feedback mit entsprechender Reaktion sind die sogenannten Feedbackschleifen (engl. Feedback-Loops). Die vier Schlüsselfaktoren für eine allgemeine Feedback-Loop sind wie in Abbildung 22 zu sehen und bereits im vorherigen Abschnitt beschrieben [Br09, Ch09, Do06]

- das Sammeln von Daten über das System und den Kontext („Collect“),
- das Analysieren dieser Daten („Analyze“),
- das Entscheiden, ob das System oder die Software angepasst werden soll („Decide“),
- das Handeln bzw. Ausführen der Entscheidung („Act“).

Mit der Zeit haben sich neben der allgemeinen Feedbackschleife verschiedene Ausprägungen entwickelt, u.a. das Rainbow-Framework von [GCS03] in 2003 und später in 2007 die Shaw-Feedback-Loop [MPS08].

Das Rainbow-Framework unterteilt dabei wie in Abbildung 23 zu sehen in das zu überwachende System auf der einen und einem überwachenden Architektur-Manager auf der anderen Seite. Anhand von Monitoring-

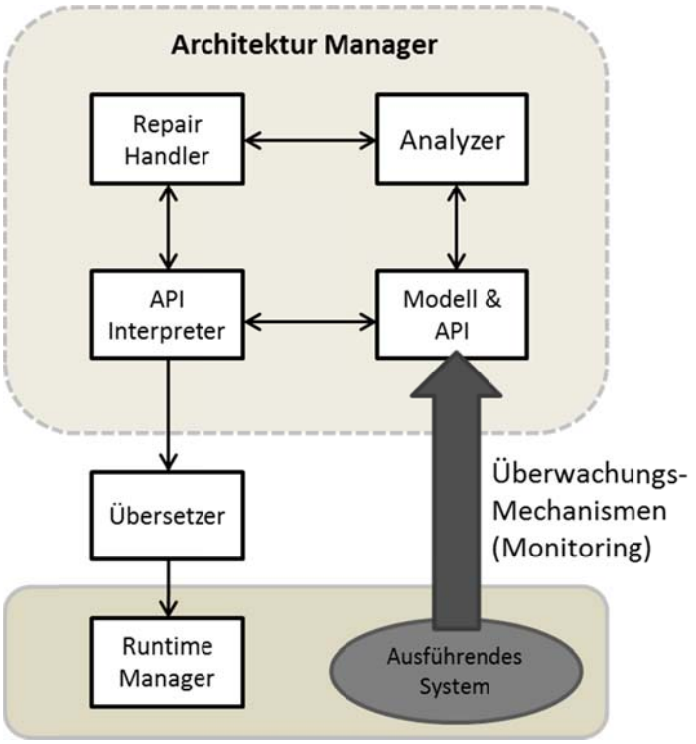

Abbildung 23 Rainbow-Framework basierend auf [GCS03]

Mechanismen wird das System überwacht und diese Daten in den „Architektur Manager“ übertragen, wo sich das Modell des Systems und die API befinden. Anhand der übertragenen Daten und des Modells werden die Daten auf Abweichungen hin im Analyzer analysiert und falls nötig wird im „Repair Handler“ eine Anpassung geplant. Diese wird über den „API Interpreter“ und den „Übersetzer“ zurück in das System übertragen, in diesem Fall dem „Runtime Manager“.

Auch in der Shaw-Feedback-Loop nach [MPS08] werden wie in Abbildung 24 zu sehen das ausführende System sowie die Betriebsumgebung von der eigentlichen Feedbackschleife abgekapselt. Daten-Stichproben werden sowohl von der Umgebung als auf vom System selbst gemessen und an die Feedbackschleife übertragen. Zusätzlich werden Prognosen gesammelt, wie sich alles voraussichtlich weiter entwickelt. Ähnlich wie im Rainbow-Framework werden die Daten anhand eines aktuellen und eines mit den bekannten Daten erstellten zukünftigen System-Modells analysiert und miteinander verglichen.

In diesen Vergleich fließen für die Entscheidung, ob Korrekturen notwendig sind, zusätzlich die Zielvorstellungen, welche anfangs erstellt worden sind, mit ein. Diese werden anschließend in einem nächsten Schritt geplant und die Korrekturen werden mit Hilfe von Kommandos in das System zurück überführt.

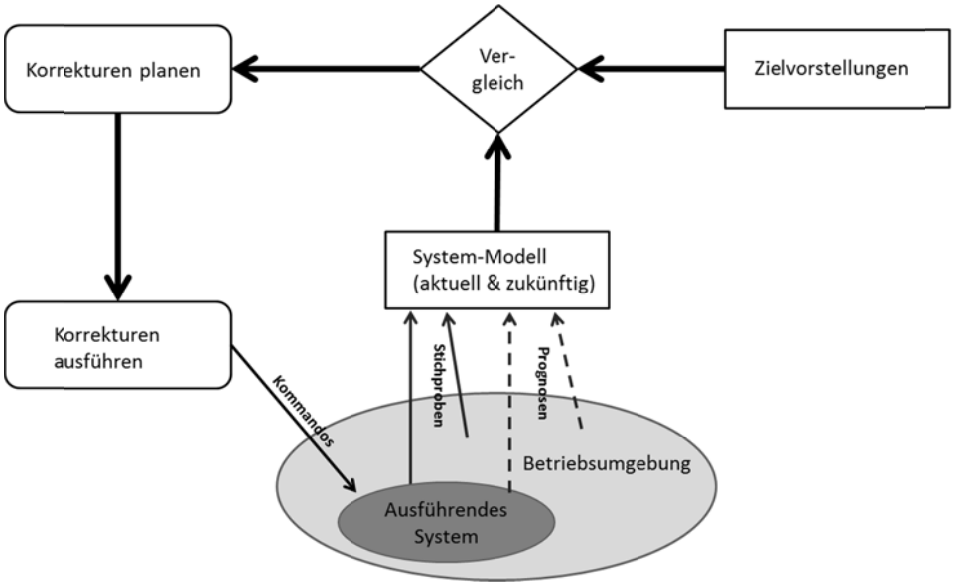

Abbildung 24 Shaw-Feedback-Loop basierend auf [MPS08]

Ähnlich diesen beiden Modellen hat sich eine der bekanntesten Feedbackschleifen, die MAPE-K-Feedbackschleife [Br09] entwickelt, welche mittlerweile größtenteils bei selbst-adaptiven Systemen eingesetzt wird. Diese wurde von Kephart und Chess [KC03] entwickelt und durch IBM populär [IBM06] gemacht. Die Feedbackschleife besteht aus vier Phasen, welche die vier Schlüsselfaktoren abbildet und diese wie in der Shaw-Feedback-Loop um eine Planung erweitert: Monitor (Collect), Analyse (Analyze + Decide), Plan und Execute (Act). Zusätzlich gibt es eine Wissensbasis, die „Knowledge Base“, wo beispielsweise die Modelle ähnlich wie im Rainbow-Framework und in der Shaw-Feedback-Loop gespeichert werden. Die Anfangsbuchstaben der Phasen und der Knowledge Base führen letztendlich zu dem Namen „MAPE-K“. Ebenfalls sind die Phasen des MAPE-K vom System abgekapselt und werden „Autonomic Element“ genannt. Diese Feedbackschleife wird im folgenden Abschnitt genauer unter die Lupe genommen.

3.2.3 Die Feedbackschleife MAPE-K

Das “Autonomic Element”, oder auch Kontroll- bzw. Feedbackschleife genannt, wurde von Kephart und Chess [KC03] als weitere Architektur für selbst-adaptive System mit einer expliziten Feedback-Loop vorgestellt. Diese Architektur, MAPE-K genannt, wurde populär durch den “Architectural blueprint for autonomic computing” von IBM [IBM06].

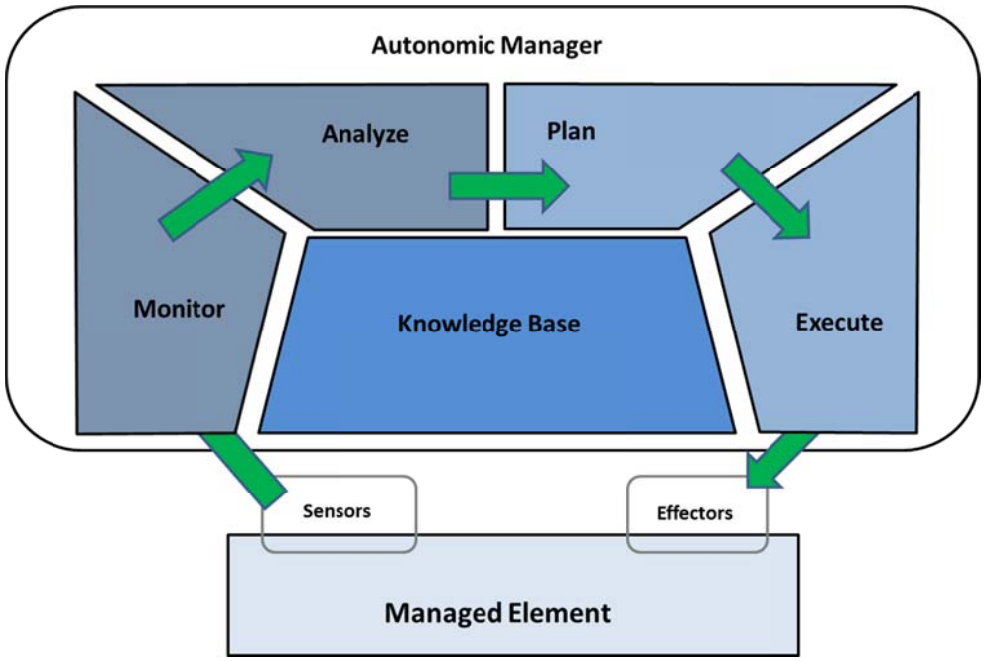

Abbildung 25 Autonomic element - MAPE-K feedback loop nach [KC03]

Das Autonomic Element besteht wie in Abbildung 25 zu sehen aus einem Element (dem System), welches gemanagt wird und aus dem „Autonomic Manager“ selbst, dessen Kern die Feedbackschleife ist. Damit eine System-Komponente sich nun selbst managen kann, muss sie eine (automatisierte) Methode haben [IBM06], welche

- Details sammelt, die sie vom System und seiner Umgebung braucht,
- diese Details analysiert um zu entscheiden, ob etwas geändert werden muss,
- einen Plan oder eine Reihe von Aktionen erstellt, welche die nötigen Änderungen spezifiziert und
- um diese Aktionen auszuführen.

Eine solche Methode ist die Implementierung des Autonomic Managers. Diese besteht wie in Abbildung 25 zu sehen aus vier Teilen, dem Monitor (M) oder Monitor-Phase, einem Analyzer (A) oder der Analyse-Phase, einem Planer (P) der sogenannten der Plan-Phase und einem Executer (E), der Execute-Phase. Alle vier Phasen teilen sich eine gemeinsame Wissensbasis (Knowledge Base, K) [Br09, IBM06]. Zusätzlich besitzt der Autonomic Manager noch zwei Schnittstellen. Zum einen die Sensoren, welche die entsprechenden Daten sammeln und zum anderen die Effektoren oder auch Aktuatoren, welche die Aktionen am Ende ausführen. Der „Architectural blueprint“ von IBM bietet ausführliche Beschreibungen an, wie die vier Phasen, die Wissens-Basis, die Sensoren und Effektoren/Aktuatoren implementiert werden können [IBM06, Br09].

In der Monitor-Phase sammelt der Monitor verschiedene Daten über das System, den Prozess, in dem es sich befindet, sowie über dessen Kontext und Umgebung. Diese Sensordaten werden gefiltert und akkumuliert, wofür es verschiedene Methoden und Möglichkeiten gibt. Abschließend werden die Daten und relevante Ereignisse in der Wissensbasis für die Zukunft gespeichert.

Der Analyzer vergleicht in der Analyse-Phase die Daten der Ereignisse mit verschiedenen Mustern aus der Wissensbasis und/oder modelliert komplexe Situationen, um mögliche Situationen in der Zukunft vorhersagen zu können.

Der Planer interpretiert in der Planungsphase die Daten und Situationen. Er erstellt mögliche Aktionen, die ausgeführt werden, um einen bestimmten Status oder gesetzte Ziele des Systems erreichen zu können. Dieser Plan wird wie beschrieben in der Execute-Phase durch den Executer und über die Effektoren ausgeführt.

Auch wenn es durch den Autonomic Manager möglich ist, die verschiedenen Phasen der Feedbackschleife zu automatisieren, möchten professionelle IT'ler manchmal nur einige Teile der potentiell automatisierten Phasen an ihn delegieren [IBM06]. In Abbildung 26 sind die vier verschiedenen Profile (Monitor, Analyse, Plan und Execute) des Autnomic Managers zu sehen. Ein Administrator möchte beispielsweise nur die Monitor-Phase automatisieren und durch den Autonomic Manager ausführen lassen. Das Ergebnis der Monitor-Phase, die gesammelten und vielleicht schon gefilterten und akkumulierten Daten möchte er dann lieber an andere Konsolen oder auch Menschen delegieren, als sie zu automatisieren, z.B. aufgrund von Erfahrungswerten etc. Natürlich können auch andere Teile zusätzlich automatisiert werden. Durch diese Aufsplittung der Phasen sind die verschiedensten Kombinationen möglich.

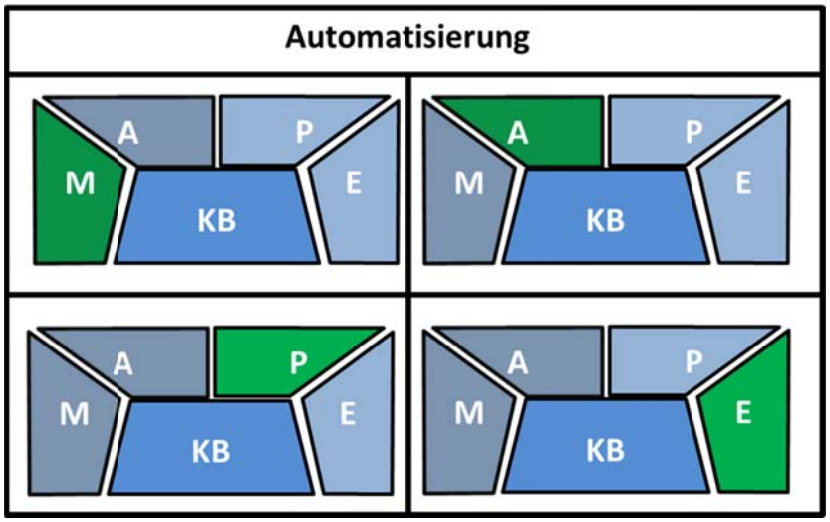

Abbildung 26 Aufsplittung der Automatisierung der einzelnen Phasen im MAPE-K nach [IBM06]

Wie vorher beschrieben, finden die vier Phasen über einer Wissensbasis statt. Diese Wissensbasis kann eine Implementierung einer Registrierung, eines Wörterbuchs, einer Datenbank oder eines Repositories sein oder eine Kombination aus allem. Zumindest stellt er den Zugriff auf „Wissen“ sicher, abhängig von den Interfaces der Architektur [IBM06]. In autonomen Systemen, besteht das Wissen aus bestimmten Typen von „management data“ mit einer bestimmten Syntax und Semantik, wie beispielsweise Symptome, Policies, Anfragen zur Änderung (Requests of Change) und Änderungspläne. Die gespeicherten Daten werden von den verschiedenen Phasen im Autonomic Manager genutzt. Zusätzlich kann die Wissensbasis Daten wie Informationen über und durch Topologien enthalten, als auch historische Logs oder verschiedene Metriken [IBM06].

Der Autonomic Manager und seine Feedbackschleife MAPE-K sind (bisher) für autonome Systeme und Systemkomponenten beschrieben. Doch bei genauer Betrachtung erfüllt diese Feedbackschleife genau die Anforderungen an unseren Ansatz für die Anpassung von Software-Engineering-Methoden.

Wie würde es nun aussehen, wenn das System durch eine Software- Engineering-Methode ersetzt würde? Ist es möglich, diese Feedbackschleife für eine selbst-adaptive Anpassung von einer SEM zur Laufzeit zu nutzen? Zunächst müsste überprüft werden, ob und wie sich die einzelnen Phasen und Funktionen darauf abbilden lassen und was sowohl die verschiedenen Phasen als auch die Wissensbasis, Sensoren und Effektoren enthalten müssten.

3.3 Beschreibung der Konzeption eines SE Method Managers

Eine Software-Engineering-Methode soll zur Laufzeit kontinuierlich überwacht und sich möglichst automatisch anpassen. Im hier vorgestellten Ansatz soll dazu der Ablauf des MAPE-K auf Software-Engineering-Methoden übertragen werden. Eine der entscheidenden Fragen ist dabei, wie die Anpassung einer Software-Engineering-Methode mit dem MAPE-K kombiniert werden kann; ist es möglich alles auf die entsprechenden Phasen abzubilden und wie müssten diese dann aussehen?

Um eine Software-Engineering-Methode mit Hilfe des MAPE-K überwachen zu können ist es wichtig zu bestimmen, was genau auf der Instanz-Ebene gemessen werden kann. Mit Hilfe der Sensoren ist es möglich, „harte“ Faktoren der Software-Engineering-Methode zu messen. Damit sind solche Faktoren gemeint, die sich direkt in Listen, in Tabellen, im Code, in Algorithmen, in Modellen usw. erfassen und auswerten lassen. Diese können mit Hilfe von Metriken die zu messenden Werte bestimmen und sie anschließend mit Hilfe von Sensoren für die weitere Verarbeitung erfassen.

Schwierig wird es bei den „weichen“ Faktoren. Damit ist beispielsweise gemeint, wenn sich zwei Personen im Team nicht verstehen, die Kommunikation untereinander nicht funktioniert, Missverständnisse entstehen usw. Dies sind Faktoren auf der psychologischen, kulturellen und zwischenmenschlichen Ebene, welche zwar Auswirkungen auf die Software-Engineering-Methode und somit auf den Erfolg des Projektes haben. Weiche Faktoren lassen sich jedoch nur schwierig bestimmen. Es lassen sich maximal die Auswirkungen solcher Faktoren beobachten und messen, wenn diese sich in den harten und somit messbaren Faktoren niederschlagen.

Auch das Projektcontrolling sagt nach [He13], dass ein Projektmanager solche strukturellen Konflikte zwar nicht grundsätzlich verhindern kann, aber er kann im Voraus versuchen diese zu reduzieren. Dies ist beispielsweise möglich, in dem er Projektrollen so definiert, dass dabei möglichst wenige Personen im selben Fachgebiet arbeiten oder dass jede Person in möglichst wenigen Fachgebieten arbeitet. Diese Aspekte können wiederum in Messwerte umgewandelt werden, zum Beispiel anhand der Qualifikationen und Zugehörigkeiten der Personen und Rollen, welche in einem Dokument festgelegt sind.

Wenn nun entsprechend die weichen Faktoren soweit wie möglich in harte Faktoren umgewandelt werden, ist eine weitere wichtige Frage, wie sichergestellt wird, dass das Richtige gemessen wird. Das heißt, dass die korrekten Werte erfasst und ausgewertet werden. Wie dies im Ansatz sichergestellt wird, beschreiben Abschnitt 4.3 und ausführlicher Abschnitt 5.3.

Um nun die Software-Engineering-Methode beobachten zu können und entsprechende Messwerte zur Auswertung zu erhalten, muss die Software-Engineering-Methode und ihre Instanziierung mit den entsprechenden Werten im Projekt zunächst selbst abgebildet werden. Dies kann hier über das Managed Element erfolgen, welches sonst das zu verwaltende System darstellt. Die resultierende „Managed SE-Methode“ muss somit alle aktuellen Daten der instanziierten Software-Engineering-Methode beinhalten, welche kontinuierlich gemessen und überwacht werden können. Daten können dabei u.a. die aktuell erstellten Artefakte, die durchgeführten Aktivitäten, die konkret ausfüllten Rollen, eingesetzte Techniken, durchgeführte Meetings usw. sein.

Anhand der Sensoren werden nun die entsprechenden Daten gemessen und in der Monitor-Phase anschließend aufgezeichnet, für die Analyse aufbereitet sowie zusätzlich in die Wissensbasis geschrieben. Anhand der Sensoren und der Monitor-Phase wäre somit sichergestellt, dass die Software-Engineering-Methode aber auch ihre Umgebung kontinuierlich überwacht werden, wie in der Anforderung A5 gefordert.

In der Analyse-Phase werden die gemessenen und aufbereiteten Daten ausgewertet. Die Daten spiegeln den aktuellen Status der Software-Engineering-Methode wider, den Ist-Zustand. Dieser wird mit vorher definierten Werten, also einem Soll-Zustand, verglichen. Falls die gemessenen Daten den Soll-Zustand nicht erfüllen, das heißt von den im Vorfeld definierten Werten beispielsweise durch Über- oder Unterschreiten abweichen, wird die Plan-Phase angestoßen, um eine Anpassung zu planen. Die Auswertung und Analyse des aktuellen Zustandes erfolgt dabei mit Hilfe der Wissensbasis.

Hat die Analyse-Phase nun ergeben, dass die Software-Engineering-Methode angepasst werden muss, werden in der Plan-Phase eine Anpassung und mögliche Varianten geplant. Dafür können verschiedene Methoden-Elemente aus einer Methoden-Basis ausgewählt werden. Diese Elemente können gegen Elemente in der aktuellen Software-Engineering-Methode ausgetauscht, gelöscht oder ihr hinzugefügt werden. Anschließend soll die Konsistenz der Anpassung in Hinblick auf die gesamte SEM überprüft werden, vor allem ob diese Anpassung an einer anderen Stelle Konflikte auslöst. Ist dies der Fall, muss eine Alternative geplant werden.

Als letztes wird die Anpassung in der Execute-Phase über Effektoren ausgeführt, das heißt die angepasste SEM wird in das aktuelle Projekt übertragen und alle Beteiligte müssen entsprechend benachrichtig werden. Mit einer solchen Anpassung wäre zum einen erfüllt, dass die Anpassung während des laufenden Projektes stattfindet (Anforderung A3). Zum anderen erfolgt die

Anpassung nicht nur während des Projektes, sondern zeitnah und immer dann, wenn ein Problem auftritt. Letztgenanntes sollte möglichst automatisch geschehen (Anforderung A11).

3.3.1. Aufbau und Beschreibung des SE Method Manager

Nachdem erörtert wurde, dass eine Abbildung auf den MAPE-K möglich ist, wird in Abbildung 27 eine detailliertere Ausarbeitung für einen solchen „SE Method Manager“ gezeigt.

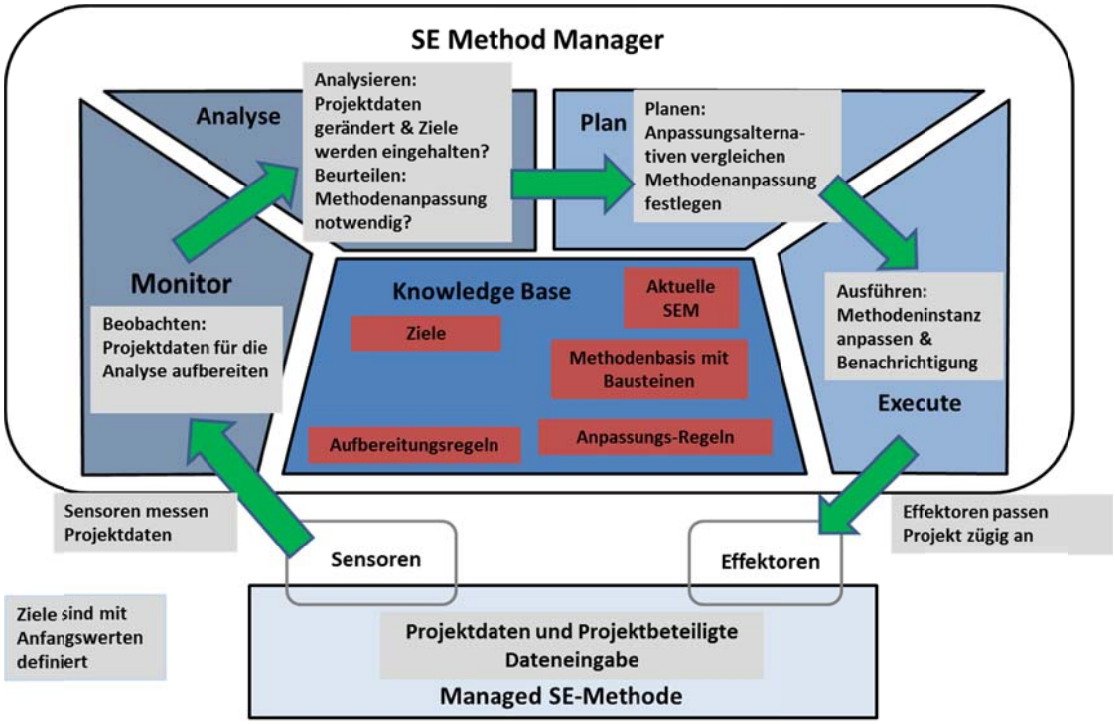

Abbildung 27 Aufbau des SE Method Managers

Das bisherige Managed Element ist hier wie bereits beschrieben nicht ein System, sondern die Software-Engineering-Methode selbst. Dieses Element, die „Managed SE-Methode“ besteht aus dem aktuellen Projekt mit den aktuellen Projektdaten sowie allen Projektbeteiligten und über das Managed Element erfolgt die Dateneingabe. Hier wird die aktuelle Software-Engineering-Methode ausgeführt und die Projektbeteiligten geben beispielsweise Daten ein, erzeugen Artefakte, dokumentieren Meetings usw.

Die beschriebene Wissensbasis beinhaltet eine Informationsbasis mit allen Informationen, die für eine Anpassung der Software-Engineering-Methode notwendig sind. Dies sind definierten Ziele, Aufbereitungsregeln, welche in der Monitor-Phase genutzt werden, sowie Regeln zu ihrer Anpassung. Außerdem enthält die Wissensbasis eine Methoden-Basis mit verschiedenen Bausteinen für eine Software-Engineering-Methode. Dies können sowohl Bausteine für eine agile als auch eine klassische SEM sein. Abschließend

enthält die Wissensbasis ein Modell der aktuellen Software-Engineering-Methode selbst.

Das Modell ist sehr wichtig, denn an diesem werden die Änderungen durchgeführt, beispielsweise das Hinzufügen eines Artefakts, die Änderung von Rollen, das Austauschen einer Technik usw. Die Modellierung einer SEM kann beispielsweise über SPEM oder MetaME erfolgen, aber auch eine einfache Form eines Modells kann gegeben sein. Wichtig ist, dass alle Artefakte, Rollen, Aktivitäten, Techniken und der Workflow sowie die Abhängigkeiten untereinander modelliert sind. Nur so kann eine Anpassung vorgenommen werden. Die verschiedenen Elemente der Wissensbasis sollten vor der Projektdurchführung erstellt werden. Durch das vorherige Erstellen vermindert sich der Aufwand während der Durchführung.

Ähnlich wie die Modelle sind die Sensoren, welche u.a. automatisch je nach Phase anhand von Regeln ausgewählt werden können, bereits vorher vorhanden. Diese Sensoren erfassen bestimmte Werte über beispielsweise das Eintragen in Programme oder Listen durch die Projektmitarbeiter. Das Eintragen der Werte würde nur einen geringen Mehraufwand für die Mitarbeiter bedeuten, da dies in ihren geregelten Tagesablauf mit eingebettet ist.

In der Monitor-Phase werden nun wie vorher beschrieben die eingegebenen Daten beobachtet und für die Analyse aufbereitet. Dies erfolgt über vorher definierte Aufbereitungsregeln mit Hilfe der Wissensbasis. Zum Beispiel ist für eine bestimmte Auswertung die Teamgröße wichtig. Die Sensoren messen kontinuierlich die Anzahl der Personen pro Team. In der Monitorphase wird dies insofern aufbereitet, als dass die Anzahl der Personen pro Team gleich der Teamgröße gesetzt wird.

In der Analyse-Phase wird zu Beginn ausgewertet, ob sich Projektdaten geändert haben. Im positiven Fall, wird überprüft, ob vorher gesetzte Ziele und Grenzwerte für diese Projektdaten weiterhin eingehalten werden. Ist dies der Fall, wird die Software-Engineering-Methode weiter fortgeführt und die Phase Plan wird nicht aufgerufen. Werden die Werte über- oder unterschritten, soll die SEM angepasst werden und die Plan-Phase wird mit den zu ändernden Daten aufgerufen.

In der Plan-Phase wird nun die Änderung der Software-Engineering-Methode geplant. Wie in 3.1.2. kann es mehrere Möglichkeiten für die Anpassung einer Methode geben. Je nach Änderungsbedarf wird ein Element hinzugefügt, ausgetauscht oder gelöscht. In Betracht kommt auch die Änderung des Ablaufs. Die Planung kann über vorhandene Erfahrungswerte aber auch über Planungs-Algorithmen erfolgen. Bevor die geplante Änderung umgesetzt wird sollte überprüft werden, ob und wie sich diese auf die gesamte Software-Engineering-Methode auswirkt. Die Frage ist, ob diese Än-

derung an dieser Stelle die SEM verbessert, oder ob sie an einer anderen Stelle dadurch weitere Probleme erzeugt. Für den Fall, dass die gesamte SEM an einer anderen Stelle stark gestört wird, muss eine Alternative geplant und mit der Gesamt-SEM sowie der anderen Planungsvariante abgeglichen werden. Ist dies erfolgt, wird die endgültige Planung festgelegt und die Execute-Phase wird aufgerufen.

Die Execute-Phase passt nun sowohl das Modell als auch die Instanz der aktuellen SEM mit der geplanten Änderung an und ermittelt, wer alles anhand von Benachrichtigungen (Notifications) über die Änderung informiert werden muss. Dies können sowohl Betroffene der Änderung sein als auch Programme oder Systemteile.

Abschließend greifen nun die Effektoren in das aktuelle Projekt und somit in die aktuelle Software-Engineering-Methode ein und setzen die Anpassung entsprechend um. Bei den Effektoren handelt es sich um ein generisches Konzept. Effektoren können sowohl Projektpläne als auch Projektparameter anpassen. Sie können ebenfalls Modelle automatisch ändern und Benachrichtigungen verschicken. Effektoren müssen ferner für die Nachvollziehbarkeit der Änderungen sorgen, z.B. über einen Mechanismus, um Änderungen in ein Log der Wissensbasis zu schreiben.

Eine selbst-adaptive Anpassung einer Software-Engineering-Methode wäre mit Hilfe von einem SE Method Manager also theoretisch möglich. Dies soll im nächsten Abschnitt mit Hilfe von einem einfachen Beispiel verdeutlicht werden.

3.3.2 Durchspielen des SE Method Managers anhand eines Beispiels

Für das Durchspielen des SE Method Managers wird das einfache Beispiel der Teamgröße in Scrum verwendet. In Scrum ist die Teamgröße des Entwicklungsteams ein wichtiger Bestandteil. Ein Team sollte nach [SS13] nicht größer als 9 und nicht kleiner als 3 Personen sein. Würden weniger als 3 Personen im Entwicklungsteam arbeiten so besteht die Gefahr, dass diese kein funktionierendes Produkt-Inkrement liefern, weil sie beispielsweise nicht über die nötigen Skills verfügen. Sind mehr als 9 Personen im Entwicklungsteam so erfordert dies nach Schwaber und Sutherland zu viel Koordination. Das zu große Entwicklungsteam erzeugt eine zu hohe Komplexität um erfolgreich durch Scrum verwaltet zu werden [SS13]. Neben der Koordination leidet zusätzlich die Kommunikation.

Auch wenn Scrum für beide Möglichkeiten Lösungsansätze vorsieht, beispielweise bei einem zu großen Team das Team zu splitten und das zusätzliche Team-Meeting Scrum of Scrums einzuführen, so wird dies in der Praxis nicht unbedingt umgesetzt. Die Erfahrung, dass ein zu großes Team nicht

entsprechend geteilt wurde, wurde auch in einem Praxis-Projekt im s-lab gemacht. Wie eine eigenständige Anpassung schnell und einfach mit dem SE Method Manager dafür aussieht, wird im folgenden Beispiel gezeigt.

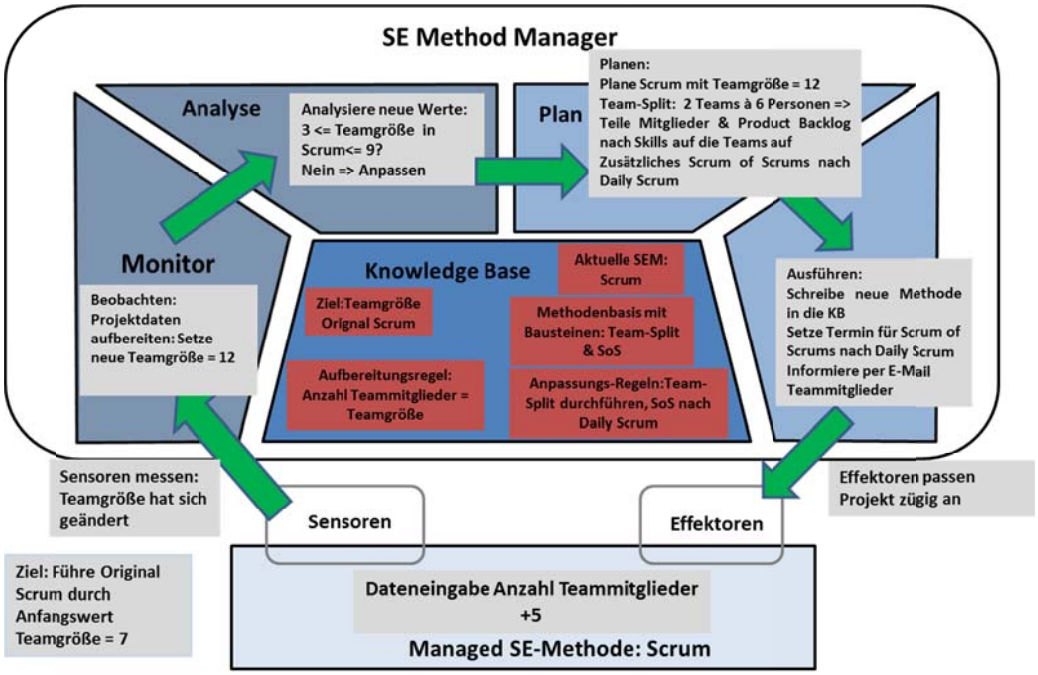

Abbildung 28 Anpassung bei einem zu großen Team mit dem SE Method Manager

Die Managed SE-Methode hier im Beispiel ist wie in Abbildung 28 zu sehen die Agile Methode Scrum im Original. Das definierte Ziel lautet hier, dass das originale Scrum durchgeführt wird und somit die optimale Teamgröße verwendet werden soll. Das Entwicklungsteam, hier einfach mit Team bezeichnet, besteht zu Anfang aus 7 Personen. Da beispielsweise gerade ein komplexer Sprint ansteht, werden dem Team 5 Mitglieder hinzugefügt. Die zusätzliche Anzahl der Mitglieder wird ein einer Liste festgehalten.

Über die Sensoren wird nun gemessen, dass sich die Teamgröße geändert hat. In der Monitor-Phase werden die neuen Projektdaten aufbereitet, die Regel für die Teamgröße besagt, dass die Anzahl der Teammitglieder = der Teamgröße ist. Die neue Teamgröße beträgt nun die Zahl 12. Das Team ist somit zu groß.

In der Analyse-Phase wird analysiert, ob das Team kleiner als drei oder größer als neun Personen ist. Die Analyse ergibt in diesem Beispiel also ein negatives Ergebnis und die Beurteilung lautet, dass die Software-Engineering-Methode angepasst werden muss.

In der Plan-Phase wird die Anpassung für das zu große Team mit 12 Personen geplant. Der Plan ist, dass ein Team-Split durchgeführt und zwei Teams mit sechs Personen entstehen. Die Personen werden gemäß ihrer Skills den

Teams zugeordnet, ebenso wie die Aufgaben aus dem Product Backlog. Zusätzlich wird geplant, dass ein Scrum of Scrums nach dem Daily Scrum durchgeführt wird. Dies ist ähnlich wie das Daily Scrum ein Informationsmeeting, wo sich Vertreter der beiden Teams treffen und entsprechend austauschen.

In der Execute-Phase wird die geplante Software-Engineering-Methode entsprechend in die Wissensbasis geschrieben und der Termin für das Scrum of Scrums wird nach dem Daily Scrum gesetzt. Abschließend werden alle Beteiligten über die Änderungen informiert und wann diese Änderungen in Kraft treten. Über die Effektoren werden die Änderungen in das aktuelle Projekt übertragen und die Benachrichtigungen versendet. Die Software-Engineering-Methode ist somit angepasst.

3.3.3 Möglichkeiten zur Automatisierung

Beim ursprüngliche Autonomic Manager ist es möglich, alle vier Phasen zu automatisieren. Trotzdem wird wie in Abschnitt 3.2.2 beschrieben dies nicht immer durchgeführt. Manchmal werden nur einzelne Phasen automatisiert oder Kombinationen von einzelnen Phasen. Das Ziel des SE Method Managers sollte ebenfalls sein, dass MAPE-K vollständig automatisiert werden kann. Insbesondere das Messen der Daten über Sensoren, die Monitor-Phase mit der Aufbereitung der Daten für die Analyse-Phase sowie die Analyse-Phase selbst bieten sich für eine Automatisierung an. Die Auswertung der Daten kann automatisch über Regeln erfolgen, welche vorher definiert und in der Wissensbasis abgelegt wurden. Ähnliches kann bei der Aufbereitung der Daten in der Monitor-Phase erfolgen. Spezielle Aufbereitungsregeln werden definiert, in der Wissensbasis abgelegt und später in der Monitor-Phase automatisch angewendet.

Auch wenn der Übergang zwischen der Analyse-Phase und der Plan-Phase automatisch erfolgen kann, beispielsweise hat die Auswertung ergeben, dass die Software-Engineering-Methode an einer bestimmten Stelle angepasst werden muss, macht es an dieser Stelle Sinn, den Menschen mit einzubinden. Ein Methoden-Engineer und/oder der Projektleiter sollten die Software-Engineering-Methode daraufhin prüfen, ob an dieser Stelle wirklich eine Anpassung nötig oder ob sie noch im Rahmen der SEM erlaubt ist.

Anschließend können sie die Freigabe für die Anpassung erteilen und den Übergang zu der Plan-Phase einläuten. Oder sie lehnen die Anpassung ab und MAPE-K wird in der Monitor-Phase weiter fortgesetzt. An dieser Stelle sollten dann die Rahmenbedingungen und die Regeln entsprechend angepasst werden. Ansonsten werden der vermeintliche Fehler und seine Anpassung im nächsten Schritt möglicherweise erneut erfolgen.

Mit den Übergängen von der Plan-Phase zur Execute-Phase sowie der endgültigen Ausführung verhält es sich ähnlich. Die Phasen an sich sollten automatisiert durchgeführt werden und auch ein automatischer Übergang ist möglich. Doch an beiden Stellen macht es Sinn, dass ein erfahrener Methoden-Engineer sich die geplante Anpassung anschaut, sie aufgrund seiner Erfahrung einschätzt und anschließend zur Durchführung freigibt. Zusätzlich kann sich am Ende der Execute-Phase vor der endgültigen Durchführung der Anpassung, der Projektleiter unter Umständen zusammen mit dem Methoden-Engineer, die nötige Anpassung ansehen und das endgültige Okay geben. Diese menschlichen Zwischenschritte nutzen die Erfahrung der Personen und erhöhen das Vertrauen in eine angepasste Methode.

3.4 Erstes Fazit und weitere Herausforderungen

In dem vorherigen Abschnitt wurde mit Hilfe des SE Method Managers basierend auf MAPE-K, ein Ansatz vorgestellt, welcher es grundlegend möglich macht, die Anpassung einer Software-Engineering-Methode selbst-adaptiv durchzuführen. In den folgenden Abschnitten wird nach einem kurzen Fazit bezüglich des Ansatzes dieser tiefergehend betrachtet und zusätzlich erörtert, welche weiteren Probleme sich im Detail ergeben.

3.4.1 Erstes Fazit bezüglich des Ansatzes

Nach einem ersten Betrachten und Durchspielen des SE Method Managers zeigt sich, dass die automatisierte Überwachung einer im Vorfeld festgelegten Software-Engineering-Methode und deren zeitnahe, selbst-adaptive und automatische Anpassung zur Laufzeit mit Hilfe von MAPE-K grundsätzlich möglich sind. Die vier Phasen des MAPE-K Monitor, Analyse, Plan und Execute, erfüllen weitestgehend die Anforderungen A1 bis A13.

Der Fokus liegt wie in A1 gefordert auf der Software-Engineering-Methode, welche das zu überwachende Element darstellt (Managed SE-Methode). Die MAPE-K-Feedbackschleife folgt einem kontinuierlichen Zyklus ähnlich den der bereits vorhandenen Ansätze und bindet Feedback mit Hilfe von Sensoren ein. Damit ist Anforderung A2 ebenfalls erfüllt. Mit Hilfe des Zyklus, der wie im vorherigen Abschnitt beschrieben teils automatisch durchgeführt werden kann, kann die Dauer der Analyse, Anpassung und insbesondere Ausführung relativ kurz gehalten werden, womit Anforderung A3 erfüllt wäre.

Die Sensoren messen die Projektdaten und die Monitor-Phase bereitet sie entsprechend für die weitere Auswertung auf. Dadurch ist nicht nur eine kontinuierliche Messung der SEM auf Instanz-Ebene, sondern auch eine durchgehende Beobachtung der SEM gewährleistet. Damit wären Anforderung A4 und durch das Messen mit Hilfe von Sensoren wäre Anforderung A5 erfüllt.

Die Analyse-Phase analysiert anhand der gemessenen Daten entsprechend den aktuellen Status und beurteilt, ob ein Anpassungsbedarf der verwendeten Software-Engineering-Methode nötig ist oder nicht. Dadurch ist Anforderung A6 erfüllt.

In der Plan-Phase wird eine entsprechende Anpassung auf der Typ-Ebene geplant, indem Methoden-Bestandteile hinzugefügt, gelöscht oder ausgetauscht werden. Dies erfüllt grundlegend die Herausforderung A8 und teilweise A9 (Beurteilung, welche Elemente hinzugefügt, gelöscht oder ausgetauscht werden).

Abschließend wird mit Hilfe der Execute-Phase und der Effektoren die Software-Engineering-Methode zeitnah angepasst, das heißt die Anpassung wird von der Typ-Ebene auf die Instanz-Ebene und somit in das laufende Projekt übertragen. Dadurch ist Anforderung A10 erfüllt. Da dies grundlegend automatisch, schnell und eigenständig möglich sein sollte, wäre voraussichtlich auch die Anforderung A11 erfüllt.

Damit die vier Phasen in der Feedbackschleife durchgeführt werden können, sind die wichtigen Daten, welche für die einzelne Schritte benötigt werden, in der Wissensbasis zentral gespeichert und es kann darauf zugegriffen werden. Damit ist auch Anforderung A7 erfüllt.

Da die MAPE-K-Feedbackbackschleife im Projekt auch nach einer Anpassung kontinuierlich mit den neuen Werten der Anpassung weiterläuft, werden somit automatisch die Ergebnisse der Anpassung mitbetrachtet und evaluiert, ob sie die Ziele entsprechend einhalten. Deswegen ist auch Anforderung A12 erfüllt.

Ferner ist es möglich, die bestehenden Daten in der Wissensbasis weiterzuverwenden und in neuen Projekten einzusetzen. Dadurch würde die Erfahrung erhalten bleiben. Allerdings kann die Wissensbasis nur in Teilen verwendet werden, da jedes Projekt eine andere Software-Engineering-Methode besitzt. Es ist also zu überlegen, wie und welche Elemente in ein neues Projekt eingesetzt werden könnten. Dann wäre Anforderung A13 ebenfalls erfüllt.

Der SE Method Manager mit MAPE-K als Kern eignet sich somit als Lösung zur selbst-adaptiven Anpassung einer Software-Engineering-Methode. Es ist noch zu überlegen, wie der zweite Teil der Anforderung A9 – die Auswirkung auf die Gesamt-SEM – erfüllt werden kann. Dies wäre zum Beispiel möglich, wenn innerhalb der Plan-Phase die geplante Anpassung in Bezug auf die Gesamt-SEM analysiert wird. Dies könnte beispielsweise mit Hilfe einer Simulation erfolgen. Dann wäre auch die Anforderung A9 vollständig erfüllt.

Im nächsten Abschnitt wird nun nach der ersten Eignung des SE Method Managers der Ansatz in der Tiefe betrachtet und herausgearbeitet, welche weiteren Herausforderungen sich ergeben.

3.4.2 Weitere Herausforderungen („Trigger-Probleme“)

Auch wenn der SE Method Manager sich nach dem ersten Fazit im vorherigen Abschnitt gut eignet, um eine Software-Engineering-Methode während eines laufenden Projektes schnell und zeitnah anzupassen, muss der Ansatz dennoch im tieferen Detail betrachtet werden.

Im Gegensatz zu dem ursprünglichen MAPE-K muss man bedenken, dass es sich während der Anpassung um eine Software-Engineering-Methode und nicht um ein System handelt. Schaut man sich den Ablauf in Abbildung 29 an, beginnt dieser korrekterweise mit der Dateneingabe. Sobald neue Daten im Projekt vorhanden sind, werden diese aufbereitet, gespeichert und analysiert.

Die erste Frage die dabei aufkommt ist, welche Daten überhaupt alle erfasst werden müssen. Was zählt alles als Dateneingabe, damit die MAPE-K-Feedbackschleife durchgeführt wird? Die Frage ist, ob ein neues Artefakt bloß durch sein Vorhandensein, beim Anlegen und Löschen die MAPE-K-Feedbackschleife anstößt („triggert“) oder ob allein ein Ändern des Artefakts die Feedbackschleife auslöst. Daran würde sich die Frage anschließen, welche Änderungen an dem Artefakt zum Auslösen der Monitor-Phase führen würden; jede Änderung oder spezielle Änderungen? Oder auch, ob ein implementierter Code zur Dateneingabe zählen würde.

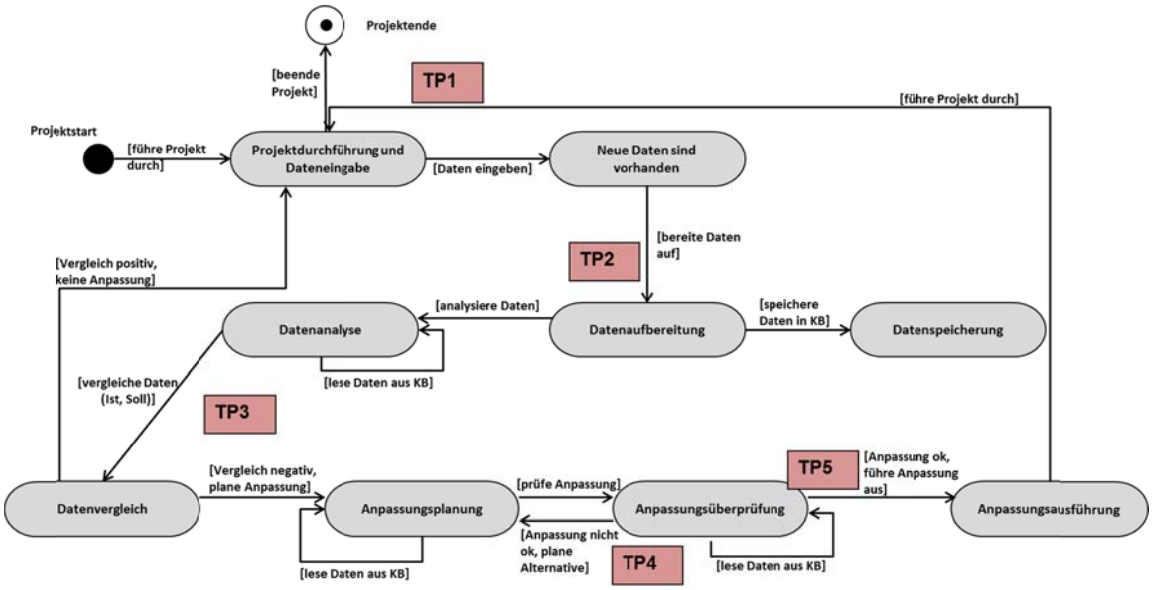

Abbildung 29 State-Chart des SE Method Managers und die „Trigger-Probleme"

Da sich dieses und die folgenden Probleme darauf beziehen, wann die MAPE-K-Feedbackschleife und ihre Phasen getriggert werden und sich dieses Anfangsproblem weiter durchzieht, werden die Herausforderungen, die sich dadurch für den Ansatz ergeben, im Folgenden auch „Trigger-Probleme“ genannt. Wie sich Folgeprobleme und zusätzliche Herausforderungen aus dem ersten Triggern ergeben wird nun weiter erläutert. Doch zunächst lautet die Kernfrage für Triggerproblem 1:

TP1: Granularität der Dateneingabe: Wie granular müssen die Daten für die Dateneingabe sein, damit die Monitor-Phase angestoßen wird?

Aus der fehlenden Granularität der Daten leitet sich direkt Triggerproblem 2 ab. Denn auch wenn bekannt ist, wie granular die Daten sein müssen, ist die Frage, ob bei allen neuen Daten getriggert werden muss. Werden alle eingegebenen Daten aufbereitet und ausgewertet? Es muss die Frage beantwortet werden, wie die relevanten Daten für die Aufbereitung und Auswertung bestimmt und von den anderen Daten herausgefiltert werden.

Triggerproblem TP2: Wie werden die relevanten Daten für die Aufbereitung und Auswertung anhand einer Datenfilterung bestimmt?

Auch wenn bekannt ist, welche Daten gemessen und aufbereitet werden müssen, ist das Folgeproblem, welches sich aus den ersten beiden ergibt, dass bei der Eingabe von vielen Daten gleichzeitig nicht bekannt ist, wann getriggert wird. Die vielen Daten könnten zur Folge haben, dass eine „Dauertriggerung“ erfolgt. Auf den ersten Blick erscheint dies nicht schlimm, aber wie wird entschieden, was wann und in welcher Reihenfolge abgearbeitet wird? Es muss geklärt werden, ob Daten gleichzeitig oder kurz nacheinander verarbeitet werden können. Bei einer Software-Engineering-Methode ist es eher sinnvoll, Daten und insbesondere eine spätere Anpassung sequentiell zu verarbeiten. Es muss also eine Vorgehensweise gefunden werden, eine Reihenfolge festzulegen, wann welche Daten ausgewertet und weiter verarbeitet werden. Dafür müssten Prioritäten vergeben werden. Jedoch ist es aktuell schwer zu entscheiden, welche gemessenen Daten eine höhere Priorität hätten als andere.

Triggerproblem TP3: Wie können Prioritäten für zu messende Daten, deren Aufbereitung und Auswertung vergeben werden, damit eine sequentielle Bearbeitung möglich ist?

Auch wenn Prioritäten vergeben sind und eine Reihenfolge für die Bearbeitung festgelegt ist, ist ein weiteres Problem, dass nicht bekannt ist, ob sich die bereits getriggerten Anpassungen gegenseitig beeinflussen. Das heißt, es muss möglich sein zu überprüfen, ob mögliche Anpassungen in Konflikt zueinander stehen. Dafür müsste zum einen bekannt sein, welche Anpas-

sung und welche Daten bearbeitet werden. Zum anderen wäre es sinnvoll zu wissen, welche Daten potentiell zueinander in Konflikt stehen, um dies eventuell im Vorfeld abzufangen und eine Anpassung entsprechend zu planen.

Triggerproblem TP4: Wie kann möglichst früh überprüft werden, ob getriggerte Daten und somit eine mögliche Anpassungen zueinander in Konflikt stehen?

Ein letzter zu betrachtender Punkt ist der Anpassungszeitpunkt. Ein System kann problemlos jederzeit und kurz hintereinander angepasst werden. Doch bei einer Software-Engineering-Methode ist es nicht möglich, sie beispielsweise stündlich oder täglich anzupassen. Dies würde vermutlich mehr Chaos auslösen, als wirklich zu helfen. Eine Anpassung einer Software-Engineering-Methode kann von daher nur in bestimmten zeitlichen Abständen vorgenommen werden. Es muss geklärt werden, wie schnell hintereinander Anpassungen zeitlich erfolgen können. Da somit eine Anpassung nicht jederzeit erfolgen kann, muss klar sein was passiert, wenn mehrere Anpassungen bis zum Anpassungszeitpunkt auflaufen. Wie können diese miteinander kombiniert werden? Eine weitere wichtige Frage in diesem Zusammenhang ist, was mit Daten passiert, die zum Anpassungszeitpunkt bereits getriggert sind und die sich somit in der Planung befinden.

Triggerproblem TP5: Wie kann der Anpassungszeitpunkt für eine Software-Engineering-Methode bestimmt werden und wie kann zu diesem Zeitpunkt eine kombinierte Anpassung erfolgen?

Diese weiteren Herausforderungen gilt es zu untersuchen und vor allem herauszufinden, wie sich diese lösen lassen. Wie ein zielorientiertes Vorgehen genutzt werden kann, um den Ansatz zu erweitern und sowohl den weiteren Herausforderungen zu begegnen als auch weiterhin die Anforderungen A1 – A13 zu erfüllen, wird im nächsten Kapitel erläutert.

Kapitel 4 Der Ansatz MAPE-K4SEM

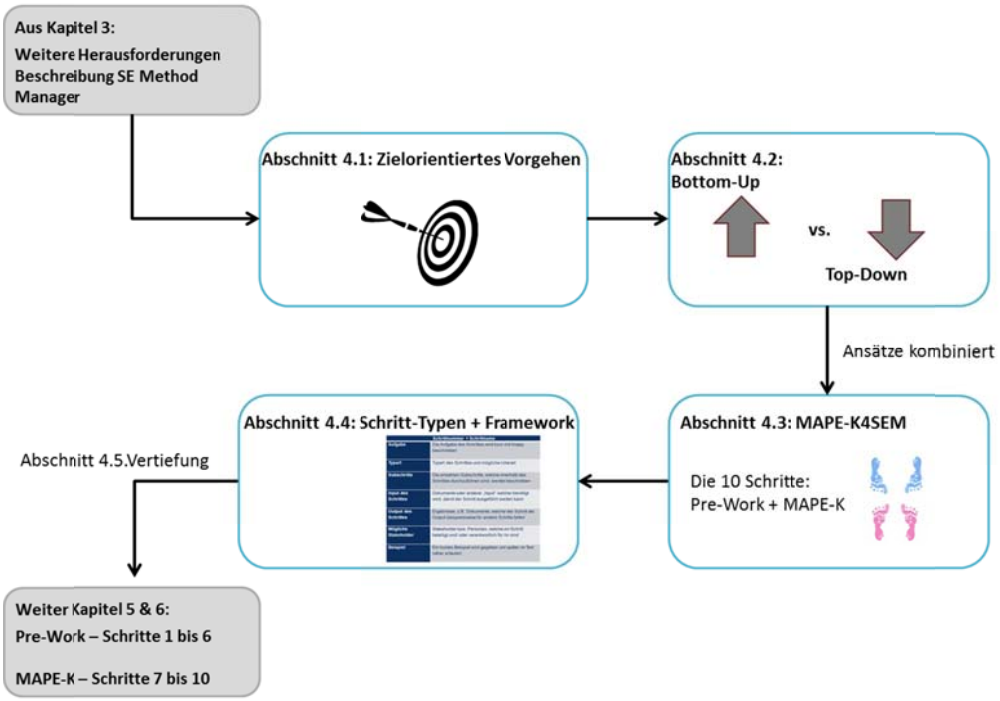

Abbildung 30 Aufbau Kapitel 4

In Kapitel 3 wurde ein Lösungsansatz vorgestellt, wie es grundsätzlich möglich ist, Software-Engineering-Methoden zur Laufzeit anzupassen. Dafür wurde der Ansatz des MAPE-K aus den selbst-adaptiven Systemen gewählt, damit sich eine SEM ebenfalls möglichst selbst-adaptiv anpassen kann. Im letzten Teil wurden weitere Herausforderungen diskutiert, die Triggerprobleme, welche bei einer detaillierten Betrachtung auftreten können. Wie diese mit Hilfe eines zielorientierten Vorgehens und des selbstentwickelten MAPE-K4SEM-Ansatzes angegangen werden können, wird in diesem Kapitel vorgestellt.

4.1 Zielorientiertes Vorgehen

Schon [Bo13, S. 54] behauptet: *„Adaptive Projekte sind zielorientiert“*. Um nun den Triggerproblemen zu begegnen, muss überlegt werden, wie eine ausreichende Granularität und darauf aufbauend eine geeignete Datenfilterung sichergestellt werden kann. Um herauszufinden, welche Daten gefiltert werden sollen, muss man wissen, welche Daten nur für die Datenanalyse und die spätere Planphase gebraucht werden. Um zu wissen, welche Daten für die Analyse benötigt werden ist es nötig herauszufinden, wo gegen analysiert wird, ähnlich wie beim Testen. Es müssen also die Anforderungen, Eigenschaften und Ziele bekannt sein, welche die Software-Engineering-Methode erfüllen muss. Dies kann angelehnt an die obige Aussage von [Bol] über ein zielorientiertes Vorgehen erfolgen.

In diesem zielorientierten Vorgehen müssen also als erstes die Ziele und Eigenschaften der Software-Engineering-Methode definiert werden. Sind die Ziele bekannt, kann im nächsten Schritt überlegt werden, was zum einen dafür nötig ist, damit diese Ziele erfüllt werden. Zum anderen ist es nötig herauszufinden, in welchen Grenzen diese Erfüllung liegt, das heißt was ist mindestens nötig, damit dieses Ziel erfüllt wird und was ist maximal nötig.

Sind diese Dinge bekannt, können daraus Regeln entwickelt werden, welche für die Analyse genutzt werden können. Liegt die Erfüllung des Ziels in den definierten Grenzen, muss nichts getan werden. Ist aber die Erfüllung des Ziels in Gefahr, das heißt der analysierte Wert liegt außerhalb der Grenzen, muss eine Anpassung der Software-Engineering-Methode vorgenommen werden. Doch durch die vorherige Definition ist bereits bekannt, um welches Ziel es sich handelt und welche Grenze über- bzw. unterschritten wurde. Da die Regeln vom vorher definierten Ziel hergeleitet wurden ist es zum Teil ebenfalls möglich. im Vorfeld zu überlegen, mit welchen Strategien dieses Ziel erreicht werden kann. Diese Strategien können vordefinierte Anpassungen für die Planung ergeben.

Ein Vorteil dieses Vorgehens ist, dass durch die definierten Regeln gut die zu messenden Werte – die Metriken – für die Sensoren und für die Monitor-Phase bestimmt werden können. Es ist bekannt, was für die Analyse gebraucht wird. Aus dieser Regel werden die Analysewerte extrahiert. Die Analysewerte sind die Werte, welche die Aufbereitung in der Monitor-Phase als Endergebnis haben muss. Aus diesen Werten kann ermittelt werden, welche Daten für die Aufbereitung gemessen werden müssen. In einfachen Fällen können die gemessenen Daten gleich den aufbereiteten Werten sein. Damit ist sowohl die nötige Granularität der Werte bekannt, die gemessen werden müssen, als auch ein Datenfilter gesetzt.

Im weiteren Vorgehen der MAPE-K-Schleife ist es nun möglich, genau zu bestimmen, was in der Analyse- und was in der Plan-Phase getriggert wird. Ein abstraktes Beispiel: zu den Zielen x, y und z gehören jeweils die Regeln rx, ry und rz sowie die zu messenden Metriken mx, my und mz. Wird nun die Metrik my gemessen, wird nur dieser Wert aufbereitet und automatisch wird Regel ry getriggert und mit den gemssenen Werten ausgewertet. Andere Werte werden im Prinzip gar nicht beachtet und triggern nicht die MAPE-K-Schleife.

Da die Ziele bekannt sind, ist es möglich, diesen Zielen Prioritäten zuzuordnen. Diese Prioritäten können sowohl an die Analyse- als auch an die Planungsmöglichkeiten usw. vererbt werden. Anhand einer geeigneten Prioritätenvergabe kann nun eine Reihenfolge für die Abarbeitung der Daten erstellt werden. Auf das Beispiel übertragen hätte Ziel x die Priorität 1, Ziel y die

Priorität 2 und Ziel z die Priorität 3. Würden nun die Metriken mx, my und mz gemessen, würde als erstes die Regel rx ausgewertet und im Falle der Grenzüber- bzw. unterschreitung würden diese Werte als erstes geplant und angepasst, da sie die geerbte Priorität 1 haben. Anschließend würde Regel ry mit Priorität 2 und abschließend Regel rz mit Priorität 3 weiter verarbeitet. Damit wäre auch Triggerproblem TP3 gelöst.

Da die Ziele bekannt sind und teilweise die Planungsmöglichkeiten und Analyseregeln im Vorfeld bestimmt werden können, ist es ebenfalls möglich, die Ziele auf mögliches Konfliktpotential zu untersuchen. Es sind am Ende sowohl die Metriken als auch die Analyseregeln und mögliche Anpassungen (zum Teil) bekannt. Diese können miteinander abgeglichen und auf Konflikte hin analysiert werden. Besteht Konfliktpotential, können im Vorfeld bereits Lösungen und Planungsalternativen erstellt werden, falls die Anpassungen im gleichen Anpassungszyklus, das heißt vom einen Anpassungszeitpunkt bis zum nächsten, gemessen werden. Damit wäre es möglich, Triggerproblem TP4 im Vorfeld zu begegnen.

Ähnlich wie das Konfliktpotential können im Vorfeld auch Kombinationsmöglichkeiten für einzelne Ziele bestimmt werden. Dazu wäre es wichtig, für die Planungsmöglichkeiten der verschiedenen Ziele jeweils Kombinationspunkte zu ermitteln. Mit Kombinationspunkten ist gemeint, an welcher Stelle ist es möglich, bei einer geplanten Anpassung anzuknüpfen, z.B. können bei der Erstellung eines Artefaktes verschiedene Techniken angewandt werden. Dabei wäre die Technik ein Kombinationspunkt. Damit wäre Triggerproblem TP 5 ebenfalls größtenteils gelöst. Es ist zusätzlich nötig, für die Software-Engineering-Methode einen geeigneten Anpassungszeitpunkt festzulegen. Dieser kann beispielsweise anhand der Eigenschaften festgelegt werden.

4.2 Bottom-Up vs. Top-Down

Für die Analyse von Prozessen aber auch im Bereich der Programmentwicklung und im Software Engineering gibt es zwei Ansätze, die genau gegensätzlich aufgebaut sind. Diese beiden Ansätze, „Top-Down“ und „Bottom-Up“ genannt, werden in vielen Bereichen eingesetzt, neben der Softwareentwicklung beispielsweise auch in der Konzeption von Algorithmen [CGL05].

Der Top-Down-Ansatz (engl. von oben nach unten) startet dabei mit der Beschreibung von etwas Abstrakten bzw. Allgemeinen und spezifiziert abwärts immer weitere Details. Dies wird auch Deduktion genannt. In Multi-Agenten-Systemen startet der Ansatz beispielsweise damit, globale Anforderungen für die Agenten zu spezifizieren, um diese nachher im Detail auf die einzelnen Eigenschaften der Agenten hinunterzubrechen [CGL05]. Wird

der Ansatz in der Algorithmik eingesetzt so heißt dies, dass ein Algorithmus zunächst sehr allgemein und umgangssprachlich definiert wird. Im weiteren Verlauf wird er immer detaillierter spezifiziert und in zusätzliche Einheiten aufgeteilt, bis am Ende der fertige Algorithmus steht.

Der Bottom-Up-Ansatz (engl. von unten nach oben) arbeitet genau gegensätzlich. Hier werden zunächst einzelne Details spezifiziert, in Agentensystemen beispielsweise die einzelnen Eigenschaften der Agenten oder in der Algorithmik einzelne Funktionen und Bestandteile eines Algorithmus. Beim weiteren Vorgehen wird ein immer komplexeres und allgemeineres Ganzes spezifiziert. Dieser Vorgang wird auch Induktion genannt.

Diese beiden Ansätze werden ebenfalls in der Prozessverbesserung eingesetzt. Der Top-Down-Ansatz vergleicht dabei den eigentlichen Prozess, z.B. der Organisation, mit bereits vorhandenen oder generellen Standards im Unternehmen. Während des Verbesserungsprozesses werden anschließend die Unterschiede zwischen beiden herausgearbeitet. Die Annahme ist hier, dass, sobald der Prozess im Detail geändert wurde, automatisch die erzeugten Produkte ebenfalls verbessert werden [Thomas in TM94].

Der Bottom-Up-Ansatz geht hier hingegen davon aus, dass die Änderung und Verbesserung eines Prozesses von folgenden Elementen ausgehen sollte: den allgemeinen Organisationszielen, ihren Charakteristiken, den Produkten und ihren Attributen sowie von der vorhandenen Erfahrung. Die Veränderung sollte sich auf der lokalen Ebene abspielen anstatt allgemeine und universelle Elemente zu nutzen. [McGarry in TM94].

Doch bei beiden Ansätzen geht es hauptsächlich darum, das Produkt zu verbessern, wobei jeweils davon ausgegangen wird, dass sich bei einer Prozessverbesserung auch das Produkt verbessert.

Die Frage ist nun, wie sich diese Ansätze auf die Anpassung einer Software-Engineering-Methode übertragen lassen und welcher Ansatz verwendet werden sollte. Schaut man sich die Vorgehensweise von MAPE-K an, so verwendet dieser eine Art Bottom-Up-Ansatz. Das System bzw. im SE Method Manager ist die Software-Engineering-Methode bekannt. Um diese anzupassen, werden alle detaillierten Daten der Software-Engineering-Methode gemessen, analysiert und ausgewertet um zu bestimmen, ob die SEM angepasst werden soll oder nicht. Hier werden also die detaillierten Daten genommen um auf das allgemeinere Modell und deren Anpassung zu schließen. Sie werden also genutzt, um eine Anpassung an dem Modell und somit an der Software-Engineering-Methode selbst vorzunehmen.

Am Ende von Kapitel 3 in Abschnitt 3.4 hat sich gezeigt, dass diese Vorgehensweise einige neue Herausforderungen mit sich bringt. Vor allem ist nicht klar, wie detailliert die Datenmessung sein soll, in welcher Reihenfolge analysiert oder was analysiert wird usw. Im vorherigen Abschnitt wurde ein zielorientiertes Vorgehen vorgestellt, welches diese Herausforderungen bewältigen soll. Dies würde einen Top-Down-Ansatz darstellen, da zunächst die allgemeinen Ziele und Eigenschaften der Software-Engineering-Methode definiert und beschrieben werden, um diese dann in detailliertere Regeln und abschließende Metriken aufzubrechen. Doch wie kann nun dieser Top-Down-Ansatz genutzt werden, um die Software-Engineering-Methode anzupassen?

Die Idee ist, die beiden Ansätze miteinander zu kombinieren. Wie schon in Abschnitt 2.3 „Change Management“ und von [No14] erwähnt, sollten die Veränderungen im Change Management sowohl Top-Down als auch Bottom-Up geplant und durchgeführt werden. Das heißt, dass die Rahmenbedingungen und die Vorgehensweise werden von oben vorgegeben (top-down for targets) werden. Die spätere konkrete und inhaltliche Umsetzung wird von unten mit Hilfe der Betroffenen umgesetzt (bottom-up for how to do it). Dies kann ebenfalls für den Ansatz zur selbst-adaptiven Anpassung einer Software-Engineering-Methode verwendet werden.

Dafür sollten wie in Abschnitt 4.1 beschrieben zunächst Top-Down die Ziele und Eigenschaften definiert und in Regeln, Möglichkeiten zur Planung und Metriken weiter verfeinert werden. Sobald dies vorhanden ist, kann die konkrete und inhaltliche Umsetzung „bottom-up-mäßig“ mit Hilfe von MAPE-K umgesetzt werden. Dies erfolgt, indem die eigentliche MAPE-K-Feedbackschleife durchgeführt wird, beginnend mit der Messung der Daten, diesmal allerdings anhand der konkreten Metriken.

Der Top-Down-Ansatz leistet sozusagen die Vorarbeiten, damit MAPE-K, also der Bottom-Up-Ansatz, später mit den konkreten Werten erfolgreich durchgeführt werden kann. Wie dies genau aussehen kann, wurde in einem 10-Schritte-Ablauf erarbeitet, dem MAPE-K4SEM-Ansatz, welcher im nächsten Abschnitt vorgestellt wird.

4.3 MAPE-K4SEM – die 10 Schritte

Für das beschriebene zielorientierte Vorgehen wurde der Ansatz MAPE-K4SEM entwickelt, ein Ablauf in 10 Schritten. Dieser stellt außerdem sicher, dass nicht nur die richtigen Werte am Ende gemessen, sondern zusätzlich die benötigten Werte für die Durchführung des MAPE-K korrekt ermittelt werden.

Der Ansatz gliedert sich dabei in zwei Teile, in die zu Beginn erforderlichen Schritte – der „Pre-Work“ – und dem danach folgenden MAPE-K. In der Pre-Work werden alle Schritte durchgeführt, um die in Abschnitt 3.4.4 beschriebenen Elemente wie Ziele, Analyseregeln, Metriken usw. herzuleiten. Sind diese Dinge bekannt, kann der MAPE-K darauf aufbauend durchgeführt werden. Mit der Kombination dieser beiden Teile ist am Ende eine selbst-adaptive Anpassung und somit eine „selbst-adaptive Software-Engineering-Methode“ möglich. Im Folgenden werden die 10 Schritte des MAPE-K4SEM kurz vorgestellt.

4.3.1 Pre-Work – Schritte 1 bis 6

Die Pre-Work beinhaltet die Schritte für die Vorarbeiten, also um alle benötigten Daten wie Regeln, Metriken usw. anhand der Ziele für die Durchführung des MAPE-K zu ermitteln. Bevor die Daten allerdings hergeleitet werden können, müssen erst die Ziele definiert und priorisiert werden.

4.3.1.1 Schritt 1: Definition der Ziele

In Schritt 1 werden wie der Name schon sagt, die Ziele der Software-Engineering-Methode definiert. Dies sind insbesondere die Eigenschaften und teilweise Regeln, welche die SEM erfüllen muss um erfolgreich zu sein. In diese Ziele fließen auch die Umgebung, in welcher das Projekt und somit die SEM durchgeführt werden, mit ein. Dazu zählen das Unternehmen und seine Kultur selbst sowie der Kontext, in dem das Projekt durchgeführt wird. In Scrum sind beispielsweise verschiedene Aktivitäten vorgegeben, welche eingehalten werden sollen, wie etwa ein Sprint Planning-Meeting oder das Review-Meeting. Auch gibt es für jede Iteration ein „Sprint-Goal“ welches erreicht werden soll.

4.3.1.2 Schritt 2: Priorisierung der Ziele

Nachdem die Ziele definiert sind, ist es nun wichtig, diesen verschiedene Prioritäten zu geben. Dabei sollte darauf geachtet werden, dass die Prioritäten nicht zu grobgranular gewählt werden. Dies kann ansonsten dazu führen, dass es schwierig wird, eine Reihenfolge für die Auswertung von Daten zu erstellen. Sind sowohl die Ziele definiert als auch priorisiert, können nun die weiteren Daten und Regeln ermittelt werden. Dabei können die nächsten beiden Schritte parallel durchgeführt werden.

4.3.1.3 Schritt 3: Ableitung von Analyseregeln

In Schritt 3 werden nun aus den priorisierten Zielen die Grenzwerte ermittelt und in Analyseregeln überführt. Eine erste Idee ist dabei, die Regeln in der Form „Wenn…, dann…“ zu formulieren, beispielsweise „Wenn Wert x Grenze über- oder unterschreitet, dann Planung mit Aktion y sonst Projektdurchführung“.

4.3.1.4 Schritt 4: Ableitung von Planungsmöglichkeiten

Auf der einen Seite werden die Analyseregeln abgeleitet, auf der anderen Seite können durch die SEM-Beschreibung und durch das vorhandene Modell mögliche Planungen für eine Anpassung hergeleitet werden. Die Grenzen sind bekannt und in einigen Fällen gibt es schon Vorschläge, was getan werden soll, wenn diese Grenzen über- oder unterschritten werden. Daraus können nicht nur mögliche Anpassungen sondern auch zusätzliche Varianten bestimmt werden, falls eine bestimmte Anpassung nicht möglich ist.

Zusätzlich kann in diesem Zuge das mögliche Konfliktpotential der einzelnen Ziele ermittelt und es können gleich entsprechende Lösungen geplant werden, falls dieser Konflikt auftritt. Ein dritter denkbarer Unterschritt ist die Bestimmung von Kombinationsmöglichkeiten und Kombinationspunkten, was mit Hilfe des Modelles möglich ist.

4.3.1.5 Schritt 5: Ableitung von Metriken

Nachdem Schritt 3 und Schritt 4 durchgeführt wurden, kann nun im Folgenden bestimmt werden, welche Metriken für die Analyseregeln gemessen, aufbereitet und ausgewertet werden müssen. Die Metrik wird dabei im abstrakten Beispiel aus Schritt 3 im Prinzip durch „Wert x“ bestimmt. Zur Verdeutlichung wird wieder das der Teamgröße in Scrum aufgegriffen. Hier durfte das Team nicht größer als neun und nicht kleiner als drei Personen sein. Es müsste nun also die Teamgröße ausgewertet werden. Die zu messende Metrik wäre somit die „Anzahl der Personen pro Team“.

4.3.1.6 Schritt 6: Ableitung von Ausführungsregeln und Benachrichtigungen

Auf der anderen Seite sind nun die Planungsmöglichkeiten aus Schritt 4 bekannt und somit, welche Art von Anpassung durchgeführt werden soll. Es wird zum Beispiel an einer Stelle im Modell etwas hinzugefügt, gelöscht oder ausgetauscht. Zusätzlich ist bekannt, wer alles über die Anpassung informiert werden muss anhand der Rollen und den Verantwortlichkeiten, die zu dem entsprechenden Element im Modell gehören. Aus der geplanten Anpassung und der Information, wer Bescheid wissen muss, können dann entsprechende Ausführungsregeln und Benachrichtigungen hergeleitet werden.

Mit diesen sechs Schritten sind die Vorarbeiten und somit die Pre-Work abschlossen. Die Ziele, Regeln und hergeleiteten Daten werden entsprechend in der Wissensbasis gespeichert, damit sie in dem folgenden MAPE-K-Teil verwendet werden können.

4.3.2 MAPE-K – Schritte 7 bis 10

Die Pre-Work ist an dieser Stelle abgeschlossen und alle Daten, um MAPE-K erfolgreich zur Anpassung einer Software-Engineering-Methode einzusetzen, sind gewonnen und in der Wissensbasis gespeichert. Im Folgenden

werden kurz die vier weiteren Schritte vorgestellt, welche bei der Durchführung ausgeführt werden und was sie beinhalten sollten.

4.3.2.1 Schritt 7: Werte messen und aufbereiten

MAPE-K beginnt in Schritt 7 mit der Messung der Metriken über entsprechende Sensoren. Diese gemessenen Werte werden in der Monitor-Phase anschließend aufbereitet und in die Wissensbasis geschrieben, damit sie in der Analyse-Phase ausgewertet können. Am Beispiel der Teamgröße würde nun also die Anzahl der Personen über Sensoren gemessen und in der Monitor-Phase würde der Wert insofern aufbereitet, als dass die Anzahl der Personen gleich der Teamgröße gesetzt werden würde (Anzahl Personen/Team = Teamgröße).

4.3.2.2 Schritt 8: Werte analysieren und bewerten

Sind die Werte gemessen und aufbereitet, werden sie anschließend in der Analyse-Phase entsprechend ihren Regeln analysiert und ausgewertet. Liegt der Wert innerhalb der Grenzen, wird das Projekt weiter durchgeführt (es passiert nichts). Liegt der Wert außerhalb, wird die Plan-Phase mit den entsprechenden Werten über den Dann-Teil getriggert. Auch wenn dies automatisch möglich ist, kann es an dieser Stelle sinnvoll sein, dass der Projektleiter oder ein Methoden-Engineer sich die Auswertung anschaut und das endgültige Okay für eine Anpassung gibt. Oder er beschließt, dass mit diesen Werten die SEM weiter durchgeführt werden kann. Dann sollten allerdings die entsprechenden Regeln und Grenzen angepasst werden.

4.3.2.3 Schritt 9: Anpassung planen

Die Auswertung in Schritt 8 hat ergeben, dass eine Anpassung nötig ist. Der Dann-Teil hat die entsprechenden Daten geliefert, zum Beispiel, welche Planungsmöglichkeit oder Planungsvariante genutzt werden soll.

Zusätzlich muss nach der geplanten Anpassung in einem zweiten Teil dieses Schrittes überprüft werden, ob der Anpassungszeitpunkt bereits erreicht ist. Ist dies der Fall, kann die Anpassung sofort über die Execute-Phase ausgeführt werden. Ist dies nicht der Fall, muss die Anpassung in einem „Anpassungs-Pool“ gespeichert werden. Ist der Zeitpunkt anschließend erreicht, muss der Pool überprüft und gegebenenfalls müssen geplante Anpassungen zu einer Anpassung erst kombiniert, bevor sie ausgeführt werden.

4.3.2.4 Schritt 10: Anpassung ausführen

Im finalen Schritt ist die Anpassung für die Software-Engineering-Methode nun bekannt und kann automatisch über die Effektoren mit Hilfe der Ausführungsregeln und Benachrichtigungen ausgeführt werden. Eine Anpassung ist nach diesem Schritt abgeschlossen und die Schritte 7 bis 10 werden entsprechend wiederholt, beginnend mit der Messung der Daten über die Sensoren.

4.3.3 Schalen-Modell und Ableitungsbaum

Schaut man sich die Schritte näher an, ist zu erkennen, dass sich diese auf verschiedenen Ebenen bewegen und somit verschiedenen „Schalen“ zugeordnet werden können. Die oberste Schale enthält wie in Abbildung 31 zu sehen, dass Unternehmen mit seiner Kultur und all seinen Gegebenheiten sowie den Kontext, in dem es sich bewegt. Die Schale darunter beinhaltet dann konkreter den Projektkontext, welcher sich aus dem übergeordneten Kontext ableitet. Dieser Projektkontext und das Unternehmen selbst werden benötigt, um eine konkrete Software-Engineering-Methode für das durchzuführende Projekt spezifisch zu erstellen oder eine Software-Engineering-Methode vor Projektbeginn entsprechend zuzuschneiden. Das konkrete Modell der SEM wird anschließend in der Wissensbasis abgelegt, um es für die Schritte 7 bis 10 zu nutzen. Diese beiden Schalen bilden im Schalenmodell die „Kontext-Schicht“.

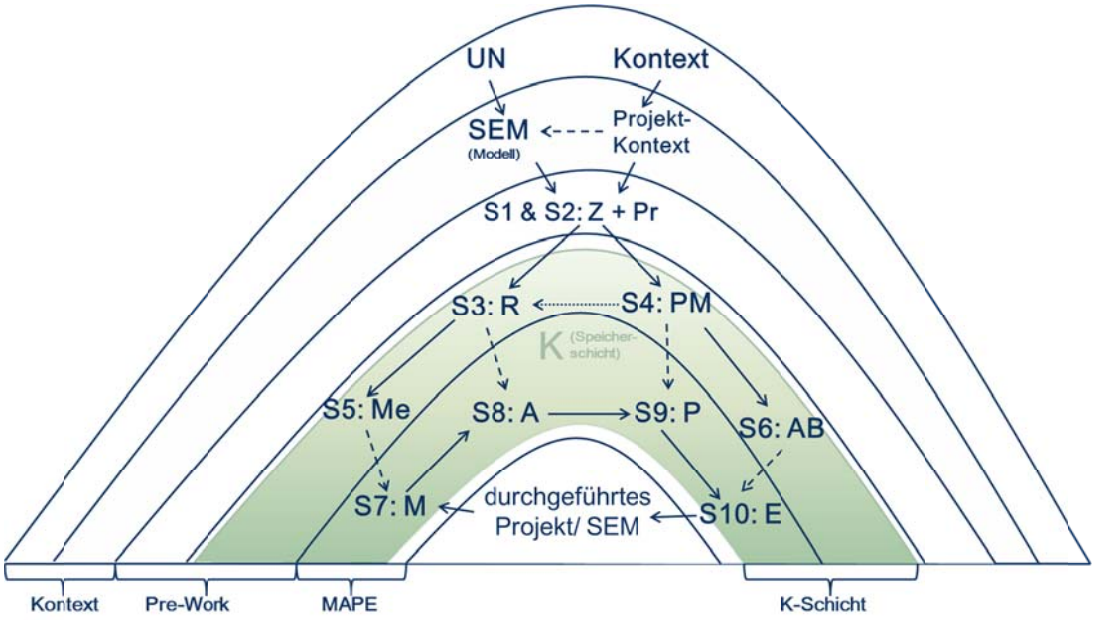

Abbildung 31 Das Schalenmodell inklusive des Ableitungsbaumes

Insbesondere der Projektkontext und die Software-Engineering-Methode werden nun wie in Abschnitt 3.4.5.1 beschrieben dazu genutzt, zunächst die Ziele (Z) zu definieren und anschließend die Prioritäten (Pr) zu vergeben. Diese Ziele werden ebenfalls in der Wissensbasis – Knowledge Base – (K) hinterlegt. Die nächste Schale beinhaltet die nächsten vier Schritte der Pre-Work, denn diese Schritte lassen sich jeweils von priorisierten Zielen her ableiten.

Auf der linken Seite sind dies zunächst wie beschrieben die Analyseregeln (R), woraus sich die Metriken (Me) ableiten lassen. Auf der rechten Seite werden aus den Zielen die Planungsmöglichkeiten (PM) abgeleitet. Diese wirken sich auf die Regeln aus, da sie den späteren Dann-Teil in den Analy-

seregeln bestimmen. Dies wird ausführlich erläutert in Kapitel 5. Im letzten Schritt werden in dieser Schale auf der rechten Seite die Ausführungsregeln und Benachrichtigungen (AB) abgeleitet. Die beschriebenen beiden Schalen bilden damit die „Pre-Work-Schicht“. Wie nun auch schon zu erkennen ist, bilden die einzelnen Schritte sowie die Schalen darüber einen Ableitungsbaum, wie sich die verschiedenen Daten gewinnen lassen, von der obersten Schale hinunter bis zur letzten Schale der Pre-Work.

Vor allem die gewonnen Daten aus der letzten Schale der Pre-Work (S3-S6) wirken sich jeweils auf die darunterliegende Schale aus. Diese Schale beinhaltet die Schritte der MAPE-K-Feedbackschleife und bildet damit zugleich die MAPE-Schicht. Dabei werden die Daten von S5 (Me) in S7 (M) genutzt, von S3 (R) in S8 (A), von S4 (PM) in S9 (P) und zu guter Letzt die Daten aus S6 (AB) in S10 (E). Dies wird im Schalenmodell durch die gestrichelten Pfeile verdeutlicht. Die Pfeile in der MAPE-Schale verdeutlichen dabei die genaue Reihenfolge, in welcher die einzelnen Schritte innerhalb von MAPE-K durchgeführt werden.

Die unterste Schale und zugleich unterste Schicht beinhaltet das durchgeführte Projekt und somit die durchgeführte Software-Engineering-Methode (SEM) auf der Instanz-Ebene. Diese wirkt sich sowohl in die darüber liegende Schale aus, indem die Daten in der Monitor-Phase bzw. Schritt 7 genutzt werden. Und die darüber liegende Schale wirkt sich umgekehrt über Schritt 10 auf die SEM aus, durch die durchgeführte Anpassung.

Zuletzt gibt es noch eine dahinterliegende Schicht, welche in grün gekennzeichnet ist. Dies ist die „K-Schicht“ und enthält die Wissensbasis (K). Die Daten aus der Pre-Work-Schicht werden alle in der Wissensbasis hinterlegt, damit sie von der darunter liegenden MAPE-Schicht genutzt werden können. Die oberste Schicht und zweitoberste Schale greifen nur insofern auf die K-Schicht zu, in dem das erstellte Modell der Software-Engineering-Methode dort ebenfalls abgelegt wird. Die Kontextinformationen werden zunächst nicht in der K-Schicht abgelegt und gebraucht.

Die genaue Nutzung des Ableitungsbaumes der ersten vier Schalen wird ausführlich in Kapitel 5 erläutert, wie sich die einzelnen Schritte und die entsprechenden Daten mit seiner Hilfe herleiten lassen.

4.4 Schritt-Typen und Framework zur Charakterisierung

Bevor in den nächsten beiden Kapiteln die einzelnen Schritte des MAPE-K4SEM detaillierter erläutert werden, wird zunächst erörtert, welche verschiedenen Typen von Schritten es gibt. Anschließend wird ein kurzes Framework erarbeitet, welches jeden Schritt zu Beginn gleichermaßen beschreibt und kurz erläutert, was dieser Schritt beinhaltet.

4.4.1 Schritt-Typen

Jeder der 10 Schritte im MAPE-K4SEM-Ansatz hat einen bestimmten „Typ“. Der Typ eines jeden Schrittes beschreibt, was den Schritt ausmacht und was seine genaue Aufgabe ist. Im Software Engineering gibt es verschiedene Bereiche, wo die Beschreibung anhand eines Typs wichtig ist und Sinn macht. In der Programmierung hat beispielsweise jede Funktion einen Typ wie boolean, int, double oder String. Diese Typen sagen aus, um was für eine Art von Funktion es sich handelt, was ihr Rückgabewert ist und wozu sich dieser verwenden lässt. Auch Fehler lassen sich anhand eines Typs klassifizieren, der aussagt, wie kritisch ein Fehler ist. Beispiele sind hier Blocker, kritischer Fehler oder leichte Fehler wie „Schönheitsfehler“.

Die 10 Schritte im MAPE-K4SEM erfüllen alle eine bestimmte Funktion. Diese werden entsprechend anhand des Typs charakterisiert und beschreiben, wozu der jeweilige Schritt verwendet werden kann. Dies kann beispielsweise wichtig für die spätere Wiederverwendung der einzelnen Schritte werden. Sollte der 10-Schritte-Ablauf des MAPE-K4SEM einmal angepasst, erweitert oder nur Teile daraus in einem anderen Kontext wiederverwendet werden, so ist es möglich, anhand der Schritt-Typen zu erkennen, wo diese sich einsetzen lassen und wie sie verwendet werden können.

Betrachtet man nun die 10 Schritte genauer, ist zum einen festzustellen, dass es nicht zehn verschiedene Schritt-Typen gibt, sondern dass einige Schritte einen gleichen oder ähnlichen Schritt-Typ enthalten. Zum anderen kann ein Schritt durch mehr als einen Schritt-Typ charakterisiert werden. Dies ist bei den späteren MAPE-K-Schritten der Fall. Diese besitzen einen „Haupt-Typen“ und zum Teil einen „Sub-Typ“, welcher die Subschritte beschreibt. Dies wird ausführlicher erläutert in Kapitel 6.

Werden die Schritte des MAPE-K4SEM aus Kapitel 4 betrachtet, ergeben sich die folgenden verschiedenen Typen:

1. Erstellen – Im Schritt dieses Typs wird etwas für die weitere Verwendung erstellt und festgelegt, z.B. die Ziele und Eigenschaften einer SEM.

2. Ableiten – Im Schritt dieses Typs wird etwas für die weitere Verwendung abgeleitet, z.B. Metriken. Insbesondere von diesem Typen gibt es weitere Sub-Typen.

3. Priorisieren – Im Schritt dieses Typs wird etwas für die weitere Verwendung priorisiert, z.B. die Ziele der SEM aus Schritt 1.

4. Messen – Im Schritt dieses Typs werden Werte für die weitere Verwendung gemessen, z.B. Messwerte der Metriken.

5. Analysieren – Im Schritt dieses Typs wird etwas für die weitere Verwendung analysiert, z.B. die gemessenen Werte der SEM.

6. Planen – Im Schritt dieses Typs wird etwas für die weitere Verwendung geplant, z.B. die Anpassung für eine SEM.

7. Ausführen – Im Schritt dieses Typs wird etwas für die weitere Verwendung ausgeführt, z.B. die geplanten Änderungen.

Wie bereits im zweiten Punkt erwähnt, kann man bei genauer Betrachtung des MAPE-K4SEM, den Typ „Ableiten“ in weitere Sub-Typen unterteilen, je nachdem, WAS abgeleitet wird. Dann ergeben sich folgende Sub-Typen:

a. Regeln,
b. Metriken,
c. Planungsmöglichkeiten,
d. Konflikte und Konfliktlösungsmöglichkeiten,
e. Ausführungen und Benachrichtigungen.

Für den hier vorgestellten Ansatz sind die Typen vollständig, da sie alle Schritte des MAPE-K4SEM abdecken und beschreiben. Für andere Beschreibungen, welche ebenfalls mit Ableitungen oder Schritten arbeiten, können sich weitere Schritte mit anderen Typen ergeben. Überträgt man die Typen beispielsweise auf den GQM-Ansatz (Goal – Question – Metric), auf welchen in Kapitel 5 noch näher eingegangen wird, so können hier zwei der Schritte entsprechend wiederverwendet werden. Die Definition der Ziele (Goal) ließe sich mit dem Typ „Erstellen“ abdecken und das Ableiten der Metriken (Metric) mit dem Schritt Ableiten und dem Subtyp „Metriken“. Für den Schritt des Ableitens der Fragen (Question) kann ebenfalls der Typ „Ableiten“ verwendet werden, aber es gebe hier einen neuen Subtypen, „Fragen“.

Um die einzelnen Schritte sowohl näher als auch auf einen Blick zu beschreiben und diese genauer zu charakterisieren wurde nicht nur das Prinzip der Typen eingeführt, sondern es wurde ein Framework erarbeitet, in welchem u.a. die Schritt-Typen verwendet werden. Dieses wird im nächsten Abschnitt erläutert.

4.4.2 Framework zur Charakterisierung

Um die einzelnen Schritte näher zu charakterisieren, wurde ein kurzes Framework erarbeitet. Dieses Framework gibt in zusammengefasster Form verschiedene Elemente, welche den Schritt beschreiben, wieder. Das Framework charakterisiert den Schritt kurz anhand seiner Aufgabe, seines Typs, den enthaltenen Subschritten, welche zur Durchführung des Schrittes nötig sind, sowie welchen Input der Schritt zur Ausführung benötigt und

was das Ergebnis – der Output – des Schrittes ist. Zusätzlich wird aufgeführt, welcher Stakeholder, das heißt, welche Verantwortlichen für den Schritt nötig sind.

Ähnlich wie die verschiedenen Typen der Schritte, kann dieses Framework nicht nur in diesem Ansatz, sondern auch in anderen Kontexten, welche ebenfalls mit Schritten arbeiten, wieder verwendet werden. Beispielsweise kann das Framework ebenfalls verwendet werden, um einzelne Schritte oder auch Aktivitäten innerhalb einer Software-Engineering-Methode zu beschreiben. Es kann auf einen Blick erfasst werden, was die Aufgabe des Schrittes ist, wie er charakterisiert ist (Typ), was mögliche Schritte innerhalb des Schrittes oder der Aktivität sind, was als Input benötigt wird, was der Output dieses Schrittes und wer dafür verantwortlich ist.

Schrittnummer + Schrittname	
Aufgabe	Die Aufgabe des Schrittes wird kurz und knapp beschrieben
Typ	Typ des Schrittes und möglicher Sub-Typ
Subschritte	Die einzelnen Subschritte, welche innerhalb des Schrittes durchzuführen sind, werden beschrieben
Input des Schrittes	Dokumente oder anderer Input, welcher benötigt wird, damit der Schritt ausgeführt werden kann
Output des Schrittes	Ergebnisse, z.B. Dokumente, welcher der Schritt als Output beispielsweise für andere Schritte liefert
Mögliche Stakeholder	Stakeholder bzw. Personen, welche am Schritt beteiligt und/oder für ihn verantwortlich sind

Abbildung 32 Framework zur Charakterisierung

Das Framework kann ebenfalls wenn der Kontext es erfordert um weitere Kategorien erweitert werden. In manchen Fällen kann es Sinn machen, ein einfaches Beispiel hinzuzufügen, um auf einen Blick zu erläutern, was hier gemeint ist. Des Weiteren kann es an manchen Stellen Sinn machen, anzugeben, mit welchem Schritt oder welchen Schritten der beschriebene Schritt zusammenhängt, z.B. wer sein Vorgänger oder sein Nachfolger ist. Da die Schritte in diesem Ansatz größtenteils sequentiell aufeinander aufbauen, wurde hier auf diese Kategorie verzichtet.

Wenn das Framework für eine Software-Engineering-Methode genutzt würde, dann würde das Einfügen der Kategorie Vorgänger/Nachfolger wiede-

rum mehr Sinn machen, um die Zusammenhänge im Überblick zu behalten. Ferner kann es bei einer SEM beispielsweise Sinn machen, eine Kategorie „Werkzeuge“ oder „Technik“ mit hinzuzufügen, wenn diese für einen Schritt verwendet werden sollen. Anhand dieser Beispiele ist zu sehen, dass eine Wiederverwendung und Übertragbarkeit des Frameworks gegeben ist.

In Kapitel 5 und Kapitel 6 findet das Framework seine Anwendung und es wird beschrieben, wie der jeweilige Schritt und seine Subschritte im Detail aussehen, ebenso wie der Input genutzt und der Output im Laufe des Schrittes erzeugt wird.

4.5 Vertiefung Messen, Monitor- und Analyse-Phase

In den folgenden Kapiteln 5 und 6 soll der vorgestellte Ansatz MAPE-K4SEM mit seinen zehn Schritten vertieft dargestellt werden. Da der Ansatz des SE Method Managers und die verschiedenen Phasen sehr komplex sind, sollen in dieser Arbeit insbesondere die Schritte der Pre-Work und die ersten beiden Phasen des MAPE-K, die Monitor- und die Analyse-Phase, vertiefend dargestellt werden.

Das Hauptaugenmerk liegt hierbei auf dem Messen der Daten und welche Daten für das Gelingen des MAPE-K gewonnen werden können. Gerade das Messen der Daten ist besonders wichtig, denn ohne die entsprechenden Werte kann die Feedbackschleife des MAPE-K nicht durchgeführt werden. Dabei ist zunächst wichtig zu wissen, was später überhaupt mit den entsprechenden Sensoren an Daten gemessen werden kann. Um dies herauszufinden, sind ebenfalls die Schritte 1-3 und 5 besonders wichtig und werden in Kapitel 5 vertiefend behandelt.

Auch wenn die anderen Schritte erläutert werden, liegt zusätzlich neben dem Messen das weitere Augenmerk auf den ersten beiden Phasen Monitor und Analyse, da diese eng verknüpft sind mit der Gewinnung der Daten. Ohne das Aufbereiten der gewonnenen Daten und einer korrekten Analyse mit den entsprechenden Regeln, kann eine automatische und somit selbstadaptive Anpassung einer Software-Engineering-Methode nicht funktionieren.

In diesem Kapitel wurde ein zielorientiertes Vorgehen und im Speziellen der Ansatz MAPE-K4SEM, bestehend aus der Kombination der Pre-Work und dem anschließenden MAPE-K, vorgestellt, um insbesondere den in Abschnitt 3.4 definierten Herausforderungen TP1 bis TP5 zu begegnen. Ferner wurden Schritt-Typen definiert und ein Framework zur Charakterisierung der einzelnen Schritte erarbeitet. Wie die genauen Schritte der Pre-Work und der MAPE-K-Feedbackschleife im Detail aussehen, wird in den Kapiteln 5 (Pre-Work) und 6 (MAPE-K) nun näher vorgestellt.

Kapitel 5 Pre-Work

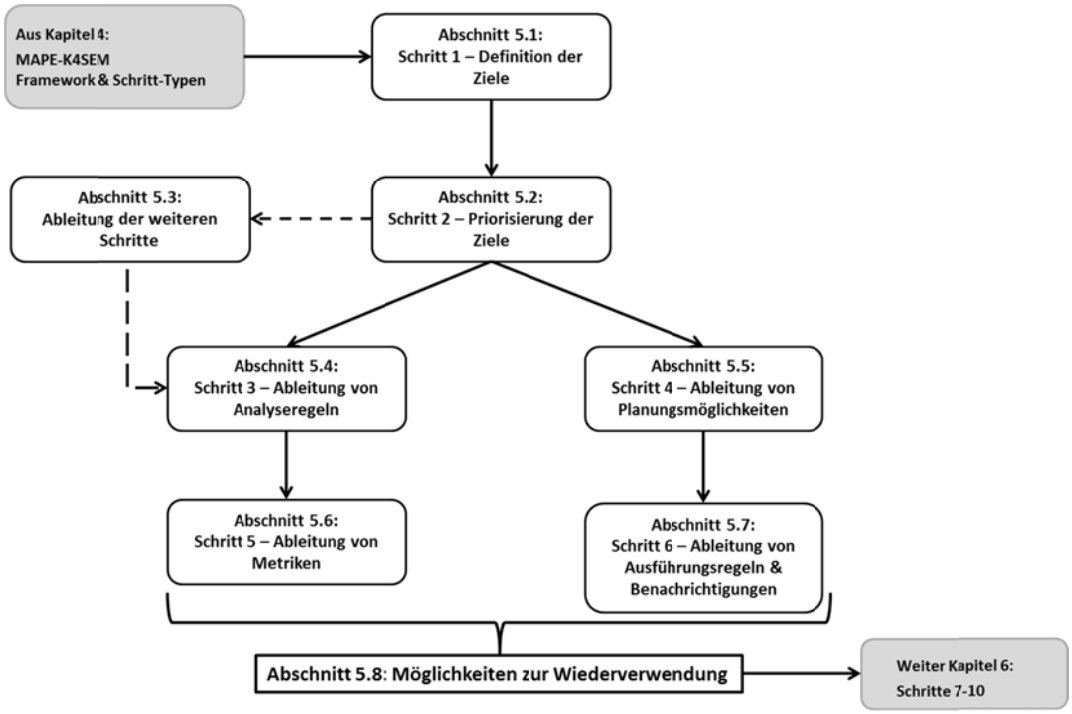

Abbildung 33 Aufbau Kapitel 5

Im vorherigen Kapitel wurde der MAPE-K4SEM-Ansatz mit seinen 10 Schritten vorgestellt. Diese Schritte beschreiben, wie sich die nötigen Inhalte für die MAPE-K-Feedbackschleife herleiten und anschließend verwenden lassen, um den beschriebenen Herausforderungen am Ende von Kapitel 3 zu begegnen. In diesem Kapitel werden nun die sechs Schritte der Pre-Work im Detail beschrieben sowie eine Möglichkeit zu ihrer Wiederverwendung vorgestellt. Zu Beginn jeden Schrittes werden sowohl das Framework als auch die Schritt-Typen aus dem vorherigen Kapitel verwendet, um den jeweiligen Schritt auf einen Blick in Kurzform darzustellen.

5.1 Schritt 1 – Definition der Ziele

Schritt 1: Definition der Ziele	
Aufgabe	Die einzelnen Ziele und Eigenschaften der Software-Engineering-Methode werden definiert
Typ	Erstellen – Die Ziele werden erstellt
Subschritte	1. Methode, Eigenschaften und Regeln der Software-Engineering-Methode als Ziele definieren (Hauptziel und Subziele) 2. Weitere Ziele/Anforderungen aus Kontext de-

	finieren (falls nötig) 3. Zeitpunkte der Schritte definieren 4. Bereiche für Abweichungen definieren
Input des Schrittes	Genutzte Software-Engineering-Methode (z.B. Scrum in der Originalversion oder Scrum mit Anpassungen etc.)
Output des Schrittes	Die definierten Ziele
Mögliche Stakeholder	Methoden-Engineer, aber auch Kunde, Management, Team usw.; alle, welche die Ziele mitdefinieren

Einer der wichtigsten Schritte zur Ableitung der verschiedenen Regeln und Metriken, damit die MAPE-K-Feedbackschleife anschließend eingesetzt werden kann, ist der erste Schritt – die Definition der Ziele. Die Ziele geben an, welche Vorgaben die Software-Engineering-Methode erfüllen muss, damit das Projekt erfolgreich und die damit zu entwickelnde Software von guter Qualität ist. Diese Vorgaben oder auch Regeln beschreiben die Eigenschaften der Software-Engineering-Methode, welche im weiteren Verlauf ebenfalls als Ziele definiert sind.

Zusätzlich zu den Eigenschaften der SEM soll noch der Kontext oder besser sollen die Eigenschaften des Kontextes in Bezug auf das Projekt mit betrachtet und als Ziele hergeleitet werden.

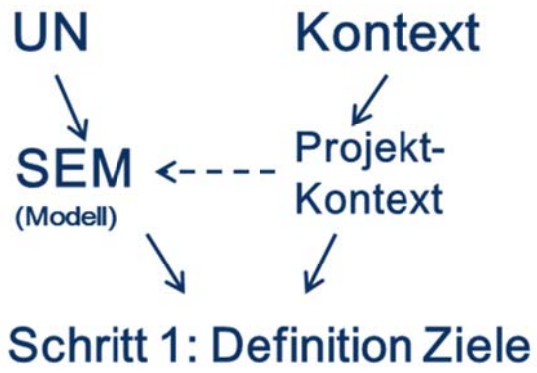

Abbildung 34 Herleitung und Definition der Ziele anhand der SEM und des Kontextes

Somit wirken auf die Definition der Ziele verschiedene Faktoren ein, auf der einen Seite die Softaware-Engineering- Methode selbst und auf der anderen Seite der Kontext.

Wie in Abbildung 34 zu sehen, hängt die Software-Engineering-Methode unter anderem vom Unternehmen ab. Jedes Unternehmen hat seine eigene SEM, diese kann jedoch von Projekt zu Projekt unterschiedlich sein. Des Weiteren kann sich der Kontext des Projektes auf die Software-Engineering-Methode auswirken. Ein gegebenes Modell einer SEM kann einmal für den Projektkontext anhand von Situational Method Engineering erstellt worden sein, oder es wurde für das Projekt zugeschnitten (Tailoring).

Im ersten Subschritt werden zunächst das konkrete Modell und die Beschreibung der Software-Engineering-Methode selbst betrachtet. Diese bringt Regeln, Eigenschaften, Vorgehensmethode etc. mit, welche für den Erfolg der SEM wichtig sind. Diese sollen von einem Methoden-Engineer oder einer anderen entsprechend geeigneten Person extrahiert und als Ziele erfasst werden. Ein kleines Beispiel ist, dass als Software-Engineering-Methode Scrum genutzt wird. Eine bereits erwähnte Regel von Scrum ist die Teamgröße, welche mindestens drei aber nicht mehr als neun Personen betragen darf, damit ein Team gut arbeiten und das Projekt Erfolg hat. Als Ziel definiert wäre dies nun: Die Teamgröße soll eingehalten werden und darf nicht mehr als 9 oder weniger als 3 Personen betragen.

So können im ersten Schritt die verschiedenen Ziele festgelegt und mit weiteren Verantwortlichen, welche für den Erfolg des Projektes wichtig sind, abgesprochen werden, so dass am Ende eine finale Version der Ziele steht. Dieser Schritt ist nicht nur der wichtigste, sondern er kann auch am längsten dauern. Die Definition und Abstimmung der Ziele nimmt einige Zeit in Anspruch, ähnlich wie bei dem Erstellen von Anforderungen. Die Ziele können auch als die Anforderungen an die Software-Engineering-Methode angesehen werden, damit die SEM am Ende von Erfolg gekrönt ist.

Jedoch ähnlich wie bei der Definition von Anforderungen kann die Anzahl der definierten Ziele je nach Komplexität der gewählten Software-Engineering-Methode relativ hoch sein. In diesem Schritt ist deswegen darauf zu achten, die Ziele nicht zu feingranular zu fassen. Die weitere Verfeinerung erfolgt später anhand der Regeln, Planungsmöglichkeiten, Metriken usw. Eine Möglichkeit, die Anzahl zu reduzieren wäre beipsielsweise falls Ziele einen gleichen Aspekt behandeln ist zu überlegen und zu diskutieren, ob diese zu einem Ziel zusammengefasst werden können. Dies kann beispielsweise über ein „Hauptziel“ und „Subziele“ erfolgen. In Schritt 2 werden anschließend nur die Hauptziele priorisiert.

Ein Beispiel wäre, dass in Scrum verschiedene Meetings, beispielsweise das tägliche Daily Scrum-Meeting oder am Ende das Review-Meeting, durchgeführt werden sollen. Das Hauptziel würde heißen: Es sollen alle Meetings entsprechend ihrer Reihenfolge durchgeführt werden. Das Subziel könnte

dann lauten: Es soll täglich ein Daily Scrum-Meeting von 15 Minuten Dauer immer zur selben Zeit durchgeführt werden.

Im nächsten Subschritt wird nun die rechte Seite aus Abbildung 34 betrachtet und falls nötig werden weitere Ziele und Anforderungen aus dem Kontext heraus definiert. Auch hier sind der Methoden-Engineer und bei Bedarf weitere Verantwortliche nötig. Da jedes Projekt in einem anderen Kontext stattfindet, gibt es andere Voraussetzungen, unter denen die Software-Engineering-Methode funktionieren muss. Einen Einfluss können beispielsweise verschiedene Standards und Normen haben, welche ein Unternehmen einsetzen möchte oder sogar muss. Ein Beispiel kann sein, dass das Unternehmen einen Standard oder eine Norm bezüglich der Gewinnung von Testdaten nutzen möchte. Das Ziele wäre dann: Nutze den aktuellen Standard bezüglich der Gewinnung von Testdaten.

Ferner kann es vorkommen, dass Ziele nur zu bestimmten Zeitpunkten, beispielsweise in einer bestimmten Phase, greifen. Somit kann bestimmt werden, wann ein Ziel „aktiv“ wird, d.h. dass die später abgeleiteten Metriken nur zu diesem Zeitpunkt gemessen werden. Neben den Zielen zu bestimmten Zeitpunkten kann es welche geben, die immer greifen sollen, beispielsweise das Standards eingehalten werden müssen. Für das Review-Meeting oder Planungsmeeting in Scrum würden Ziele beispielsweise nur zu Beginn eines Sprints (Planung) oder am Ende (Review) aktiv werden. Die verschiedenen Zeitpunkte sollen bei der Priorisierung mit beachtet werden.

Ein anderes Beispiel wäre die Einbeziehung des Kunden, ähnlich dem Beispiel „Quasi-Scrum“ aus Kapitel 3.1.3.1. Dort hat sich am Ende herausgestellt, dass es wichtig ist, den Kunden mehr mit einzubeziehen (durch eigene Aufgaben) als vorgegeben. Dies hatte sich über die Kundenzufriedenheit oder besser die Kunden**un**zufriedenheit anhand vieler (Fehler-) Meldungen herausgestellt. Das Hauptziel würde hier lauten: Die Kundenzufriedenheit im Projekt soll hoch sein. Ein Subziel könnte dann heißen: Es darf nur eine bestimmte Anzahl, welche entsprechend vorher definiert werden muss, von negativen Meldungen oder Fehlermeldungen vom Kunden vorhanden sein. Dafür muss natürlich sichergestellt sein, dass der Kunde regelmäßig die Software begutachten und Fehlermeldungen verfassen darf. Dies kann in Scrum gut realisiert werden und wurde teilweise bereits realisiert. Je nachdem, ob der Kunde nur nach der Auslieferung des Inkrements oder die ganze Zeit über Meldungen verfassen darf, wird das Ziel immer oder nur nach dem Sprint geschaltet.

Sind die Ziele sowohl aus der Beschreibung der Software-Engineering-Methode selbst extrahiert als auch anhand des Kontextes erfasst, erfolgt ein dritter Subschritt. In diesem Schritt werden, soweit dies bei dem Ziel mög-

lich ist, die Grenzen, in denen sich das Ziel bewegen darf, definiert. Diese können bei der Beschreibung der Software-Engineering-Methode bereits vorgegeben sein oder können entsprechend während der Besprechung und dem Erstellen der Ziele definiert oder geändert werden.

Im ersten Beispiel bezüglich der Teamgröße in Scrum sind die Grenzen zum einen vorgegeben, die Teamgröße muss sich zwischen mind. 3 und max. 9 Personen bewegen. Diese Grenzen können zum anderen aber auch angepasst werden, bei der Teamgröße können beispielsweise beide Grenzen auf 4 (untere Grenzen) oder auf 10 (obere Grenze) angehoben werden.

5.2 Schritt 2 – Priorisierung der Ziele

Schritt 2: Priorisierung der Ziele	
Aufgabe	Für die einzelnen Ziele werden Prioritäten vergeben, z.B. Wichtigkeit des Ziels und/oder Kritikalität des Ziels (ähnlich einer Fehlerpriorisierung)
Typ	Priorisieren
Subschritte	Keine Subschritte vorhanden
Input des Schrittes	Konkrete Ziele aus Schritt 1
Output des Schrittes	Priorisierte Ziele
Mögliche Stakeholder	Kunde, Management, Team; alle, welche die Ziele mitpriorisieren

Sobald die Ziele der Software-Engineering-Methode festgelegt sind, müssen diese nun im zweiten Schritt priorisiert werden. Dieser Schritt ist besonders wichtig für die spätere Durchführung der Analyse und Planung von gemessenen Werten, um eine mögliche Reihenfolge dafür festzulegen. Wie in Kapitel 3.4.2. beschrieben, hilft dieser Schritt vor allem dabei, das Triggerproblem TP3 zu lösen.

Dabei müssen die Verantwortlichen bei der Priorisierung des Ziels betrachten, wie wichtig das Ziel für den Erfolg des Projektes und somit für den Erfolg der Software-Engineering-Methode ist. Die Wichtigkeit des Ziels bestimmt, unter Einhaltung von bestimmten Grenzen (siehe Schritt 1), später die Sortierung für die Analyse, die Planungsdurchführung und zum Teil auch für die Anpassungsausführung.

Für die Priorisierung eines Zieles anhand seiner Wichtigkeit gibt es verschiedene Möglichkeiten. Eine einfache Möglichkeit wäre dabei, den Zielen Zahlen zuzuordnen, welche die Priorität darstellen. Ein Beispiel wären hier Schulnoten, wobei 1 die höchste Priorität widerspiegelt und 6 die niedrigste Priorität. Umgekehrt wäre es möglich, den Zielen eine Zahl von beispielsweise 1 bis 10 zuzuordnen, wobei 10 (= hohe Zahl) die höchste Priorität ist und 1 die niedrigste.

Eine andere bekannte Methode, welche häufig in den agilen Methoden wie beispielsweise Scrum zur Priorisierung des Product Backlogs eingesetzt wird, ist die sogenannte MusCoW-Methode [Gl08]. MuSCoW ist dabei die Abkürzung für „Must“, „Should“, „Could“ und „Would not“. Must bedeutet dabei, dass eine Anforderung oder ein Ziel ein Muss ist. Wird dieses Ziel verletzt oder diese Anforderung nicht umgesetzt, funktioniert die ganze Software nicht und der Erfolg des Projektes ist hochgefährdet. Should ist in diesem Zusammenhang zweitrangig, das heißt, die Anforderung bzw. das Ziel sollte umgesetzt werden. Wird das Ziel nicht umgesetzt, ist der Erfolg gefährdet, aber nicht so stark wie bei Must. Could bedeutet hier, dass es schön wäre, wenn das Ziel umgesetzt wird (nice to have). Wird dies nicht umgesetzt, ist der Erfolg wenig gefährdet. Interessant ist bei der MuSCoW-Methode das Would not. Hier wird zusätzlich definiert, was NICHT umgesetzt werden soll. Bei der ursprünglichen Methode wird damit definiert, welche Anforderungen auf keinen Fall umgesetzt werden sollen. Auf Ziele einer Software-Engineering-Methode umgemünzt hieße dies, welche Ziele und Eigenschaften der Methode zum Misserfolg des Projektes führen würden, wenn sie umgesetzt sind. Von der Wichtigkeit kann das Would not in diesem Fall zwischen Must und Should angesiedelt werden. Übersetzt man die MuSCoW-Methode in Zahlen, wären dies Prioritäten von 1 bis 4, wobei Must die Priorität 1, Would not die Priorität 2, Should die Priorität 3 und Could die Priorität 4 erhalten würde.

Eine andere Möglichkeit wäre es, die Ziele der Software-Engineering-Methode anhand ihrer Kritikalität zu beurteilen, ähnlich einer Fehlerpriorisierung beim Testen. In Testprojekten, z.B. bei Nutzung der Tools JIRA [At14] oder HP Quality Center Enterprise [HP14], können die Fehler bewertet werden und bekommen Attribute wie beispielsweise Blocker oder leichter Fehler wie Schönheitsfehler. Dabei bekommt ein blockierender Fehler in der Software die höchste Priorität, denn dies heißt, wenn dieser Fehler nicht (sofort) behoben wird, blockiert die ganze Software und stürzt im schlimmsten Fall ab. Der Fehler muss schnellstmöglich behoben werden. Ein Schönheitsfehler besitzt hier die niedrigste Priorität und ist beispielsweise beim Testen von Software bei der Realisierung der GUI zu finden, z.B. dass eine Farbe falsch gesetzt ist oder ein Feld nicht korrekt dargestellt wird. Dieser Fehler sollte behoben werden, aber die Software funktioniert

auch ohne die Behebung einwandfrei. Zwischen diesen Kategorien kann es verschiedene Abstufungen geben, beispielsweise insgesamt fünf Stufen oder mehr.

Um die spätere Erstellung der Reihenfolge für die Analyse zu vereinfachen, sollte die Priorisierung aus einer Zahlenfolge bestehen. Dabei können sowohl die MuSCoW-Methode als auch die Methode zur Fehlerpriorisierung helfen, diese Reihenfolge anhand von Prioritäten zu erstellen.

Wichtig ist es darauf zu achten, die Abstufungen nicht zu feingranular (zu viele Abstufungen zu wählen) aber auch nicht zu grobgranular (zu wenig Abstufungen). Ist die Abstufung zu grobgranular, kann dies dazu führen, dass keine Reihenfolge während der Analyse erstellt wird, da die Ziele dieselben Prioritäten haben. Ist die Abstufung zu feingranular gewählt bedeutet dies einen sehr hohen Aufwand, wenn die Menge der Ziele sehr hoch ist. Von daher muss die Abstufung in Abhängigkeit von der Anzahl der Ziele gewählt werden. Zusätzlich sollte mit betrachtet werden, wann das Ziel aktiv sein wird. Greift beispielsweise zwei Ziele zu völlig unterschiedlichen Zeitpunkten, können sie dieselbe Priorität erhalten.

Sollte es dennoch vorkommen, dass zwei Ziele gleichzeitig gemessen UND dieselbe Priorität besitzen, muss der Methoden-Engineer oder der Projektleiter eingreifen und beurteilen, welches Ziel zuerst analysiert wird. Ansonsten kann die Durchführung automatisiert ohne das Eingreifen von Verantwortlichen erfolgen.

5.3 Ableitung der weiteren Schritte

Die Ziele der Software-Engineering-Methode sind nun definiert und wurden priorisiert. In den folgenden Schritten geht es jetzt darum, aus diesen Zielen zunächst wie in Kapitel 4.3.2. im Ableitungs- und Schalenmodel zu sehen, die Regeln für die Analyse, die möglichen Planungsmöglichkeiten und daraus jeweils Metriken und Ausführungsregeln und Benachrichtigungen weiter ab- und herzuleiten.

Mit Hilfe der Ableitungsschritte wird, ausgehend von den Zielen, die Frage beantwortet, was gemessen werden muss und wie die zu messenden Werte gefunden werden können. Die Werte ergeben am Ende in Schritt 5 die Metriken. Diese sind die Grundlage für die anschließenden Sensoren im späteren MAPE-K, welche die entsprechenden Werte für die Metriken messen.

Dies ist aber, wie auch im Ableitungsbaum zu sehen, nur die eine Seite (links). Auf der anderen Seite steht die Frage, was angepasst wird und was geplant werden kann. Dies erfolgt ebenfalls mit Hilfe der Ableitung ausgehend von den Zielen. Sobald die Planungsmöglichkeiten bekannt sind, kann

mit ihrer Hilfe bestimmt werden, welche „Anpassungs- und Benachrichtigungs-Metriken“ ausgeführt werden.

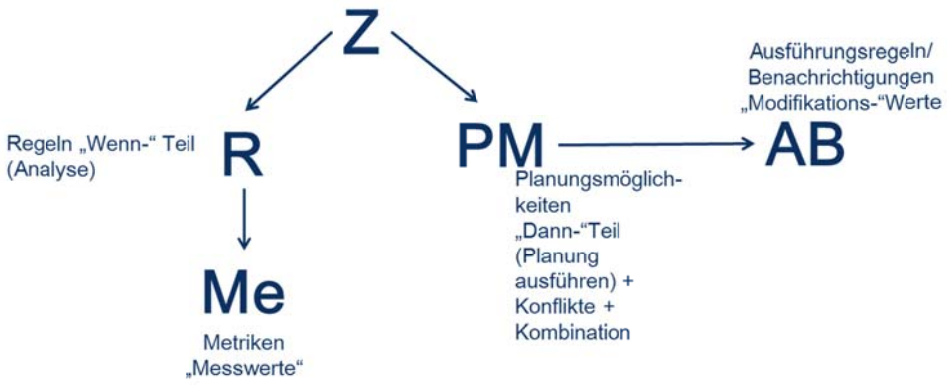

Abbildung 35 Weitere Ableitungen

Die Ableitungsreihenfolge, welche ebenfalls im Ableitungsbaum in Kapitel 4.3.2. zu finden ist, wird noch einmal konkret für die Pre-Work in Abbildung 35 dargestellt. Aus den priorisierten Zielen werden die Regeln für die Analyse und daraus die Metriken für die Messwerte hergeleitet. Auf der anderen Seite werden die Planungsmöglichkeiten sowie mögliche Konflikte und Kombinationen für Anpassungen hergeleitet. Aus diesen ergeben sich anschließend wie beschrieben die genauen Ausführungsregeln und Benachrichtigungen, genauer die Metriken und Werte zur Modifikation.

Wie ist es nun möglich, aus den priorisierten Zielen die entsprechenden Punkte herzuleiten? Nach [BCR94] ist es nötig, die Herleitung nach einem Top-Down-Ansatz vorzunehmen und sie sollte sich dabei auf spezifische Ziele oder auch Modelle fokussieren. Um messbare Ziele zu definieren und entsprechende Metriken herleiten zu können, ist ein Ansatz das bekannte GQM-Model, wobei GQM für „Goal“, Question“ „Metric“ steht [BCR94, SB99].

Für diesen Ansatz ist ebenfalls die Idee, einen Teil des GQM-Modells anzuwenden, um aus den priorisierten Zielen Regeln und Metriken herzuleiten. Das Modell wird typischerweise als eine Verfahrensweise verwendet, um ein Qualitätsmodell zu erstellen. Dieses ist hierarchisch aufgebaut und besteht am Ende aus Metriken sowie einem spezifischen Set an Regeln, welche für die Interpretation der gemessenen Daten wichtig sind. Das Modell enthält nachher drei verschiedene Ebenen [BCR94]:

1. Die Konzeptions-Ebene – die Ziele (Goals) – welche für ein bestimmtes Objekt mit verschiedenen Sichtweisen definiert sind. Dies können in der Softwareentwicklung verschiedene Arten sein. Beispiele sind das Produkt (Artefakte, Inkremente etc.), die Prozesse

oder Phasen im Entwicklungsprozess (Spezifikation, Design, Test usw.) oder auch für die Ressourcen, welche für die Entwicklung genutzt werden.
2. Die Operations-Ebene (Question), in der eine Anzahl von Fragen erarbeitet wird. Diese Fragen charakterisieren, wie ein bestimmtes Ziel erreicht oder gemessen wird.
3. Die Quantitative-Ebene (Metric), in der eine Anzahl von Daten erarbeitet wird, um die Fragen in einer quantitativen Form zu beantworten. Dies kann sowohl in objektiver (z.B. Versionsnummer von Dokumenten, Aufwand für Aktivitäten, Lines of Codes usw.) als auch in subjektiver Form (z.B. Zufriedenheit der Benutzer, Lesbarkeit eines Textes usw.) geschehen.

Diese drei Ebenen können in drei Fragen umgewandelt werden, die es zu beantworten gilt. Sie werden genutzt, um das Modell zu erstellen:

1. Welches Ziel soll durch die Messung erreicht werden? (Goal)
2. Was soll gemessen werden; welche Fragen soll die Messung beantworten? (Question)
3. Welche Metrik(en) sind in der Lage, die notwendigen Eigenschaften zu beschreiben? (Metric)

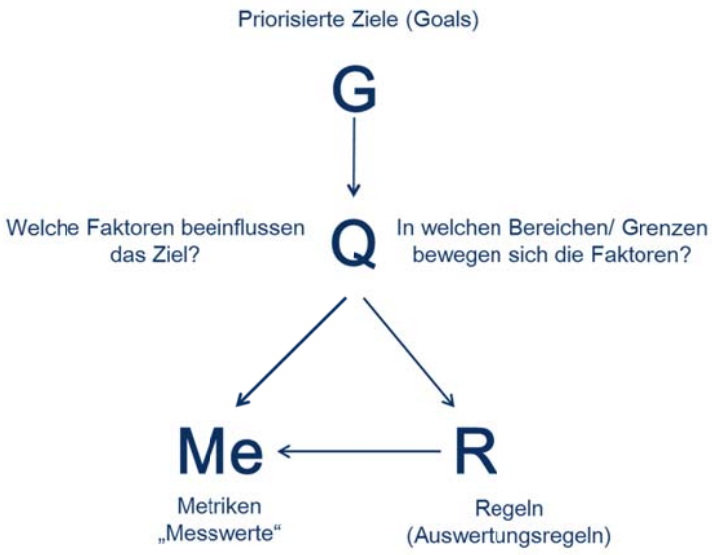

Abbildung 36 Ableitung GQM-Variante linke Seite

Genau diese Fragen sollen in ähnlicher Weise durch die Pre-Work sowohl für die linke, als auch die rechte Seite des Ableitungsbaums beantwortet und können diesem somit zugrunde gelegt werden. Allerdings müssen die Fragen dann jeweils umformuliert werden. Als oberstes bleiben wie in Abbildung 36 zu sehen, immer noch die Ziele erhalten. Um die jeweils weiteren

Schritte und Ergebnisse zu erhalten, werden zunächst entsprechende Fragen gestellt, um anschließend für die linke Seite erst die Regeln und dann die Metriken herzuleiten:

1. Welches Ziel oder welche Eigenschaft soll bei der Software-Engineering-Methode erhalten bleiben und kann durch eine Messung sowie Analyse überprüft werden? (Goal, Schritt 1 + 2)
2. Was soll dafür analysiert und welche Grenzen können dafür eingehalten werden? Welche Fragen muss ich für diese Analyse beantworten um die Regeln zu erhalten? (Questions)
3. Welche Metriken sind in der Lage, die nötigen Werte für die Analyse der Regeln zu liefern? Welche Faktoren beeinflussen das Ziel?(Regeln, Schritt 3, Metric Schritt 5)

Mögliche Fragen können hier sein:

a. Wie stelle ich das Ziel sicher, was ist dafür nötig?
b. Zu welchem Zeitpunkt greift das Ziel?
c. Welche Faktoren beeinflussen das Ziel und müssen beobachtet werden?
d. In welchen Bereichen/Grenzen bewegen sich diese Faktoren damit das Ziel erreicht wird und erhalten bleibt?

Wie in Abbildung 36 zu sehen ergibt hierbei die Leitfrage c) die späteren Metriken und Leitfrage d), ergibt anschließend die Regel für die Auswertung. Beide Leitfragen können durch weitere Fragen verfeinert werden. Wichtige Fragen, um eine Software-Engineering-Methode anzupassen und um mögliche Schwächen oder Abweichungen festzustellen, könnten außerdem noch sein:

- Welche Abhängigkeiten in der Software-Engineering-Methode sind kritisch?
- Welche Veränderungen am Projekt können Einfluss auf die Software-Engineering-Methode nehmen?
- Wie wird sichergestellt, dass alle Beteiligten über eine Anpassung und somit eine Veränderung in der SEM informiert sind?
- Wie werden neue Risiken und Gefahren für die Software-Engineering-Methode erfasst und in MAPE-K integriert?
- Welche Übergänge im adaptiven Zyklus lassen sich wie managen?

Ein ähnliches Vorgehen kann für die Schritte der rechten Seite vorgenommen werden, welche in Abbildung 37 zu sehen sind. Dabei bleibt der erste Punkt gleich und anschließend werden mit Hilfe der Fragen zunächst die Planungsmöglichkeiten und dann die Ausführungs- und Benachrichtigungs-Metriken hergeleitet:

1. Welches Ziel soll bei der Software-Engineering-Methode erhalten bleiben und kann durch eine Messung und Analyse überprüft werden? (Goal, Schritt 1 + 2)
2. Was muss unter bestimmten Voraussetzungen geplant werden, damit das Ziel weiter eingehalten wird. Welche Fragen muss ich stellen, damit eine Anpassung geplant werden kann, um das Ziel weiter einzuhalten? (Questions)
3. Welche Anpassungsschritte sind nötig; an welchen Stellen muss die Software-Engineering-Methode angepasst und wer muss benachrichtig werden? Wie sehen diese „Anpassungs-Metriken“ aus? (Planungsmöglichkeiten Schritt 4, Metric, Schritt 6)

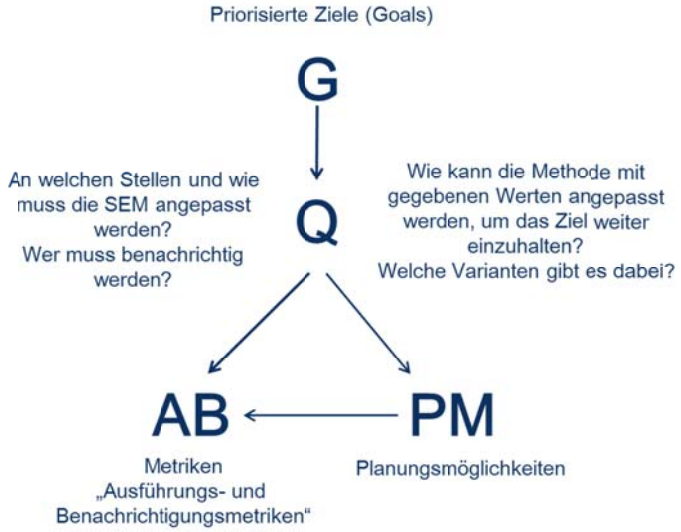

Abbildung 37 Ableitung GQM-Variante rechte Seite

Mögliche Fragen können hier sein:

a. Wie stelle ich das Ziel sicher, was ist dafür nötig?
b. An welchen Stellen kann eine SEM angepasst werden und wie kann eine SEM angepasst werden (siehe dazu Kapitel 3.1.2.)?
c. Wer muss bei einer Anpassung alles benachrichtig werden?
d. Wie kann die gegebene SEM mit den gemessenen und analysierten angepasst werden, damit das Ziel weiter eingehalten wird?
e. Welche Varianten gibt es dabei?
f. Welche Konflikte können bei den jeweiligen Zielen entstehen?
g. Wie können Anpassungen kombiniert werden, gibt es bestimmte Kombinationspunkte?

Die Leitfragen b) und c) ergeben mit Hilfe der Planungsmöglichkeiten die Ausführung-Metriken. Die Beantwortung der restlichen Leitfragen ergibt mit möglichen Verfeinerungsfragen die Planungsmöglichkeiten.

Wie genau die weiteren Ableitungen und somit die Schritte im Detail aussehen, wird in den Abschnitten 5.4. bis 5.7. beschrieben.

5.4 Schritt 3 – Ableitung von Analyseregeln

Schritt 3: Ableitung von Analyseregeln	
Aufgabe	Für die Analyse-Phase werden anhand der Ziele mögliche (Auswertungs-)Regeln abgeleitet
Typ	Ableiten Sub-Typ: Regeln
Subschritte	1. Auswertungsteil ableiten (Wenn-Teil) 2. Aktionsteil ableiten (Dann-Teil)
Input des Schrittes	Priorisierte Ziele aus Schritt 2 und Grenzwerte
Output des Schrittes	Regeln für die Analyse
Mögliche Stakeholder	Methoden-Engineer, evtl. Projektmanagement

Nachdem die Ziele nun priorisiert sind, ist der nächste Schritt, die Analyseregeln für die spätere Analyse-Phase zur Durchführung des MAPE-K abzuleiten. Wie im vorherigen Abschnitt beschrieben, wird dazu eine Form des GQM-Modells verwendet. In Schritt 1 wurden die Ziele definiert u.a. durch die Eigenschaften der Software-Engineering-Methode, welche während der Durchführung der SEM eingehalten werden sollen. Zusätzlich wurden Toleranzbereiche und mögliche Abweichungen angegeben, in denen sich die Ziele zur Einhaltung bewegen dürfen. Diese Angaben müssen nun mit Hilfe der Leitfrage „In welchen Bereichen/Grenzen bewegen sich diese Faktoren damit das Ziel erreicht wird und erhalten bleibt?“ in eine Regel-Form überführt werden.

Die Analyseregeln bestehen später aus zwei Teilen, aus einem Auswertungsteil, dem Wenn-Teil und einem Aktionsteil, dem Dann-Teil. In diesem Schritt geht es hauptsächlich um den Auswertungsteil, welcher aus den Zielen abgeleitet wird. Er ist dafür zuständig zu überprüfen, ob ein bestimmtes Ziel sich noch innerhalb der gesetzten Grenzen befindet und ob es noch erfüllt wird oder nicht. Je nach Ergebnis der Auswertung, beschreibt der Aktionsteil was entsprechend zu tun ist.

Der Aktionsteil ist eng mit Schritt 4 verknüpft, der Ableitung von Planungsmöglichkeiten und deren Alternativen. Durch den Dann-Teil wird

beschrieben, welche Planungsmöglichkeit oder Alternative aufgerufen wird. Falls eine bestimmte Voraussetzung, welche durch den Wenn-Teil ausgewertet wird, erfüllt ist, wird eine entsprechende Anpassung der Software-Engineering-Methode aufgerufen. Es kann allerdings Situationen geben, wo die Analyseregel ergibt, dass sich das Erfüllen des Ziels nicht mehr in seinen Grenzen befindet, aber auch keine Planungsmöglichkeit vorhanden ist. Die Anpassung muss dann während der Planphase bestimmt werden.

Daraus ergeben sich drei mögliche Regel-Formen, welche u.a. davon abhängen, ob es eine Planungsmöglichkeit gibt oder nicht

1. Wenn „Angabe/ Ziel nicht erfüllt“ dann „Planungsmöglichkeit mit Werten aus Monitor“

2. Wenn „Angabe/ Ziel nicht erfüllt“ dann „Neue Planung mit Werten aus Monitor + Ziel“

3. Wenn „Angabe/ Ziel erfüllt“ dann Schritt 7 (Monitor-Phase)

Hier soll noch einmal das in Abschnitt 3.3.2 eingeführte Beispiel der Teamgröße in Scrum verwendet werden. Ist das Team zu groß, ist die Regel, dass das Team gesplittet und ein Scrum of Scrums durchgeführt wird (Planungsmöglichkeit Variante 1). Ist das Team zu klein, muss eine weitere Person mit entsprechenden Skills gesucht und dem Team hinzugefügt werden (Planungsmöglichkeit Variante 2). Daraus ergeben sich nach erstens und drittens folgende Regeln:

a. Wenn Teamgröße > 9 dann Planungsmöglichkeit Variante 1 (Team zu groß, aktuelle Teamgröße x)
b. Wenn Teamgröße < 3 dann Planungsmöglichkeit Variante 2 (Team zu klein, aktuelle Teamgröße y)
c. Wenn Teamgröße 3 <= Teamgröße <= 9 dann Schritt 7 (Monitor-Phase)

Wie in 4.1. und genauer in 4.3.1.4 beschrieben sind typischerweise die Planungsmöglichkeiten bereits vorhanden. Es gibt einige Situationen, in denen der Kontext eine Planung ohne vorherige Planungsmöglichkeiten triggern kann. Ein Beispiel dafür ist, dass das Ziel ist, dass die SEM bestimmte Normen und Standards einhalten soll, z.B. den aktuellen Standard zum Testen. Wenn sich dieser nun ändert, ist es nötig, die Software-Engineering-Methode darauf abzustimmen. Die Regel würde nach zweitens dafür lauten:

- Wenn aktuelle Norm/aktueller Standard != aktuell genutzte Norm/ Standard, dann Neue Planung mit neuer Norm/ Standard (auslesen aus Datenbank) *else* Schritt 7 (Monitor-Phase)

Damit diese Regel funktioniert, muss zunächst an einer Stelle, z.B. in der Wissensbasis, immer bekannt sein, welche Norm/ welcher Standard aktuell verwendet wird. Zum anderen muss es entweder möglich sein, den Kontext, in diesem Fall beispielsweise eine entsprechende Seite im Internet, zu scannen und auf eine mögliche neue Norm oder einen neuen Standard hin zu überprüfen. Im einfacheren Fall wird regelmäßig von einem Teammitglied oder einem Mitarbeiter des Unternehmens überprüft, ob es neue Normen oder Standards gibt. Diese werden dann in eine entsprechende Datenbank eingetragen.

5.5 Schritt 4 – Ableitung von Planungsmöglichkeiten

Schritt 4: Ableitung von Planungsmöglichkeiten	
Aufgabe	Aus den priorisierten Zielen (und der SEM) werden, soweit möglich, Planungsmöglichkeiten abgeleitet
Typ	Ableiten Sub-Typ: Planungsmöglichkeiten
Subschritte	1. Ab- bzw. Herleitung von Planungsmöglichkeiten 2. Variantenbestimmung 3. Bestimmung von Konfliktpotential (und mögliche Lösungen) 4. Kombinationsmöglichkeiten und -punkte
Input des Schrittes	Priorisierte Ziele und Software-Engineering-Methode
Output des Schrittes	Möglichkeiten für die Planung
Mögliche Stakeholder	Methoden-Engineer

Ähnlich wie aus der vorgegebenen Software-Engineering-Methode, den Projektzielen und einem Toleranzbereich, Regeln abgeleitet werden können, können im Vorfeld Anpassungs- bzw. Planungsmöglichkeiten bereits hergeleitet werden. Die Herleitung der Planungsmöglichkeiten ist für die meisten, aber für einige Ziele wie das Einhalten eines Standards, nicht möglich.

Wie in Abschnitt 5.3 beschrieben, sind zunächst die einzelnen Ziele und Eigenschaften zu betrachten, für welche bereits im die Regeln und Toleranzbereiche bestimmt wurden. Nun ist zu überlegen, was getan werden muss, wenn diese Toleranzbereiche über- oder unterschritten werden und somit die Gefahr besteht, dass das Ziel nicht mehr eingehalten wird. Es ist

nun die Leitfrage zu beantworten, was geplant werden muss, damit das Ziel weiter eingehalten werden kann und somit die SEM weiter Erfolg hat (Schritt „Q“ aus GQM).

5.5.1 Herleitung

Da durch das Ziel, den Regeln und dem Toleranzbereich genau bekannt ist, an welcher Stelle in der SEM ich mich gerade befinde, müssen verschiedene Dinge betrachtet werden. Zunächst lautet jeweils die Frage, was genau angepasst werden soll, wenn die Regel nicht eingehalten wird. Es muss dafür genauer überlegt werden,

a. **Wo** muss im Modell der Software-Engineering-Methode angepasst werden?
b. **Was** kann an dieser Stelle angepasst werden, z.B. ein Artefakt, eine Rolle, eine Aktivität, eine Kombination daraus usw.?
c. **Wie** muss an dieser Stelle angepasst werden, damit das Ziel weiter sichergestellt wird, z.B. muss etwas hinzugefügt, etwas ausgetauscht, etwas gelöscht werden usw.?
d. **Welche** Verbindungen müssen dabei neu gesetzt werden und welche bleiben erhalten, z.B. welche Rollen gehören zu welchen Aktivitäten, bleibt der Workflow erhalten oder muss er neu gestaltet werden (neue Verbindungen setzen) usw.?
e. **Wer** muss alles über die Änderungen benachrichtigt werden, z.B. welche neuen und alten Teammitglieder bekommen eine Benachrichtigung? Oder erhält das System eine Nachricht, wenn etwas automatisch durchgeführt werden kann usw.?

Auch wenn durch die Ziele und Regeln bekannt ist, an welcher Stelle angepasst werden muss, ist das gegebene Modell der Software-Engineering-Methode wichtig um die Stelle des Ziels und der Regeln im Modell genau zu definieren, also wo genau ich mich befinde (Punkt a). Nur anhand des Modells und durch das Bekanntsein der Stelle, wo ich mich nun im Modell befinde, ist es möglich herauszufinden, was an dieser Stelle angepasst werden kann. Denn durch das Modell sind die Elemente, welche sich an der anzupassenden Stelle befinden, genau definiert.

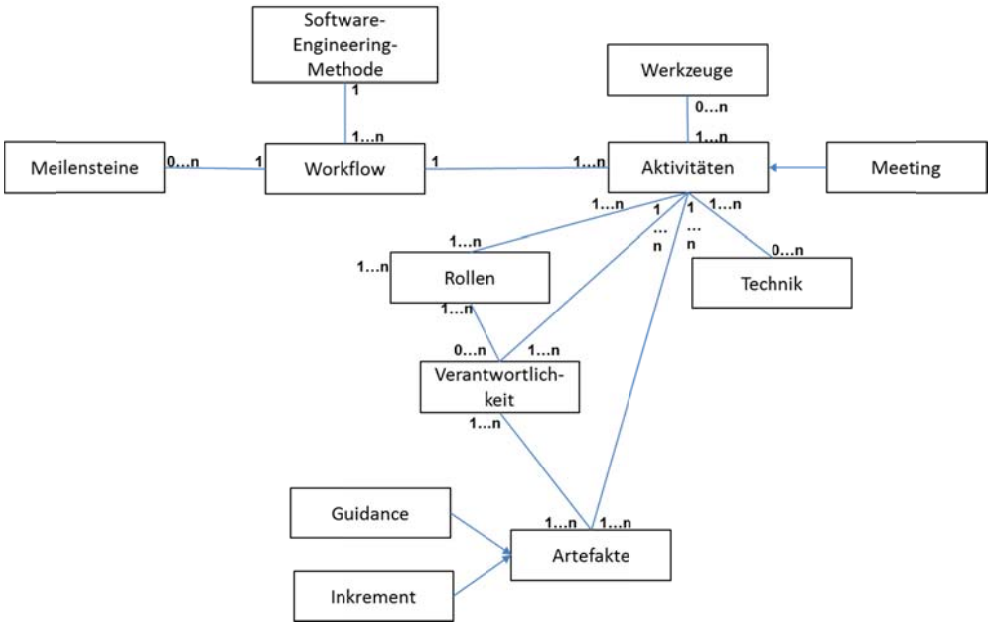

Abbildung 38 Begriffe einer Software-Engineering-Methode (vergl. Abschnitt 2.1.1)

In Abschnitt 2.1.1 wurde bereits beschrieben, welche Elemente eine Software-Engineering-Methode enthalten kann, also WAS (Punkt b) angepasst werden kann. Die wichtigsten Elemente, die in Betracht gezogen werden müssen, sind wie auch in Abbildung 38 zu sehen:

- Aktivitäten (= Aufgaben), welche innerhalb einer SEM durchgeführt werden müssen.
- Rollen (z.B. Projektleiter, Teammitglied, Tester, Architekt usw.) zugeordnet zu Aktivitäten.
- Artefakte (z.B. Dokumente, Arbeitsergebnisse, fertige Software usw.), welche bei einzelnen Aktivitäten entstehen bzw. für die Durchführung weiterer Aktivitäten benötigt werden.
- Verantwortlichkeiten, die eine Rolle besitzt und Aktivitäten oder Artefakten zugeordnet sind.
- Workflow (= Zusammenhang und Reihenfolge) der einzelnen Aktivitäten in Kombination mit Rollen und Artefakten.
- Meetings (= Aktivität), welche regelmäßig in SEMs durchgeführt werden.
- Guidance (= Hilfsmittel, Anleitungen), alles was eingesetzt wird, um bei der Durchführung einer Aktivität zu helfen bzw. um das gesetzte Ziel zu erreichen. Dabei können Technik und Werkzeuge ebenfalls als Unterkategorien angesehen werden.

- Technik (z.B. Priorisierungstechnik, Pair Programming bei der Entwicklung usw.), welche für die Durchführung einer Aktivität und das Erstellen eines Artefakts eingesetzt wird.
- Werkzeuge (z.B. Software-Tools), welche für die Durchführung einer Aktivität eingesetzt werden und diese unterstützen.
- Qualifikation (= Skills), welche eine Person besitzen muss, um für eine bestimmte Aktivität eingesetzt zu werden. Dies kann auch als Unterkategorie für die Rolle angesehen werden oder wie in Abschnitt 2.1.1 beschrieben zugehörig zu einer Person sein.
- Meilensteine, welche den Abschluss eines bestimmten Zeitraums symbolisieren, zu dem bestimmte Dinge (z.B. Arbeitsergebnisse, fertige Software usw.) erreicht sein müssen/sollen.

Diese Elemente wurden ausgewählt, da sie die in Abschnitt 2.1.1 definierten Elemente widerspiegeln. Die Hauptelemente sind dabei neben dem Workflow die Elemente Rolle, Artefakt und Aktivität. Diese Hauptelemente werden ebenfalls in anderen Modellen von Software-Engineering-Methoden als Hauptelemente definiert, manchmal unter einem anderen Begriff.

Durch die Bestimmung des Anpassungspunktes im Modell der Software-Engineering-Methode ist es nun besonders wichtig zu bestimmen, welches Element/welche Elemente WIE (Punkt c) an dieser Stelle angepasst werden. In Kapitel 3.1.2. wurden dafür bereits die verschiedenen Anpassungs-Arten näher erläutert. Mit Hilfe der Ziele ist nun von einem Methoden-Engineer zu bestimmen, was getan werden kann, um das Ziel weiter einzuhalten, z.B. ob ein Element ausgetauscht, hinzugefügt oder gelöscht werden muss. In manchen Fällen ist bereits in der Beschreibung einer Software-Engineering-Methode vorhanden, was in bestimmten Fällen durchgeführt werden muss. Hat der Methoden-Engineer bestimmt, was bei der möglichen Anpassung durchgeführt wird, muss er noch überprüfen (Punkt d):

a. Ob die Reihenfolge in der SEM weiter eingehalten wird oder ob diese entsprechend über Verbindungen neu gesetzt werden muss.
b. Ob die Verantwortlichen für die Elemente weiter gesetzt sind oder diese neu gesetzt werden müssen.

Als einen letzten Schritt muss der Methoden-Engineer bestimmen, wer am Ende alles über die Änderungen benachrichtig werden muss. Durch das Modell und die entsprechende Zuordnung der Rollen kann schnell ermittelt

werden, welche Rollen, sowohl alt als auch neu, von der Änderung betroffen sind und somit informiert werden müssen. Wichtig ist dabei zu beachten, dass die Benachrichtigung bei weggefallenen Rollen nicht vergessen wird. Durch die Zuordnung der konkreten Personen zu den Rollen auf der Instanz-Ebene wird in der Ausführung bestimmt, wer die entsprechende Benachrichtigung bekommt, z.B. anhand einer Nachricht in E-Mail-Form.

Sind die verschiedenen Schritte vom Methoden-Engineer durchgeführt, hat er für die Planungsmöglichkeit am Ende bestimmt**,** an **welcher Stelle** im Modell (Wo) bei einer Verletzung der Regel **welche Elemente** (Was), in **welcher Art** (Wie) angepasst werden, **welche Verbindungen** (Welche) erhalten oder neu gesetzt werden müssen und an wen **welche Benachrichtigung** (Wer) geschickt werden muss. Diese fünf Punkte können in Schritt 6 in konkrete Ausführungs- und Benachrichtigungs-Metriken für die Phase Execute überführt werden.

Zur Veranschaulichung wird wieder das Beispiel der Teamgröße in Scrum aufgegriffen. Das Team ist zu groß und beinhaltet mehr als 9 Personen. Der Methoden-Engineer überlegt nun zunächst, wo und was er ändern kann, wenn das Team zu groß wird. In der Beschreibung von Scrum selbst wird dazu geraten, das Team dann aufzusplitten (Team-Split). Das bedeutet, dass mehrere Teams gemeinsam am Product Backlog arbeiten und diese sich in Form eines Scrum of Scrums täglich austauschen müssen.

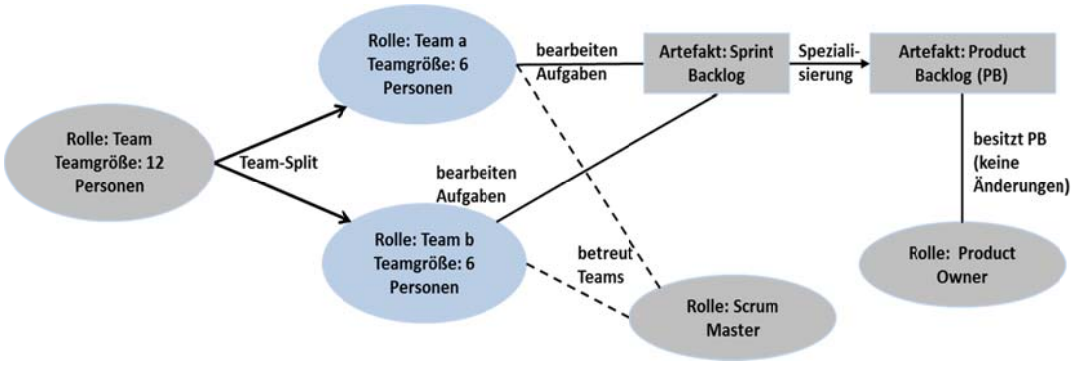

Abbildung 39 Team-Split: Aufteilung in zwei Teams und neue Zuordnung

Im Modell werden nun zum ursprünglichen Team ein oder mehrere Teams (je nach Anzahl der Personen) hinzugefügt, bei einem Beispiel von 12 Personen würden wie in Abbildung 39 zu sehen zwei neue Teams mit jeweils 6 Personen entstehen. Die Aufgaben im Product Backlog werden auf die Teams aufgeteilt. Grau bedeutet dabei, dass sich diese Rolle bzw. dieses Artefakt nicht ändert. Die Personen werden anhand ihrer Qualifikationen, ableitend beispielsweise aus den Aufgaben im Product Backlog, auf die Teams aufgeteilt. Der Product Owner bleibt weiterhin der Besitzer des Pro-

duct Backlogs und für ihn ändert sich nichts. Der Scrum Master an sich ändert sich ebenfalls nicht, aber er ist nun für zwei Teams verantwortlich.

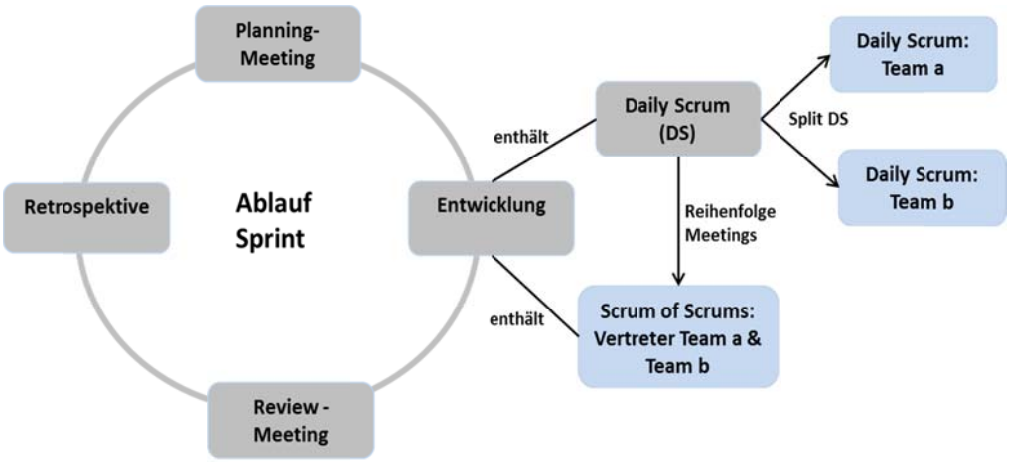

Abbildung 40 Split Daily Scrum und Einfügen Scrum of Scrums nach Daily Scrum

Im Modell wird ein Meeting, also eine Aktivität, hinzugefügt und zwar wie in einem stark vereinfachten Modell in Abbildung 40 zu sehen nach dem Daily Scrum. Außerdem wird das Daily Scrum in zwei aufgeteilt, eines wird durchgeführt von Team a, das andere von Team b. Im Scrum of Scrums treffen sich gewählte Vertreter aus den beiden Teams und besprechen ähnlich wie im Daily Scrum, was die Teams zum einen getan haben und zum anderen als nächstes tun wollen.

Die Verantwortlichkeiten müssen nun bei dem alten Team neu gesetzt werden. Es muss beispielsweise mindestens eine Person am Scrum of Scrums teilnehmen und ist dafür verantwortlich. Beim neuen Team müssen die Verbindungen sowohl zum Product Backlog als auch zum neuen Meeting gesetzt werden. Bei den Benachrichtigungen müssen alle alten und neuen Teammitglieder benachrichtig werden.

5.5.2 Variantenbestimmung

Bei vielen Anpassungen kann es verschiedene Möglichkeiten geben. Gerade wenn ein Toleranzbereich vorhanden ist, gibt es mindestens eine Möglichkeit zur Anpassung, wenn der Bereich überschritten und eine, wenn der Bereich unterschritten wurde. Zusätzlich kann es noch weitere Varianten geben, muss es aber nicht. Die Anzahl der Varianten bestimmt sich also aus dem Toleranzbereich und auch aus dem, was der Methoden-Engineer an Möglichkeiten erkennt. Gerade deshalb ist es wichtig, dass die Herleitung der Planungsmöglichkeiten eine erfahrene oder entsprechend ausgebildete Person übernimmt.

Sind die Varianten nach dem Vorgehen aus 5.5.1. bestimmt, so muss noch bestimmt werden, wann welche Variante eingesetzt wird. Ist die Variante

anhand der Toleranzüber- bzw. -unterschreitung bestimmt, so kann dies anhand der entsprechenden Regel geschehen, wie etwa im Beispiel im vorherigen Kapitel 5.4. Dort ist jeweils eine Variante angegeben, wenn das Team größer ist als der Toleranzbereich und eine zweite Variante, wenn das Team kleiner ist als der Toleranzbereich. Je nachdem, welcher Fall zutrifft, wird die entsprechende Variante ausgeführt.

Sind mehrere Varianten für einen Fall angegeben, muss zur Laufzeit, also zur Ausführung der Plan-Phase bestimmt werden können, welche der Varianten am besten geeignet ist zu dem jeweiligen Zeitpunkt. Dafür muss es eine entsprechende Vergleichsmöglichkeit geben, um die beste Variante zu bestimmen. Dabei sind mit die beiden wichtigsten Faktoren:

- Grad der Anpassung (muss viel oder nur sehr wenig angepasst werden).
- Auswirkung der Anpassung auf der Gesamt-SEM. Das heißt, wie stark beeinflusst die Anpassung die restliche Software-Engineering-Methode, das heißt gibt es Konflikte, muss an einer anderen Stelle ebenfalls etwas angepasst werden etc.

Gerade die Auswirkung einer Anpassung auf die Gesamt-SEM ist sehr wichtig. Durch das Modell ist Rückverfolgung (Traceability) und die Verbindung in der gesamten Methode gegeben. Mit Hilfe von Mustern (Pattern), Traceability-Algorithmen oder Simulationen der Anpassung in der Gesamt-SEM kann überprüft werden, wie sich Anpassung auswirkt und ob es an anderen Stelle Probleme gibt. Zum Beispiel kann an einer anderen Stelle ein anderes Ziel nicht mehr eingehalten werden oder die Software-Engineering-Methode verlangsamt sich etc.

Wurde herausgefunden, dass die erstgewählte Variante ein Problem darstellt und einen Konflikt in der Gesamt-SEM auslöst, muss bestimmt werden, welche weitere Variante genutzt werden kann. Es muss über alle möglichen Varianten überprüft werden, welche sich am besten für die Anpassung eignet und am wenigsten Probleme, wenn überhaupt, hervorruft. Wird festgestellt, dass jede Variante große Konflikte hervorruft, muss unter Umständen eine neue Alternative mit den gegebenen Werten geplant werden, ähnlich einer Planung ohne vorherige Planungsmöglichkeiten

5.5.3 Konfliktpotential

Ein zusätzlicher Punkt, welcher im Vorfeld betrachtet werden kann und welcher die Auswahl der Alternativen unterstützen kann ist, die möglichen Ziele und somit ihre Planungsmöglichkeiten im Vorfeld auf ihr Konfliktpotential hin zu untersuchen. Die zu beantwortende Frage ist hier: Welche Konflikte können bei den jeweiligen Zielen entstehen?.

Dabei muss in einem ersten Teil betrachtet werden, welches Konfliktpotential die Ziele untereinander besitzen können. Dies ist wichtig zu analysieren, wenn in der späteren Durchführung von MAPE-K mehrere Anpassungen gleichzeitig durchgeführt werden müssen. Ähnlich wie bei der Analyse der Varianten in Bezug auf die Gesamt-SEM müssen die Ziele zueinander in Beziehung gesetzt werden. Das heißt, die Ziele müssen dahingehend betrachtet werden, ob sie in Konflikt zueinander stehen, was eher selten der Fall sein sollte.

Zusätzlich müssen die erstellten Planungsmöglichkeiten und ihre Varianten betrachtet werden, ob diese untereinander in Konflikt stehen, sollten sie gleichzeitig zur Anwendung kommen. Eine Möglichkeit der Überprüfung wäre ebenfalls die Anwendung von Simulationen sowie von der Verwendung von Pattern und deren Analyse.

Anhand des Ergebnisses dieser Simulationsalgorithmen und Pattern-Analyse ist es möglich die Konflikte zu identifizieren, aber auch um Lösungsmöglichkeiten für diese Konflikte bereits im Vorfeld zu finden. Ein wichtiger Punkt sind dabei die Prioritäten der Ziele. Stehen beispielsweise zwei Ziele mit unterschiedlichen Prioritäten in Konflikt, so wird das Ziel mit der höheren Priorität zuerst in Betracht gezogen und sticht das andere Ziel unter anderem aus, sofern es keine Lösung für diesen Konflikt gibt. Allerdings sind vorher die Konsequenzen in Bezug auf die Gesamt-SEM zu bewerten.

5.5.4 Kombinationsmöglichkeiten

Für die Planung ist als letzter Punkt zu überlegen, wie Planungsmöglichkeiten und die daraus folgenden Anpassungen miteinander kombiniert werden können. Da nicht jede Anpassung sofort sondern erst zu einem definierten Zeitpunkt ausgeführt werden kann, ist es möglich, dass im MAPE-K mehrere Anpassungen aufgelaufen sind. Da diese zum Anpassungszeitpunkt nicht alle direkt nacheinander in das Modell überführt werden sollen, muss eine Kombination der aufgelaufenen Anpassungen erfolgen, um ein optimales Ergebnis zu erzielen.

Um eine solche kombinierte Anpassung zu erstellen muss überlegt werden, ob es bestimmte Kombinationspunkte innerhalb der Anpassungen gibt und wie diese aussehen. Anzusetzen ist dabei an den beiden Punkten aus Abschnitt 5.5.1 WO angepasst wird, WAS (welche Elemente) angepasst und WIE (Art der Anpassung) angepasst werden kann.

Zu betrachten sind im ersten Schritt gleichartige Elemente, da diese zu einem neuen gleichartigen Element verschmolzen werden können. Sollen beispielsweise zwei Rollen angepasst werden, muss zunächst untersucht werden, ob sie an derselben Stelle angepasst werden. Ist dies der Fall, kann dies

zu einer gemeinsamen Rolle kombiniert werden. Sind die Rollen an verschiedenen Stellen, werden sie in einem Anpassungsschritt angepasst, sofern sie nicht in Konflikt zueinander stehen. In einem zweiten Schritt wird betrachtet, wie das Element angepasst wird, beispielsweise ob die Rolle hinzugefügt oder ausgetauscht wird.

Schwierig wird es, wenn sich die Anpassung nicht ausschließlich auf ein gleichartiges Element bezieht, sondern es sich um genau dasselbe Element an derselben Stelle im Modell handelt, da die Anpassungen sehr wahrscheinlich in Konflikt zueinander stehen. Dann muss abgewogen werden, welche von beiden Anpassungen den größten Nutzen in der Einhaltung der Ziele bringt, z.B. anhand der Prioritäten.

Die letzten drei Abschnitte Variantenbestimmung, Konfliktpotential und Kombinationsmöglichkeiten haben verschiedene Punkte gezeigt, welche für die spätere Planung einer Anpassung wichtig sind. Ebenso haben sie Herausforderungen und erste Ideen zur Lösung gezeigt. In dieser Arbeit soll sich zu diesem Zeitpunkt darauf beschränkt werden, doch diese Punkte bieten Potential für weitere Forschungsarbeiten.

5.6 Schritt 5 – Ableitung von Metriken

Schritt 5: Ableitung von Metriken	
Aufgabe	Aus den Analyseregeln werden entsprechende Metriken für die Messungen abgeleitet
Typ	Ableiten Sub-Typ: Metriken
Subschritte	Aufbereitungsregeln herleiten
Input des Schrittes	Analyseregeln
Output des Schrittes	Metriken für die Messung, Aufbereitungsregeln für Monitor
Mögliche Stakeholder	Methoden-Engineer

Nachdem im dritten Schritt die Regeln für die Analyse hergeleitet wurden, ist es nun möglich, mit dieser Eingabe die Metriken für die Messungen mit Hilfe der Sensoren herzuleiten. Die Metriken werden aus dem Wenn-Teil der Regeln abgeleitet, da dieser Teil als Eingabe für die Auswertung der Software-Engineering-Methode die entsprechenden Messwerte benötigt.

In diesem Schritt wird nun ermittelt, welche Messwerte nötig sind, um die entsprechende Analyseregel auswerten zu können. Dabei ist es möglich, dass bei komplexen Regeln mehrere Messwerte nötig sind, welche zu einem Auswertungswert in der Monitor-Phase zusammengefasst werden. Somit ergeben sich in einem zweiten Schritt die Aufbereitungsregeln für die spätere Monitor-Phase. Zur Verdeutlichung einige Beispiele:

- Beispiel 1: Für das Beispiel aus Schritt 3 wird in der Analyseregel die Teamgröße in Scrum ausgewertet. Gemessen werden dafür die Anzahl der Mitarbeiter in einem Scrum- oder Projektteam. Dies ergibt nun die Metrik: Anzahl Mitarbeiter pro Team, was entsprechend gemessen wird.

  Um aus diesem gemessenen Wert anschließend wieder den aufbereiteten Wert für die Auswertung zu bekommen, wird die Herleitung praktisch rückwärts angewandt. Es entsteht folgende Aufbereitungsregel für die Monitor-Phase: Setze Teamgröße = Anzahl Mitarbeiter pro Team.

- Beispiel 2: In Schritt 3 wurde ebenfalls das Beispiel bezüglich der Nutzung von aktuellen Standards und Normen eingeführt. Es wurde beschrieben, dass es möglich sein muss, über einen Scan oder Datenbankeintrag die aktuelle Norm/den aktuellen Standard messen zu können. Ein solcher Eintrag in der Datenbank wäre damit die Metrik, welche für diese Auswertungsregel gemessen werden muss. Sie würde wie folgt lauten: Datenbankeintrag Normen bzw. Datenbankeintrag Standard.

  In der Aufbereitung würde der entsprechend ausgelesene Wert, in diesem Fall die Versionsnummer der Norm oder des Standards sowie das Datum, auf den Wert „aktuelle Norm gesetzt“.

- Beispiel 3: In einem komplexeren Beispiel soll die Zufriedenheit der Kunden ausgewertet werden. Sobald diese einen bestimmten Grenzwert unter- oder überschreitet, muss die Software-Engineering-Methode überdacht werden. Die Kundenzufriedenheit kann dabei über die Anzahl und Kritikalität der Fehlermeldungen gemessen werden, welche die Metrik ergeben. Die Messwerte an den Sensoren wären damit zum einen die Anzahl Fehlermeldungen und zum anderen die Kritikalität der Fehlermeldung.

  In der Aufbereitung würden diese beiden Werte miteinander kombiniert werden. Nach der Aufbereitung wären dann zum einen die Gesamtanzahl der Fehlermeldungen, z.B. 10, und zum anderen die zusätzliche Aufschlüsselung anhand der Fehlermeldungen bekannt: Gesamtzahl Fehlermeldung = 10, davon 3 Blocker, 5 mittelkritische Fehler und 3 Schönheitsfehler.

5.7 Schritt 6 –Ableitung von Ausführungsregeln und Benachrichtigungen

Schritt 6: Ableitung von Ausführungsregeln und Benachrichtigungen	
Aufgabe	Aus den Planungsmöglichkeiten werden Ausführungsregeln und Benachrichtigungen abgeleitet
Typ	Ableiten Sub-Typ: Ausführungen + Benachrichtigungen
Subschritte	1. Ausführungsregeln ableiten 2. Benachrichtigungen ableiten
Input des Schrittes	Planungsmöglichkeiten + mögliche Anpassungen
Output des Schrittes	Ausführungsregeln + Benachrichtigungen
Mögliche Stakeholder	Methoden-Engineer

Im letzten Schritt der Pre-Work müssen nun noch die Ausführungsregeln und Benachrichtigungen bekannt sein, um am Ende in der Execute-Phase die Software-Engineering-Methode letztendlich im laufenden Projekt anzupassen. Durch die Planungsmöglichkeiten und die verschiedenen Varianten aus dem vorherigen Schritt ist nun bekannt, WAS an den jeweiligen Stellen im Modell zum entsprechenden Ziel angepasst werden muss, z.B. ob eine Rolle ausgetauscht, ein Artefakt hinzugefügt oder gelöscht werden muss usw. Durch die Verantwortlichen und Beteiligten, welche den jeweiligen Aktivitäten etc. zugeordnet sind kann sofort ermittelt werden, wer alles benachrichtig werden muss.

Die Ausführungsregeln geben im Detail an, wie die Anpassung der Software-Engineering-Methode erfolgen soll. Die Anpassung erfolgt in der Ausführungsphase in verschiedenen Schritten. Zunächst wird das (allgemeine) Modell der SEM, welches in der Wissensbasis liegt, auf Typ-Ebene angepasst. Dort werden entsprechend wie im vorherigen Abschnitt beschrieben, Aktivitäten etc. hinzugefügt, gelöscht ausgetauscht usw. Zusätzlich zu den geänderten Aktivitäten usw. müssen, falls nötig, die entsprechenden Verantwortlichen und/oder die Ausführenden der Aktivität geändert werden. Diese Änderung ergeben am Ende die Personen, welche benachrichtigt werden müssen, egal ob sie eine neue Aktivität durchführen müssen oder ihre alte gestrichen wurde. In einigen Fällen kann es sogar sein, dass eine Nachricht ans System geschickt werden muss, wenn sich eine automatische Aktivität etc. geändert hat.

Die Regel selbst hat am Ende folgende Form:

1. Anpassung Modell (Aktion x Element m (Name)), Aktion y Element n (Name), …); wobei Aktion = löschen, hinzufügen, austauschen etc. ist und Element = Artefakt (Name Artefakt), Aktivität (Name Aktivität), Rolle (Name Rolle) usw. ist
2. Änderung Stakeholder (Stakeholder x, Element m (Name), Aktion t); wobei Stakeholder die entsprechende Rolle und somit Person im Team angibt, Element wie im vorherigen Schritt das zugehörige Element (Aktivität, Artefakt) angibt und die Aktion definiert, ob die Person gelöscht oder hinzugefügt wird
3. Benachrichtigung Beteiligte (Beteiligter x, Änderung m); wobei der Beteiligte die konkrete Person auf Instanz-Ebene ist, an welche die Nachricht geht, mit der Änderung m, welche neue Aufgabe sie nun durchführen muss.

Wenn wir dies wieder auf unser einfaches Beispiel mit der Teamgröße beziehen, würde bei einem zu großen Team die Planungsmöglichkeit besagen, dass ein Team-Splitting erfolgen soll und ein Scrum of Scrum durchgeführt werden muss. Für die Ausführungsregel besagt dies, dass es zwei neue Rollen und somit zwei Teams gibt, welche dem Artefakt Product Backlog zugeordnet werden. Bei einer neuen Teamgröße von beispielsweise 10 Personen würde dies zwei Teams mit jeweils 5 Personen ergeben. Die Teams werden nach Skills aufgeteilt und die Aufgaben im Product Backlog entsprechend zugeordnet.

Zusätzlich gibt es die neue Aktivität Scrum of Scrums, welche nach dem sogenannten Daily Scrum mit eingefügt wird, die Dauer von 15 Minuten hat und als Verantwortliche dieser Aktivität jeweils der Scrum Master und 1-2 Mitglieder der beiden Teams zugeordnet werden. Die jeweiligen Teammitglieder werden über die Aktivität und die neue Zuordnung informiert. Die Daten werden vorher entsprechend in das Modell in der Wissensbasis geschrieben. Die Ausführungsregel würde dann ungefähr so aussehen:

1. Anpassung Model (Hinzufügen Team 2, Ändern Team 1 (Anzahl Personen), Hinzufügen Meeting (Scrum of Scrums, nach Daily Scrum), Ändern Product Backlog (neue Zuordnung Aufgaben)
2. Änderung Stakeholder (Teammitglied 1, Rolle (Team 1), hinzufügen (für alle Teammitglieder durchführen); Scrum Master, Meeting (Scrum of Scrums), Hinzufügen, Teammitglied 1-2 (Team 1), Meeting (Scrum of Scrums), Hinzufügen; usw.
3. Benachrichtigung Beteiligte (Scrum Master, neue Aktivität (Scrum of Scrums), Teammitglieder 1 – 10, (neue Aktivität, neue Teamzuordnung, neue Aufgaben Product Backlog)

Aufgrund dieser Regeln ist es dann in der Execute-Phase möglich, mit Hilfe der Effektoren die Anpassung in das laufende Projekt zu übertragen.

5.8 Möglichkeiten zur Wiederverwendung der Pre-Work – Ableitungsblock

Die Pre-Work ist zu Beginn relativ aufwendig und benötigt zum einen mehrere Personen, insbesondere zur Bestimmung und Priorisierung der Ziele. Zum anderen braucht es einen erfahrenen Methoden-Engineer oder zumindest eine Person mit ausreichend Erfahrung, um aus den priorisierten Zielen die Regeln, Metriken, besonders aber auch die Planungsmöglichkeiten usw. abzuleiten. Auch wenn die Arbeit einiges an Zeit in Anspruch nehmen kann, bevor das Projekt beginnt, so wird bei der Durchführung des MAPE-K im Projekt selbst entsprechend Zeit wieder gespart. Gerade dann ist im Projekt die Zeit ein kritischer Faktor und es muss möglichst schnell sowie selbstadaptiv gehandelt werden. Dies ermöglichen die durchgeführten Schritte in der Pre-Work.

Eine weitere Frage ist aber, wie die erfolgte Arbeit für spätere Projekte genutzt werden kann. Es wäre sehr nützlich, wenn die Schritte aus der Pre-Work in neuen, vielleicht ähnlichen Projekten, nicht vollständig wiederholt werden müssen, sondern erfolgte Schritte zum Beispiel in der Wissensbasis gespeichert und, mit eventuellen Anpassungen, wiederverwendet werden können.

Die Idee ist hier, einen Block zu entwickeln, welcher in der Wissensbasis für zukünftige Projekte gespeichert werden kann und zu einem beschriebenen Ziel die entsprechenden Regeln, Planungsmöglichkeiten, Metriken sowie Ausführungsregeln enthält. Zusätzlich muss es möglich sein, das Ziel und somit den Block priorisieren zu können, denn die Prioritäten für ein Ziel sind nicht in jedem Projekt gleich. Ferner muss der Block Konfigurationspunkte im Ziel-Teil enthalten, um beispielsweise die Toleranzbereiche anpassen zu können.

Die Herleitung bzw. der Aufbau eines solchen „Wiederverwendungsblockes“ könnte wie in Abbildung 41 beschrieben aussehen. Dabei wird wieder das Beispiel von vorher bezüglich der Teamgröße in Scrum genutzt. Die Software-Engineering-Methode ist im ersten Schritt mit ihren Regeln, Verhaltensbeschreibungen, Workflow usw. gegeben, am besten aufbereitet in der Form „Beschreibung A, Beschreibung B, Beschreibung C“ usw.

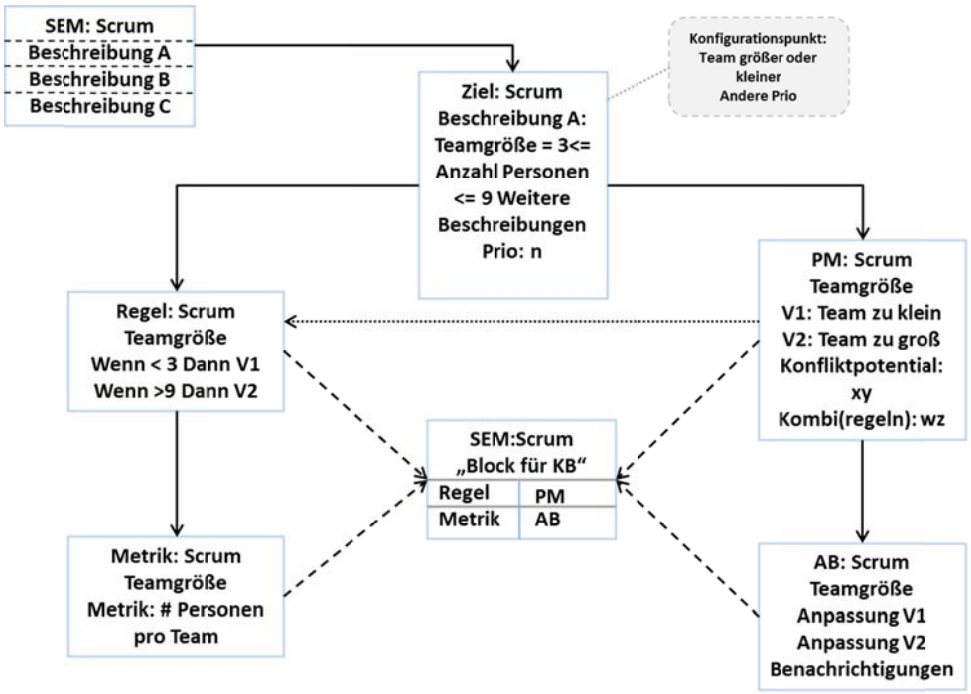

Abbildung 41Ableitungen der einzelnen Punkte und Zusammenfassung zu einem Block für die Wissensbasis

Aus diesen Beschreibungen lassen sich einfach die Ziele ableiten. Die jeweilige Beschreibung gibt in aufbereiteter (Kurz-) Form das entsprechende Ziel an, mit dem Zusatz, welche Software-Engineering-Methode genutzt wird. Im Beispiel würde dies für das Ziel lauten:

- Ziel genutzte SEM: Scrum (Original)
- Beschreibung A: 3 <= Teamgröße <= 9
- Priorisierung: n
- Weitere Beschreibungen können, falls nötig, im Block enthalten sein

Dies wäre der erste Eintrag im Wiederverwendungsblock. Die Priorität stellt dabei einer der möglichen Konfigurationspunkte dar. Ein weiterer Konfigurationspunkt wäre in diesem Beispiel die Teamgröße. Sie könnte auf maximal 10 Personen oder auch das Minimum könnte auf 4 Personen erhöht werden.

Die daraus resultierenden Planungsmöglichkeiten auf der rechten Seite enthalten im Block unter dem Abschnitt PM, wie oben in Schritt 4 beschrieben, die verschiedenen Varianten zur Anpassung für dieses Ziel. Zusätzlich sind dort das mögliche Konfliktpotential beschrieben, sowie Kombinationspunkte, wo andere Bausteine beispielsweise ansetzen können, um sie miteinander zu verknüpfen oder sogar zu vermischen.

Aus den Varianten können für den Block wie in Schritt 6 beschrieben die entsprechenden Ausführungsgsregeln und Benachrichtigungen, welche bei diesem Ziel nötig wären, im Block unter dem Abschnitt AB gespeichert werden.

Ähnlich lassen sich die Abschnitte Regel und Metrik im Block von der linken Seite herleiten. Unter Regel sind die hergeleiteten Regeln beschrieben, welche sich aus den Toleranzbereichen und Varianten wie unter Schritt 3 beschrieben zusammensetzen. Die daraus resultierenden Metriken werden in einem weiteren Abschnitt festgehalten.

Wird nun ein weiteres Projekt aufgesetzt, welches ebenfalls die Software-Engineering-Methode Scrum nutzen soll, kann der Block „Scrum, Teamgröße“ wiederverwendet werden und die Pre-Work muss nicht neu hergeleitet werden. Über die möglichen Konfigurationspunkte können die Unterabschnitte automatisch angepasst werden. Wird beispielsweise der Toleranzbereich um eine Person erhöht, werden die Regeln links entsprechend angepasst (Wenn <4 dann V1, Wenn >10 dann V2). Die restlichen Abschnitte bleiben in diesem Fall gleich und müssen nicht mehr angepasst werden.

In diesem Kapitel wurde die Pre-Work im Detail beschrieben, welche die ersten sechs Schritte des Ablaufes aus Kapitel 4 enthält. Dabei wurde zusätzlich erläutert, wie sich mit Hilfe einer GQM-Variante die Schritte 3-6 aus den priorisierten Zielen herleiten lassen. Abgeschlossen wurde das Kapitel mit der Idee für die Erstellung eines Blockes pro Ziel zur Wiederverwendung für weitere Projekte. Diese Blöcke lassen sich mit Konfigurationspunkten für die Bedürfnisse eines neuen Projektes anpassen.

Im nächsten Kapitel werden die Schritte 7-10 im Detail beschrieben, wie die Schritte im MAPE-K aufgebaut sind und ablaufen.

Kapitel 6 MAPE-K

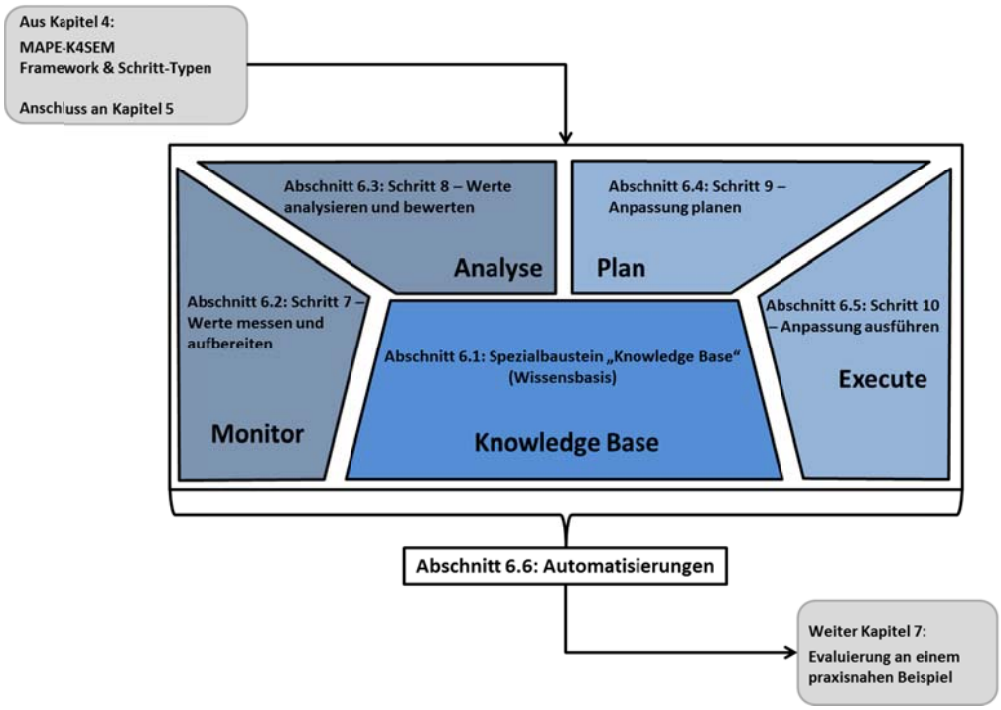

Abbildung 42 Aufbau von Kapitel 6

Nachdem in Kapitel 5 die Pre-Work im Detail vorgestellt wurde, werden in diesem Kapitel die folgenden Schritte 7-10, die Phasen und Durchführung des MAPE-K im Detail erläutert. Das Kapitel beginnt mit der Wissensbasis – der Knowledge Base – im MAPE-K. Dies ist ein essentieller Baustein, welcher alle wichtigen Elemente zur Durchführung der Feedbackschleife enthält. Nach der Beschreibung der Knowledge Base werden die weiteren Schritte und somit die einzelnen Phasen im Detail erläutert, beginnend mit Schritt 7, dem Messen der Werte und deren Aufbereitung, was sowohl die Sensoren als auch die Monitor-Phase widerspiegelt. Darauf aufbauend werden in Schritt 8 in der Analyse-Phase die Werte analysiert und bewertet, ob eine Software-Engineering-Methode entsprechend angepasst werden muss. Daran knüpft die Beschreibung des Schrittes 9 an, mit der Planung der Anpassung und Beschreibung des Anpassungszeitpunktes. Im nächsten Abschnitt werden der letzte Schritt und die Phase Execute beschrieben, in welcher die Anpassung ausgeführt wird. Dieses Kapitel schließt mit der Beschreibung von Möglichkeiten zur Automatisierung des vorgestellten Ansatzes.

6.1 Der Spezial-Baustein im SE Method Manager – die Knowledge Base

Die Wissensbasis, englisch Knowledge Base (KB) selbst gehört nicht zur eigentlichen Feedbackschleife des MAPE-K und ist kein eigener Schritt im MAPE-K4SEM. Dennoch ist dieser Baustein essentiell für das Gelingen des MAPE-K und ebenfalls für das Gelingen des hier entwickelten Ansatzes. Denn in der Wissensbasis werden alle nötigen Daten gespeichert, sowohl aus der Pre-Work als auch während der Ausführung. Während der Durchführung der einzelnen Phasen bzw. der Schritte 7-10, interagiert jeder einzelne Schritt mit der Knowledge Base, indem er Daten von dort abruft oder wieder dort abspeichert.

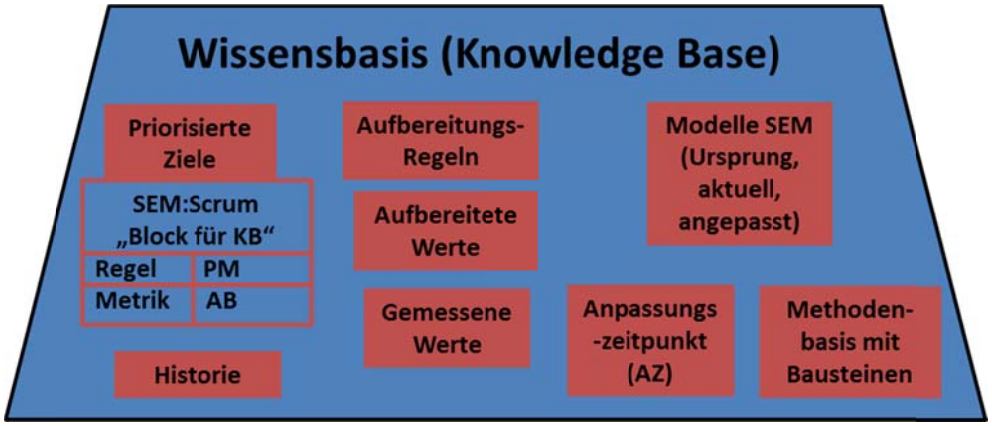

Abbildung 43 Inhalte der Knowledge Base

Die nötigen Inhalte, die für den SE Method Manager und den Ablauf der zehn Schritte gespeichert sind, sind in Abbildung 43 dargestellt. Dabei handelt sich um:

- Die priorisierten Ziele aus Schritt 1 & Schritt 2
- Die abgeleiteten Werte aus den Schritten 3-6
  - Regeln für die Analyse
  - Abgeleitete Metriken für die Sensoren und die Monitor-Phase
  - Die Planungsmöglichkeiten
  - Die Ausführungs- und Benachrichtigungsregeln
- Die gemessenen Werte
- Aufbereitungsregeln zur Aufbereitung der gemessenen Werte
- Die aufbereiteten Werte
- Der Anpassungszeitpunkt (AZ)

- Das Modell der Software-Engineering-Methode, sowohl in seiner Ursprungsform als auch das aktuelle instanziierte Modell und das angepasste Modell
- Eine Methoden-Basis mit entsprechenden Methoden-Bausteinen
- Eine Historie über die Werte und dabei vor allem über die Anpassung

Einzelne Elemente werden in verschiedenen Abschnitten des MAPE-K4SEM erarbeitet, gespeichert und für die weitere Verwendung wieder abgerufen. Es ist allerdings die Voraussetzung, dass das Ursprungsmodell der Software-Engineering-Methode bereits festgelegt wurde. Dabei ist es egal, ob es sich um eine klassische SEM, eine agile SEM, eine situationsspezifische SEM oder eine SEM, die für das Projekt zugeschnitten wurde, handelt.

Wichtig ist, dass die Software-Engineering-Methode als ein Modell vorliegt. Die Modellierung kann beispielsweise anhand der in Abschnitt 2.1.2 vorgestellten Techniken wie SPEM, ISO 24744, MeteME oder einer vereinfachten Form dieser Techniken erfolgen. Das Modell der Software-Engineering-Methode ist für die Anpassung wichtig, damit klar ist, an welchen Stellen im Modell die Anpassung der einzelnen Elemente, wie hinzufügen oder löschen, erfolgen kann. Außerdem sind anhand des modellierten Workflows die Abhängigkeiten zwischen den einzelnen Elementen ersichtlich.

Anhand der aktuellen Instanz der Software-Engineering-Methode ist klar, in welcher Ausprägung sich das Projekt aktuell befindet. Dies ist ebenfalls wichtig für die aktuelle Messung und Auswertung der einzelnen Messwerte. Die Auswertung der Messwerte bezüglich einer Anpassung kann beispielsweise an einem bestimmten Punkt (Planning-Meeting in Scrum oder Design-Phase in RUP) in der Software-Engineering-Methode anders ausfallen als zu einem früheren oder späteren Punkt (Review-Meeting in Scrum oder Test-Phase in RUP).

Abschließend ist das angepasste Modell der Software-Engineering-Methode wichtig, welche nach der Anpassung in das Projekt übertragen wird. Durch die Übertragung in das Projekt wird die angepasste Software-Engineering-Methode zu der aktuellen Instanz der SEM im Projekt. Diese Transformation erfolgt in der Execute-Phase.

Neben den Modellen und insbesondere dem Ursprungsmodell ist zu Beginn die Methoden-Basis gegeben. In dieser werden alle Elemente (Bausteine) gespeichert, die bei einer Anpassung und hier besonders in der Planung benötigt werden können. Das können Elemente von einzelnen Artefakten über Aktivitäten, Rollen bis hin zu (Teil-)Workflows sein.

Dabei ist es nötig zu wissen, welchen Typ sie besitzen, zu welchem Punkt sie in einer Software-Engineering-Methode eingesetzt werden können oder was ihre Aufgabe ist. Ebenso muss bekannt sein, ob Verantwortlichkeiten benötigt werden, ob diese Elemente einen Input erfordern oder ob sie einen Output für andere Elemente erzeugen. Diese Punkte müssen in den Elementen gespeichert sein. Für eine solche gespeicherte Übersicht kann beispielsweise eine angepasste Form des in Abschnitt 4.3.2 vorgestellten Frameworks genutzt werden. Die Methoden-Basis ist dabei nicht in sich abgeschlossen, sondern kann kontinuierlich erweitert und in anderen Projekten verwendet werden.

Neben diesen Inhalten der Wissensbasis, welche teils zu Beginn vorhanden sein müssen, werden in ihr ebenfalls die erarbeiteten Ergebnisse aus den ersten 6 Schritten abgelegt. Dies umfasst zunächst einmal die Ziele und Eigenschaften der Software-Engineering-Methode, welche in Schritt 1 definiert und Schritt 2 entsprechend priorisiert wurden.

Wichtig für die nächsten 4 Schritte sind die aus den Zielen abgeleiteten Analyseregeln, die daraus abgeleiteten Metriken, die Planungsmöglichkeiten sowie die Ausführungsregeln und Benachrichtigungen. Des Weiteren sind die in unter Schritt 5 erarbeiteten Varianten, Konfliktpotentiale bezüglich der einzelnen Ziele sowie deren Kombinationspunkte bekannt.

Dabei werden die Metriken in Schritt 7 verwendet. Es ist mit ihrer Hilfe möglich, über die Sensoren die erforderlichen Daten entsprechend zu messen. Die Analyseregeln in Schritt 8 für die Analyse und Auswertung der gemessenen Werte benötigt. Die Planungsmöglichkeiten sind ebenso wie die Konfliktpotentiale und Kombinationsmöglichkeiten für Schritt 9 wichtig. Abschließend werden die Benachrichtigungs- und Ausführungsregeln in Schritt 10 für die Durchführung der Anpassung gebraucht.

Für die Analysephase sind die aufbereiteten Werte wichtig, denn diese werden für die Analyseregeln verwendet. Die von den Sensoren und mit Hilfe der Metriken gemessenen Werte werden entsprechend ihrer Aufbereitungsregeln für die weitere Verwendung aufbereitet. Die Aufbereitungsregeln werden neben den Metriken in Schritt 5 hergeleitet und in der Wissensbasis abgespeichert.

Des Weiteren ist in der Wissensbasis der Anpassungszeitpunkt festgelegt und abgespeichert. Dieser ist nötig für die spätere Ausführung der Anpassung, wann diese im Endeffekt durchgeführt wird. Der Anpassungszeitpunkt und seine Bedeutung werden in Abschnitt 6.4 noch näher erläutert.

Abschließend wird in der Wissensbasis eine Historie der verschiedenen Anpassungen gespeichert. Diese ist wichtig für die Nachverfolgung der einzel-

nen Anpassungen, um zu sehen, wie wirksam die einzelnen Anpassungen waren oder ob eine Anpassung wieder rückgängig gemacht wurde. Ferner kann diese Historie interessant sein, wenn die Erfahrung aus dem gemachten Projekt für die Wiederverwendung übertragen werden soll. Anhand der Historie können die Anpassungen, welche in bestimmten Situationen durchgeführt wurden, erfasst und als Erfahrung für ein kommendes Projekt mit ähnlichem Kontext und ähnlicher Software-Engineering-Methode verwendet werden. So können beispielsweise mögliche Situationen vorhergesehen und nötige Anpassungen eventuell vermieden werden.

Die Wissensbasis kann als eine Datenbank umgesetzt werden. Wichtig ist dabei zu beachten, welcher Schritt oder welche Phase während des MAPE welche Schreib- und Leserechte besitzt. Die ersten sechs Schritte haben dabei Leserechte für das Element „Modell“, da dieses für die Herleitung der einzelnen Schritte, insbesondere der Anfangsschritte benötigt wird. Zusätzlich haben Schritt 1 und 2 Schreibrechte für das Element „priorisierte Ziele“, sowie Schritt 3-6 Schreibrechte für die Elemente die jeweils zu ihrem Schritt korrespondieren.

Die Sensoren aus Schritt 6 bekommen Schreibrechte für das Element „gemessene Werte“, da die Sensoren die Werte dort entsprechend ihrer Messung hinterlegen. Ebenso bekommt anschließend die Monitor-Phase Leserechte auf dieses Element, auf das Element „Aufbereitungsregeln“, sowie Schreibrechte für das Element „aufbereitete Werte“, wo das Ergebnis der Aufbereitung abgespeichert wird. Schritt 8 hat Leserechte bezüglich der Analyseregeln.

Schritt 9 hat für die Planung Leserechte auf das Element der Planungsmöglichkeiten, sowie auf den Anpassungszeitpunkt. Die Schreibrechte für das angepasste Modell der Software-Engineering-Methode erhält erst Schritt 10, da in der Execute-Phase das Modell in die Wissensbasis geschrieben und die Anpassung über die Effektoren in das Projekt zurückübertragen wird. Dafür hat Schritt 10 zusätzlich Leserechte auf die Ausführungsregeln und Benachrichtigungen.

Wie die genaue Durchführung der einzelnen Schritte der MAPE-Feedbackschleife mit Hilfe dieser Werte und Elemente der Wissensbasis aussieht, wird in den folgenden Abschnitten beschrieben.

6.2 Schritt 7 – Werte messen und aufbereiten

Schritt 7: Werte anhand der Metriken messen und aufbereiten	
Aufgabe	Die Werte/ Metriken der SEM werden gemessen und für die Speicherung in der Knowledge Base und die anschließende Analyse aufbereitet
Typ	Messen + Erstellen
Subschritte	1. Werte messen 2. Werte aufbereiten
Input des Schrittes	Metriken, gemessene Werte durch die Sensoren
Output des Schrittes	Aufbereitete Messwerte
Mögliche Stakeholder	System (Monitor-Phase, Wissensbasis)

Nachdem die Pre-Work abgeschlossen und die Wissensbasis mit den verschiedenen Inhalten (Elemente aus der Pre-Work, Modell der SEM, Aufbereitungsregeln usw.) gefüllt wurde, kann die eigentliche Durchführung des Projektes und somit des MAPE-K beginnen. Die Managed SE-Methode, welche nun durch den MAPE-K überprüft wird, ist durch die aktuelle Instanz der Software-Engineering-Methode bereits festgelegt. Die entsprechenden Inhalte, wie Rollen und die konkreten Personen, die ersten auszuführenden Aktivitäten, die zu erstellenden Dokumente etc. sind auf die Ausgangswerte gesetzt. Die Durchführung der Feedbackschleife beginnt mit Schritt 7 zum Start des Projektes.

6.2.1 Messen anhand von Sensoren

Bevor die eigentliche Monitor-Phase startet, müssen zunächst die zu beobachtenden Werte mit Hilfe der Sensoren gemessen werden. Die Sensoren ergeben sich aus den in Schritt 5 abgeleiteten Metriken. Die Metriken müssen dafür entsprechend in Sensoren umgewandelt werden. Das heißt, es gilt im Vorfeld festzulegen, wie und in welcher Form die Metrik gemessen wird. Dafür kann es möglicherweise mehrere Arten geben, die bei gleichem Ergebnis einen unterschiedlichen Aufwand erzeugen.

Die Sensoren überprüfen in den meisten Fällen Einträge in verschiedenen Artefakten, sowohl im eigentlichen Projekt als möglicherweise auch im gesamten Unternehmen oder der Umgebung. Zu der Umgebung können je nach Metrik auch spezielle Einträger in einer Internetdatenbank oder auf einer Webseite sein. Des Weiteren kann es spezielle Umgebungssensoren geben, die beispielsweise messen, ob es zusätzliche Anweisungen von der

Unternehmensleitung, dem Projektleiter oder ähnliches gibt. Diese Anweisungen werden in eine Liste bzw. Datenbank geschrieben oder aus E-Mails mit einem entsprechenden Schlagwort in der Betreffzeile ausgelesen.

Zu Beginn des Projektes sind alle Sensoren mit den Initialwerten versehen. Sobald eine Änderung an den Werten vorgenommen wird, registriert der Sensor dies, er misst den entsprechenden Wert und gibt diesen an die Monitor-Phase weiter. Für ein kurzes Beispiel zur Verdeutlichung wird wieder die Teamgröße aus Scrum verwendet.

Die Metrik, die sich für die Teamgröße in Schritt 5 ergab, war die Anzahl der Personen pro Team. Die Teammitglieder sind in diesem Beispiel in einer Liste eingetragen. Dies kann einerseits eine Liste mit den Namen der Personen sein, welche in einem Team sind. Oder es gibt im Unternehmen eine vollständige Personenliste, in der es für jede Person einen Eintrag gibt, welchem Team sie zugeteilt ist.

Der Sensor für Fall 1 überprüft die Teamliste und registriert eine Änderung in der Liste. Der Sensor misst die Zahl der Personen, indem er in der Liste die Personen zählt und das Ergebnis an die Monitor-Phase weitergibt. Im zweiten Fall überprüft der Sensor die Spalte Team in der gesamten Personenliste des Unternehmens und registriert, wenn sich an dieser Spalte etwas ändert. Dann zählt er die Personen für das Projektteam und gibt diese wiederum an die Monitor-Phase zur Aufbereitung weiter. Hier muss in der gesamten Personenliste jeweils überprüft werden, ob eine Person zum aktuellen Projektteam gehört oder nicht. Dadurch wäre der zweite Fall aufwendiger.

In einem anderen Beispiel besagt eine Metrik, dass die aktuelle Norm oder ein aktueller Standard beispielsweise für das Testen benutzt werden soll. Dafür ist in einem Wert festgelegt, welche aktuelle Norm im Projekt genutzt wird (aktuelleNorm_Projekt). Der Sensor überprüft zum Beispiel auf der Webseite der Norm täglich, welche Norm aktuell ist. Dies kann beispielsweise über das Auslesen der Versionsnummer erfolgen. Die Nummer wird gemessen und in der Wissensbasis nach der Aufbereitung für die Analyse abgespeichert.

Sind die Sensoren mit Hilfe der Metriken alle festgelegt, kann das Messen, Beobachten und Aufbereiten beginnen.

6.2.2 Die Monitor-Phase – Werte aufbereiten

Nachdem die Sensoren die verschiedenen Werte gemessen haben, müssen diese für die weitere Analyse aufbereitet werden. Dies erfolgt mit der Hilfe von in Schritt 5 hergeleiteten Aufbereitungsregeln. Diese Regeln enthalten einen oder mehrere gemessene Werte, welche für den Analysewert zusammengefasst werden. Im einfachsten Fall ist der Messwert gleich dem Analy-

sewert. Ist der Fall komplexer, müssen die Werte entsprechend miteinander kombiniert werden. Die aufbereiteten Werte werden anschließend in die Wissensbasis geschrieben. Bevor der Wert in die Wissensbasis geschrieben wird, wird er mit dem aktuellen Wert in der Wissensbasis verglichen. Ist der Wert gleich, wird der Wert nicht neu gespeichert. Sind der aktuelle und der neue Wert nicht gleich, wird der neue Wert in die Wissensbasis geschrieben.

Für eine Verdeutlichung werden die Beispiele aus Schritt 5 erneut aufgegriffen. Im Beispiel der Teamgröße ist die Aufbereitung des Wertes relativ einfach. Die Teamgröße wird gleich dem Messwert, also die Anzahl der Personen, gesetzt. Ist der Wert anders als der vorherige Wert, so wird der neue Wert in die Wissensbasis geschrieben und die Analyse-Phase wird gestartet, da sich das Team vergrößert oder verkleinert hat.

In Schritt 5 wurde ebenfalls das komplexere Beispiel vorgestellt, die Kundenzufriedenheit zu beobachten. Fällt die Kundenzufriedenheit unter einen bestimmten Schwellenwert, das heißt der Kunde ist unzufrieden, muss etwas geändert werden. Die Metriken für die Kundenzufriedenheit waren zum einen die Anzahl der Fehlermeldungen, die der Kunde in eine Liste einstellt oder die von Mitarbeitern eingestellt werden mit dem Vermerk „Fehler Kunde“ sowie die Kritikalität dieser Fehlermeldungen. Die Kritikalität steht als Eintrag in den Fehlermeldungen, es muss dafür nur jeweils dieses Feld ausgelesen werden.

Um die Kundenzufriedenheit auswerten zu können, soll die Gesamtzahl der Fehlermeldungen der Kunden und die Aufschlüsselung anhand der Kritikalität bekannt sein. Dafür werden, sobald eine neue Fehlermeldung eingestellt wurde, alle Kunden-Fehlermeldungen aufaddiert. Zusätzlich wird die jeweilige Kritikalität ausgelesen und entsprechend der jeweiligen Kritikalität aufaddiert. Diese Werte werden in der Aufbereitung miteinander kombiniert, mit dem Eintrag in der Wissensbasis verglichen und gegebenenfalls wird der Wert neu in die Wissensbasis geschrieben. Ein solcher aufbereiteter Eintrag könnte bei der gemessenen Anzahl von 10 Fehlermeldungen folgendermaßen aussehen: Gesamtzahl Fehlermeldung=10, davon Blocker=3, mittelkritische Fehler=5, Schönheitsfehler=3. In der Analyse-Phase wird anschließend überprüft, ob etwas geändert werden muss oder nicht.

Durch die Kombination aus „Messung mit Hilfe der Sensoren“ und „Aufbereitung dieser Werte für die Analyse-Phase“ wird der aktuelle Status der Software-Engineering-Methode auf der Instanz-Ebene kontinuierlich erfasst und überwacht. Das Messen und Aufbereiten der Werte wird dabei vollständig vom System übernommen.

6.3 Schritt 8 – Werte analysieren und bewerten

Schritt 8: Werte analysieren und bewerten	
Aufgabe	Die gemessenen Werte werden analysiert; als Ergebnis kommt heraus, ob die SEM angepasst werden muss oder nicht
Typ	Analysieren
Subschritte	1. Werte analysieren => Auswertung des Wenn-Teils 2. Anpassung ja/ nein => Ausführung des Dann-Teils
Input des Schrittes	Aufbereitete Messwerte; Analyseregeln
Output des Schrittes	Anpassung ja/ nein + Aktion; evtl. Analysedokument
Mögliche Stakeholder	System (Analyse-Phase), evtl. Methoden-Engineer und/ oder Projektleiter zur Freigabe der Anpassung

In Schritt 7 wurden die gemessenen Werte gemäß ihrer Aufbereitungsregeln aufbereitet und in die Wissensbasis geschrieben. Damit sind die Werte nutzbar für die Analyse, welche durch das Schreiben in die Wissensbasis angetriggert wurde. Die Auswertung der Werte erfolgt mit Hilfe der in Schritt 3 abgeleiteten Analyseregeln und am Ende wird entschieden, ob eine Anpassung erfolgt oder nicht.

6.3.1 Die Analyse-Phase

Von der Monitor-Phase bekommt die Analyse-Phase über einen Trigger Bescheid, dass sich ein bestimmter Wert geändert hat und aufbereitet wurde. Dieser Wert wird entsprechend aus der Wissensbasis ausgelesen und in der Analyseregel verwendet. Die Analyseregel wurde wie in Schritt 3 beschrieben hergeleitet und ist ebenfalls in der Wissensbasis gespeichert.

Die Regel bestand dabei aus zwei Teilen, einem Wenn-Teil und einem Dann-Teil. Für die Auswertung ist vor allem der Wenn-Teil wichtig, wohingegen der Dann-Teil angibt, welche Aktion als nächstes ausgeführt werden soll. Für die Analyseregeln gibt es wie in Abschnitt 5.4 beschrieben die drei folgenden Formen:

1. Wenn „Angabe/ Ziel nicht erfüllt“ dann „Planungsmöglichkeit mit Werten aus Monitor“

2. Wenn „Angabe/ Ziel nicht erfüllt“ dann „Neue Planung mit Werten aus Monitor + Ziel“

3. Wenn „Angabe/ Ziel erfüllt“ dann Schritt 7 (Monitor-Phase)

Durch die Regel ist bekannt, in welchen Grenzen der aufbereitete Wert liegen muss. Ist er innerhalb dieser Grenze, ist das Ergebnis der Bewertung, dass die Software-Engineering-Methode nicht angepasst werden muss. Die MAPE-K-Feedbackschleife wird an dieser Stelle für den gemessenen Wert und das entsprechende Ziel abgebrochen und beginnt wieder bei Schritt 7, den Sensoren und der Monitor-Phase, da das Ziel weiterhin erfüllt wird.

Liegt der Wert nicht innerhalb der Grenzen oder unter- bzw. überschreitet er einen bestimmten Schwellenwert, so ist das Ergebnis, dass die Software-Engineering-Methode angepasst werden muss. Dabei können zwei Fälle auftreten: Zum einen sind bereits Planungsmöglichkeiten und die entsprechenden Varianten bekannt und die Plan-Phase kann mit den aufbereiteten Werten gestartet werden. Zum anderen kann es aber in einigen Fällen durchaus vorkommen, dass Planungsmöglichkeiten noch nicht vorhanden sind. Da bekannt ist, dass mit den aktuellen Werten mindestens ein Ziel nicht erfüllt werden kann, wird die Plan-Phase gestartet, mit den Werten aus der Monitor-Phase und dem Ziel, welches es zu erfüllen gilt. Je nach Ziel können zusätzliche Informationen benötigt werden. Diese werden für die Plan-Phase entsprechend ausgelesen und gespeichert.

Zur Verdeutlichung der verschiedenen Fälle werden hier noch einmal die Beispiele aus Abschnitt 5.4 herangezogen und erläutert. Für den Fall der Teamgröße in Scrum gibt es zwei Möglichkeiten zur Auswertung. Zum einen, ob das Team zu groß ist und gesplittet werden muss. Zum anderen, ob das Team zu klein ist und eine oder mehrere Personen zum Team hinzugefügt werden muss/müssen. Bewegt sich das Team innerhalb der gesetzten Grenzen, erfolgt keine Anpassung.

Bekommt die Analyse-Phase nun einen neuen Wert für die Teamgröße geliefert, wird zunächst überprüft, ob die Größe innerhalb der Grenzwerte liegt, hier im Beispiel sind dafür die Grenzen 3 und 9 nach der ursprünglichen Scrum-Beschreibung gewählt. Liegt der Wert innerhalb dieser Grenzen, wird zurück zu Schritt 7 gegangen. Liegt der Wert außerhalb der Grenzen, das heißt die Regel liefert ein „false“ zurück, dann wird im nächsten Schritt überprüft, ob das Team zu groß ist. Liefert die Regel „true“, wird die Plan-Phase mit der entsprechenden Planungsmöglichkeit gestartet.

Liefert die Regel wiederum false, so ist das Team zu klein und es kann die Planung mit der zweiten Variante für ein zu kleines Team beginnen. Die

ursprünglichen 3 Regeln aus Schritt 3 lassen sich somit folgendermaßen für die Analyse zusammenfassen:

Wenn Teamgröße 3 <= Teamgröße <= 9 dann Schritt 7 *sonst*
Wenn Teamgröße > 9 dann Planungsmöglichkeit Variante 1 („Team zu groß“, aktuelle Teamgröße x) *sonst*
Planungsmöglichkeit Variante 2 („Team zu klein“, aktuelle Teamgröße y)

Für den zweiten Fall in der Analyse, dass die Planungsmöglichkeiten noch nicht bekannt sind, wird das Beispiel genutzt, dass aktuelle Standards und Normen genutzt werden sollen, z.B. ein aktueller Standard zum Testen. Wenn sich dieser nun ändert ist es nötig, die Software-Engineering-Methode darauf abzustimmen. Zunächst wird für die Auswertung des Wenn-Teils die aktuelle Version des Standards im Projekt ausgelesen, hier im Beispiel wird dafür die Versionsnummer genutzt. Zusätzlich wird aus der Wissensbasis die Versionsnummer für den aktuellen Standard allgemein ausgelesen. Diese beiden Werte werden in der Analyse miteinander verglichen. Ist die Versionsnummer aus dem Projekt gleich der allgemeinen Versionsnummer, so ist keine neue Planung nötig. Doch sind beide Versionsnummer ungleich, so wird die Plan-Phase gestartet. Dafür wird zusätzlich die neue Version, sofern dies möglich ist, heruntergeladen und mitgeliefert. Die Auswertung würde also folgendermaßen lauten:

> Wenn Versionsnummer_Standard_Testen_Projekt != Versionsnummer_Standard_Test_aktuell dann Plan neuer Standard(lese Test_Standard aus) *sonst*
> Schritt 7

Es ist zu sehen, dass es mit Hilfe der verschiedenen Analyseregeln einfach möglich ist, die verschiedenen Werte bezüglich der Ziele auf Abweichungen hin zu analysieren und zu bestimmen, ob eine Anpassung nötig ist oder nicht. Die Analyse kann somit vollständig automatisiert über das System erfolgen. Doch für eine Absicherung kann eingeführt werden, bevor die Planung endgültig angestoßen wird, dass sich der Projektleiter und/oder der Methoden-Engineer das Ergebnis der Analyse anschaut und freigibt, ob die Anpassung erfolgen soll oder nicht.

Wird die Anpassung nicht freigegeben, sondern bestimmt, dass die Software-Engineering-Methode mit diesen Werten weiterlaufen soll und die Ziele weiterhin erfüllt werden, so müssen die Regeln, Ziele und Werte entsprechend in der Wissensbasis entsprechend angepasst werden. Doch dies sollte nur in Ausnahmefällen durchgeführt werden, da dies je nach Ziel Auswirkungen auf weitere Ziele und die gesamte Software-Engineering-Methode haben kann. Die Ziele müssen zuerst überprüft werden, dass sie

nicht verletzt werden. Das kann unter Umständen zu einem höheren Aufwand führen, als die weitere Planung und Ausführung.

6.3.2 Architekturmöglichkeiten für die Analyse

Um die Analyse konkret umzusetzen und durchzuführen, gibt es verschiedene Überlegungen und Möglichkeiten. Als Grundlage für die Analyse und Bewertung dienen die in Schritt 3 erstellten Bewertungsregeln. Diese bestehen aus einer einfachen Form, einem Wenn-Teil zur Auswertung und einem Dann-Teil für weitere Ausführungen. Die erste einfache Möglichkeit ist es, alle Regeln in der Form „if… then… else“ zu programmieren. Doch dies würde zu einem hohen Aufwand und verbrauchtem Speicherplatz führen. Außerdem müssten die Funktionen auf die Einträge in einer Datenbank zugreifen, die in der einfachen Form noch nicht betrachtet sind.

6.3.2.1 Aktive Datenbanken

Eine erste Überlegung ist es, für die Speicherung, das Auslesen und weitere Auswerten der aufbereiteten Werte eine Datenbank und ein Datenbanksystem zu nutzen. Für Datenbanken gibt es die verschiedensten Formen und Funktionen, welche in einem Datenbankmodell festgelegt sind. Ein verwendetes Model ist die relationale Datenbank bzw. Datenbankmanagementsysteme.

Eine Datenbankmanagementsystem (DBMS) ist ein Softwaresystem, welches für die effiziente Erstellung und Verwaltung von Datenbanken verantwortlich ist. Dabei implementiert ein DBMS die Speicherung und Wiederherstellung von Daten sowie die Speicherung und Verwaltung von Objekten wie Zugriffspfaden, Clustering von Daten usw [DGG95]. Die eigentliche Speicherung der Daten mit allen relevanten Informationen erfolgt in der Datenbank (data base).

Für den Aufbau einer Datenbank gibt es verschieden Möglichkeiten, welche im Datenbankschema definiert sind. Der Zugriff und Änderungsmöglichkeiten werden im Datenmodell festgelegt. Das DBMS und die konkrete Datenbank bilden zusammen das Datenbanksystem, engl. database system (DBS). Eine aktive Datenbank oder mit anderen Worten ein aktives Datenbankmanagementsystem (active database management system, ADBMS) erweitert das passive DBMS mit der Möglichkeit, reaktives Verhalten auf spezielle Ereignisse wie beispielsweise Änderungen zu spezifizieren [DGG95, MD89, PD99].

Diese auslösenden Ereignisse, auch Trigger genannt, erfolgen zur Laufzeit und können weitere Aktionen oder auszuführende Skripte innerhalb der Datenbank anstoßen. Das dies zur Laufzeit möglich ist, ist für die Ausführung der Analyse und Bewertung essentiell.

Ein solches Trigger-System wird für die Ausführung der Aktivitäten benutzt und als Regel definiert. Die Trigger-Regel triggert in einem auslösenden Ereignis und unter bestimmten Bedingungen eine weitere Aktion, wenn der Trigger true zurückliefert. Die bekannteste Implementierung solcher Regeln, welche in DBMS bzw. ADBMS eingesetzt werden, sind die ECA-Regeln [PD99, Qi07, DGG95].

6.3.2.2 ECA-Regeln (Event-Condition-Action-Rules)

Die Buchstaben ECA stehen für die Abkürzung „Event“ (E), „Condition“ (C), und „Action“ (A). Zusammengefasst bestehen diese ECA-Regeln aus einem auslösenden Ereignis (Event), den Bedingungen, welche das Ereignis erfüllen (Condition) muss sowie der Aktion (Action), welche bei Erfüllung der Bedingungen als nächstes ausgeführt wird. Nach [DGG95, S. 4-5] besagt eine ECA-Regel: *„when an event occurs, check the condition and if it holds, execute the action".* Diese Beschreibung der ECA-Regeln kommt der definierten Analyseregel mit ihrer „Wenn…, Dann…“- Spezifikation sehr nah.

Analog zu einer „Data definition language“ (DDL) um eine Datenstruktur zu modellieren, liefert eine aktive Datenbank eine „Rule definition language (RDL), welche dazu genutzt werden kann, um die ECA-Regeln zu spezifizieren [DGG95]. Diese Sprache beinhaltet dabei Konstruktoren für die Definition der eigentlichen Regeln, also der Ereignisse, der Bedingungen, der Aktionen sowie Constraints zur Ausführung. Zusätzlich benötigt die ADBMS eine Möglichkeit, um Ereignisse zu entdecken und die ECA-Regeln anzuwenden. Dies kann beispielsweise über Sensoren oder Detektionsregeln erfolgen.

Übertragen auf die Analyseregel würde dies bedeuten, dass der aufbereitete Wert das auslösende Ereignis ist. Genauer ist die Speicherung dieses neuen Wertes in die aktive Datenbank das auslösende Event. Diesem Wert, sind die Bedingungen zugeordnet. Die Bedingungen sind hier die Abfrage, ob der Wert innerhalb der Grenzen liegt oder ein definierter Schwellenwert über- oder unterschritten wird. Dies ist der ursprüngliche Wenn-Teil der Analyseregeln. Der Dann-Teil der Regeln beschreibt die Aktion in den ECA-Regeln, welche bei Erfüllung der Bedingung durchgeführt wird.

Sind nun alle Regeln in der ECA-Form definiert und in der Wissensbasis in einer aktiven Datenbank entsprechend gespeichert, so kann die MAPE-K-Feedbackschleife mit Hilfe der Sensoren die relevanten Ereignisse überwachen. Sobald ein Ereignis, das heißt die Speicherung eines relevanten Wertes erfolgt ist, wird die entsprechende Regel-Komponente angesprochen, welche für die Ausführung der Regel zuständig ist. Dies wird auch *„Signalisierung des Ereignisses“* genannt [DGG95, S.5]. Zusätzlich zu dem Ereig-

nis kann auch ein Zeitstempel mitgegeben werden, wann das Ereignis eingetreten ist und würde entsprechend als ein Paar abgespeichert (Event, Zeit).

Auch wenn der Zeitstempel hier in der MAPE-K-Feedbackschleife nicht unbedingt erforderlich ist, soll dieser beibehalten werden. Wichtiger ist, dass dieses Paar um die Priorität ergänzt wird. Die Priorität ist gegeben durch das Ziel, für welches der Wert ausgewertet wird. Dies ist nötig, wenn zwei Werte gleichzeitig gespeichert werden, damit das Ereignis mit der höheren Priorität zuerst ausgewertet wird. Die aktive Datenbank muss also bei getriggerten Ereignissen zusätzlich den Mechanismus enthalten zu überprüfen, dass Ereignisse innerhalb desselben Zeitraumes, z.B. innerhalb einer Minute, entdeckt werden und bei ihnen die Prioritäten gegeneinander abgeglichen werden. Ist dieser Mechanismus gegeben und die Wissensbasis sowie die Analyse-Phase als aktive Datenbank mit ECA-Regeln realisiert, dann kann diese Phase vollständig automatisch durchgeführt werden.

Die Aufbereitung der Monitor-Phase kann als eine vereinfachte Form der Regel realisiert werden. Hier würde die zu überprüfende Bedingung wegfallen, denn ein gemessener Wert soll immer entsprechend aufbereitet werden. In Wenn-Dann-Form würde die Regel lauten: Wenn Wert xy gemessen, dann (bereite Wert auf mit Regel xy; schreibe Wert in KB; starte ECA mit Ereignis „Wert“). Mit dieser gekoppelten Form wird der gemessene Wert automatisch aufbereitet, in die Wissensbasis geschrieben und es wird automatisch das auslösende Ereignis entdeckt. Die Auswertung der Bedingungen wird entsprechend selbstständig gestartet. Ist die Auswertung positiv, so wird die Plan-Phase gestartet; ist die Auswertung negativ, wird zur Monitor-Phase übergegangen.

Somit wären mit Hilfe einer aktiven Datenbank und den ECA-Regeln die Phasen Monitor und Analyse abgedeckt und eine Automatisierung ermöglicht.

6.4 Schritt 9 – Anpassung planen

Schritt 9: Anpassung planen	
Aufgabe	Die Anpassung und mögliche Alternativen werden in diesem Schritt geplant. Herangezogen werden die vorher bestimmten Planungsmöglichkeiten, sowie die gemessenen Werte anhand derer Planungsmöglichkeiten bestimmt werden
Typ	Planen
Subschritte	1. Anpassung planen 2. Anpassung analysieren 3. Anpassungszeitpunkt

	4. Konflikte analysieren 5. Gegebenenfalls kombinierte Anpassung erstellen
Input des Schrittes	Ergebnis aus der Analyse, Planungsmöglichkeiten, Konfliktpotential, Kombinationsmöglichkeiten, gegebenenfalls Daten aus der Methoden-Basis
Output des Schrittes	Geplante Anpassung
Mögliche Stakeholder	System, bei Bedarf Methoden-Engineer und/ oder Projektleiter

Ist das Ergebnis der Analyse, dass die Software-Engineering-Methode angepasst werden muss, so wird mit Hilfe des Denn-Teils oder einer Action in einer ECA-Architektur die Plan-Phase angestoßen. Anders als im ursprünglichen MAPE-K besteht sie hier wie in Abbildung 44 zu sehen aus zwei Teilen. Der erste Teil entspricht der Plan-Phase im MAPE-K, hier wird die eigentliche Anpassung geplant und erstellt. Wie schon im Abschnitt vorher angesprochen, kann die Planung dabei mit Hilfe von bereits erstellten Planungsmöglichkeiten oder die Anpassung muss ohne Planungsmöglichkeiten mit Hilfe der Ziele und gemessenen Werten erfolgen.

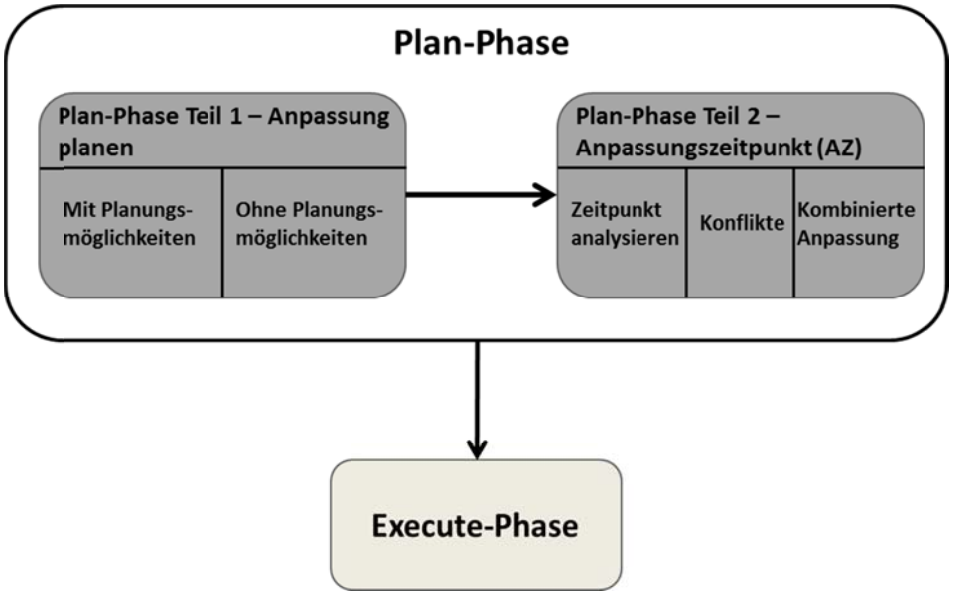

Abbildung 44 Die zwei Hauptbestandteile der Plan-Phase

Der zweite Teil der Plan-Phase berücksichtigt hier den Anpassungszeitpunkt. Wie bereits in Abschnitt 3.4.2 unter der Herausforderung TP 5 ange-

sprochen, muss bei der Anpassung einer Software-Engineering-Methode der Anpassungszeitpunkt (AZ) mit betrachtet werden. Es ist nicht wie bei einem System möglich, eine Software-Engineering-Methode zu jedem Zeitpunkt anzupassen, sie sollte nicht einmal täglich angepasst werden.

In diesem zweiten Teil wird zunächst der Zeitpunkt nach einer erfolgten Anpassung analysiert, das heißt ob der Anpassungszeitpunkt bereits erreicht ist oder nicht. Sind mehrere Anpassungen zum AZ vorhanden, müssen diese zunächst auf Konflikte hin analysiert werden und es muss gegebenenfalls eine kombinierte Anpassung erstellt werden, bevor zur Execute-Phase übergegangen wird.

6.4.1 Das Planen einer Anpassung

Sind Planungsmöglichkeiten bereits vorhanden, so wird die Anpassung entsprechend mit ihrer Hilfe durchgeführt. In den Planungsmöglichkeiten wurde bestimmt, wie in einer bestimmten Situation eine Anpassung vorgenommen werden soll. Das heißt, es ist angegeben, an **welcher Stelle** im Modell (Wo) bei einer Verletzung der Regel **welche Elemente** (Was), in **welcher Art** (Wie) angepasst werden, **welche Verbindungen** (Welche) erhalten oder neu gesetzt werden müssen und an wen **welche Benachrichtigung** (Wer) geschickt werden muss (vergl. Abschnitt 5.5 Ableitung von Planungsmöglichkeiten).

Dadurch ist klar, welche Elemente aus der Methoden-Basis für beispielsweise einen Austausch oder ein Hinzufügen genommen werden können. Zusätzlich ist klar, an welcher Stelle im Modell der Software-Engineering-Methode Elemente geändert werden und wie die neuen Abhängigkeiten aussehen. Außerdem wurde festgelegt, wer in der Ausführung alles benachrichtigt wird.

Für das Beispiel mit der Teamgröße in Scrum würden also bei einem zu großen Team die entsprechende Planungsmöglichkeit angegeben und in der Plan-Phase aufgerufen. Wie bereits in Abschnitt 5.5 beschrieben ist in der Planung durch den Aufruf festgelegt, dass das Team gesplittet wird und wie viele Personen jeweils ein neues Team enthält. Die konkrete Aufteilung der Personen und der jeweiligen Aufgaben erfolgt nun in der Plan-Phase. Dies kann zum einen automatisch mit Hilfe von Zuteilungsfunktionen erfolgen, welche als Eingabe die Aufgaben und die jeweiligen Skills der Personen kennen. Eine andere Möglichkeit ist, dass an dieser Stelle der Scrum Master mit einbezogen wird und dass dieser die Aufteilung vornimmt.

Das neue Meeting kann wieder automatisch mit Hilfe der Planungsmöglichkeiten eingefügt werden, da hier genau bekannt ist, welches Element (Scrum of Scrums) an welche Stelle (täglich nach dem Daily Scrum) eingefügt werden muss. Ferner ist bekannt, an wen alles eine Benachrichtigung geschickt

werden muss. Diese Angaben sowie die Zuordnung werden entsprechend an die Execute-Phase übergeben.

Sind Planungsmöglichkeiten nun nicht vorhanden, so muss die Planung wie in Abbildung 45 zu sehen mit Hilfe der gemessenen Werte und des Ziels durchgeführt werden. Für die Planung der Anpassungen müssen zunächst die genaue Abweichung und die Folgen bestimmt werden. In einem nächsten Schritt wird das Ziel herangezogen und es muss bestimmt werden, mit welchen Elementen, in welcher Art und an welcher Stelle dieses Ziel weiter eingehalten wird. Dies kann mit Hilfe von entsprechenden Methoden-Mustern (Method Pattern) erfolgen. Eine weitere Möglichkeit wäre, Planungsalgorithmen an dieser Stelle mit hinzuzuziehen.

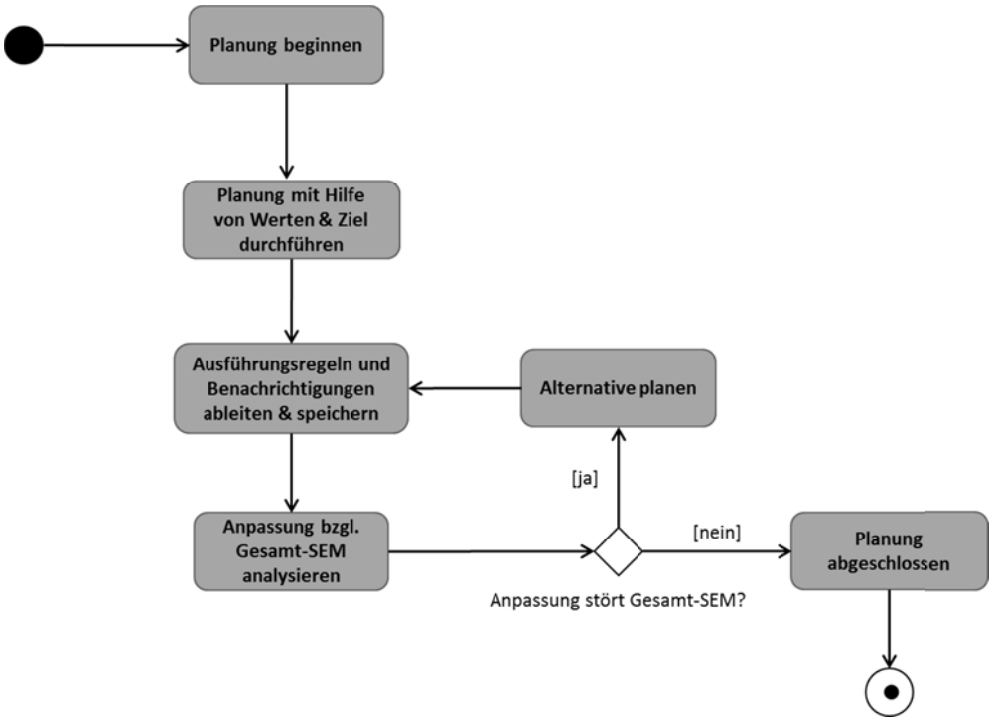

Abbildung 45 Planungsablauf ohne Planungsmöglichkeiten

Ist eine neue Anpassung mit Hilfe von Method Pattern und/ oder Planungsalgorithmen erstellt, so müssen daraus entsprechend für die Execute-Phase die Ausführungsregeln und Benachrichtigungen abgeleitet und in der Wissensbasis gespeichert werden. Die Ableitung kann mit Hilfe von Schritt 6 aus der Pre-Work erfolgen.

Zusätzlich zu der Herleitung der Ausführungsregeln und Benachrichtigungen ist es in einem weiteren Schritt wichtig, die geplante Anpassung bezüglich ihrer Auswirkung auf die Gesamt-SEM hin zu analysieren. Dies ist nötig um herauszufinden, ob diese neue Anpassung Auswirkungen auf andere Abschnitte der Software-Engineering-Methode hat und ob diese Anpassung bewirkt, dass andere Ziele verletzt werden. Die Analyse kann beispielsweise

mit einer Simulation und Simulationsalgorithmen durchgeführt werden. Dafür wird die Anpassung in eine Kopie des aktuellen Modells der SEM übertragen und in einer Simulation ausgeführt. Dadurch können die entsprechenden Auswirkungen bereits im Vorfeld abgeklärt werden. Eine andere Möglichkeit wäre, eine Analyse über den Workflow durchzuführen, beispielsweise ob alle Abhängigkeiten gegeben sind und entsprechend erreicht werden können.

Wie diese Analyse und die vorher beschriebene Planung mit Hilfe von Algorithmen oder Pattern im Detail aussehen, wird an dieser Stelle nicht im Detail betrachtet. Es wird als gegeben angesehen, dass sowohl eine Simulation oder Analyse zur Bewertung der Gesamtauswirkung als auch „Planungshilfen“ gegeben sind.

Ergibt die Bewertung, dass die Anpassung Auswirkungen auf die Gesamt-SEM hat, das heißt, dass andere Ziele verletzt werden, so muss eine alternative Anpassung geplant werden. Diese Alternative wird am Ende ebenfalls überprüft, ob sie weitere Ziele verletzt oder nicht. Es ist weiterhin zu überlegen und festzulegen, wie oft eine Alternative geplant werden soll. Im schlimmsten Fall kann es vorkommen, dass keine Alternative möglich ist, welche kein Ziel verletzt. Somit muss zusätzlich ermittelt werden, welche Art von Zielen verletzt wird, also welche Priorität die jeweiligen Ziele besitzen. Ist das verletzte Ziel beispielsweise von wesentlich geringerer Priorität, kann entschieden werden, dass die Anpassung trotzdem durchgeführt und das andere Ziel entsprechend angepasst wird. Ist die Priorität des verletzten Ziels höher als die der Anpassung, so muss eine Alternative geplant werden. Auch wenn solche Bestimmungen entsprechend für eine automatische Ausführung festgelegt werden können, sollte in einem solchen Fall der Projektleiter und/oder ein Methoden-Engineer hinzugezogen werden.

Ist für die Anpassung entschieden, dass diese die Gesamt-SEM nicht beeinflusst, also keine Ziele verletzt werden, so ist die Planung abgeschlossen und es kann zum zweiten Teil der Plan-Phase übergangen werden, dem Anpassungszeitpunkt.

6.4.2 Zwischen Planung & Ausführung – der Anpassungszeitpunkt

Wie bereits zu Beginn des Abschnitts 6.4 erwähnt, kann die Anpassung einer Software-Engineering-Methode nicht andauernd und schnell hintereinander erfolgen. Zu viele Anpassungen kurz nacheinander würden ebenso den Erfolg der Software-Engineering-Methode und somit des gesamten Projektes gefährden, wie wenn überhaupt keine Anpassung bei Problemen durchgeführt werden würde. Ferner würden zu viele Anpassungen zu Verwirrung und unter Umständen zu Unzufriedenheit bei den Teammitgliedern führen, wenn sich ihre Aufgaben ständig verändern. Würden sich die Auf-

gaben in einer hohen Taktzahl ändern, kann es passieren, dass Aufgaben nicht zu Ende gebracht werden können, da andere wichtiger sind und irgendwann das ganze Projekt hinter seiner Zeit zurückfällt.

Der Zeitpunkt der Anpassung erfolgt zum einen je nach Kritikalität der Anpassung, zum anderen je nach Dauer und Komplexität der Software-Engineering-Methode. Anpassungen sollten möglichst immer im gleichen Intervall folgen, beispielsweise einmal in der Woche, alle zwei Wochen etc. Der Anpassungszeitpunkt soll zu Beginn des Projektes mit allen Beteiligten besprochen und festgelegt werden. Ähnlich wie bei den Agilen Methoden sollen alle, die davon betroffen sind, also Teammitglieder, Projektleiter usw., mit einbezogen werden und sich austauschen. Diese Personen können am besten beurteilen, in welchen Intervallen sie vollständige Anpassung der Software-Engineering-Methode bewerkstelligen können, ohne dass es die Qualität der Arbeit beeinflusst und ihr Projekt gefährdet. Zusätzlich soll in die Software-Engineering-Methode ein regelmäßiger Zeitpunkt mit eingeplant werden, zu dem der Anpassungszeitpunkt besprochen und gegebenenfalls angepasst wird.

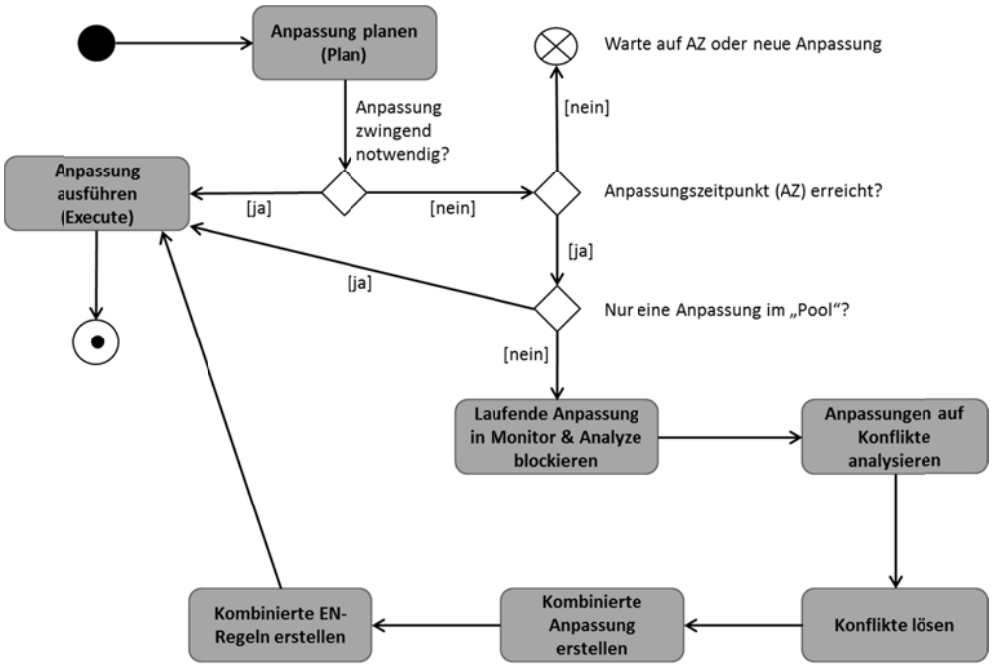

Abbildung 46 Durchführungsschritte zum Anpassungszeitpunkt

Ist im ersten Teil der Plan-Phase nun eine Anpassung geplant worden, wird nach der fertigen Anpassung zunächst wie in Abbildung 46 zu sehen die Kritikalität der Anpassung überprüft. Besitzt die Anpassung die Priorität 1, also die höchste Priorität, so heißt dies, dass die Anpassung zwingend notwendig ist, damit der Erfolg der Software-Engineering-Methode nicht ge-

fährdet wird. Dies ist der einzige Punkt, wo eine Anpassung der SEM nicht zum Anpassungszeitpunkt erfolgt und sofort zur Execute-Phase übergegangen wird.

Ist die Anpassung nicht zwingend erforderlich, so wird zuerst überprüft, ob der Anpassungszeitpunkt bereits erreicht ist. Ist dies nicht der Fall, so wird gewartet, bis der Anpassungszeitpunkt erreicht oder eine neue Anpassung in Teil 1 erfolgt ist. Die Anpassung wird solange in einem „Anpassungs-Pool" gespeichert. Ist der Anpassungszeitpunkt erreicht, wird die Anzahl der Anpassungen im Pool überprüft. Ist dort nur eine Anpassung zu finden, so wird zur Phase Execute übergegangen und die Anpassung wird direkt ausgeführt.

Sind bereits mehrere Anpassungen im Pool, so wird zunächst eine mögliche laufende Anpassung in den Phasen Monitor und Analyse blockiert, damit diese nicht weiter ausgeführt wird und mit den nun zu kombinierenden Anpassungen in Konflikt stehen. Die bereits fertigen Anpassungen werden zunächst ebenfalls dahingehend analysiert, ob sie bzw. ihre Ziele möglicherweise in Konflikt zueinander stehen und sich gegenseitig beeinflussen. Für diese Analyse kann das in Abschnitt 5.5.3 bestimmte Konfliktpotential herangezogen werden. Stehen eine oder mehrere Anpassungen in Konflikt zueinander, so muss zunächst nach einer Lösung zur Beseitigung der Konflikte gesucht werden. Dies kann unter Umständen zu einer Umplanung einer Anpassung führen oder sogar dazu, dass eine Anpassung nicht durchgeführt und verworfen wird. Dies kann nur erfolgen, wenn die Priorität der einen Anpassung höher ist als die der anderen.

Sind alle Konflikte gelöst, kann nun eine kombinierte Anpassung geplant und erstellt werden, da nicht alle Anpassungen gleichzeitig oder hintereinander durchgeführt werden. Mit der kombinierten Anpassung ist ebenfalls sichergestellt, dass die verschiedenen Abhängigkeiten eingehalten und beachtet worden sind. Um eine solche kombinierte Anpassung erstellen zu können, kann die in Abschnitt 5.5.4 vorgestellten Kombinationsmöglichkeiten genutzt werden. Dort wurde bestimmt, ob und welche Kombinationspunkte es innerhalb der jeweiligen Anpassungen gibt und wie diese aussehen. Anzusetzen ist dabei an den beiden Punkten aus Abschnitt 5.5.1 **Wo** angepasst wird, **Was** (welche Elemente) angepasst und **Wie** (Art der Anpassung) angepasst werden kann. Mit Hilfe dieser Kombinationspunkte können am Ende die verschiedenen Anpassungen zu einer einzigen verschmolzen werden.

Ist die kombinierte Anpassung erstellt, so ist die Plan-Phase abgeschlossen und es wird die Execute-Phase mit der kombinierten Anpassung angestoßen, damit sie entsprechend ausgeführt und in das Projekt zurück übertragen werden kann.

6.5 Schritt 10 – Anpassung ausführen

Schritt 10: Anpassung ausführen	
Aufgabe	Die geplante Anpassung wird ausgeführt
Typ	Ausführen
Subschritte	1. Anpassung ausführen & Modell anpassen 2. Modelle speichern 3. Benachrichtigungen durchführen
Input des Schrittes	Anpassung zur Ausführung (Modell, Ausführungsregeln, Benachrichtigung)
Output des Schrittes	Angepasste Software-Engineering-Methode
Mögliche Stakeholder	System (Execute), Projektleiter (gibt das okay zur Ausführung)

Sind die Anpassungen bestimmt und ist der Anpassungszeitpunkt erreicht oder eine sofortige Anpassung erforderlich, so wird die Execute-Phase angestoßen. Die Ausführung der Anpassung und die Übertragung der angepassten Software-Engineering-Methode erfolgt mit Hilfe der in Schritt 6 bestimmten Ausführungsregeln und Benachrichtigungen.

Die Execute-Phase läuft wie bereits im Framework unter Subschritte beschrieben, in drei Schritten ab:

1. Die Anpassung der Software-Engineering-Methode wird mit Hilfe der Anpassungsregeln in die Modelle der SEM übertragen.
2. Die angepasste SEM wird in der Wissensbasis für die weitere Durchführung gespeichert. Das alte Modell wird gleichzeitig in die Historie mit Datum übertragen.
3. Es werden Benachrichtigungen an alle Beteiligten geschickt.

Sobald die Execute-Phase angestoßen wurde, werden anhand der geplanten Anpassung die entsprechenden Ausführungsregeln und Benachrichtigungen ausgelesen. Diese wurden in Schritt 6 in der Pre-Work oder in der vorherigen Plan-Phase erstellt und abgespeichert. Mit Hilfe dieser Regeln wird das aktuelle Modell der Software-Engineering-Methode entsprechend für die weitere Durchführung aktualisiert.

Anhand der jeweiligen Regel werden automatisch die entsprechenden Aktivitäten, Techniken, Artefakte etc. im Modell hinzugefügt, gelöscht oder

ausgetauscht usw. Zusätzlich zu diesen geänderten Elementen müssen, falls nötig, die konkreten Verantwortlichen bzw. die Ausführenden des Elements geändert werden. Ferner ist durch die Regel bekannt, welche Abhängigkeiten, wenn nötig, neu im Modell geknüpft und geändert werden müssen.

Dieses neue Modell wird entsprechend in der Wissensbasis gespeichert und wird für die weitere Durchführung mit Hilfe der Effektoren freigegeben. Das heißt, die Anpassung ist beendet und der neue Status des aktuellen Projektes kann über die angepasste Software-Engineering-Methode abgerufen werden. In demselben Schritt, wenn das neue Modell in die Wissensbasis gespeichert wird, wird das alte Modell nicht nur überschrieben, sondern eine Kopie dieses alten Modells wandert mit einem Zeitstempel und einer Markierung, was eine Anpassung hervorgerufen hat, in die Historie. Die Historie kann zu einem gewählten Zeitpunkt und besonders nach Beendigung des Projektes hinsichtlich aller Anpassungen, die während des Projektes stattgefunden haben, abgerufen und ausgewertet werden. Diese Erfahrung kann beispielsweise für das Aufsetzen von neuen Projekt genutzt und gemachte Fehler können im Vorfeld mit beachtet und vermieden werden.

Als letzten Schritt in der Execute-Phase und somit in der MAPE-Feedbackschleife werden Benachrichtigungen an alle Beteiligten geschickt, die von der Änderung betroffen sind. Wer von den einzelnen Änderungen betroffen ist, wurde bereits in Schritt 6 entsprechend herausgearbeitet, so dass in der Execute-Phase eine entsprechende Nachricht automatisch vom System geschickt werden kann. Die Nachrichten sind insofern personalisiert, als dass der Betroffene jeweils die Änderung mitgeteilt bekommt, welche ihn betrifft.

Im Beispiel der Teamgröße in Scrum hieße das, wenn ein Team zu groß ist und gesplittet wurde, wird über die Benachrichtigungen jedem Teammitglied mitgeteilt, welchem Team er oder sie nun angehört, welche Aufgaben zugeteilt wurden und wer am ersten Scrum of Scrums teilnimmt. Ferner wird ihnen der Zeitpunkt mitgeteilt, ab wann die Änderung greift. In diesem Beispiel wird dafür der Anfang des nächsten Sprints gewählt, so dass alle Beteiligten erst ihre entsprechenden Aufgaben beenden können.

Auch wenn mit Hilfe der Regeln und Benachrichtigungen die Schritte automatisch ausgeführt werden können, so kann es Sinn machen, dass vor Beginn der eigentlichen Durchführung oder dem Versenden der Benachrichtigungen zunächst der Projektleiter und/ oder der Methoden-Engineer mit auf die fertige Anpassung draufschaut und diese freigibt. Ist die Freigabe erteilt, die Anpassung durchgeführt und sind alle Beteiligten benachrichtigt, so ist die Anpassung abgeschlossen und eine angepasste Software-Engineering-Methode liegt vor. Die Evaluierung und Überwachung dieser SEM beginnt

nun von neuem mit Schritt 7, dem Messen der Werte anhand der Sensoren und der Monitor-Phase.

6.6 Automatisierungen

In Abschnitt 3.3.3 wurden bereits Möglichkeiten für die Automatisierung des Ansatzes vorgestellt und kurz diskutiert. Das Ziel war es, dass der SE Method Manager mit der Hilfe von MAPE-K automatisiert werden kann, wie es beim ursprünglichen Autonomic Manager der Fall ist. Es hat sich herausgestellt, dass eine Automatisierung in den meisten Fällen möglich ist. Doch an einigen Stellen im MAPE-K4SEM ist es sinnvoll, eine oder mehrere Personen einzusetzen. Auch wenn es beim Autonomic Manager möglich ist, alle Phasen zu automatisieren, geschieht dies nicht immer bei allen Phasen.

Im vorliegenden Ansatz ist es wie beschrieben so, dass er größtenteils automatisch abläuft und Personen nur punktuell eingesetzt werden. Mit Hilfe von definierten Sensoren erfolgt die Messung der spezifischen Daten automatisch. Anhand der definierten Aufbereitungsregeln werden diese in der Monitor-Phase selbstständig vom System aufbereitet. Wie in Abschnitt 6.3.1. dargelegt, kann mit der einfachen Form der Regel und der Kombination mit ECA-Regeln die Monitor-Phase vollständig automatisch ablaufen und die Analyse-Phase wird über den Einsatz von ECA-Regeln automatisch angestoßen.

Ebenfalls ist es wie in Abschnitt 6.3.1. beschrieben mit einer aktiven Datenbank und dem Einsatz von den beschriebenen ECA-Regeln möglich, dass die Analyse vollständig automatisiert durchgeführt wird. Sobald die aufbereiteten Werte gespeichert sind, wird damit die entsprechend dem Wert definierte ECA-Regel ausgelöst und die Werte werden anhand der Bedingung ausgewertet. Je nachdem, ob die Bedingung positiv (true) oder negativ (false) ist, wird die Plan-Phase mit der entsprechenden Planungsmöglichkeit oder den gemessenen Werten und Ziel angestoßen oder es wird wieder zu Schritt 7 übergegangen.

Der erste Zeitpunkt, an dem eine Person hinzugezogen werden kann, ist während des Übergangs zwischen der Analyse- und der Plan-Phase, auch wenn dieser automatisch möglich ist. Es kann hier Sinn machen, dass sich ein Methoden-Engineer und/oder der Projektleiter zu diesem Zeitpunkt Software-Engineering-Methode anschaut, ob an dieser Stelle wirklich eine Anpassung nötig oder ob sie noch im Rahmen ist. Dies kann allein schon aus Erfahrungsgründen der Personen sinnvoll sein. Anschließend können sie die Freigabe für die Anpassung erteilen und den Übergang zu der Plan-Phase einläuten. Oder sie lehnen die Anpassung ab und MAPE-K wird in Schritt 7 weiter fortgesetzt.

Wurde manuell beschlossen, dass die Anpassung nicht nötig ist, müssen an dieser Stelle die Rahmenbedingungen sowie die Regeln entsprechend manuell angepasst werden, ansonsten werden der vermeintliche Fehler und seine Anpassung im nächsten Schritt möglicherweise wieder erfolgen. Dabei sollte bedacht werden, dass diese manuelle Änderung wiederum Aufwand und mögliche Zeiteinbußen mit sich bringen kann. Die manuelle Änderung sollte daher nur in dringenden Fällen angewendet werden.

Die Plan-Phase selbst kann wie beschrieben ebenfalls in den meisten Fällen automatisch erfolgen, vor allem dann, wenn bereits Planungsmöglichkeiten vorhanden sind und diese nur ausgeführt werden müssen. Sind keine Planungsmöglichkeiten vorhanden, kann es unter Umständen sein, dass an dieser Stelle ein Methoden-Engineer hinzugezogen werden muss. Eine Automatisierung oder zumindest eine Teil-Automatisierung ist an dieser Stelle möglich, wenn entsprechende Planungsalgorithmen vorliegen und angewandt werden können. Die Ableitung der Ausführungsregeln und Benachrichtigungen sollten allerdings entsprechend Schritt 6 manuell durchgeführt werden. Die Auswirkung der Anpassung auf die Gesamt-SEM sollte wiederum möglichst automatisiert mit Hilfe von beispielsweise Simulationen durchgeführt werden. Manuell würde dies einen zu großen Aufwand bedeuten, da verschiedene Möglichkeiten per Hand durchgespielt werden müssen um herauszufinden, ob und welche Auswirkungen die Anpassung auf die Gesamt-SEM hat. Der Übergang zum Anpassungszeitpunkt erfolgt ebenfalls automatisiert.

Der Anpassungszeitpunkt kann wiederum automatisch ausgewertet werden und falls die Blockierung der vorherigen Phasen nötig ist, kann dies ebenfalls vom System selbstständig erfolgen. Je nach Implementierung mit Hilfe der Konfliktmöglichkeiten kann das Ermitteln der Konflikte der einzelnen Anpassungen zueinander noch automatisch erfolgen. Doch für die Auflösung der Konflikte und für eine Kombination der Anpassungen kann ein Methoden-Engineer nötig sein. Sind allerdings entsprechende Planungsalgorithmen und Kombinationspunkte bekannt, so kann eine Automatisierung möglich sein.

Mit dem Übergang von der Plan-Phase zur Execute-Phase und der endgültigen Ausführung verhält es sich ähnlich wie beim Übergang von der Analyse zur Planung. Der Übergang von der Plan- zur Execute-Phase ist automatisch möglich. Doch an dieser Stelle kann es ebenfalls Sinn machen, dass sich ein erfahrener Methoden-Engineer die geplante Anpassung anschaut, sie aufgrund seiner Erfahrung einschätzt und anschließend zur Durchführung freigibt. Diese menschlichen Zwischenschritte nutzen die Erfahrung der Personen und erhöhen das Vertrauen in die angepasste Methode.

Die Execute-Phase mit der Durchführung der Anpassung, der Änderung des Modells mit Hilfe der Ausführungsregeln und das Senden der Benachrichtigungen kann entsprechend automatisch ausgeführt werden. Mit dem letzten Schritt, dem Versenden der Benachrichtigungen, ist eine Anpassung abgeschlossen und sie wird automatisch durch das kontinuierliche Ausführen des SE Method Managers evaluiert und überwacht.

Das kurze Fazit bezüglich der Automatisierungen lautet, dass MAPE-K4SEM automatisch durchgeführt werden kann. Somit ist auch sie selbst-adaptive Anpassung einer SEM, also eine „selbst-adaptive Software-Engineering-Methode“, ist. Es macht an den genannten Übergängen und in der Plan-Phase jedoch Sinn, einen erfahrenen Methoden-Engineer einzusetzen.

Kapitel 7 Evaluierung

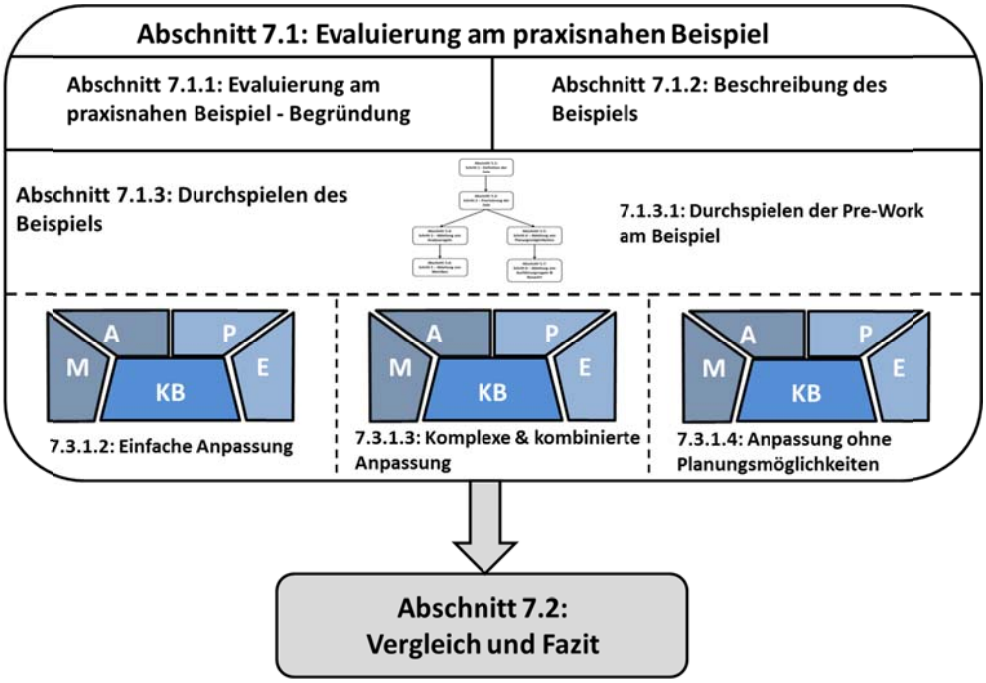

Abbildung 47 Aufbau Kapitel 7

In den Kapiteln 3 bis 6 wurde MAPE-K4SEM – ein Ansatz für die selbst-adaptive Anpassung von Software-Engineering-Methoden – vorgestellt. In diesem Kapitel soll der Ansatz an einem praxisnahen Beispiel durchgespielt und evaluiert werden. Nach einer Begründung für die Wahl dieses Beispiels zur Evaluierung wird es im nächsten Abschnitt im Detail beschrieben. Das Beispiel ist dabei angelehnt an das konkrete Projekt „Quasi-Scrum“ aus dem s-lab – Software Quality Lab, welches bereits in vorherigen Kapiteln erwähnt wurde. An einigen Stellen wird dieses Beispiel zusätzlich mit Erfahrung aus einem anderen Projekt angereichert. Nach der Beschreibung des Beispiels beginnt das Durchspielen mit den Schritten 1bis 6, der Pre-Work. Nachdem die einzelnen Elemente erstellt sind, beginnt der eigentliche Durchlauf mit MAPE-K, die Schritte 7 bis 10. Im Beispiel werden drei verschiedene Anpassungen durchgespielt, zunächst eine einfache Anpassung, die mehrfach erwähnte Teamgröße. Daran schließt sich eine komplexe und kombiniert Anpassung sowie eine Anpassung ohne Planungsmöglichkeiten an. Das Kapitel schließt mit einem Vergleich der Ergebnisse mit den Praxis-Projekt und einem Fazit.

7.1 Evaluierung an einem praxisnahen Beispiel

Das Durchspielen eines praxisnahen Beispiels soll hier zeigen, wie die Pre-Work und später die Durchführung des MAPE-K in der Praxis aussehen und funktionieren können. Für das Durchspielen wurden dabei sowohl eine Änderung aus dem Projektcontrolling gewählt, welche eine Anpassung nur auf der Instanz-Ebene bewirkt, als auch eine komplexere und kombinierte Anpassung einer Software-Engineering-Methode sowie einer Anpassung ohne Planungsmöglichkeiten. Um im Beispiel die verschiedenen Änderungen besser zeigen zu können, wurde jeweils ein Ausschnitt des stark vereinfachten Modells der Software-Engineering-Methode gewählt. Dieses Modell wird im Beispiel beibehalten und kontinuierlich mit der jeweiligen Anpassung erweitert.

7.1.1 Evaluierung am praxisnahen Beispiel – Begründung

Für diese Arbeit wurde die Evaluierung am praxisnahen Beispiel gewählt. Gerade im Bereich der Durchführung und Überprüfung von Software-Engineering-Methoden ist es schwierig, diese direkt zu überprüfen und mit dem ursprünglichen Fall zu vergleichen. Zwar ist die Dauer der Anpassung im vorgestellten Ansatz in der Durchführung zeitnah und größtenteils automatisch möglich, doch die Überprüfung müsste an einem Projekt während der Durchführung über einen längeren Zeitraum erfolgen. Auch müsste das Projekt von Anfang an mit begleitet werden, da die Pre-Work zu Beginn des Projektes durchführt werden muss.

Der Ansatz sollte zudem in einem Projekt in der Praxis, das heißt in einem Unternehmen umgesetzt und überprüft werden. Doch für eine Evaluierung ist es schwierig, ein Unternehmen zu finden, welches bereit ist, diese durchzuführen, denn im Unternehmen geht es bei einem Projekt um Zeit und hauptsächlich auch um Kosten. Um zu überprüfen, ob der Ansatz nicht nur funktioniert, sondern bessere Ergebnisse für die Anpassung liefert als ohne den eingesetzten Ansatz, müssten für einen direkten Vergleich dasselbe Projekt praktisch zweimal unabhängig voneinander gestartet werden, einmal mit und einmal ohne den Ansatz. In der Unternehmenspraxis ist dies gar nicht oder nur sehr schwer möglich.

Von daher wurde für das Durchspielen des Ansatzes ein Beispiel gewählt, welches sich nah an der Praxis befindet. Zwar ist das Beispiel konstruiert, aber der Kern des Beispiels wurde in ähnlicher Form bereits in einem Projekt ohne den Einsatz des Ansatzes durchgeführt. Auch wenn das Beispiel mit weiteren Aspekten angereichert wird, so stammen diese ebenfalls alle aus der Praxiserfahrung und sind bereits vorgekommen. Durch diesen Praxisbezug ist es am Ende möglich, den Ansatz mit den ursprünglichen Projekten zu vergleichen und schlussendlich ein Fazit zu ziehen.

7.1.2 Beschreibung des Beispiels

Das Beispiel, welches hier für die Evaluierung verwendet wird, ist im Kern an das Praxisprojekt Quasi-Scrum [EG09] angelehnt. Dieses Projekt wurde vom s-lab – Software Quality Lab – durchgeführt und in den vorherigen Kapiteln bereits mehrfach erwähnt. In diesem Projekt wurde die Software-Engineering-Methode angepasst, allerdings ohne die Hilfe von MAPE-K4SEM und erst sehr spät im Projekt, als es schon in einigen Punkten „vor die Wand gelaufen“ war. Dadurch hat das Unternehmen die Laufzeit verlängern und einen weitere Iteration zur Reparatur einschieben müssen, was im Unternehmen zusätzliche Kosten verursacht hat.

In diesem Beispiel soll jedoch nicht nur diese Anpassung aus diesem Projekt gezeigt werden, sondern es wird mit der Erfahrung aus einem weiteren Praxisprojekt angereichert sowie mit Empfehlungen aus einem Artikel, welcher weitere Erfahrungen widerspiegelt. Es sollen wie bereits geschrieben in diesem Beispiel sowohl eine einfache Anpassung als auch komplexe Anpassungen gezeigt werden. Zusätzlich wird an einem einfachen Beispiel gezeigt, dass es mit dem Ansatz ebenfalls möglich ist zu erkennen, dass Ressourcen fehlen oder wegfallen und diese mit Hilfe der Planung schnellstmöglich ersetzt und ausgetauscht werden können.

Im Projekt Quasi-Scrum wurde eine Kreditkalkulationssoftware entwickelt. Dafür wurde die Agile Methode Scrum eingesetzt. Da Scrum aus Sicht der Beteiligten nicht optimal war, wurde die Software-Engineering-Methode bereits im Vorfeld mit Hilfe von Tailoring angepasst. Die Ausgangslage soll hier im Beispiel ebenfalls verwendet werden, um möglichst nah an der Praxis zu bleiben. Die Daily Scrums finden im Beispielprojekt anstatt täglich 15 Minuten zweimal in der Woche (dienstags und donnerstags) jeweils 30 Minuten statt. Der Product Owner ist keine einzelne Person sondern besteht aus einem Lenkungskreis von 5 Personen. Das Review-Meeting war im Projekt kein Informationsmeeting sondern ein Abnahmemeeting. In diesem Beispiel bleibt das ursprüngliche Review-Meeting allerdings erhalten. Die restlichen Regeln und Eigenschaften von Scrum bleiben ebenfalls erhalten. Wie sich die Anpassungen auf Regeln und Metriken auswirken, wird beim Durchspielen des Beispiels erläutert.

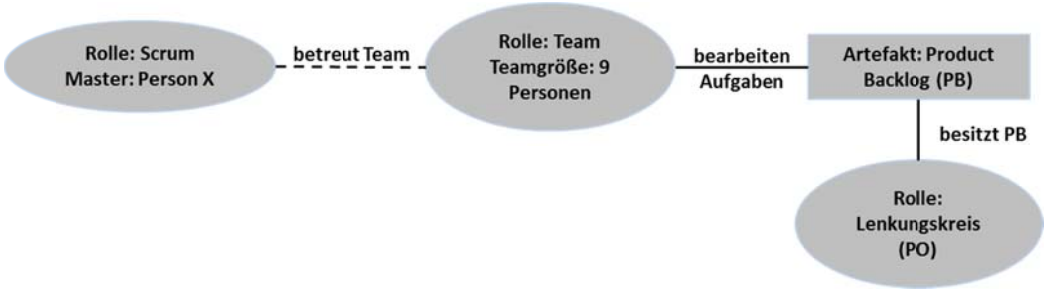

Abbildung 48 Hauptrollen in Quasi-Scrum

Die Hauptrollen im Beispiel teilen sich wie in Abbildung 48 zu sehen wie folgt auf:

- Es gibt einen Scrum Master, welcher das Team begleitet und durch Person X im konkreten Projekt realisiert wird.
- Der Scrum Master betreut das Team, welches zu Beginn des Projektes bzw. zum Start der Software-Engineering-Methode konkret aus 9 Personen besteht.
- Der Product Owner wird hier durch einen Lenkungskreis dargestellt. Dieser ist Besitzer des Hauptartefaktes Product Backlog, in welchem alle Anforderungen und Aufgaben für die Entwicklung der Software beinhaltet sind. Der Lenkungskreis wählt die verschiedenen Aufgaben aus und priorisiert diese.
- Das Team bearbeitet während des Sprints die Aufgaben im Product Backlog.

Der Ablauf eines Sprints zu Beginn des Beispiels wird in stark vereinfachter Form in Abbildung 49 dargestellt, wobei diese Vereinfachung zur Verdeutlichung der verschiedenen Anpassungen beibehalten wird. Die Sprintlänge, also die Länge einer Iteration, beträgt hier 4 Wochen. Dabei beginnt der Sprint montags mit dem Planning-Meeting, in dem gemeinsam im Team die Aufgaben für den aktuellen Sprint besprochen werden. Der Scrum Master hat die priorisierten Aufgaben bereits vom Lenkungskreis erhalten. Das Team bestimmt nun, welche Aufgaben es in den nächsten vier Wochen realisieren kann.

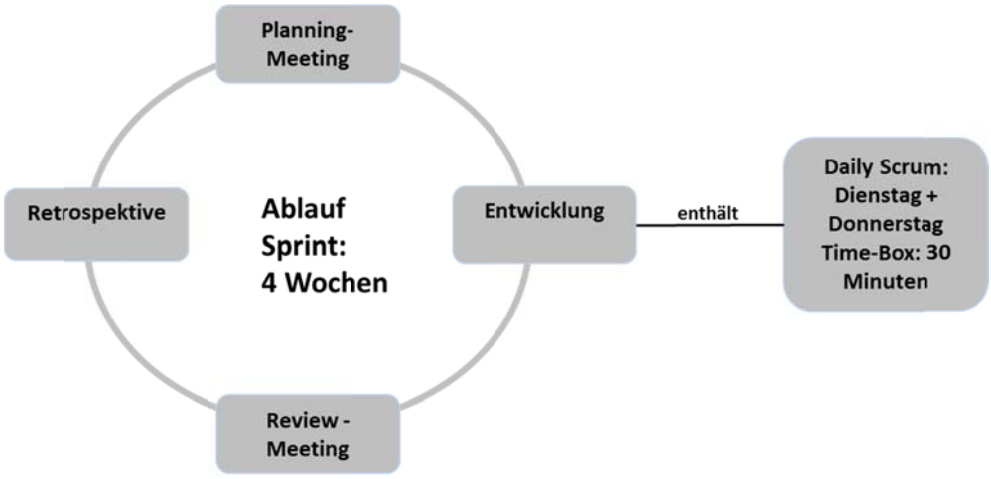

Abbildung 49 Vereinfachter Ablauf in Quasi-Scrum

Während der Entwicklung führt das Team jeden Dienstag und Donnerstag ein Daily Scrum von 30 Minuten durch, wo besprochen wird, was die einzelnen Teammitglieder bisher getan haben und was sie in den nächsten Tagen an Aufgaben bearbeiten wollen. Zusätzlich nimmt der Scrum Master die Probleme des Teams auf und versucht diese zu lösen. Ferner zeigt er mit Hilfe von sogenannten BurnDown-Charts, wo das Team aktuell steht und was es bisher geschafft hat. Nach der Entwicklung findet zusammen mit dem Lenkungskreis ein Review-Meeting statt, in dem der Status nach dem Sprint vorgestellt wird. Daran schließt sich die Retrospektive an, bei der die Teammitglieder besprechen, was im Sprint gut und was schlecht gelaufen ist sowie was sie im nächsten Sprint ändern möchten. Im Original-Projekt fand die Retrospektive erst ab Sprint 7 statt, im Beispiel wird sie von Beginn an eingesetzt.

Im Folgenden soll dieses Grundszenario im Laufe der verschiedenen Sprints mit Hilfe von MAPE-K4SEM angepasst werden. Für die Anpassungen wurden verschiedene Szenarien überlegt, die der Praxis entspringen. Eine einfache, aber dennoch sehr wichtige Anpassung, die ausschließlich auf Instanz-Ebene erfolgt, bezieht sich auf die Erkennung, wenn eine Person plötzlich krank wird, Urlaub hat oder sogar das Unternehmen verlässt. Diese Person muss zeitweise oder vollständig im Team durch eine andere mit den entsprechenden Fähigkeiten für die Aufgaben ersetzt werden. Eine andere Möglichkeit kann sein, dass die Aufgaben der Person je nach Aufwand durch das Team kompensiert werden können. Besonders wichtig ist, dass die Teammitglieder über die Änderung der Verantwortlichkeit informiert werden.

Auch wenn dies eine sehr einfache Erkennung und Anpassung während der Laufzeit der Software-Engineering-Methode und eher der Projektkontrolle zuzuordnen ist, so ist diese Erkennung doch essentiell. In der Praxis kommt dieser Fall, sei es durch Krankheit oder Urlaub, regelmäßig vor. Wo Urlaub im Voraus relativ gut einzuplanen ist, ist eine Vertretung bei Krankheit hingegen wesentlich schwieriger „auf die Schnelle“ zu finden.

Die Praxiserfahrung aus einem anderen Projekt hat gezeigt, wie wichtig diese Erkennung bei großen und verteilten Projekten ist. In diesen Projekten geht schnell der Überblick über die einzelnen Personen verloren und bei Abwesenheit oder gar Verlassen des Unternehmens sind die Verantwortlichkeiten teilweise nicht mehr klar. In der Praxis ist es im angesprochenen Projekt mehrfach vorgekommen, dass Personen aus einem anderen Team das Unternehmen verlassen haben und die anderen Teams nicht darüber informiert wurden. Einmal hat ein solcher Fall das Team mehrere Tage gekostet, um den neuen Ansprechpartner zu finden. Durch MAPE-K4SEM

hätte dies verhindert werden können. Deshalb soll diese einfache Anpassung mit ins Beispiel aufgenommen werden.

Eine weitere Anpassung ist das mehrfach genannte Beispiel der Teamgröße in Scrum. Auch wenn der Team-Split in Scrum vorgesehen ist, so wird er in der Praxis nicht unbedingt vorgenommen. Auch im Praxisprojekt vergrößerte sich das Team zeitweise, wurde aber nicht aufgeteilt, was zu Kommunikationsschwierigkeiten und Verzögerungen führte. Aus diesem Grund wird diese Anpassung ebenfalls mit aufgenommen.

Nachdem die Anpassung vorgenommen wurde, wird in einer ersten komplexen Anpassung die Anpassung des ursprünglichen Quasi-Scrum-Projektes mit Hilfe von MAPE-K4SESM durchgeführt. Um die insbesondere nicht-funktionalen Eigenschaften wie Performanz besser erkennen und im Product Backlog priorisieren zu können, wurde eine neue Phase, die „Kunden-Test-Tage“, mit zusätzlichen Rollen in die Software-Engineering-Methode eingeführt.

In einer weiteren komplexen Anpassung wird betrachtet, dass das Testen in Scrum nicht immer optimal läuft. Auch wenn Testen einen wichtigen Aspekt in den Agilen Methoden einnimmt, so ist es doch nur wenig und unstrukturiert beschrieben. Dies kann zu unzureichenden Tests und späteren Fehlern in der Software führen. Durch das Hinzufügen von speziellen Testaktivitäten kann dem entgegen gewirkt werden. Um ebenfalls eine kombinierte Anpassung zu zeigen, werden im Beispiel diese Anpassung und die Kunden-Test-Tage zu einer zusammengefasst. Wie dies zur Laufzeit mit Hilfe von MAPE-K4SEM funktioniert, wird in Abschnitt 7.1.3.4 gezeigt.

Beendet wird die Evaluierung mit einer Planung ohne Planungsmöglichkeit, dem bereits genannten Beispiel, dass ein aktueller Standard verwendet werden soll.

7.1.3 Durchspielen des Beispiels

Das Grundszenario des Beispiels ist erläutert, ebenso was in den Sprints im Laufe des Projektes angepasst werden soll. In den folgenden Abschnitten wird nun erläutert, wie diese Anpassungen im Detail mit Hilfe von MAPE-K4SEM aussehen. Zunächst werden dabei die ersten 6 Schritte – die Pre-Work – durchgespielt und die Details für die einzelnen Beispiele werden festgelegt und hergeleitet. Dabei ist zu erwähnen, dass die Ziele, Regeln, Metriken usw. hier im Beispiel nur auszugsweise für genau die einzelnen Anpassungen festgelegt werden. Das Definieren und Herleiten aller Ziele, Regeln und Metriken würde den Rahmen der Evaluierung sprengen.

Im weiteren Verlauf dieses Abschnittes werden für die verschiedenen Anpassungen die Schritte 7 bis 10 im MAPE-K durchlaufen und erläutert. Im

abschließenden Abschnitt 7.2 werden die Pre-Work und die verschiedenen Anpassungen soweit möglich mit der Praxis und den ursprünglichen Projekten verglichen. Darauf aufbauend wird ein abschließendes Fazit gezogen.

7.1.3.1 Durchspielen der Pre-Work am Beispiel

Bevor die einzelnen Schritte der Pre-Work für die verschiedenen Beispiel-Anpassungen durchgespielt werden, muss zunächst erläutert werden, was bereits vorab als gegeben angesehen wird.

Voraussetzungen und Festlegungen

Bevor die Pre-Work durchgespielt werden kann, gibt es verschiedene Elemente, welche in der Wissensbasis gespeichert werden, die bereits vorhanden und gegeben sein müssen. Dies sind die Voraussetzungen, damit die z10 Schritte des MAPE-K4SEM durchgeführt werden können. Das wichtigste Element, die eigentliche Software-Engineering-Methode muss gegeben und mit allen Beteiligten abgestimmt sein. In diesem Beispiel wird wie geschrieben als Software-Engineering-Methode die Agile Methode Scrum verwendet.

Falls nötig, muss vor Beginn die Software-Engineering-Methode mit Hilfe von Tailoring für eine Situation angepasst worden sein oder sie wurde mit Situational Method Engineering erstellt. In diesem Beispiel wurde die Software-Engineering-Methode bereits angepasst. Um den MAPE-K4SEM nun durchführen zu können, muss die gegebene Software-Engineering-Methode als Modell vorliegen. Die Modellierung kann mit den in Abschnitt 2.1.2 beschrieben Möglichkeiten oder auch einer abgeschwächten Form davon erfolgen. Wichtig ist, dass der Workflow und alle Aktivitäten, Rollen, Artefakte, Techniken usw. modelliert worden sind, damit die Anpassungen am Modell entsprechend vorgenommen werden können.

Zusätzlich soll bereits eine Methoden-Basis mit verschiedenen Bausteinen vorhanden und in der Wissensbasis abgespeichert sein. Diese Methoden-Basis kann zum einen im Unternehmen von erfahrenen Methoden-Engineern selbst entwickelt worden sein. Oder es können bereits vorhandenen Methoden-Basen, z.B. aus dem Internet oder von anderen Projekten, übernommen und abgespeichert werden. Dies ist wichtig, falls eine Anpassung ohne Planungsmöglichkeiten geplant werden muss. Ist dies nötig, so können verschiedene Bausteine aus der Methoden-Basis entnommen und eingesetzt werden. Die Methoden-Basis ist um zusätzliche Bausteine erweiterbar. Die Methoden-Basis im Beispiel besteht aus Bausteinen der verschiedenen Agilen Methoden mit zusätzlichen Elementen aus den klassischen Methoden.

Des Weiteren ist es nötig, im Vorfeld den Anpassungszeitpunkt für die Software-Engineering-Methode festzulegen. Wie bereits im vorherigen Kapitel angesprochen ist dies wichtig, da eine SEM nicht zu jedem Zeitpunkt angepasst werden kann. Der Anpassungszeitpunkt hängt zum einen von der Software-Engineering-Methode und der Komplexität des Projektes ab. Zum anderen setzen sich alle Beteiligten vor Beginn des Projektes zusammen und bestimmen gemeinsam einen Anpassungszeitpunkt, welcher im Laufe des Projektes ebenfalls überdacht und angepasst werden kann.

In einem Scrum-Projekt bietet sich als erstes an, sehr komplexe Anpassungen jeweils zu Beginn eines neuen Sprints durchzuführen. In einem normalen Sprint wäre für die Planung einer entsprechenden Anpassung auf Typ-Ebene nur wenig bis gar keine Zeit vorhanden, da solche Anpassungen nur indirekt vorgesehen sind. Von daher würde es Sinn machen, die Software-Engineering-Methode zu diesem Zeitpunkt anzupassen, vor allem wenn kurze Sprints an der Tagesordnung sind. Besitzt der Sprint die Länge von 4 Wochen, so können ein bis mehrere Anpassungszeitpunkte zwischen den einzelnen Sprints bestimmt werden. Der Abstand von Anpassungszeitpunkten von weniger als einer Woche würde allerdings kaum Sinn machen, da die Software-Engineering-Methode zu oft geändert und Aufgaben unter Umständen nicht beendet werden würden.

Für dieses Beispiel wird der Anpassungszeitpunkt von zunächst alle 2 Wochen festgelegt, zur Halbzeit und vor Beginn eines neuen Sprints. Durch die Priorisierung der einzelnen Ziele kann es wie beschrieben dennoch vorkommen, dass einige Anpassungen sofort vorgenommen werden müssen, da der Erfolg sonst gefährdet ist. Darunter würde beispielsweise die erste Anpassung bei Krankheit zählen.

Schritt 1 + 2: Definition und Priorisierung der Ziele

Sind das Projektteam und die Software-Engineering-Methode bestimmt, so können der Methoden-Engineer und die entsprechenden Beteiligten mit dem ersten Schritt des MAPE-K4SEM beginnen. Wie in Kapitel 5 beschrieben beginnt das Team damit, die Ziele der Software-Engineering-Methode zu definieren. Hier im Beispiel wird nur ein Auszug der Ziele beschrieben. Es werden die Ziele definiert, die für die späteren Anpassungen nötig sind und es werden ein paar weitere mögliche Beispiele gegeben. Normalerweise wird den verschiedenen Zielen noch zugeordnet, zu welchem Zeitpunkt sie „aktiv“ sind. Hier im Beispiel sind alle Ziele immer aktiv und werden kontinuierlich überprüft. Ausnahme sind die Ziele, welche mit einem Meeting in Verbindung stehen, beispielsweise die beiden folgenden Ziele bzgl. der Original-Methode oder die späteren Ziele der angepassten Methode (ZAM2 und ZAM3). Diese sind nur zur Zeit des jeweiligen Meetings aktiv.

Zu Beginn wird wie in Abschnitt 5.1 beschrieben die angepasste Software-Engineering-Methode Scrum mit ihren Eigenschaften betrachtet. Ziele, die sich von den Eigenschaften und Regeln bei Scrum ableiten lassen sind beispielsweise:

- ZOM1: Alle Meetings werden eingehalten und durchgeführt.
- ZOM2: Die Time-Boxes von Meetings werden eingehalten.

ZOM steht dabei für „Ziel Original-Methode“.

In diesem Projekt wird ein angepasstes Scrum verwendet und ein Lenkungskreis anstatt des ursprünglichen Product Owners eingesetzt. Ziele, die sich daraus und aus dem Original-Scrum ergeben sind beispielsweise:

- ZAM1: Der Lenkungskreis priorisiert die Anforderungen im Product Backlog.
- ZAM2: Der Lenkungskreis schlägt die Aufgaben für einen neuen Sprint und somit für das Sprint Backlog vor.
- ZAM3: Das Team sucht im Planning-Meeting die Aufgaben aus und legt das endgültige Sprint Backlog fest.
- ZAM4: Der Scrum Master übernimmt keine Entwicklungsaufgaben.

ZAM steht hier für „Ziel angepasste Methode“:

Im Beispiel sind gerade die Ziele wichtig, welche für die hier durchgeführten Anpassungen relevant sind. Für die ersten Beispielanpassungen, dass eine Ressource ausfällt (jemand wird krank) und dass die Teamgröße in Scrum eingehalten wird, lauten die Ziele:

- ZBA1: Ressourcen optimal nutzen.
- ZBA2: Ressourcenveränderungen abfangen.
- ZBA3: Teamgröße in Scrum einhalten.

ZBA steht dabei für „Ziel Beispielanpassung.

In der nächsten Anpassung geht es, wie im Prinzip in jedem Projekt gewünscht, darum, dass die Kundenzufriedenheit im Projekt hoch sein und die Software möglichst fehlerfrei sein soll. Gerade in Scrum ist es gewollt, dass der Kunde frühzeitig Feedback gibt. Jedes ausgelieferte Produkt-Inkrement ist prinzipiell einsatzfähig und kann vom Kunden eingesetzt werden. Der Kunde kann des weiteren Feedback und Fehlermeldungen einstellen. Beides wird anschließend in den nächsten Sprints umgesetzt bzw. die Fehler werden behoben. Ein Sub-Ziel der Kundenzufriedenheit könnte dann heißen: Es dürfen nur eine bestimmte Anzahl von negativen Meldungen oder Fehlermeldungen vom Kunden vorhanden sein. Außerdem sollen nicht-

funktionale Anforderungen wie Performanz und Usability ebenfalls mit betrachtet und eingehalten werden. Abgeleitete Ziele heißen dann:

- ZBA4: Die Kundenzufriedenheit soll im Projekt hoch sein.
    - Es soll nur eine bestimmte Anzahl an Fehlermeldungen/negativen Feedbackmeldungen vom Kunden vorliegen.
- ZBA5: Es dürfen keine kritischen Fehler (Blocker) vorliegen und nur eine bestimmte Anzahl an nicht-funktionale Beschwerden.

Um die Kundenzufriedenheit und eine möglichst fehlerfreie Software einhalten zu können ist es ebenfalls wichtig, dass die Software ausreichend und strukturiert getestet wird. In Scrum wird zwar erwähnt, dass Testen sehr wichtig ist, aber es wird nicht vorgegeben, wie das Testen stattfinden soll. Ferner wird im Original-Scrum erwähnt, dass es keine expliziten Tester geben soll. Wichtig ist aber, dass zumindest alle Anforderungen getestet werden. Als besonders wichtig werden in Scrum die Unit-, Integrations- und User Acceptance-Tests angesehen. Daraus können sich beispielsweise als Ziele ergeben:

- ZBA6: Das Testen wird strukturiert durchgeführt.
- ZBA7: Es gibt für die Software verschiedene Testarten und -stufen, insbesondere Unit- und Integrationstests.

Sind wichtige Ziele für die Software-Engineering-Methode hergeleitet, so wird zusätzlich der Kontext des Projektes und des Unternehmens mit betrachtet. Dabei kann es für das Unternehmen wichtig sein, dass alle Normen und Standards, beispielsweise im Testen eingehalten werden. Ebenso kann es vorkommen, dass Anweisungen vom Projektleiter oder von der Unternehmensleitung etc. eingehalten und umgesetzt werden müssen. Als Ziele, ZK steht dabei für „Ziel Kontext“, ließen sich diese folgendermaßen formulieren:

- ZK1: Die Standards für das Software-Testen sollen eingehalten werden. Es soll immer der aktuellste Standard verwendet werden.
- ZK2: Die Anweisungen vom Unternehmen sollen eingehalten werden.
- ZK3: Anweisungen vom Projektleiter sind einzuhalten.

In einem weiteren Schritt, der teilweise direkt mit der Zieldefinition erfolgen kann, werden die Rahmenbedingungen oder mögliche Abweichungen für die Ziele festgelegt. Dies ist wichtig für die weiteren Schritte und zur Ableitung der Analyseregeln. Für die Ziele der Original-SEM werden nur für das dritte Ziel die Time-Boxen der verschiedenen Meetings festgelegt. Die anderen beiden Ziele sind eindeutig und brauchen keine weiteren Rahmenbedingungen. ZOM2 würde dann lauten:

- ZOM2: Die Time-Boxes von Meetings werden eingehalten. Die Time-Boxes für einzelnen Meetings sind:
    - Planning-Meeting: 1 Tag = 8 Stunden
    - Daily Scrum am Dienstag und Donnerstag: 30 Minuten
    - Review-Meeting: 4 Stunden
    - Retrospektive: 1 Stunde
    - Gesamter Sprint: 4 Wochen

Hauptsächlich sollen hier für die Beispiele die Rahmenbedingungen festgelegt werden. Für das erste Beispiel ist es wichtig, dass die Teamgröße vom Original-Scrum eingehalten wird. Dabei soll jede Ressource im Team, hier die Ressource Person entsprechend genutzt und Veränderungen sollen gemeldet werden.

- ZBA1: Die Ressource Person soll optimal genutzt werden.
- ZBA2: Jede Veränderung der Ressource Person soll überprüft werden. Bei einer Veränderung müssen die Aufgaben der Person, die im aktuellen Sprint zu erledigen sind, weiterhin bearbeitet und abgeschlossen werden können.
- ZBA3: Die Teamgröße von Scrum soll eingehalten werden. Ein Team soll nicht größer als neun aber auch nicht kleiner als drei Personen sein.

Für die Ziele ZBA4 und ZBA5 müssen die Anzahl der Fehlermeldungen und der Feedbackmeldungen etc. als Rahmenbedingungen bestimmt werden. Für das Beispiel würden sie folgendermaßen lauten:

- ZBA4: Die Kundenzufriedenheit im Projekt soll hoch sein. Es sollen maximal 10 Fehlermeldungen/negative Feedbackmeldungen vom Kunden vorliegen.
- ZBA5: Es dürfen keine besonders kritischen Fehler, sogenannte Blocker, und maximal 3 kritische Fehler vorliegen. Ebenfalls dürfen maximal nur 2 nicht-funktionale Beschwerden vorliegen. Die Anzahl der Schönheitsfehler ist nicht begrenzt.

Für die letzten zwei Beispielziele können als weitere Rahmenbedingungen festgelegt werden:

- ZBA6: Das Testen wird strukturiert durchgeführt, jede Person im Team soll seine eigenen Aufgaben testen.
- ZBA7: Es gibt für die Software Unit-Tests und Integrationstests. Jede Aufgabe besitzt einen Unit-Test und es wird pro Sprint ein Integrationstest durchgeführt. Weitere Teststufen und Testarten werden nur durchgeführt, wenn nötig.

Für die Kontext-Ziele wird keine Definition von weiteren Rahmenbedingungen benötigt. Nachdem die Ziele bestimmt sind, können diese nun von den Beteiligten entsprechend priorisiert werden. Hier im Beispiel wird die Einstufung nach dem Fehlerprinzip und eine Abstufung von 1 bis 5 gewählt, wobei 1 für Blocker steht, das heißt ohne dieses Ziel geht es nicht und eine Anpassung muss sofort umgesetzt werden. Hingegen steht 5 für die niedrigste Stufe, das heißt die Anpassung gefährdet den Erfolg kaum oder nur sehr wenig und muss nicht direkt umgesetzt werden. Auch wenn nur fünf Abstufungen vorhanden sind und somit verschiedene Ziele dieselbe Priorität haben können, wird hier davon ausgegangen, dass nur äußerst selten der Fall vorkommt, das zwei Ziele mit derselben Priorität gleichzeitig getriggert werden.

Da für den weiteren Verlauf in erster Linie die Ziele der Beispiele ZBA1 bis ZBA7 sowie von ZK1 wichtig sind, werden nur diese Ziele priorisiert und im weiteren Verlauf für die anderen Schritte verwendet.

Vor allem bei der Änderung einer Ressource beispielsweise durch Krankheit, Verlassen des Unternehmens oder ähnlichem, ist es wichtig, dass sofort darauf reagiert wird, damit mögliche Engpässe oder nicht erledigte Aufgaben abgefangen werden. Deswegen bekommt dieses Ziel die Priorität 1. Damit hängt zusammen, dass die Ressourcen optimal genutzt werden, es ist aber nicht ganz so essentiell wie ZBA2 und bekommt von daher die Priorität 2. Dass die Teamgröße eingehalten wird, ist dem Projektteam wichtig und wird hier mit einer mittleren Priorität (3) bewertet.

Die Kundenzufriedenheit in ZBA4 ist ebenfalls sehr wichtig, aber kein Blocker und bekommt hier die Priorität 2 ebenso wie das Ziel ZBA5. Das Testen ist beim Team zwar wichtig, wird aber eher als eine mittlere Kategorie, jedoch nicht als nicht sehr wichtig angesehen. Von daher bekommt ZBA6 die Priorität 3 und ZBA7 die Priorität 4. Da die Einhaltung von Normen und Standards im Beispiel-Unternehmen als sehr wichtig angesehen wird, wird hier die Priorität 1 vergeben.

Als eine niedrige Priorität (5) könnte beispielsweise das Einhalten der Time-Boxes angesehen werden. Hingegen würde das Einhalten von Anweisungen des Projektleiters oder der Unternehmensleitung in die Kategorie 2 oder sogar 1 fallen. Für die sieben Ziele der Beispiele und des Kontext-Ziels ergibt sich somit folgende Priorisierung:

- ZBA 1: Prio 2
- ZBA 2: Prio 1 (wird immer angepasst)
- ZBA 3: Prio 3
- ZBA 4: Prio 2

- ZBA 5: Prio 2
- ZBA 6: Prio 3
- ZBA 7: Prio 4
- ZK 1: Prio 1

Sind die beiden ersten Schritte durchgeführt und alle Ziele sowie ihre Priorisierung bestimmt, können nun wie in den Abschnitten 5.3 bis 5.7 beschrieben die weiteren nötigen Informationen für die MAPE-K-Feedbackschleife hergeleitet werden.

Schritt 3 + 4: Ableitung der Analyseregeln und Planungsmöglichkeiten

Für die Ableitung der Analyseregeln und Planungsmöglichkeiten sind die Schritte 3 und 4 eng miteinander verknüpft. Wie in Abschnitt 5.4 beschrieben besteht eine Analyseregel aus einem Wenn- und einem Dann-Teil. Der Dann-Teil beschreibt dabei die Aktion, welche je nach Analyse-Ergebnis ausgeführt werden soll und beinhaltet dabei je nach Regel Varianten der Planungsmöglichkeiten

Zunächst wird aber der Wenn-Teil abgeleitet, also was überprüft werden soll, damit das Ziel weiter eingehalten werden kann. Für das Ziel ZBA1 wird überprüft, ob alle Ressourcen, in diesem Fall Personen, genutzt werden, das heißt ob sie in einem aktuellen Projekt sind. Sind alle Personen in einem Projekt und besitzen Aufgaben, passiert nichts. Ansonsten werden sie einem Projekt oder ihnen werden Aufgaben zugeordnet. Nach Abschnitt 5.4 würden die beiden Regeln dann lauten:

a) RZBA1: Wenn Person Y 0 Aufgaben zugeordnet, dann Planung (Ordne Aufgaben nach Fähigkeiten zu)
b) RZBA1: Wenn Person Y Aufgaben zugeordnet, dann Schritt 7

Bei ZBA2 hingegen muss überprüft werden, ob eine Änderung an den Ressourcen stattgefunden hat, z.B. dass eine Person krank geworden ist oder das Unternehmen verlassen hat. Um dies zu ermitteln, könnte jeder Person ein Status im Team zugeordnet sein beispielsweise „aktiv“, „Urlaub“, „krank“, „gelöscht“. Ändert sich dieser Status, sind mögliche Planungsschritte, dass entweder die Aufgaben je nach Kapazität vom Team übernommen werden oder dass eine neue Person mit den entsprechenden Fähigkeiten eingesetzt wird.

Das Übernehmen der Aufgaben oder Einsetzen einer Person sollte je nach Ausfall der zu ersetzenden Person geplant werden. Ist beispielsweise eine Person nur 3 Tage krank, braucht dafür typischerweise keine neue Person eingesetzt werden (Variante 1). Ist eine Person 3 Wochen im Urlaub, müssen die Aufgaben entweder gut vom Team geplant oder von einer anderen

Person übernommen werden (Variante 2). Fällt eine Person vollständig weg, muss diese ersetzt werden (Variante 3). Daraus würden sich folgende Regeln ergeben:

a) RZBA2: Wenn Statusänderung == krank & Fehltage <= 5 dann Planungsmöglichkeit Variante 1
b) RZBA2: Wenn Statusänderung == krank oder Urlaub & Fehltage >5 dann Planungsmöglichkeit Variante 2
c) RZBA2: Wenn Statusänderung == Person gelöscht dann Planungsmöglichkeit Variante 3

Wenn eine Person nicht anwesend ist, wären ihr somit keine Aufgaben zugeordnet, was in diesem Fall korrekt ist. Von daher muss die Regel aus dem vorherigen Fall angepasst werden und würde lauten:

- RZBA12: Wenn Person Y 0 Aufgaben zugeordnet & Status == aktiv, dann Planung (Ordne Aufgaben nach Fähigkeiten zu)

In allen drei Varianten ist es wichtig, dass in der Planungsphase die Aufgaben der abwesenden Person bekannt sind und aus der Wissensbasis ausgelesen werden müssen. Ferner müssen in der Planungsphase die Skills der einzelnen Personen bekannt sein, ebenso wie deren Auslastung. Dies ist wichtig, damit der richtigen Person die neuen Aufgaben zugewiesen werden können.

In Schritt 4 sollen, soweit möglich, bereits im Vorfeld Konfliktmöglichkeiten zu den Zielen analysiert und gegebenenfalls Lösungen aufgezeigt werden. Hier im Beispiel muss bei der Verteilung der Aufgaben darauf geachtet werden, dass es beispielsweise keinen Konflikt mit Ziel ZOM1 gibt, dass der Scrum Master keine Entwicklungsaufgaben übernehmen darf. Fällt ein Mitglied aus dem Team aus, so darf der Scrum Master dessen Aufgaben nach dieser Regel nicht übernehmen. Diese Konflikt-Regel wird in der Wissensbasis für die Plan-Phase gespeichert.

ZBA3 spiegelt Teamgröße in Scrum wider. Die Analyseregeln und Planungsmöglichkeiten wurden dabei bereits in den Abschnitten 5.4. bzw. 5.5. beschrieben und lauten:

a) RZBA3: Wenn Teamgröße > 9, dann Planungsmöglichkeit Variante 1 („Team zu groß“, aktuelle Teamgröße x)
b) RZBA3: Wenn Teamgröße < 5 dann, Planungsmöglichkeit Variante 2 („Team zu klein, aktuelle Teamgröße y)
c) RZBA3: Wenn Teamgröße 3 <= Teamgröße <= 9, dann Schritt 7

Wie bereits in Abschnitt 5.4 beschrieben besagt die Planungsmöglichkeit bei einem zu großen Team, dass ein Team-Splitting durchgeführt und ein neues Meeting Scrum of Scrums eingeführt werden soll. Da in diesem Projekt das Daily Scrum angepasst wurde und nur an 2 Tagen stattfindet, wird hier die Planungsmöglichkeit ebenfalls entsprechend angepasst. Das Scrum of Scrums soll ebenfalls nur an zwei Tagen stattfinden und zwar um einen Tag verschoben zu den Daily Scrums, um eine gute Kommunikation zwischen den Teams zu ermöglichen. Diese Anweisung spiegelt sich nachher in Schritt 6 wider.

Die nächsten Ziele für die komplexeren Beispiel ZBA4 bis ZBA7 sind alle relativ ähnlich und hängen miteinander zusammen, da sie alle den Bereich Kundenzufriedenheit bzw. das Testen betreffen. Gerade die Kundenzufriedenheit ist in Projekten besonders wichtig. Ein Vorteil bei Scrum ist, dass die Möglichkeit frühzeitig Feedback zu bekommen gegeben ist. Von daher ist es hier im Projekt dem Kunden möglich, Feedback über Fehlermeldungen oder allgemeine Meldungen zu geben, z.B. als Eintrag zu einem Product Backlog. Bei Scrum soll nach jedem Sprint ein fertiges Produkt-Inkrement ausgeliefert werden. Je nach Unternehmen wird dieses Inkrement schon eingesetzt oder nicht. Anhand dieser Meldungen ist es nach den Zielen ZBA4 und ZBA5 möglich, den entsprechenden Wenn-Teil festzulegen.

Um die Kundenzufriedenheit mit Hilfe einer Änderung der Software-Engineering-Methode zu beheben, kann es verschiedene Möglichkeiten geben. Zum einen kann ein Methoden-Engineer aus seiner Erfahrung schöpfen, was in einer gegebenen Situation als Anpassung möglich ist, um die Situation zu verbessern (Variante 1). Zum anderen ist es in diesem Ansatz möglich, die Planungsmöglichkeiten „außen vor“ zu lassen und die entsprechende Anpassung je nach den angegebenen Werten erst in der Plan-Phase zu planen (Variante 2).

Hier im Beispiel weiß der Methoden-Engineer, dass in Scrum der Fokus nicht unbedingt auf den nicht-funktionalen Anforderungen sondern mehr auf den Funktionen liegt. Von daher tauchen diese Anforderungen nicht unbedingt im Product Backlog und somit im Sprint Backlog auf, weil dem Kunden die Wichtigkeit nicht bewusst ist [EG09]. Tauchen nun vermehrt negative Meldungen (Regel ZBA4) nach der Auslieferung in Verbindung mit nicht-funktionalen Anforderungen (Regel ZBA5) auf, so muss sichergestellt werden, dass die Betrachtung von nicht-funktionalen Anforderungen verstärkt wird. Ferner kann es sinnvoll sein, den Kunden zu Anfang mit einzubinden, damit die Fehler nicht erst im Produktionsbetrieb auftauchen. Dies könnte beispielsweise darüber erfolgen, dass der Kunde am Ende für einen User Acceptance Test (UAT) direkt mit eingebunden wird.

Die Regeln für die Ziele ZBA4 und ZBA5 werden teilweise zusammengefasst. Liegt ein Blocker vor, muss dieser zwar behoben, aber die Software-Engineering-Methode nicht sofort angepasst werden. Es muss also nur auf der Instanz-Ebene eine Aufgabe hinzugefügt werden, damit der Blocker sofort angepasst wird. Summieren sich die Fehler allerdings oder treten ähnliche Fehler immer wieder auf, so kann dies auf eine Anpassung hindeuten. Von daher kann in einer Regel für RZBA5 die Zeit bzw. Historie der Meldungen eine Rolle spielen. Am Ende ergeben sich daraus die Regeln:

a) RZBA4: Wenn Anzahl Fehlermeldungen/negatives Kundenfeedback < 10, dann Monitor
b) RZBA45: Wenn Anzahl Fehlermeldungen/negatives Kundenfeedback >10 & Meldungen nicht-funktionale Eigenschaften >= 2, dann Planungsmöglichkeit Variante 1 (Kunden für UAT einbinden)
c) RZBA45 Wenn Anzahl Fehlermeldungen/ negatives Kundenfeedback > 10 & Meldungen nicht-funktionale Eigenschaften < 2, dann Plan (Anzahl Blocker, kritischer Fehler, Schönheitsfehler)
d) RZBA5: Wenn Anzahl Blocker > 0 oder kritische Fehler > 3, dann Planungsmöglichkeit (Fehler sofort beheben)

Die Ziele ZBA6 und ZBA7 beziehen sich nicht auf die Kundenzufriedenheit, sondern auf das Testen selbst. Da Fehlermeldungen und Testen allerdings sehr eng miteinander verbunden sind, ist zu überprüfen, ob hier Kombinationsmöglichkeiten vorliegen. Für das Ziel ZBA6 muss zunächst überprüft werden, ob jeder Entwickler seine eigene Aufgabe testet, z.B. über Unit-Tests oder einen User Acceptance Test. Es sollte somit pro Aufgabe jeweils ein Test zur Verfügung stehen, der vom Entwickler durchgeführt wird. Ist dies nicht der Fall, wird der Entwickler vom System angewiesen, einen Testfall zu erstellen.

Für das Ziel ZBA7 wird zunächst überprüft, ob jede Aufgabe einen Unit-Test besitzt, was sich mit dem Ziel ZBA6 kombinieren lässt. Ferner wird analysiert, ob im Sprint ein Integrationstest geplant ist. Zusätzlich lässt sich das Ziel ZBA7 hier sowohl mit dem Ziel ZBA5, aber auch mit dem Ziel ZBA4 kombinieren, um zu überprüfen, ob eine weitere Teststufe oder Testart notwendig ist. Dies können beispielsweise Inkrement- oder Systemtests sein. Gibt es mehr Fehler als den vorgegebenen Blocker oder kritischen Fehler, die regelmäßig wiederkehren, sollten mehr Tests durchgeführt werden. Welche Art von Tests notwendig ist, müsste genauer in der Plan-Phase geplant werden (Variante 1). Ist das Kundenfeedback regelmäßig negativ, sollte ein allgemeiner Inkrementtest durchgeführt werden (Variante 2).

Zusätzlich kann sich bei den Planungsmöglichkeiten ergeben, um ZBA6 und ZBA7 zu erfüllen und das Testen strukturierter durchzuführen sowie mehr Teststufen mit einzubinden, dass ein bis zwei explizite Personen als Tester und/oder Test-Designer nötig sind. Auch wenn Scrum diese definitiv nicht vorsieht, zeigt die Erfahrung, dass der Einsatz sehr sinnvoll ist und in Scrum mit eingebunden werden kann [GG12]. Deswegen soll dies im Beispiel ebenfalls mit in die Software-Engineering-Methode bei Bedarf eingebunden werden.

Als Regeln würden sich ergeben:

a) RZAB67: Wenn jede Aufgabe Unit-Test vom Entwickler besitzt, dann Monitor
b) RZAB67: Wenn Aufgabe keinen Unit-Test besitzt, dann Plan (Benachrichtige Entwickler Erstellung Tesfall)
c) RZAB7: Wenn Sprint keinen Integrationstest besitzt, dann Plan (plane Integrationstest)
d) RZBAkombiniert: Wenn Blocker >= 0 oder kritische Fehler >= 3 & Fehlerhistorie > 2, dann Planungsmöglichkeit Variante 1 (zusätzliche Tests, zusätzliche Personen)
e) RZABkombiniert: Wenn Kunden-Feedback > 10 & Anzahl Kundenfehler-Historie > 2, dann Planungsmöglichkeit Variante 2 (Inkrementtest)

Für das Kontext-Ziel ZK1 muss analysiert werden, ob der aktuelle Test-Standard verwendet wird oder nicht. Dafür wird die Versionsnummer verglichen und ist diese höher (also neuer) als die aktuelle Versionsnummer, so soll die Software-Engineering-Methode an den neuen Standard angepasst werden. Da es nicht möglich ist vorherzusehen, welche Änderung ein neuer Standard mit sich bringt, muss die Plan-Phase ohne vorherige Planungsmöglichkeiten gestartet werden. In der Plan-Phase muss dafür der neue Standard ausgelesen und hinsichtlich der Änderungen mit dem alten Standard verglichen werden. Die Regel würde also lauten:

- RZK1: Wenn Versionsnummer Test-Standard > als Versionsnummer aktueller Test-Standard, Dann Plan (alter Test-Standard, neuer Test-Standard)

Die Analyseregeln und Planungsmöglichkeiten für die verschiedenen Ziele sind nun abgeleitet, ebenso mögliche Konflikt- und Kombinationsmöglichkeiten bedacht. Somit kann zu Schritt 5 und Schritt 6 übergegangen werden. In diesen Schritten werden die Metriken sowie die konkreten Ausführungsregeln und Benachrichtigungen abgeleitet.

Schritt 5: Ableitung der Metriken

Die Analyseregel und die Planungsmöglichkeiten sind bestimmt. Nun können zunächst die Metriken für die einzelnen Regeln und im nächsten Schritt die konkreten Ausführungsregeln und Benachrichtigungen abgeleitet werden. Sind in diesem Schritt die einzelnen Metriken bestimmt, werden für die spätere Monitor-Phase in einem Subschritt die Aufbereitungsregeln bestimmt. Dafür wird wie in Abschnitt 5.6 beschrieben die Herleitung praktisch rückwärts vorgenommen, um den entsprechenden Wert für die Analyse zu erhalten.

Für die Regel RZBA1 (a und b) muss gemessen werden, wie viele Aufgaben einer Person zugeordnet sind, die Metrik wäre also #Aufgabe/ Person Y. Für die beiden Regeln entspricht der aufbereitete Wert dem gemessenen Wert.

Für die Regeln RZBA2 (a,b,c) ist es nötig, den aktuellen Status einer Person zu messen. Jeder Person im Unternehmen ist dazu in beispielsweise einer „Personenliste“ der Status zugeordnet, ob diese krank, im Urlaub, aktiv oder gelöscht ist (ehemalige Mitarbeiter). Dieser Status wird entsprechend ausgelesen. Die Metrik würde dann lauten Status/ Person Y. Zusätzlich müssen für die Regel RZBA2 (a und b) die Fehltage ermittelt werden. Ist die Person aktiv, sind die Fehltage = 0. Die Metrik würde hier lauten #Fehltage/ Person Y. Die Aufbereitung für die Regel RZBA2 (c) ist die Metrik bezüglich des Status. Für die anderen beiden Regeln von RZBA2 werden die beiden Metriken zusammengefasst, da beide Werte für die Analyse benötigt werden. Überliefert wird also die Kombination (Status & Fehltage)/Person Y.

Für die Regel RZBA12 greifen die beiden Metriken Status/ Person Y und #Aufgaben/ Person Y. Die Monitor-Phase würde also am Ende liefern (Status & #Aufgaben)/Person Y.

Die Metriken für die Ziele RZBA3 (a – c) wurden bereits in Abschnitt 5.6 im ersten Beispiel beschrieben. Hier ist die Metrik #Personen/ Team. Die Aufbereitung für die Teamgröße lautet dann entsprechend Teamgröße = #Personen/ Team.

Für die nächsten Regeln RZBA4, RZBA5 und RZBA45 wird das Feedback des Kunden und seine Fehlermeldungen, genauer die Art der Fehlermeldung (nicht-funktional, Blocker, kritisch, Schönheitsfehler), ermittelt. Dafür wird im Product oder Sprint Backlog oder auch in einem Fehlermanagement-System zunächst vermerkt, ob eine Meldung vom Kunden oder vom Team ist. Zusätzlich wird für eine Fehlermeldung eingestellt, ob diese funktional oder nicht-funktional ist sowie die Kritikalität des Fehlers. Diese Werte ergeben dann die Metriken und später die entsprechenden Sensoren.

Im Detail lauten die Metriken jeweils:

- RZAB4 (a): #Fehlermeldungen/ negatives Feedback/ Kunde (damit ist das gesamte Feedback über den Kunden gemeint, nicht eine einzelne Person vom Kunden)
- RZBA45 (b und c): #Fehlermeldungen/ negatives Feedback/ Kunde und #Fehlermeldungen nicht-funktionale Eigenschaften
- RZBA5 (d): #Blocker, #kritische Fehler, #Schönheitsfehler

Für die Aufbereitung ist in Fall a) der gemessene Wert gleich dem aufbereiteten Wert. Für die Fälle b) und c) werden die Werte wieder zusammen überliefert als #Fehlermeldung/ negatives Feedback pro Kunde + #Fehlermeldung nicht-funktionale Eigenschaften. Für den letzten Fall wird für die Analyse nur das Tuple (#Blocker, #kritische Fehler) übergeben, da die Schönheitsfehler nicht relevant sind.

Die Ziele RZBA6 und RZBA7 beschäftigen sich mit den Testarten, welche ausgelesen werden müssen. Für RZBA67 muss für jede Aufgabe ausgelesen werden, ob überhaupt ein Unit-Test vorliegt und ob dieser vom entsprechenden Entwickler ist. Es muss also in der Aufgabe stehen, wem die Aufgabe zugeordnet ist (Verantwortlicher Aufgabe) und wer den Unit-Test erstellt hat (Verantwortlicher Unit-Test). Die einzelnen Metriken würden lauten #Unit-Test/ Aufgabe Y; Verantwortlicher Aufgabe Y und Verantwortlicher Unit-Test Aufgabe Y. Für die Aufbereitung würde jeweils das Triple dieser drei Werte zurückgegeben (#Unit-Test/ Aufgabe Y & Verantwortlicher Aufgabe Y & Verantwortlicher Unit-Test Aufgabe Y). Für RZBA7 muss der Wert #Integrationstest/ Sprint ausgelesen werden. Dieser Wert ist auch der Wert für die Aufbereitung.

Für die beiden kombinierten Ziele muss zusätzlich die Fehlerhistorie der jeweiligen Fehler ausgelesen werden. Jeder Fehler wird in der Historie gespeichert. Tritt derselbe Fehler oder ein sehr ähnlicher Fehler erneut auf, so wird die Historie um +1 hochgezählt. Dies muss zum Teil von den Teammitgliedern manuell erfolgen, insbesondere wenn die Fehler nur sehr ähnlich sind. Dasselbe gilt für die Fehlerhistorie von Kundenmeldungen. Wird beispielsweise ein Performanz-Fehler vom Kunden gemeldet, so wird dieser in der Historie abgespeichert. Wird ein Fehler derselben Art vom Kunden gemeldet, so wird die Historie um +1 nach oben gezählt. Dasselbe gilt auch für die funktionalen Fehler.

Daraus ergeben sich an weiteren zu messenden Werten die #Fehler X in Historie und Anzahl Fehler Y in Kundenhistorie. Für die Aufbereitung würde in Fall d) das Triple (#Blocker, #kritische Fehler, #Fehler X in Historie)

zurückgeliefert. In Fall e) wäre es hingegen das Tuple (#Fehlermeldungen/ negatives Feedback/ Kunde, #Fehler Y Kundenhistorie).

Die Metrik für das Kontext-Ziel RZK1 ist die Versionsnummer des Test-Standards. Dieser muss nur von einer Webseite oder aus einem manuellen Eintrag in einer Liste ausgelesen werden. In der Aufbereitung wird diese Versionsnummer auf „aktueller Test-Standard = Versionsnummer“ gesetzt.

Um die Werte für die einzelnen Metriken zu messen, müssen diese entsprechend in die Sensoren umgewandelt werden. Wie im vorherigen Beispiel gezeigt, muss die Versionsnummer des Test-Standards von der entsprechenden Seite oder aus einer entsprechenden Liste ausgelesen werden. In diesem Beispiel wird einmal am Tag die Versionsnummer auf der Webseite geprüft.

Für die Historien müssen diese entsprechend jeweils in der Wissensbasis vorliegen und werden manuell vom Scrum Master befüllt. Die Fehlerhistorien werden jeweils hochgezählt. Für die Kritikalität der Fehler ist eine entsprechende Einstellung im Fehlermanagementsystem vorhanden, die ausgelesen werden kann. Ebenso ist ein Eintrag möglich, ob es sich um einen funktionalen oder nicht-funktionalen Fehler handelt. Ferner gibt es einen Verantwortlichen für die Fehlermeldung, für den zusätzlich vermerkt ist, ob dieser ein Kunde oder ein Teammitglied ist. Die Fehlermeldungen, die pro Aufgabe eingestellt werden, sind in diesem Beispiel gleichzeitig das Feedback des Kunden.

Im Sprint Backlog sind die Aufgaben vorhanden und für jede Aufgabe ist ein Verantwortlicher eingetragen, der entsprechend ausgelesen werden kann. Diesen Aufgaben und dem ganzen Sprints können Tests zugeordnet werden. Die Tests erhalten zusätzlich den Eintrag, welche Art sie sind, z.B. Unit-Test, Integrationstest, Performanz-Test usw. Außerdem erhalten die Tests jeweils einen Verantwortlichen.

Wie schon in Abschnitt 6.2.1. erläutert sind für das Ziel RZBA3 die Teammitglieder in einer Liste eingetragen. Dies kann einerseits eine Liste mit den Namen der Personen, welche in einem Team sind, sein oder es gibt im Unternehmen eine vollständige Personenliste, in der es für jede Person einen Eintrag gibt, welchem Team sie zugeteilt ist.

Für die Regeln RZBA1 und RZBA2 gibt es wie eingangs beschrieben jeweils einen Eintrag zum Status der Person sowie einen Eintrag zu ihren Fehltagen. Somit sind die Sensoren für die Beispielziele entsprechend vorhanden.

Schritt 6: Ableitung der Ausführungsregeln und Benachrichtigungen

Auf der einen Seite sind nun die Metriken erstellt, um die Werte für die Analyse zu bekommen. Auf der anderen Seite müssen noch die konkreten Ausführungsregeln und Benachrichtigungen abgeleitet werden, um in der Execute-Phase die Änderungen vornehmen zu können. Die Ableitungen aus den Planungsmöglichkeiten ergeben die in Abschnitt 5.7 beschriebenen Regeln zur Anpassung des Modells der Software-Engineering-Methode. Dies sind die Anpassung des Modells, Änderung der Verantwortlichkeiten, also der Stakeholder und die Benachrichtigung an die entsprechenden Beteiligten. In diesem Abschnitt werden nun die Ausführungsregeln und Benachrichtigungen beschrieben, die im nächsten Abschnitt für die verschiedenen Anpassungen benötigt werden.

Für den einfachen Fall nach RZBA1 bzw. RZBA12, dass einer Person neue Aufgaben zugeordnet werden, die entsprechend ihrer Fähigkeiten in der Planung ausgewählt wurden, würden die Ausführungsregeln und Benachrichtigung lauten:

1. Anpassung Modell (keine Anpassung), da die Aufgaben bereits bestehen, nur noch keinen Verantwortlichen besitzen, der Status des Stakeholders wird geändert
2. Änderung Stakeholder (Person Y, Aufgabe Y, hinzufügen); (Person Y, Aufgabe Z, hinzufügen);
3. Benachrichtigung Beteiligte (Person Y, neue Aufgaben);

Dieser einfache Fall erfordert zwar keine Anpassung der Software-Engineering-Methode aber, es ist zu sehen, dass hier mit dem Ansatz ebenfalls Änderungen auf der Instanz-Ebene ausgeführt werden können. Ähnlich gelagert ist der Fall der Anpassung der Regel RZBA2. Die Regel, dass die Aufgaben im Team verteilt werden, wenn eine Person für ein paar Tage ausfällt, gleicht denen im vorherigen Fall. Der Unterschied ist an dieser Stelle, dass unter 2. nur eine Person einer Aufgabe hinzugefügt wird, während gleichzeitig die aktuelle Person gelöscht wird. Da der Status der Person nicht aktiv ist, reicht es, nur den Beteiligten mit den neuen Aufgaben zu benachrichtigen.

Auch wenn eine Person das Unternehmen verlässt, also gelöscht ist, sehen die Regeln im Prinzip gleich aus. Denn hier werden einer neuen Person alle Aufgaben der vorherigen Person zugeordnet. In allen drei Fällen ist allerdings ein Konflikt zu beachten: Fällt ein Teammitglied aus, dürfen die Aufgaben nicht dem Scrum Master zugeordnet werden. Fällt der Scrum Master aus, dürfen seine Aufgaben keinem Teammitglied zugeordnet werden. Beide Fälle würden gegen Ziel ZOM1 verstoßen.

Trifft die Regel RZBA3 mit dem Fall das das Team zu groß ist zu, so erfolgt die Anpassung wie in Schritt 5.6. beschrieben nach den folgenden Regeln und Benachrichtigungen:

1. Anpassung Model (Hinzufügen Team 2, Ändern Team 1 (Anzahl Personen), Hinzufügen Meeting (Scrum of Scrums, nach Daily Scrum), Ändern Product Backlog (neue Zuordnung Aufgaben)
2. Änderung Stakeholder (Teammitglied 1, Rolle (Team 1), hinzufügen (für alle Teammitglieder durchführen); Scrum Master, Meeting (Scrum of Scrums), hinzufügen, Teammitglied 1-2 (Team 1), Meeting (Scrum of Scrums), hinzufügen; usw.
3. Benachrichtigung Beteiligte (Scrum Master, neue Aktivität (Scrum of Scrums), Teammitglieder 1 – 10, (neue Aktivität, neue Teamzuordnung, neue Aufgaben Product Backlog)

Da die Anpassung, dass das Team zu klein ist, in diesem Beispiel nicht vorkommen soll, werden die Ausführungsregeln hier nicht beschrieben.

Wird nun nach Regel RZBA45 (Fall b) festgestellt, dass der Kunde für User Acceptance Tests mit eingebunden werden soll, so würde die Planungsmöglichkeit vorsehen, dass dies am Ende des Sprints über eine eigene Phase von ca. 1 bis 3 Tagen erfolgt. Dafür wird ein Team vom Kunden zusammengestellt, welches aus Kunden-Testern besteht. Diese neue Phase erfolgt am Ende der Entwicklung, wenn das Inkrement fertig ist, aber vor dem Review-Meeting. Die Kunden bekommen die entsprechenden Aufgaben im Sprint Backlog zugewiesen. Die Regeln und Benachrichtigungen würden dann wie folgt lauten:

1. Anpassung Model (Hinzufügen Rolle (Kunden-Test-Team (4 Personen)), Hinzufügen Rolle (Kunden-Tester), Hinzufügen Aktivität (Kunden-Testtage (Dauer 2 Tage), nach Entwicklung vor Sprint), Hinzufügen Aktivität (Testen der Aufgaben), Hinzufügen Zuordnung (Zuordnung Aufgaben))
2. Änderung Stakeholder (Kunden-Tester1, Rolle (Kunden-Test-Team), hinzufügen (für alle neuen Kunden-Tester durchführen); Scrum Master, Aktivität (Kunden-Test-Tage), hinzufügen; Kunden-Tester Zuordnung (Test der Aufgaben)).
3. Benachrichtigung Beteiligte (Scrum Master, neue Aktivität (Kunden-Test-Tage), Kunden-Tester 1 – 4, (neue Aktivität, neue Aufgaben Backlog))

Ergeben in einem anderen Fall die kombinierten Regeln, dass zusätzliche Tests mit Hilfe von expliziten Testern und/oder Test-Designern ebenso wie ein Inkrementtest nötig sind, so sollte dieses Test-Team wie ein weiteres Team behandelt werden. Das heißt, es bekommt ein eigenes Test-Sprint

Backlog und trifft sich ähnlich wie im Fall eines zu großen Teams im Scrum of Scrums, um sich mit dem Entwickler-Team auszutauschen. Der zusätzliche Inkrementtest sollte am Ende der Entwicklung durchgeführt werden. Die entsprechenden Regeln würden folgendermaßen aussehen:

1. Anpassung Model (Hinzufügen Test-Team (2 Personen), Hinzufügen Rolle (Tester), Hinzufügen Rolle (Test-Designer), Hinzufügen Artefakt (Test Sprint Backlog), Hinzufügen Aktivität (Durchführung der Test-Aktivitäten) Ändern Aktivität Entwicklung (Entwicklung + Test), Hinzufügen Meeting (Scrum of Scrums, nach Daily Scrum, Time-Box: 40 Minuten), Hinzufügen Aktivität (Inkrementtest, Ende Entwicklung + Test))
2. Änderung Stakeholder (Test-Teammitglied 1, Rolle (Tester), hinzufügen, Test-Teammitglied 2, Rolle (Test-Designer), Rolle Scrum Master, Meeting (Scrum of Scrums), hinzufügen, Test-Teammitglied 1-2, Meeting (Scrum of Scrums), hinzufügen, Test-Teammitglied 1-2, Artefakt (Test Sprint Backlog), hinzufügen, Test-Teammitglied 1-2, Aktivität (Test-Durchführung), hinzufügen, Test-Teammitglied 1-2, Aktivität (Inkrementtest), hinzufügen)
3. Benachrichtigung Beteiligte (Scrum Master, neue Aktivität (Scrum of Scrums), Test-Teammitglieder 1 – 2, (neue Rolle, neue Aktivitäten, neues Artefakt Test Sprint Backlog)

Da für die Regel RZK1 keine Planungsmöglichkeiten vorhanden sind, können somit dafür keine Ausführungsregeln und Benachrichtigungen erstellt werden.

Die einzelnen Schritte in der Pre-Work sind nun durchgeführt. Die erarbeiteten Ergebnisse werden vor Beginn des Projektes und somit vor Beginn des Durchlaufs der MAPE-K-Feedbackschleife entsprechend der Beschreibung in Abschnitt 6.1 in der Wissensbasis gespeichert. Zusätzlich zu der Historie über die Anpassung wird hier eine Historie über die Fehlermeldungen und die Kundenfehlermeldungen angelegt. Die Analyseregeln wurden vor Beginn entsprechend in die ECA-Regeln übertragen und eine aktive Datenbank wurde für die Wissensbasis erstellt. Die einzelnen Anpassungen werden im weiteren Verlauf an einfachen Modellen erläutert.

7.1.3.2 Durchführung einer einfachen Anpassung

Zu Beginn des Projektes, dem Start der Ausführung der Software-Engineering-Methode und somit der Durchführung von den Phasen MAPE bzw. den Schritten 7 bis 10, werden die entsprechenden Einträge mit den Initialwerten versehen. Alle Beteiligten werden dem Team über die Teamliste zugeordnet, alle Teammitglieder bekommen den Status „aktiv“, die Teamgröße beträgt zu Beginn bereits 9 Personen, dem Lenkungskreis wird

die Rolle des Product Owners zugeordnet, das Daily Scrum findet mit 30 Minuten dienstags und donnerstags statt, die Liste und das System für die Fehlermeldungen wird initialisiert, das Product Backlog enthält alle Anforderungen usw. Der erste Sprint beginnt mit den Sprint Planning und es wird das Sprint Backlog erstellt. Hier im Beispiel laufen die ersten zwei Sprints, die noch relativ einfache Umsetzungen enthalten, nach Plan und es erfolgen keine Anpassungen.

Beispiel Projektcontrolling: Krankheit oder Verlassen des Unternehmens, Verteiltes Arbeiten

Am Anfang des nächstens Sprints, beispielsweise an Tag 6 nach dem Sprint Planning und während der Entwicklung, ereignet es sich, dass der Scrum Master einen Unfall hat und mindestens bis zum Ende des Sprints ausfällt. Der Scrum Master meldet seine Krankheit dem Unternehmen und wie lange er voraussichtlich ausfällt. Dies wird, zum Beispiel von der Sekretärin, entsprechend in die Liste eingetragen, der Status des Scrum Masters ändert sich auf „krank“ und die Fehltage von 21 Tagen werden ebenfalls eingetragen.

Diese Werte werden nun über die Sensoren gemessen und durch die Aufbereitung in der Monitor-Phase wird (Status „krank“ & Fehltage = 21/ Person = Scrum Master) zurückgeliefert. Die Einträge haben sich also in der Statuszeile und bei der Anzahl der Fehltage für den Scrum Master, geändert. Dies ist das auslösende Ereignis für die Analyse anhand der ECA-Regeln.

Die Auswertung der Analyse ergibt anschließend, dass die Person länger ausfällt und somit wird die Planung mit Variante 2 angestoßen. Hierzu werden die Aufgaben des Scrum Masters in der Plan-Phase ausgelesen. Da die Person länger krank und zusätzlich der Konflikt bekannt ist, dass das Team die Aufgaben des Scrum Masters nicht übernehmen darf, wird in der Personenliste eine Person gesucht. Diese soll die passenden Fähigkeiten für den Scrum Master besitzen, muss Zeit für die Aufgaben haben oder bekommt die entsprechende Zeit dafür vom Unternehmen. Beispielsweise werden die aktuellen Aufgaben dieser Ersatz-Person automatisch vom System im anderen Team neu verplant.

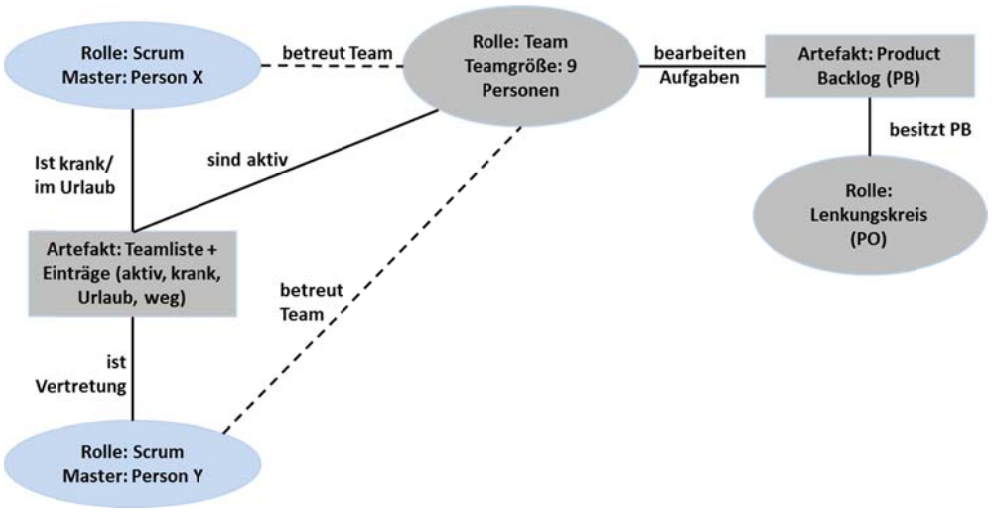

Abbildung 50 Austausch Scrum Master bei Krankheit, Verlassen des Unternehmens etc.

Die Planung ergibt also wie in Abbildung 50 zu sehen, dass die Person des Scrum Masters X durch die Person Y für die Vertretungszeit ersetzt wird. Die Einträge dafür wurden aus der Teamliste entnommen. Die Person Y wird dem Team als Vertretung des Scrum Masters zugeordnet. An der eigentlichen Software-Engineering-Methode ändert sich nichts, da die Person nur auf der Instanz-Ebene ausgetauscht wird.

Da das Ziel, wenn sich der Status einer Person verändert, die Priorität 1 hatte, wird der Anpassungszeitpunkt nicht abgewartet, sondern die Anpassung wird sofort vorgenommen. In der Execute-Phase wird die Anpassung entsprechend vorgenommen, die Teamliste erhält den neuen Eintrag bezüglich des Scrum Masters und wer seine Vertretung ist. Dafür wird der neue Status „Vertretung“ eingetragen, damit bekannt ist, wann der Scrum Master voraussichtlich zurück in sein altes Team wechselt. Alle Teammitglieder und der neue Scrum Master werden entsprechend über die Änderung informiert. Der Scrum Master tritt am nächsten Tag bis zum Ende des Sprints seine neuen Aufgaben an.

Beispiel Team-Vergrößerung

Am Anfang des nächsten Sprints (Nummer 4) ist der alte Scrum Master wieder gesund und kann wie vorgesehen seine Aufgaben übernehmen. In der Teamliste wird entsprechend sein Status auf „aktiv“ gesetzt und seine Vertretung kehrt in ihr altes Team zurück. Da die Sprints langsam komplexer werden und der Lenkungskreis mehr Aufgaben realisiert haben möchte, hat dieser beschlossen, dass das Team durch 7 weitere Personen unterstützt wird. Diese werden entsprechend in der Liste dem Team zugeordnet.

Die Sensoren lesen im weiteren Verlauf die Anzahl der Personen pro Team aus, die nun 16 beträgt. Durch die Aufbereitung in der Monitor-Phase wird

dieser Wert der Teamgröße zugewiesen. Da sich dieser Wert geändert hat, ist dies das auslösende Ereignis für die Analyse-Phase. Die Auswertung ergibt, dass das Team zu groß ist und die entsprechende Planung wird angestoßen.

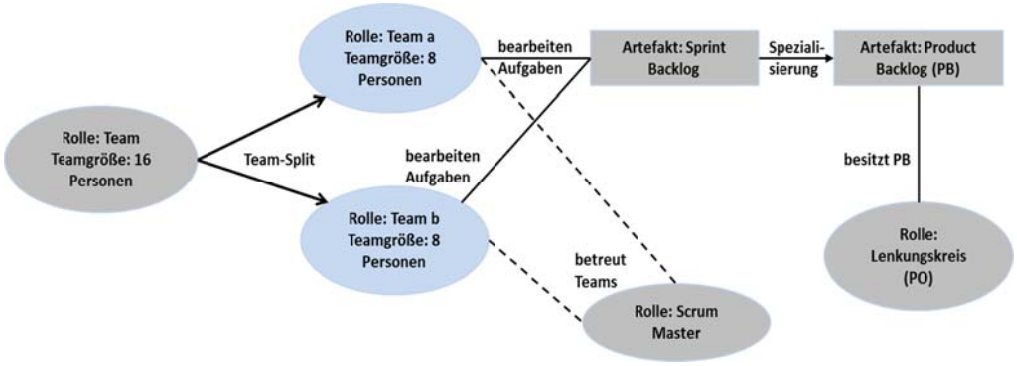

Abbildung 51 Team-Split in Quasi-Scrum bei 16 Personen

Für ein zu großes Team ist ein Team-Split vorgesehen, bei 16 Personen ergibt dies wie in Abbildung 51 zu sehen zwei Teams von je acht Personen. Den Personen werden dabei jeweils die verschiedenen Aufgaben aus dem Sprint Backlog zugewiesen. Der Scrum Master ist für beide Teams der Ansprechpartner. Bei der Verteilung der Aufgaben für beide Teams achtet das System darauf, dass die Aufgaben, welche die Teammitglieder im aktuellen Sprint besitzen, ihnen entsprechend erhalten bleiben. Dadurch gibt es im weiteren Sprint keine Konsistenz-Probleme, zum Beispiel das Aufgaben abgebrochen und nicht weiter ausgeführt werden.

Beide Teams erhalten im weiteren Verlauf wie in Abbildung 52 zu sehen ein eigenes Daily Scrum, welches sie dienstags und donnerstags jeweils 30 Minuten wie bisher abhalten. Zusätzlich wird das Scrum of Scrums hinzugefügt, bei dem sich die Vertreter beider Teams jeweils mittwochs und freitags für 30 Minuten treffen, um entsprechend den Fortschritt beider Teams zu besprechen und sich zu koordinieren. Der Vertreter wird von den Teams selbst bestimmt und muss nicht immer dieselbe Person sein. Von daher wird dies vom System nicht vorgegeben.

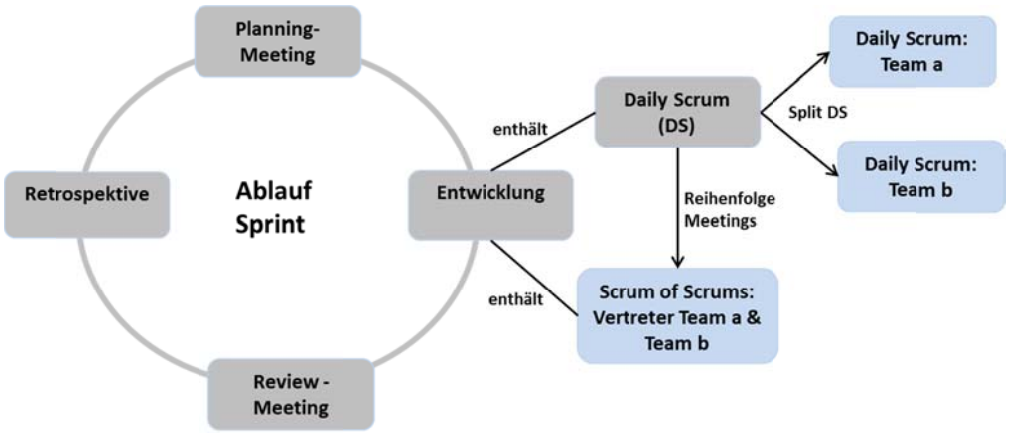

Abbildung 52 Vereinfachter Ablauf nach Team-Split in Quasi-Scrum

Nachdem die Planung entsprechend erfolgt ist, wird überprüft, ob der Anpassungszeitpunkt bereits erreicht ist, da das Ziel nicht die Priorität 1 hatte und somit keine sofortige Anpassung erfolgt. Der Anpassungszeitpunkt ist noch nicht erreicht (Mitte des Sprints, Ende des Sprints), von daher wird die Anpassung gespeichert und es wird bis zum Anpassungszeitpunkt gewartet.

Ist der Anpassungszeitpunkt erreicht, wird wie in Abschnitt 6.4.1 beschrieben überprüft, ob nur eine Anpassung vorhanden ist. Da dies der Fall ist, kann die Anpassung nun ausgeführt werden. Alle Teammitglieder werden entsprechend über den Team-Split und somit welchem Team sie zugeordnet sind, benachrichtigt. Zusätzlich erhalten sie die Information über das Daily Scrum, den Zeitpunkt des Scrum of Scrums sowie das die neue Zuordnung ab dem nächsten Tag, also in der Mitte des Sprints, erfolgt.

7.1.3.3 Durchführung einer komplexeren und kombinierten Anpassung

Da im weiteren Verlauf das größere Team beibehalten wird und somit mehr Aufgaben pro Sprint erledigt werden können, werden entsprechend fertige Produkt-Inkremente geliefert, die der Kunde bereits anfängt einzusetzen. Von daher kann dieser bereits wichtiges Feedback geben und stellt, wenn nötig, Fehlermeldungen ein. In Sprint 6 kommt es nun vor, dass die Fehlermeldungen vom Kunden immer weiter zunehmen, da ihm während des Einsatzes vom Inkrement aus dem fünften Sprint wesentlich mehr Fehler und Dinge auffallen, mit denen er nicht zufrieden ist. Vor allem erscheint dem Kunden und auch seinen Anwendern das ausgelieferte Inkrement viel zu langsam zu sein. Er erfasst entsprechend Fehlermeldungen bezüglich der Performanz, 3 sind bereits vorhanden. Insgesamt sind nun 11 Meldungen im System vorhanden. Die Rückmeldung ist der kombinierte Wert beider Werte und die Änderung ist das auslösende Ereignis in der Monitor-Phase. Die Analyse ergibt, dass die Software-Engineering-Methode gemäß Regel RZBA45 (Fall b) angepasst werden muss (#Fehlermeldungen Kunde > 10

und #nicht-funktionale Meldungen >2). Der Kunde soll schon während des Sprints für User Acceptance Tests mit eingebunden werden.

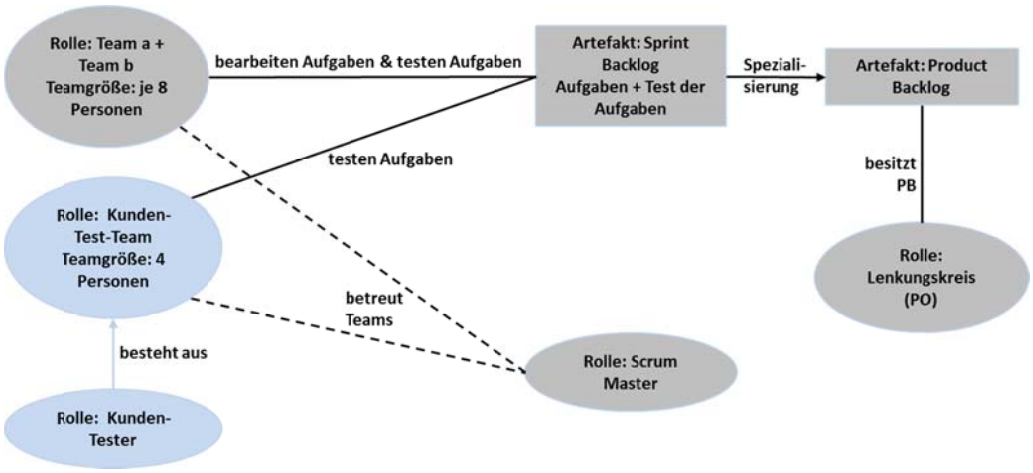

Abbildung 53 Quasi-Scrum Kunden-Test-Team und Kunden-Tester

Die Planung ergibt wie in Abbildung 53 zu sehen, dass als weitere Rolle ein Kunden-Test-Team mit der Teamgröße von 4 Personen mit eingebunden werden soll. Das Team besteht aus einzelnen Kunden-Testern. Die Kunden-Tester können auf das Backlog zugreifen und sollen die jeweiligen Aufgaben des aktuellen Sprints testen. Der Scrum Master ist ebenfalls der Ansprechpartner für das Kunden-Test-Team.

Das Kunden-Test-Team soll das fertige Inkrement des jeweiligen Sprints testen. Die dort erarbeiteten Informationen fließend in das Review-Meeting mit ein und möglichen Problemen kann besser entgegen gewirkt werden. Diese zusätzlichen Tage zum Testen durch den Kunden können durch den Methoden-Baustein „Kunden-QS-Tage“ realisiert werden und haben die Länge von 1-3 Tagen. Hier im Beispiel sollen sie 2 Tage betragen und sollen wie in Abbildung 54 zu sehen nach der Entwicklung, aber vor dem Review-Meeting stattfinden. Die Kunden-Tester bekommen als Aktivität entsprechend das Testen der Aufgaben zugewiesen. Da dies eine vollständig neue Aktivität ist, die nicht in die ursprüngliche Entwicklung eingreift, muss während der Anpassung nicht darauf geachtet werden, ob Aufgaben bereits beendet sind oder nicht.

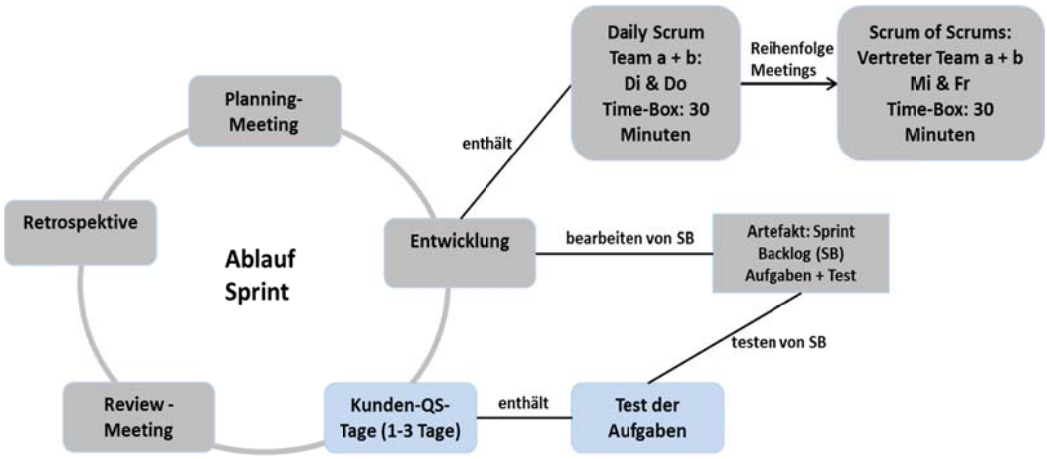

Abbildung 54 Quasi-Scrum: zusätzliche Kunden-QS-Tage

Bevor die Anpassung ausgeführt wird, wird wiederum überprüft, ob der Anpassungszeitpunkt erreicht ist oder nicht, da die Priorität des ursprünglichen Ziels nicht die Priorität 1 beträgt. Der Anpassungszeitpunkt ist noch nicht erreicht und die Anpassung wird zunächst gespeichert.

Bevor die Anpassung ausgeführt wird, wird in der Monitor-Phase wiederum die Analyse ausgelöst. Aktuell sind im Fehlermanagementsystem 4 kritische Fehler vermerkt, wobei 2 der Fehler so oder in ähnlicher Form mittlerweile zum dritten Mal auftreten. Ferner ist aus dem vorherigen Fall bekannt, dass die Anzahl der Kundenrückmeldungen mehr als 10 beträgt und davon sind ebenfalls Fehler mittlerweile zum dritten Mal aufgetreten (z.B. die Performanz-Fehler). Die aufbereiteten Werte sind die auslösenden Ereignisse und in der Analyse-Phase greifen hier die beiden kombinierten Regeln RZBAkombiniert (d und e).

Diese Regeln stoßen die Plan-Phase an, so dass zum einen mehr Tests und explizite Tester mit eingebunden werden. Zum anderen soll am Ende des Sprints ein vollständiger Inkrementtest (Test des gesamten Systems bzw. des Inkrements) erfolgen sollen.

Dafür wird wie in Abbildung 55 zu sehen ein Test-Team eingeführt mit der Teamgröße von 2 Personen. Eine Person nimmt dabei die Rolle des Test-Designers ein, die andere Person ist der Tester, welcher die Tests durchführt. Dieses Team wird ebenfalls vom Scrum Master betreut. Für diese Tests gibt es ein neues Artefakt, analog zum Sprint Backlog gibt es ein Test Sprint Backlog, in dem alle Tests bezüglich der Aufgaben enthalten sind. Das heißt, mögliche Tests werden aus den Aufgaben herausgezogen und in das Test Sprint Backlog verlagert.

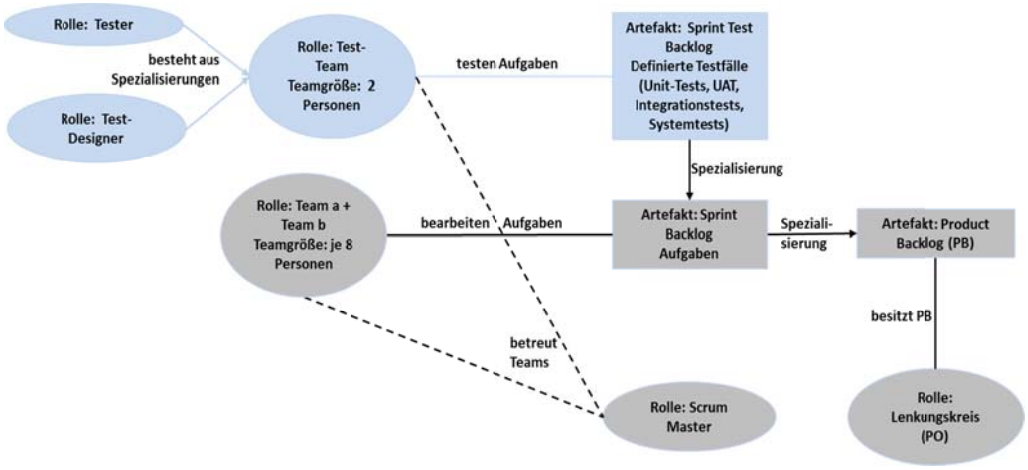

Abbildung 55 Neue Rollen bezüglich Testen und ein neues Artefakt

Das Testen soll wie auch in dem Papier von [GG12] beschrieben während der Entwicklung durchgeführt werden. Somit wird die Entwicklung wie in Abbildung 56 zu sehen um das explizite Testen erweitert. Zusätzlich zu den Entwicklungsaufgaben wird die neue Aktivität „Durchführung der Testaufgaben“ eingeführt. Die beiden Mitglieder des Test-Teams nehmen ebenfalls am Scrum of Scrums der beiden Teams teil, um sich mit den beiden Teams abzusprechen und zu koordinieren. Von daher wird die Time-Box des Scrum of Scrums auf 40 Minuten erhöht, die beiden gewählten Wochentage werden aber beibehalten.

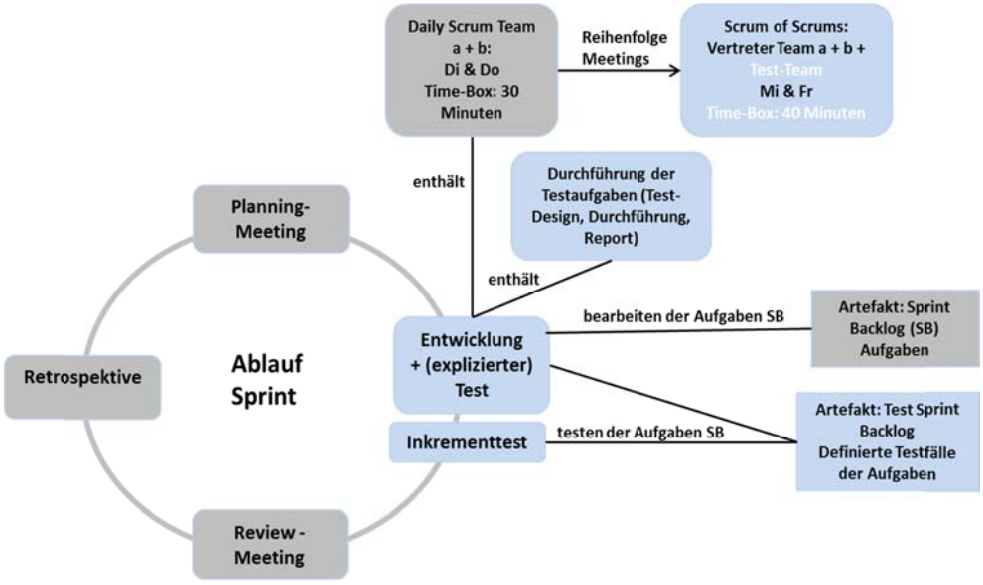

Abbildung 56 Hinzufügen der Testaktivitäten im Scrum-Ablauf

Zusätzlich zu den Testaktivitäten wird ein expliziter Inkrementtest hinzugefügt, welcher nach der Entwicklung stattfindet. Dieser ist ebenfalls im Test

Sprint Backlog vermerkt. Der Inkrementtest wird vom Test-Designer erstellt und vom Tester entsprechend durchgeführt.

Nachdem die Anpassung erstellt ist, wird wie im vorherigen Fall überprüft, ob der Anpassungszeitpunkt erreicht ist. Dieser ist ebenfalls noch nicht erreicht und die Anpassung wird entsprechend gespeichert.

Kombinierte Anpassung Kunden-QS-Tage + Testaktivitäten

Da beide Anpassungen erst in der zweiten Sprinthälfte geplant wurden, erfolgt der Anpassungszeitpunkt erst zum Ende des Sprints. Zum Anpassungszeitpunkt wird festgestellt, dass zwei Anpassungen gespeichert wurden. Diese müssen nun für die Ausführung miteinander zu einer gemeinsamen Anpassung kombiniert wird. Beide Ziele stehen nicht miteinander in Konflikt und können somit gut verbunden werden.

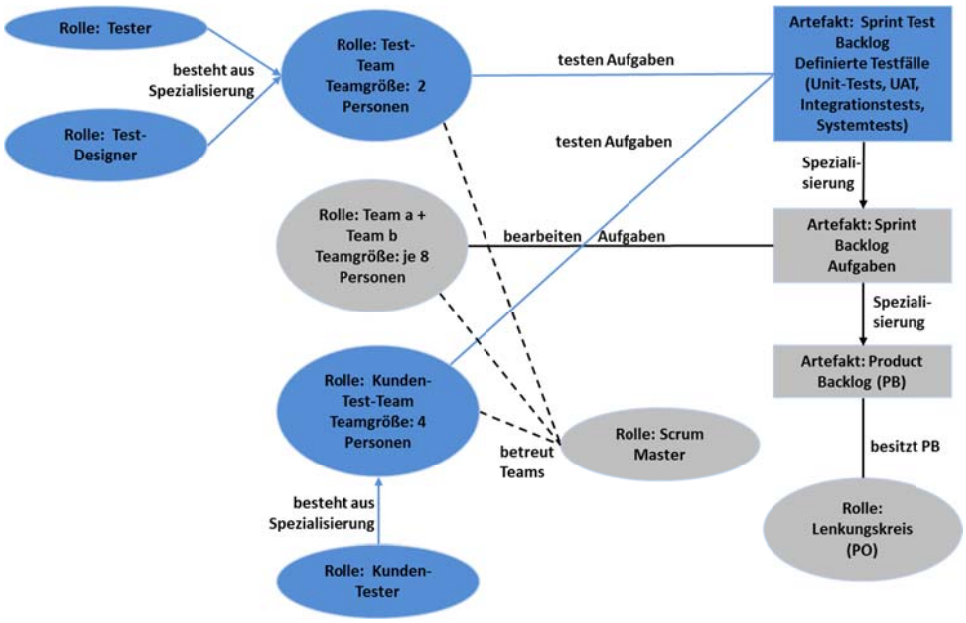

Abbildung 57 Die neuen Rollen in der kombinierten Anpassung - Kunden-Test-Team & Test-Team

Wie in Abbildung 57 zu sehen, werden für die kombinierte Anpassung beide Teams entsprechend ihrer vorher geplanten Anpassung dem Modell hinzugefügt. Da es nun das neue Artefakt „Test Sprint Backlog“ gibt und das Kunden-Test-Team die Aufgaben nicht entwickeln, sondern testen soll, greifen sie anstatt auf das Sprint Backlog nun auf das Test Sprint Backlog zu, so dass beide Teams dieselben Tests durchführen können.

Die Aktivitäten werden in dem vereinfachten Modell wie in Abbildung 58 zu sehen ebenfalls entsprechend ihrer vorherigen Anpassung gemeinsam eingefügt. Die Entwicklung erhält die zusätzlichen Testaktivitäten und das Scrum of Scrums wird entsprechend erweitert. Am Ende der Entwicklung

findet zunächst der Inkrementtest statt, bevor die Kunden-QS-Tage starten. Die eigentlichen Tests stehen im Test Sprint Backlog und werden zunächst vom Team und anschließend zur nochmaligen Validierung vom Kunden-Test-Team durchgeführt. Die Ergebnisse werden gemeinsam im Review-Meeting besprochen.

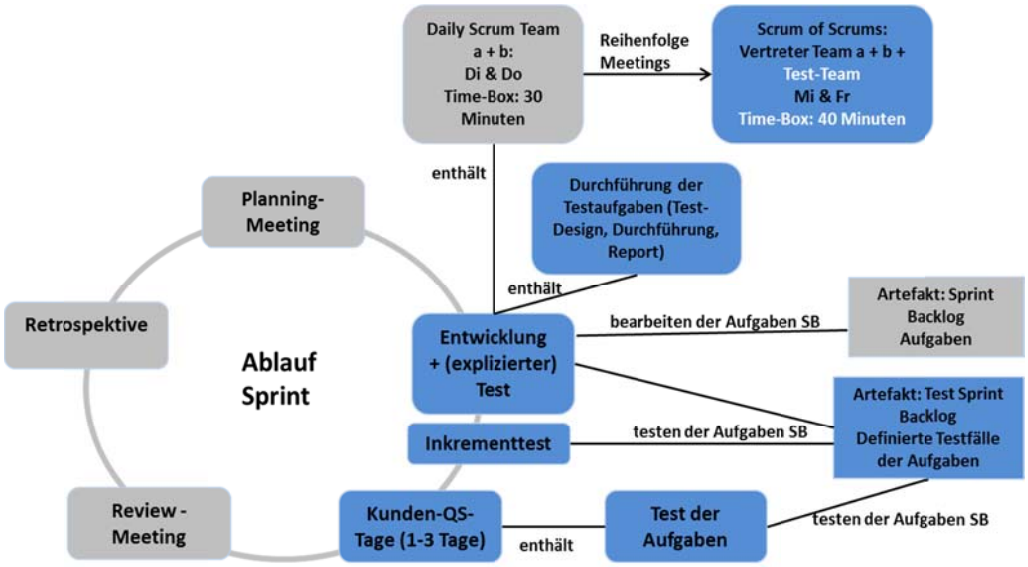

Abbildung 58 Die kombinierte Anpassung

Die kombinierten Ausführungsregeln und Benachrichtigungen sind dadurch komplexer und lauten dann:

1. Anpassung Model (Hinzufügen Rolle (Kunden-Test-Team (4 Personen)), Hinzufügen Rolle (Kunden-Tester), Hinzufügen Test-Team (2 Personen), Hinzufügen Rolle (Tester), Hinzufügen Rolle (Test-Designer), Hinzufügen Artefakt (Tes –Sprint Backlog), Hinzufügen Aktivität (Durchführung der Test-Aktivitäten) Ändern Aktivität Entwicklung (Entwicklung + Test)), Hinzufügen Meeting (Scrum of Scrums, nach Daily Scrum, Time-Box: 40 Minuten), Hinzufügen Aktivität (Inkrementtest, Ende Entwicklung + Test), Hinzufügen Aktivität (Kunden-QS-Tage (Dauer 2 Tage), nach Inkrementtest vor Review-Meeting), Hinzufügen Aktivität (Testen der Test-Aufgaben durch Kunden-Tester))
2. Änderung Stakeholder (Kunden-Tester 1, Rolle (Kunden-Test-Team), hinzufügen (für alle neuen Kunden-Tester durchführen); Scrum Master, Aktivität (Kunden-Test-Tage), hinzufügen, Kunden-Tester Artefakt (Test Sprint Backlog) hinzufügen, Kunden-Tester 1-4, Aktivität (Test-Durchführung) hinzufügen, Test-Teammitglied 1, Rolle (Tester), hinzufügen, Test-Teammitglied 2, Rolle (Test-Designer), Rolle Scrum Master, Meeting (Scrum of Scrums), hinzufügen, Test-Teammitglied 1-2, Meeting (Scrum of Scrums), hinzu-

fügen, Test-Teammitglied 1-2, Artefakt (Test Sprint Backlog), hinzufügen, Test-Teammitglied 1-2, Aktivität (Test-Durchführung), hinzufügen, Test-Teammitglied 1-2, Aktivität (Inkrementtest), hinzufügen)).

3. Benachrichtigung Beteiligte (Scrum Master, (neue Aktivität (Kunden-Test-Tage), neue Aktivität (Scrum of Scrums), neue Rollen (Test-Team, Kunden-Test-Team)) , Kunden-Tester 1 – 4, (neue Rollen, neue Aktivitäten, neues Artefakt Test Sprint Backlog), Test-Teammitglieder (neue Rolle, neue Aktivitäten, neues Artefakt Test Sprint Backlog))

Während die neue Anpassung erstellt wird, werden mögliche neue Anpassungen in den Phasen Monitor und Analyse blockiert, damit es zu keinen weiteren Überschneidungen kommt. Da die Anpassung am Ende des Sprints erfolgt und Personen und Aktivitäten etc. neu hinzugefügt werden, entsteht hier kein Konflikt mit aktuellen Aufgaben, Artefakten etc. die gegebenenfalls hätten beendet werden müssen.

Am Ende des Sprints wird die kombinierte Anpassung ausgeführt, welche ab dem nächsten Sprint zum Einsatz kommt.

7.1.3.4 Durchführung einer Anpassung ohne vorherige Planungsmöglichkeiten

Im Laufe des nächsten Sprints (Sprint 7) kommt es nun vor, dass der Test-Standard, welchen das Team verwendet, aktualisiert wird. Die Versionsnummer des Test-Standards wurde auf deren Webseite entsprechend hochgezählt und der neue Standard wird zum Download angeboten. Der Sensor, welcher täglich die Webseite für eine neue Versionsnummer scannt, entdeckt die neue Versionsnummer, liest diese aus und speichert sie entsprechend in der Wissensbasis. Dies ist das auslösende Ereignis für die Analyse. Dort ergibt die Auswertung, dass die neue Versionsnummer höher als die alte ist und somit eine aktuellere Version vorliegt.

In der Plan-Phase wird der neue Test-Standard ausgelesen und mit dem alten verglichen. Dies muss entweder manuell erfolgen oder es besteht die Möglichkeit für ein Vergleichsprogramm ähnlich dem Vergleich in einer Versionskontrolle, wo die geänderten Teile sofort angegeben werden. Der Vergleich ergibt, dass nach dem neuen Test-Standard keinen Kundendaten mehr als Testdaten verwendet werden dürfen. Diese wurden für einige Tests bisher vom Team verwendet. Das heißt, es muss eine neue Aktivität für die Testdatengenerierung erstellt werden, welche vom aktuellen Test-Team verwendet wird.

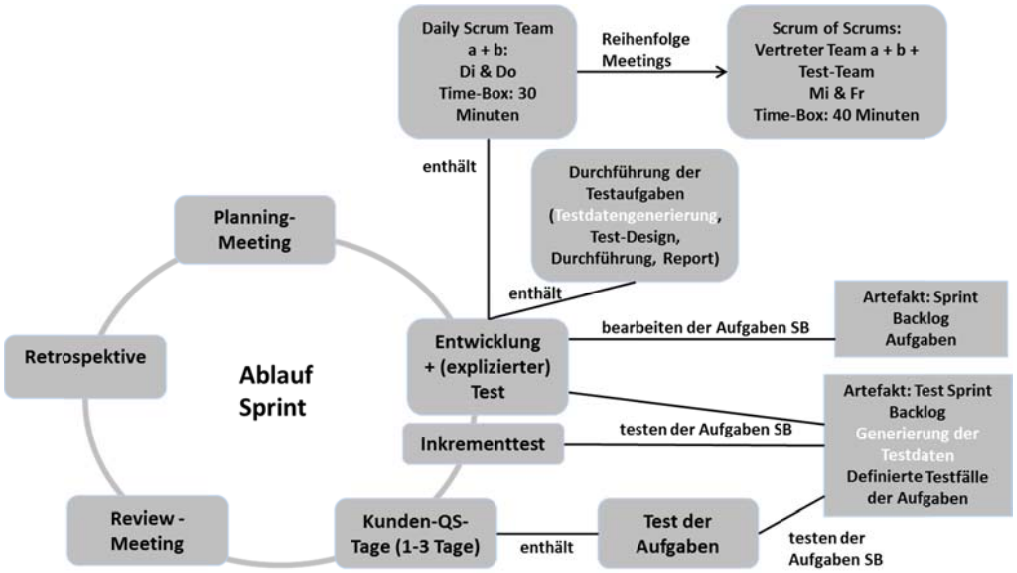

Abbildung 59 Anpassung auf Instanz-Ebene durch Hinzufügen im Test Sprint Backlog

Für die Anpassung gibt es nun zwei Varianten. Die erste Variante passt die Software-Engineering-Methode nur auf der Instanz-Ebene an. Dafür wird wie in Abbildung 59 zu sehen im Test Sprint Backlog eine neue Aufgabe erstellt, welche besagt, dass neue Testdaten generiert werden müssen. Diese Aufgabe bekommt die Priorität 1, so dass sie nach Beendigung der aktuellen Aufgaben als erstes für weitere Tests durchgeführt wird. Das Generieren der Testdaten wird ebenfalls in die Aktivität der Testdurchführung mit eingefügt.

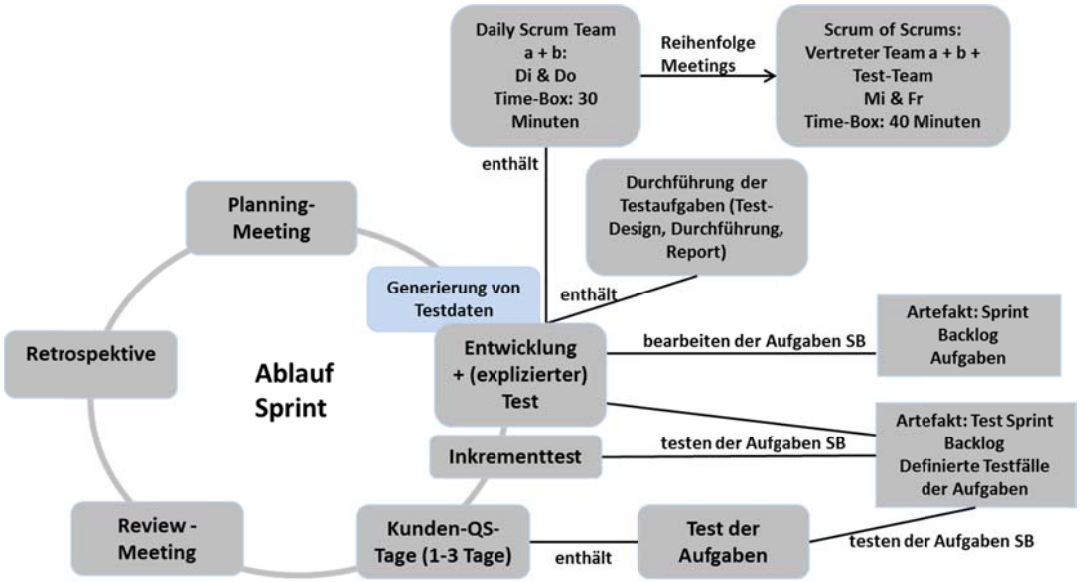

Abbildung 60 Neue Aktivität "Generierung von Testdaten"

Die zweite Variante ist wie in Abbildung 60 zu sehen das Hinzufügen einer neuen Aktivität zu Beginn der Entwicklung. Die Testdaten werden in dieser Aktivität generiert und für die folgenden Tests aus dem Test Sprint Backlog verwendet, sowohl vom Test-Team als auch dem Kunden-Test-Team.

Die Entscheidung, welche Variante gewählt wird, soll zum Zeitpunkt der Anpassung gefällt werden. Ist der Anpassungszeitpunkt während des Sprints, wird die Variante 1 gewählt. So können aktuelle Test-Aufgaben in der Durchführung zunächst beendet werden, bevor die neuen Daten generiert werden (Priorität 1). Konflikte werden somit vermieden.

Ist der Anpassungszeitpunkt zum Ende des Sprints, wird die Variante 2 gewählt. So können die Testdaten für die jeweiligen Tests immer zu Beginn der Entwicklung generiert und im weiteren Verlauf des Sprints verwendet werden, es entstehen ebenfalls keine Konflikte. Wird die Anpassung ausgeführt, werden in beiden Fällen das Test-Team und das Kunden-Test-Team über die Änderungen benachrichtigt, da diese von der Änderung betroffen sind.

Da hier im Beispiel die Änderung des Test-Standards in der ersten Sprint-Hälfte entdeckt wurde, greift für den aktuellen Sprint die erste Variante. Doch da es Sinn macht, ab dem nächsten Sprint die zweite Variante zu nutzen, wird eine kombinierte Anpassung insofern vorgenommen, als dass vermerkt wird, dass ab dem nächsten Sprint Variante 2 greift und damit Variante 1 ablöst.

Auch wenn diese Anpassung ohne vorherige Planungsmöglichkeiten relativ einfach gewesen ist, so kann je nach Ziel und Werten die Planung schnell komplex werden. Zum einen muss eine Anpassung mit den Werten vollständig neu geplant werden, was entweder manuell durch einen erfahrenen Methoden-Engineer erfolgt, oder automatisch über entsprechende Planungsalgorithmen. Zum anderen müssen Konflikte am Ende ausgeschlossen und Alternativen wie in Abschnitt 6.4.2 beschrieben geplant werden.

7.2 Vergleich und Fazit

In den vorherigen Abschnitten wurden mit Hilfe von einfachen und komplexeren Beispielen verschiedene Anpassungsmöglichkeiten des Ansatzes durchgespielt. Dabei wurden sowohl Anpassungen auf der Instanz-Ebene als auch Anpassungen der Software-Engineering-Methode auf Typ-Ebene gezeigt. Zusätzlich wurde eine kombinierte Anpassung und eine Anpassung ohne Planungsmöglichkeiten dargestellt. In diesem Abschnitt werden die Beispiele mit den ursprünglichen Praxis-Projekten verglichen, welche ohne den MAPE-K4SEM-Ansatz durchgeführt wurden. Am Ende erfolgt ein Fazit bezüglich der Evaluierung.

7.2.1 Vergleich mit den ursprünglichen Praxis-Projekten

Der Vergleich der durchgeführten Beispiele findet hier speziell mit dem Projekt Quasi-Scrum statt, welches im s-lab zusammen mit einem Projektpartner im Unternehmen durchgeführt und bereits in [EG09] beschrieben wurde. Da ein Großteil der beschriebenen Anpassungen so oder in ähnlicher Form ohne den Einsatz des MAPE-K4SEM-Ansatzes stattgefunden haben, ist eine Beurteilung gut möglich. Des Weiteren werden bei einigen Beispielen Vergleiche zu einem weiteren Projekt aus dem s-lab gezogen, welches in einem großen Unternehmen mit verteilten Teams, welche z.T. agil gearbeitet haben, stattgefunden hat.

Vergleich: Eine Person fällt aus

Gerade das erste Beispiel, wo eine Person ausfällt, kann gut mit beiden Projekten verglichen werden. Mit Hilfe des Ansatzes war es möglich, den krankgewordenen Scrum Master schnell mit einer geeigneten Person zu ersetzten, so dass das Team problemlos weiter arbeiten konnte. Der Erfolg des Sprints und somit des Projektes war nicht gefährdet.

Im Projekt Quasi-Scrum ist es ebenfalls vorgekommen, dass eine Person zwischenzeitlich krank geworden ist. Dies wurde zwar am ersten Tag bekannt, doch in diesen Fällen musste der Projektleiter entscheiden können, ob durch den Ausfall Aufgaben und somit der Sprint gefährdet sind. Hat er dieses erfasst, hat er das Team zusammen gerufen oder bis zum Daily Scrum gewartet und gefragt, wer sich im Stande sieht, die Aufgaben zu übernehmen. Hat sich niemand gemeldet, musste eine neue Person gesucht werden, die für die Zeit der Krankheit die andere Person ersetzt. Typischerweise war es vom Team möglich, die Aufgaben zu übernehmen, auch wenn dies teilweise dazu geführt hat, dass einige Aufgaben mehr Aufwand erfordert haben, als ursprünglich geplant. Ein oder zweimal ist es vorgekommen, dass eine Aufgabe bis zum Ende des Sprints nicht erfolgreich beendet werden konnte.

Hieran ist zu erkennen, dass der Ansatz zwar nur einen geringen, aber dennoch entscheidenden Unterschied macht. Dadurch, dass dieser eigenständig ermitteln kann, wer für die Aufgaben der ausfallenden Person zuständig ist und Kapazität bereitstellen kann, wird sichergestellt, dass schnell ein geeigneter Ersatz gefunden und nahtlos weitergearbeitet werden kann. Aufgaben können beendet und der Sprint eingehalten werden. Ferner wissen alle Personen über die neue Zuständigkeit Bescheid.

Wie wichtig insbesondere die Benachrichtigung an sich ist, wird im Vergleich mit dem zweiten Projekt deutlich. Hier wurde wie bereits beschrieben mit verteilten Teams gearbeitet, wodurch eine hohe Anzahl Personen im gesamten Team vorhanden gewesen ist. Zwar haben sich die Teams regel-

mäßig mit Hilfe von Vertretern der jeweiligen Teams auseinandergesetzt, doch aufgrund einer hohen Fluktuation in den verschiedenen Teams war teilweise nicht klar, wann ein Verantwortlicher das Team verlassen hat und wer der neue Ansprechpartner gewesen ist.

Ein Beispiel, welches in der Praxis vorgekommen ist: Das Test-Team hat einen Fehler gefunden, diesen erfasst und dem Zuständigen des entsprechenden Teams eine E-Mail mit der Fehlerbeschreibung geschrieben. Da der Fehler ein Blocker war, musste dieser schnellstmöglich behoben werden, um das Testen abzuschließen. Das Problem: Die Antwort auf die geschriebene E-Mail war, dass der Zuständige das Unternehmen verlassen hatte. Zunächst musste nun der neue Ansprechpartner gefunden und ihm das Problem erläutert werden. Dies allein hat zwei Tage gedauert, in denen der Fehler nicht behoben werden konnte.

Zusätzlich fehlten dem neuen Zuständigen die entsprechenden Qualifikationen, denn er nannte ein altes Problem als Fehler, welches schon lange behoben war. Das Test-Team musste dem Zuständigen erläutern, wie er dies bei sich beheben kann, damit er sich um den ursprünglichen Fehler kümmern konnte. Insgesamt hat dies eine Woche an Zeit gekostet, wodurch der Fehler nicht behoben wurde und einen Teil der Tests blockiert hat. Die Deadline musste deswegen nach hinten geschoben werden.

Gerade in diesem Projekt mit verteilten Teams hätten die Teams es gern gesehen, wenn die Zuständigkeiten jederzeit bekannt gewesen und dass sie speziell bei Veränderungen immer informiert worden wären. Dies war in mehreren Fällen nicht gegeben und hat einiges an Zeit gekostet.

Auch wenn die beschriebene Anpassung nur auf der Instanz-Ebene stattfindet, kann hier gezeigt werden, dass der Einsatz von MAPE-K4SEM dieses Problem vor allem mit Hilfe der Benachrichtigungen an die passenden Beteiligten, beheben kann. Es wäre nicht nur Zeit gespart worden, sondern es wäre die passende Person ausgewählt worden.

Vergleich: Teamgröße und zusätzliches Meeting

Der zweite Vergleich bezüglich der Teamgröße findet wiederum mit dem Projekt Quasi-Scrum statt. Hier ist das Interessante, dass das Team zwar vergrößert wurde, es betrug wie im Beispiel-Projekt zwischenzeitlich mindestens 16 Personen, doch es wurde wie in Scrum ursprünglich vorgesehen kein Team-Split vorgenommen und ein zusätzliches Scrum of Scrums eingeführt. Auch wenn sich alle Personen zu den zwei Daily Scrums getroffen haben, so führte dies zum einen dazu, dass regelmäßig die Time-Boxes überschritten wurden, was weitere Entwicklungszeit gekostet hat. Zum anderen gab es Überschneidungen bei den Aufgaben. Ein Teammitglied wollte

eine neue Aufgabe anfangen, brauchte aber Informationen aus einer anderen Aufgabe, welche noch nicht fertiggestellt war. Die Kommunikation im großen Team, von dem nicht alle in demselben Raum saßen, funktionierte nur bedingt. Dadurch verzögerten sich einige Aufgaben und der Sprint war in Gefahr. Dieser konnte manchmal nur mit Überstunden eingehalten werden.

Auch wenn der Team-Split vorgeschlagen wird, wird dieser wie auch im Beispiel zu sehen, nicht immer durchgeführt, was Absprachen und die Kommunikation verschlechtert. Durch MAPE-K4SEM wären der Team-Split und das zusätzliche Meeting automatisch gegeben und eine bessere Kommunikation sowie Absprache wären gesichert. Die Einhaltung des Sprints wäre dadurch besser gewährleistet gewesen.

Gerade in verteilten Teams ist ein zusätzliches Meeting mit Vertretern der einzelnen Teams wichtig. Dies konnte auch im anderen Projekt erfahren werden. Es gab zwar ein regelmäßiges Meeting initiiert durch das Test-Team, doch es war keine Pflicht, dass alle Vertreter der einzelnen Teams daran teilnehmen mussten; die Verantwortlichkeiten waren nicht fest gegeben. Dies führte dazu, dass häufig Verantwortliche aus einzelnen Teams fehlten und Fehler nicht besprochen werden konnten. Das führte zu Verzögerungen, was wiederum zur Gefährdung der Deadlines führte. Auch eine Eskalation nach oben hat wenig zur Verbesserung beigetragen.

Durch MAPE-K4SEM wäre dieses Meeting und besonders Verantwortliche wären automatisch festgelegt worden. Eine Besprechung der Fehler wäre möglich gewesen und Deadlines hätten voraussichtlich eingehalten werden können.

Vergleich: Zusätzliche Kunden-QS-Tage und weitere Testaktivitäten

Im Quasi-Scrum-Projekt wurden im Laufe des Projektes ebenfalls die Kunden-QS-Tage wie in [EG09] beschrieben eingeführt, ebenso wie zusätzliche Testaktivitäten. Für weitere Testaktivitäten wurde dem Team eine weitere Person hinzugefügt. Diese war ausschließlich für das Testen und dabei für die User Acceptance Tests zuständig, Unit-Tests und ein automatischer Integrationstest existierten bereits. Am Anfang hat der Tester nur verschiedene UATs durchgeführt, ohne einen konkreten Testfall zu beschreiben. Am Ende wurde die Anforderung abgenommen oder nicht und mögliche Probleme wurden entsprechend dokumentiert.

In späteren Sprints hat der Tester vorgeschlagen, das Testen mehr zu strukturieren und für jede Anforderungen ein oder mehrere Testfälle zu erstellen. Dies wurde vom Team gut angenommen und in späteren Sprints durchgeführt. Dies führte zu einer Verbesserung der Software und es gab später weniger Fehler im ausgelieferten Inkrement. Auch wenn es von der Testseite

her logisch klingt, Testfälle von Anfang an zu dokumentieren, so ist dies in den Agilen Methoden nicht unbedingt vorgegeben [vergl. GG12]. Der Product Backlog Eintrag dient dabei als Beschreibung und es wird geraten, zusätzlich entsprechende Akzeptanzkriterien zu definieren. Auch dies wird in der Praxis nicht immer vorgenommen.

Durch MAPE-K4SEM wäre das strukturierte Testen wesentlich früher und mit den entsprechenden wichtigen Aktivitäten eingebunden worden. Er hätte damit voraussichtlich zu einer Verbesserung der Testqualität und somit der Qualität des Inkrements geführt.

Deutlicher wird dies bei der Einführung der Kunden-QS-Tage, insbesondere zur Sicherstellung der Betrachtung der nicht-funktionalen Anforderungen. Der Fokus liegt bei Scrum auf den funktionalen Anforderungen, was auch in diesem Projekt der Fall gewesen ist. Es gab zwar Performanzvorgaben für die Software, doch diese haben nie einen Eintrag im Product Backlog erhalten. Auch auf einer Konferenz (Informatik 2009) wurde in einem Workshop diskutiert, ob dies von Anfang an geschehen sollte, oder erst im Laufe des Projektes. Darüber gab es geteilte Meinungen. Doch auch wenn die Anforderungen vom Anfang an im Product Backlog gestanden hätten, so müssten sie vom Product Owner entsprechend ausgewählt und vor allem priorisiert werden. Auch dies war im Projekt nicht der Fall, da wie beschrieben das Hauptaugenmerk auf den funktionalen Anforderungen lag.

Dies führte dazu, dass die Performanz der Inkremente beim Kunden immer schlechter wurde bis hin zum Absturz der Software bei zu großer Belastung. Dadurch wurde klar, dass der Kunde schon vorher mit eingebunden werden sollte mit Hilfe der Kunden-QS-Tage. Durch seine Einbindung wurde ihm sehr schnell bewusst, wie wichtig gerade nicht-funktionale Anforderungen sind. Es ergaben sich neue Einträge für die Optimierung der Performanz, welche immer hoch bis sehr hoch priorisiert wurden. Somit konnte die Performanz bis zum Ende des Projektes erhöht und gehalten werden. Doch vorher hat dies eine Menge Zeit und den Anbieter einen zusätzlichen Sprint gekostet, den er selbst bezahlen musste.

Durch MAPE-K4SEM wäre es möglich gewesen, Probleme schon automatisch vorher zu erkennen und dem entgegen zu steuern, nicht nur mit zusätzlichen Testaktivitäten, sondern auch mit der Einbindung des Kunden. Dadurch hätten insofern Kosten gespart werden können, als dass der Absturz der Software voraussichtlich nicht erfolgt und die Performanzschwierigkeiten in einem früheren Stadium entdeckt worden wäre. Der zusätzliche Sprint wäre nicht nötig gewesen.

Im Projekt mit den verteilten Teams wurde das Testen wesentlich strukturierter angegangen. Es war ein Test-Team mit verschiedenen Testern und

einem Testmanager vorhanden. Die Testfälle für einen End-to-End-Test wurden im Vorfeld definiert und entsprechend durchgeführt. Doch vorherige Tests durch das Entwicklungsteams und speziell gemeinsame Integrations- und Systemtests durch eine Zusammenarbeit der verteilten Teams wurden nicht durchgeführt. Von daher traten die Fehler erst beim abschließenden End-to-End-Test auf.

Mit einer entsprechenden Anpassung der Ziele und der Regeln im MAPE-K4SEM-Ansatz hätten diese Gegebenheiten überwacht und Probleme im Vorfeld entdeckt werden können. Eine Anpassung und das Hinzufügen einzelner Inkrement- und insbesondere gemeinsamer Integrationstests hätten hier zu einer besseren Qualität beigetragen und durch den Ansatz wäre entsprechend Zeit gespart worden.

Sowohl der Kunde als auch der Endanwender wurden in diesem Projekt gar nicht mit eingebunden. Eine endgültige Abnahme erfolgte über das Management, welches an den endgültigen Test-Reports interessiert war. Von daher ist nicht bekannt, ob es Fehler im produktiven System gab oder nicht. Durch den Einsatz des MAPE-K4SEM-Ansatzes wäre ein schnelles Einbinden von Endanwendern oder das direkte Einbinden des Managements gut möglich gewesen. Ob durch das Einbinden des Endanwenders eine weitere Steigerung der Qualität möglich gewesen wäre ist schwer abzuschätzen, da aktuelle Daten nicht bekannt sind.

Vergleich: Neuer Test-Standard

Der Einsatz von Standards, insbesondere eines Test-Standards, wurde in beiden Projekten nicht vorgenommen. Doch auch wenn ein direkter Vergleich nicht möglich ist, liegt die Vermutung nahe, dass vom Team nicht täglich überprüft wird, ob ein neuer Standard vorhanden ist oder nicht. Wäre ein neuer Standard vorhanden, so müsste manuell verglichen werden, welche Änderung vorliegt, was wiederum Zeit kostet.

Auch wenn im Beispiel angegeben ist, dass ein Vergleich möglicherweise manuell erfolgen muss, so wird zumindest rechtzeitig darüber Bescheid gegeben, dass eine Anpassung erfolgen muss. Sobald die Änderungen erkannt sind, kann die Planung gegebenenfalls mit Hilfe von Planungsalgorithmen sowie die Ausführung und Benachrichtigung wiederum automatisch stattfinden. Dies würde ebenfalls Zeit und Kosten sparen.

7.2.2 Fazit der Evaluierung

Auch wenn es sich bei dem Durchspielen der verschiedenen Beispiele um konstruierte Beispiele handelte, so sind diese wie sich vor allem im Vergleich zeigt, sehr nah an der Praxis.

Zunächst konnte anhand des Durchspielens der Pre-Work gezeigt werden, wie konkret einzelne Regeln, Metriken, Planungsmöglichkeiten sowie Ausführungsregeln und Benachrichtigungen aus den priorisierten Zielen hergeleitet werden. Dabei war auch zu sehen, wie Regeln für einzelne Ziele miteinander zusammenhängen und kombiniert werden können. Gerade die Planungsmöglichkeiten hängen eng mit der Herleitung der Ziele zusammen, da sie für den Dann-Teil der Analyseregeln wichtig sind.

Der Aufwand für die Definition der Ziele und die weiteren Schritte der Pre-Work ist im Vergleich zu den Schritten 7 bis 10 wesentlich höher, da diese größtenteils manuell vorgenommen werden müssen. Neben allen Beteiligten für die Definition der Ziele, ihren Rahmenbedingungen und der Priorisierung, wird eine erfahrene Person benötigt, beispielsweise ein Methoden-Engineer. Aber auch ein erfahrener Projektleiter kann die einzelnen Schritte vornehmen.

Obwohl der Aufwand zunächst hoch erscheint, so kann er mit dem Aufwand für das Ermitteln und Spezifizieren bzw. Beschreiben von Anforderungen an eine Software (Requirements Engineering) [He13, Ga14] verglichen werden. Das Finden und Definieren der Anforderungen an eine Software benötigt ebenfalls seine Zeit, ist aber essentiell um eine gute und qualitativ hochwertige Software zu entwickeln. Die Ziele und die daraus resultierenden Schritte sind für eine qualitativ hochwertige Software-Engineering-Methode ebenfalls essentiell. Vom Zeitpunkt her kann das Durchführen der Pre-Work gleichzeitig mit dem Requirements Engineering durchgeführt werden. Auch wenn die Einträge für ein Product Backlog anfangs teilweise gröber gehalten werden können, als in einer anderen Software-Engineering-Methode, so „fallen diese nicht vom Himmel“. Sobald die Einträge für ein Product Backlog erstellt sind, lassen sich die Sprints und die Entwicklung schneller durchführen.

Genauso verhält es sich mit dem Ansatz MAPE-K4SEM. Sind die verschiedenen Schritte der Pre-Work erstellt und entsprechend in der Wissensbasis gespeichert, erfolgt die Durchführung der Schritte 7 bis 10, der eigentliche Durchlauf der MAPE-Feedbackschleife wesentlich schneller und größtenteils automatisch.

Durch den Vergleich mit den beiden Praxis-Projekten konnte gezeigt werden, dass die Nutzung der Software-Engineering-Methode kontinuierlich während der Laufzeit überwacht werden kann. Probleme werden automatisch erkannt und entsprechend angepasst. Die Anpassungen sind zusätzlich auf der Instanz-Ebene möglich. Im Gegensatz zu Anpassungen in den Praxis-Projekten konnten im Vergleich die Anpassungen schneller, strukturierter und größtenteils eigenständig ohne Hilfe von außen durchgeführt werden. Besonders wichtig ist, dass automatisch alle erforderlichen Beteiligten vom System über die Anpassung informiert werden. Dabei bekommen die Beteiligten konkret die Benachrichtigung nur mit den Informationen, die sie betreffen, und wann die neue Anpassung greift.

Dass gerade die Benachrichtigungen beispielsweise über neue Verantwortliche sehr wichtig sind und Zeit sparen, hat das erste Beispiel im Zusammenhang mit verteilten Teams gezeigt. Der hier gezeigte Ansatz würde in einem solchen Szenario den Projektleiter insofern unterstützen, als dass er den Überblick über die verschiedenen Teams behält. Gerade bei einer hohen Fluktuation und vielen Statusänderungen der Personen kann eine automatische Übersicht und Benachrichtigung viel Zeit und Probleme ersparen. Es kann beispielsweise verhindert werden, dass jemand Informationen nicht rechtzeitig erhält.

Des Weiteren wurde in der Evaluierung gezeigt, wie Anpassungen sich zum Anpassungszeitpunkt verhalten und wie eine kombinierte Anpassung aussieht. Im Vergleich mit dem Praxis-Projekt Quasi-Scrum wären die Kunden-QS-Tage voraussichtlich früher zum Einsatz gekommen, Probleme bezüglich der Performanz wären früher erkannt und der Absturz der Software und die Kosten für einen zusätzlichen Sprint wären verhindert worden. Das Hinzufügen neuer Rollen war ebenfalls mit dem Ansatz problemlos möglich und führte zu einem strukturierteren Testen.

Auch wenn eine Planung ohne Planungsmöglichkeiten gezeigt wurde, so konnte diese nicht direkt mit der Praxis verglichen werden. In diesem Fall war die Anpassung ohne Planungsmöglichkeiten verhältnismäßig einfach. Wie schon im entsprechenden Abschnitt beschrieben, können solche Planungen wesentlich komplexer und aufwendiger sein und es wird ein erfahrener Methoden-Engineer oder es werden gute Planungsalgorithmen benötigt.

Ferner konnte an einem einfachen Beispiel gezeigt werden, dass Ziele miteinander in Konflikt stehen können und diese mit beachtet werden müssen. Im Fall des Scrum Masters, der keine Teamaufgaben übernehmen darf, war dieser Konflikt verhältnismäßig einfach zu beheben. Auch hier können sich komplexere Konflikte ergeben, die eine Lösung benötigen.

Des Weiteren wurde in den Beispielen angesprochen, dass die Anpassungen zum Zeitpunkt der Anpassung keine anderen Aufgaben beeinflusst haben und diese abgeschlossen werden konnten. Im Falle des neuen Test-Standards wurden zwei Varianten angegeben, welche dies mit beachten. Auch muss es bei komplexeren Fällen mit betrachtet werden, wie Anpassungen die aktuelle Software-Engineering-Methode beeinflussen können.

Abschließend kann gesagt werden, dass anhand der durchgeführten Beispiele und dem Vergleich mit der Praxis, der Ansatz MAPE-K4SEM für die Anpassung von Software-Engineering-Methoden zur geeignet ist und funktioniert. Obwohl die Pre-Work manuell erfolgt, können die eigentliche Überwachung, Analyse, Planung und endgültige Anpassung größtenteils automatisch und somit selbst-adaptiv durchgeführt werden.

Kapitel 8 Zusammenfassung und Ausblick

In den vorherigen Kapiteln wurde der Ansatz MAPE-K4SEM vorgestellt, welcher eine selbst-adaptive Software-Engineering-Methode ermöglicht. Es konnte gezeigt werden, dass es mit Hilfe des Ansatzes möglich war, die Software-Engineering-Methode zur Laufzeit automatisch zu überwachen, Ergebnisse der Überwachung zu analysieren und bei Bedarf die SEM entsprechend anzupassen. Im nächsten Abschnitt wird nach einer kurzen Wiederholung der Ausgangssituation eine Zusammenfassung des Ansatzes sowie der Evaluierung gegeben. Ferner wird der Beitrag dieser Arbeit erläutert. Abschließend wird ein Ausblick gegeben, welche weiteren Arbeiten diesen Ansatz zum einen weiter verfeinern und zum anderen sich daran anschließen können.

8.1 Zusammenfassung

Nach [Vo13] scheitern heute noch ca. 75% der Projekte im IT-Umfeld. Als Gründe werden dabei u.a. undefinierte Ziele, fehlende Unterstützung des Managements, unzureichend definierte Rollen und Verantwortlichkeiten, fehlender Change-Management-Prozess, aber auch nicht angemessene Beachtung von Störeinflüssen oder das Ignorieren von Warnzeichen im Projekt genannt.

Ferner können als weitere Faktoren genannt werden, dass zwar jedes Projekt ein Vorgehen, aber nicht unbedingt eine Software-Engineering-Methode besitzt. Ist eine Software-Engineering-Methode vorhanden, kann diese mit Tailoring oder Situational Method Engineering vor Beginn des Projektes angepasst werden. Während der Laufzeit des Projektes wird die SEM jedoch nicht überwacht und es fehlt die Zeit, bei Warnzeichen diese schnellstmöglich anzupassen.

Auch wenn das Projektmanagement mit der Projektkontrolle sowie das Change Management Ansätze zur Überwachung und Veränderung liefert, so sind diese meistens nicht mit der Software-Engineering-Methode selbst verzahnt und befinden sich alle auf einer anderen Ebene bzw. betrachten einen anderen Kontext. In Kapitel 2 wurden daher zunächst erste Anforderungen definiert, die an die möglichst eigenständige Anpassung einer Software-Engineering-Methode gestellt werden, beispielsweise dass der Fokus auf der SEM liegen muss. Diese Anforderungen wurden in Kapitel 3.1. noch einmal überarbeitet, verfeinert und auf die 13 Anforderungen A1 bis A13 erweitert.

Im weiteren Verlauf des zweiten Kapitels wurden sowohl verschiedene Anpassungsmethoden als auch das Projektcontrolling genauer vorgestellt. Um die Eignung dieser Ansätze einschätzen zu können, wurden die ersten groben Anforderungen in Bewertungskriterien überführt. Auch wenn die ver-

schiedenen Ansätze diverse Möglichkeiten bieten, so konnte mit Hilfe der Bewertungskriterien festgestellt werden, dass die Ansätze nicht ausreichen, eine Software-Engineering-Methode zur Laufzeit anzupassen. Insbesondere eine automatische Überwachung und möglichst eigenständige Anpassung war nicht gegeben.

In dieser Arbeit wurde mit MAPE-K4SEM ein Ansatz vorgestellt, der die Vorteile der verschiedenen Feedbackzyklen aus den bisherigen Ansätzen übernimmt. Mit dem in Kapitel 3 vorgestellten der SE Method Manager (SEMM), wurde ein Ansatz erarbeitet, welcher im Kern die Feedbackschleife MAPE-K aus den selbst-adaptiven Systemen nutzt. Mit Hilfe des Ansatzes unter der Nutzung des SE Method Managers wird eine automatische und selbst-adaptive Überwachung und Anpassung einer Software-Engineering-Methode ermöglicht.

	SME	**Tailoring**	**AM**	**PDCA**	**SS**	**8D**	**PC**	**SEMM**
Fokus	5	5	4	2	2	0	2	5
AZP	2	2	4	3	3	2	4	5
Dauer	3	3	4	3	1	3	3	5
KH	1	1	4	4	2	3	5	5
ÜW	0	0	4	4	2	2	5	5
AA	1	1	4	4	4	4	5	5
PM	0	0	2	3	5	5	3	5
EE	1	0	4	3	4	3	4	5
VS	3	3	2	5	3	1	5	4
Auto	1	0	0	0	0	0	1	4
WVE	3	3	3	3	3	3	3	4
Gesamt	Ø 1.8	Ø 1.6	Ø 3.2	Ø 3.1	Ø 2.45	Ø 2.4	Ø 3.6	Ø 4.7

Tabelle 2 Übersicht der Bewertung des Ansatzes im Vergleich zu den anderen Ansätzen.

Vergleicht man nun den SEMM mit den Bewertungskriterien aus Kapitel 2, so ist in Tabelle 2 zu sehen, dass dieser bereits eine höhere Bewertung erzielt als die vorherigen Ansätze. Der Fokus liegt bei SEMM auf der Software-Engineering-Methode und dem sich zur Laufzeit befindenden Anpassungszeitpunkt. Durch einen hohen Grad an Automatisierung ist die Dauer der Anpassung kurz und kann sowohl frühzeitig als auch weitestgehend eigenständig durchgeführt werden. Die Anpassungen erfolgen kontinuierlich und je nach Bedarf. Durch die Monitor-Phase ist eine kontinuierliche Überwachung der SEM gegeben und mit Hilfe der Analyse-Phase wird regelmäßig der aktuelle Status ausgewertet, ob eine Anpassung erforderlich ist. Bei Bedarf wird in der Plan-Phase eine entsprechende Anpassung geplant und über die Execute-Phase frühzeitig und schnell ausgeführt.

Da die MAPE-K-Feedbackschleife kontinuierlich während des Projektes ausgeführt wird, werden die Ergebnisse einer Anpassung evaluiert, ob sie entsprechend den Erfolg des Projektes erhalten oder nicht. Da zu Beginn des Projektes die Regeln, Sensoren etc. bekannt sein müssen, ist bis zu einem gewissen Grad die Vorausschau gegeben. Diese kann durch die Erweiterungen aus Kapitel 4 noch verbessert werden. Eine vollständige Automatisierung des SE Method Managers ist zwar möglich, es macht aber an einigen Stellen, beispielsweise nach der Analyse-Phase, während der Plan-Phase oder am Ende der Plan-Phase, Sinn, jeweils einen Methoden-Engineer oder den Projektleiter wie beschrieben mit einzubeziehen. Die Ergebnisse können wiederverwendet werden, insbesondere Vorarbeiten und gemachte Erfahrungen. Am Ende von Kapitel 3 konnte zunächst in Abschnitt 3.4.1 gezeigt werden, dass der Ansatz neben den Bewertungskriterien ebenso die gestellten Anforderungen A1 bis A13 erfüllt.

Kapitel 3 schloss mit der Vorstellung der weiteren Herausforderungen TP1 bis TP5. Es musste geklärt werden, wie granular die Daten für den Ansatz sein müssen, ebenso wenig, welche Daten wirklich wichtig für die Aufbereitung und die Analyse sind. Es wurde herausgearbeitet, wie sichergestellt werden kann, was gemessen und in der Analyse-Phase analysiert und ausgewertet wird. Ferner musste herausgearbeitet werden, wie eine Reihenfolge für die Analyse und Planung erstellt werden und Konflikte vermieden werden können. Ferner musste ein Anpassungszeitpunkt vorliegen, da ein System zwar zu jedem Zeitpunkt angepasst werden kann, eine Software-Engineering-Methode aber nicht. Sollten zum Anpassungszeitpunkt mehrere Anpassungen vorliegen, musste herausgearbeitet werden, wie diese miteinander kombiniert werden können.

Um diesen Herausforderungen zu begegnen wurde ein zielorientierter Ansatz verfolgt und der Ansatz MAPE-K4SEM entwickelt. Dieser Ansatz bindet den SE Method Manager als Kernvorgehen in einen 10-Schritte-Ablauf ein und setzt sich aus einer Pre-Work (Schritte 1-6) und der MAPE-K-Feedbackschleife (Schritte 7-10) und somit dem SE Method Manager zusammen. Die Pre-Work startet mit der Definition der Ziele, welche bei der Software-Engineering-Methode eingehalten werden sollen, und priorisiert diese Ziele. Diese Priorisierung setzt sich für die Analyse und die Planung der Anpassung fort, wodurch eine Reihenfolge der Anpassungen erstellt werden kann.

Im weiteren Verlauf der Pre-Work werden sowohl Regeln für die Analyse und daraus die entsprechenden Metriken, als auch Planungsmöglichkeiten für die Plan-Phase und entsprechende Ausführungsregeln und Benachrichtigungen abgeleitet. Ferner wird im Schritt der Planungsmöglichkeiten im Voraus untersucht, welche möglichen Konflikte

auftreten können und es werden erste Lösungsmöglichkeiten dafür gesucht. Ebenso wird untersucht, wie sich verschiedene Planungsmöglichkeiten kombinieren lassen können; es wird nach Kombinationspunkten gesucht. Zur Wiederverwendung der Ergebnisse wurde in Abschnitt 5.8 ein Block zur weiteren Verwendung definiert. Dieser Block enthält die Ergebnisse aus der Pre-Work für das jeweilige Ziel und sie werden in der Wissensbasis gespeichert. Die jeweiligen Blöcke können in späteren Projekten entsprechend wieder verwendet werden.

In der MAPE-K-Feedbackschleife werden die eigentlichen Schritte des SE Method Managers durchgeführt. Hierfür werden die in der Wissensbasis gespeicherten Ergebnisse aus der Pre-Work genutzt. Die Plan-Phase wird in zwei Teile unterteilt, die eigentliche Planung und den Anpassungszeitpunkt. Hier wird betrachtet, ob mehrere Anpassungen vorliegen und es wird gegebenenfalls eine kombinierte Anpassung erstellt und anschließend ausgeführt.

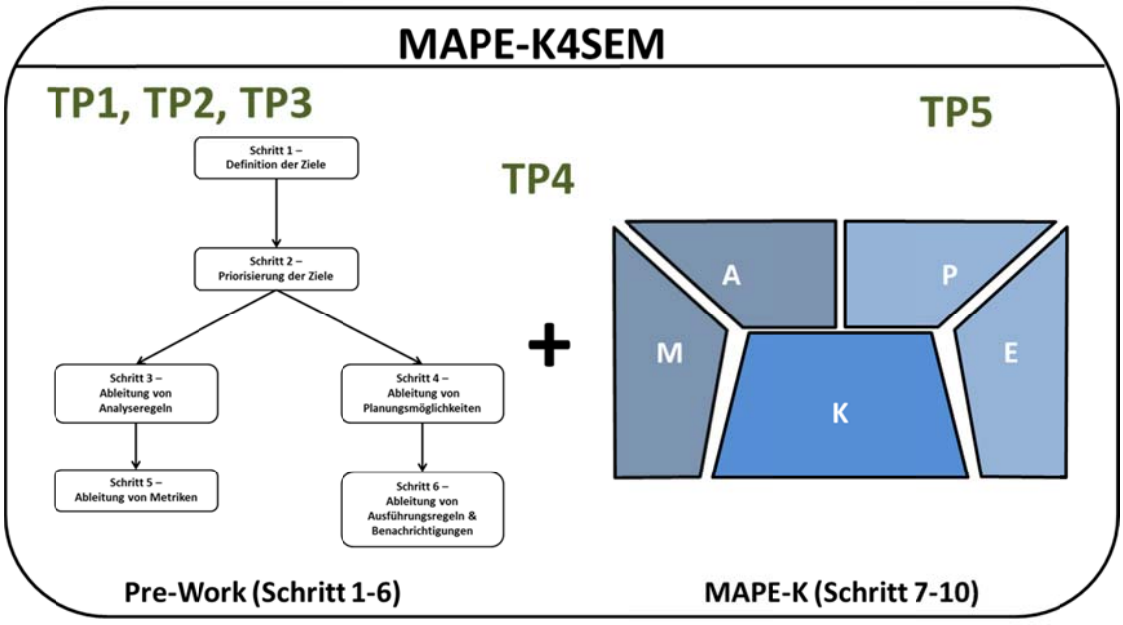

Abbildung 61 Mit Hilfe der Pre-Work + MAPE-K werden zusätzlich die Herausforderungen TP1 - TP5 gelöst

Wie in Abbildung 61 zu sehen, können mit dem MAPE-K4SEM-Ansatz nicht nur die Anforderungen A1 bis A13 erfüllt werden, sondern die Herausforderungen TP1 bis TP5 werden damit ebenfalls erfüllt.

Der wissenschaftliche Beitrag dieser Arbeit ist somit zum einen mit MAPE-K4SEM ein Ansatz, welcher eine selbst-adaptive Software-Engineering-Methode ermöglicht. Eine insbesondere automatische und somit eigenständige sowie kontinuierliche Überwachung der SEM, Analyse des aktuellen Status, Planung und Ausführung einer nötigen Anpassung war mit den bis-

herigen Ansätzen in dieser Form gar nicht oder nur in Teilen möglich. Neben der zeitnahen und eigenständigen Anpassung einer Software-Engineering-Methode zur Laufzeit ist ein weiterer Beitrag der Arbeit, dass der Ansatz MAPE-K4SEM verschiedene Themengebiete miteinander verbindet. Der Ansatz verknüpft die selbst-adaptiven System direkt mit dem Software Engineering. Der Beitrag ist hier genauer die Übertragung des MAPE-K auf die Anpassung von Software-Engineering-Methoden.

Zusätzlich beinhaltet MAPE-K4SEM Ansätze aus dem Projektmanagement und dem kontinuierlichen Verbesserungsprozess, in dem Feedbackschleifen an sich genutzt werden. Ferner beinhalten diese Ansätze beispielsweise einen Kontrollmechanismus und das Planen von Gegenmaßnahmen, allerdings in einem anderen Kontext. Der Ansatz überträgt dies auf den Kontext der Software-Engineering-Methoden mit einer erweiterten Feedbackschleife.

MAPE-K4SEM übernimmt den Fokus auf die Software-Engineering-Methode aus dem Tailoring und dem Situational Method Engineering sowie die gegebenenfalls mit diesen Ansätzen im Vorfeld angepasste Software-Engineering-Methode. MAPE-K4SEM geht dann den weiteren Schritt und überwacht eigenständig, wie sich die gegebene SEM während der Laufzeit verhält.

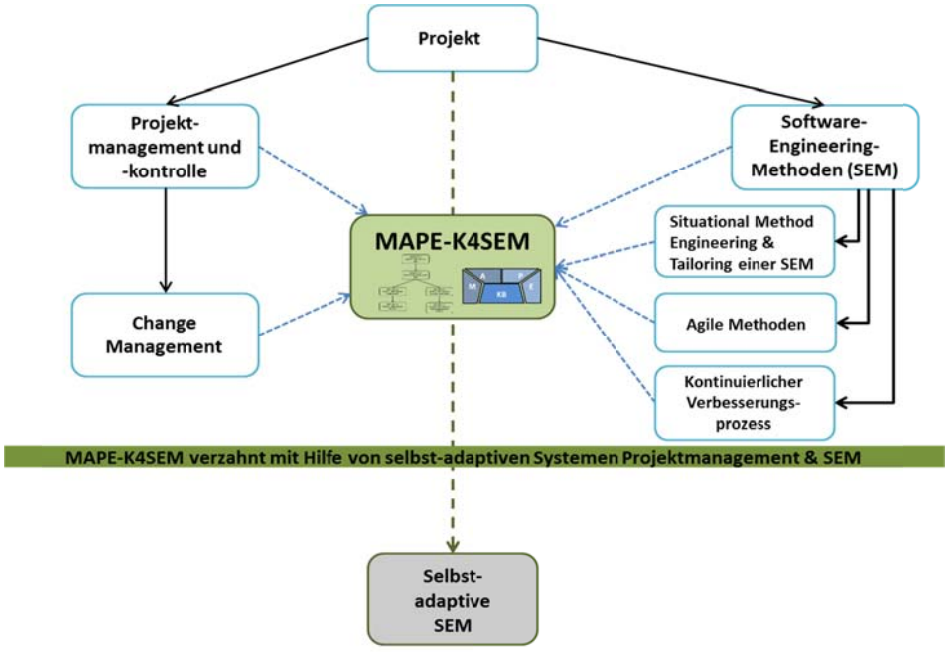

Abbildung 62 MAPE-K4SEM verzahnt mit Hilfe der selbst-adaptiven Systeme das Projektmanagement und Software-Engineering-Methoden

Als letztes beinhaltet der Ansatz die in Abschnitt 2.3 beschriebenen drei Phasen des Change Managements – Planung, Umsetzung und Evaluierung. Wie in Abbildung 62 zu sehen, verarbeitet MAPE-K4SEM die verschiedenen Einflüsse zu einem eigenen Ansatz und verzahnt dabei vor allem das Projektmanagement, hier genauer die Projektkontrolle, und Software-Engineering-Methoden mit Hilfe der selbst-adaptiven Systemen zu einer selbst-adaptiven Software-Engineering-Methode.

In der anschließenden Evaluierung beim Durchspielen der verschiedenen Beispiele konnte gezeigt werden, dass der MAPE-K4SEM-Ansatz vom Prinzip her funktioniert. Durch den anschließenden Vergleich mit zwei Praxis-Projekten wurde gezeigt, dass durch den Ansatz im Gegensatz zu den Projekten eine Anpassung eigenständig, zeitnah und schnell möglich ist. Mit Hilfe des Ansatzes konnte sowohl Zeit gespart als auch Missverständnisse beispielsweise in der Kommunikation, verhindert werden. Probleme konnten frühzeitig erkannt und ihnen entgegengewirkt werden.

Es wurde festgestellt, wie wichtig gerade die Benachrichtigungen an die Beteiligten während der Ausführung sind. Gerade in großen und verteilten Teams kann so der Überblick gewahrt werden und ein Wechsel von Verantwortlichkeiten ist den entsprechenden Beteiligten rechtzeitig bekannt. Auch wenn dies nur eine kleine Änderung auf der Instanz-Ebene und nicht an der Software-Engineering-Methode selbst ist, so kann durch das Nicht-Bekannt-Sein des Wechsels der Verantwortlichkeiten viel Zeit und dadurch Geld verloren gehen. Wie im Praxis-Beispiel zu sehen, kann dadurch sogar der Erfolg des Projektes gefährdet werden.

Es wurde gezeigt, dass eine kontinuierliche Überwachung der Software-Engineering-Methode gegeben ist und durch die Analyse und Auswertung mit Hilfe von beispielsweise ECA-Regeln, ist eine schnelle Beurteilung des aktuellen Status ebenfalls gegeben. Mit Hilfe der Pre-Work wird auf der einen Seite durch die Ableitung der Analyseregeln und den anschließenden Metriken von den vorher definierten Zielen sichergestellt, dass das Richtige gemessen wird, um den Erfolg der Software-Engineering-Methode zu sichern. Auf der anderen Seite orientieren sich die Planungen ebenfalls an den Zielen, so dass diese nach einer erfolgreichen Anpassung weiter eingehalten werden.

8.2 Ausblick

Auch wenn mit MAPE-K4SEM ein funktionsfähiger Ansatz für eine selbst-adaptive Software-Engineering-Methode vorgestellt wurde, so steht als nächstes die Implementierung der Schritte 7 bis 10, des SE Method Managers in einem Prototyp an. Dabei sollte die Wissensbasis mit einer aktiven Datenbank und den beschriebenen ECA-Regeln umgesetzt werden.

Ferner sind der Einsatz von MAPE-K4SEM und seine Evaluierung in einem konkreten Projekt in einem Unternehmen wünschenswert. Neben diesen beiden direkten Anknüpfungspunkten an diese Arbeit gibt es noch weitere Möglichkeiten für zukünftige Arbeiten.

8.2.1 Verknüpfung Requirements Engineering und Pre-Work

Wie bereits in der Evaluierung angesprochen, nimmt die Pre-Work einiges an Zeit in Anspruch. Gerade die Definition und Priorisierung der Ziele erfordern einen hohen Aufwand und auch das Einbeziehen von den Beteiligten. Die Ziele für die Software-Engineering-Methode besitzen Ähnlichkeiten mit den Anforderungen an eine Software und haben somit eine Verbindung zum Requirements Engineering. Dieses beschäftigt sich mit der Ermittlung, Analyse, Spezifizierung und Priorisierung sowie dem Management von Anforderungen sowohl an Systeme als auch an Software-Produkte [He13, Ga14].

Auch wenn die Ziele der Software-Engineering-Methode von der gegebenen SEM hergeleitet werden können, z.B. anhand gegebener Regeln und Eigenschaften, wäre in einer weiteren Arbeit zu überlegen, wie die Ähnlichkeiten dieser beiden Gebiete genutzt und Möglichkeiten sowie Techniken aus dem Requirements Engineering das Definieren der Ziele erweitern können. Gerade die Ziele, die sich aus dem Projekt- und Unternehmenskontext ergeben, könnten mit Hilfe dieser Techniken besser erfasst und präziser formuliert werden.

Umgekehrt kann untersucht werden, ob und wie sich das Definieren der Ziele einer Software-Engineering-Methode auf die Definition der Anforderungen übertragen lässt.

8.2.2 Planen ohne Planungsmöglichkeiten

In Abschnitt 6.4.1 und Abbildung 44 wurde das Planen einer Anpassung ohne Planungsmöglichkeiten beschrieben. In der Evaluierung wurde dafür das einfache Beispiel für die Anpassung bei einem neuen Test-Standard gezeigt. Hier war die Planung ohne Planungsmöglichkeiten relativ einfach. Doch für den Vergleich der beiden Test-Standards ist entweder ein manueller Vergleich oder, wie beschrieben, eine Software nötig, die zwei Versionen automatisch miteinander vergleichen kann.

Für weitere und insbesondere komplexere Planungen ohne vorher bekannte Möglichkeiten ist die erste Lösung, dass eine Person, ein Methoden-Engineer oder ein erfahrener Software-Architekt oder Projektleiter eingesetzt wird. Dieser schaut sich die gegebenen Werte und die entsprechenden Ziele an und entwirft Varianten für die Anpassungen. Doch dies nimmt wieder viel Zeit in Anspruch und könnte den Zeitvorteil des Ansatzes revidieren.

Ein erster Ansatz wäre, die Planung mit Hilfe von Mustern (Pattern) vorzunehmen. In der Arbeit von [FBL13] wird ein Ansatz vorgestellt, wie eine situationsspezifische Methode mit Hilfe von Methoden-Mustern (Method Patterns) erstellt werden kann. Es ist zu überlegen, wie diese Methoden-Muster genutzt werden können, nicht um eine situationsspezifische Methode zu erstellen, sondern um für eine bereits gegebene Software-Engineering-Methode in der gegebenen Situation mit den vorliegenden Werten gemäß ihrer Ziele die entsprechende Anpassung zu planen.

Neben dem Ansatz der Methoden-Muster kann in einer weiteren Arbeit untersucht werden, wie das Planen ohne bereits bekannte Planungsmöglichkeiten automatisch mit Hilfe von Planungsalgorithmen möglich ist. Es kann untersucht werden, welche Planungsalgorithmen bereits in der Literatur vorhanden sind und wie gut diese für den Ansatz eingesetzt werden können. Sind die Algorithmen aus der Literatur nicht ausreichend, so müssen diese entsprechend angepasst und erweitert werden.

Des Weiteren ist zu überlegen, ob es möglich ist, aus der Erfahrung zu lernen und entsprechende Algorithmen aus dem maschinellen Lernen einzusetzen. Diese können mit bereits vorhandenen Planungsmöglichkeiten und eventuellen Algorithmen trainiert werden, um anschließend auf eine neue Situation angemessen zu reagieren.

8.2.3 Kombination von Anpassungen

Bereits in der Evaluierung wurde ein Beispiel für die Kombination zweier Anpassungen zum Anpassungszeitpunkt vorgestellt und durchgespielt. Hier war eine Kombination relativ gut möglich, da sich die Ziele sehr ähnlich waren.

In einer weiteren Arbeit wäre zu untersuchen, welche Kombinationsmöglichkeiten es zwischen den verschiedenen Elementen geben kann. Wie bereits in Abschnitt 5.4.4 beschrieben ist zu untersuchen, wie es sich verhält, wenn es sich beispielsweise bei der Kombination von zwei Anpassungen nicht bloß um gleichartige Elemente, sondern um genau dasselbe Element an derselben Stelle im Modell handelt. Die eine Anpassung will dieses Element löschen, die andere Anpassung will dieses Element austauschen. Stehen diese beiden Anpassungen in Konflikt oder gibt es eine Kombinationslösung?

Eine andere zu untersuchende Frage ist, wie zwei oder mehrere Elemente miteinander verschmolzen werden können, wenn diese nicht von demselben Typ sind? Wie kann abgewogen werden, welche den höheren Nutzen bringt? In einer solchen Arbeit wäre ein Kombinationskonzept zu erstellen, dass die verschiedenen Möglichkeiten betrachtet.

8.2.4 Analyse von Konflikten und Auswirkung auf Gesamt-SEM

Sowohl in der Pre-Work als auch zum Anpassungszeitpunkt wurde bereits erwähnt, dass es zwischen Zielen aber auch zwischen geplanten Anpassungen zu Konflikten kommen kann. Ein einfaches Beispiel wurde in der Analyse gezeigt, wo der Konflikt beachtet werden muss, dass ein Scrum Master keine Entwicklungsaufgaben übernehmen darf. Neben diesem einfachen Fall kann es zu wesentlich komplexeren Fällen kommen.

In einer weiteren Arbeit wäre zu überlegen, wie mögliche Konflikte zwischen einzelnen Zielen bereits im Detail in der Pre-Work analysiert werden können und welche Lösungsmöglichkeiten es geben kann. Sind Konflikte zwischen den Zielen bekannt, ist die Frage zu klären, ob diese nur möglich sind, wenn bereits Planungsmöglichkeiten vorhanden sind oder ob und welche Konflikte es geben kann, wenn eine Anpassung erst während der Plan-Phase erstellt wird. Wie können diese Konflikte aussehen und wie können diese gelöst werden?

In einem weiteren Schritt kann untersucht werden, wie sich eine Planung ohne vorherige Planungsmöglichkeiten auf die gesamte Software-Engineering-Methode auswirkt. Wie bereits in den Abschnitten 5.5.2 „Variantenbestimmung“ und 6.4.1 „Planen einer Anpassung“ sowie in Abbildung 44 beschrieben, muss überprüft werden, wie sich eine Anpassung auf die gesamte Software-Engineering-Methode auswirkt. Es muss sichergestellt sein, dass die geplante Anpassung nicht in Konflikt zur restlichen SEM steht. Dies wäre mit Hilfe von Traceability- und Simulationsalgorithmen möglich. Dazu muss untersucht werden, welche Algorithmen es bereits gibt und ob diese für den Ansatz genutzt werden können oder ob sie angepasst oder erweitert werden müssen.

Ferner ist zu untersuchen, wann eine Variante bzw. Alternativen-Planung abgeschlossen und überprüft wird, welcher Konflikt eine höhere Priorität hat. Dies ist nötig um zu verhindern, dass die Planung ohne Planungsmöglichkeiten in einen Deadlock läuft. Der aktuelle Vorschlag ist, dass nach maximal 3 geplanten Alternativen untersucht wird, welcher der Konflikte die niedrigste Priorität hat, damit eine Anpassung erfolgen kann.

Ein anderer Konflikt der entstehen kann, sind Aufgaben, die zur Zeit der Anpassung noch nicht beendet sind. Die Frage ist, ob diese Aufgaben zunächst beendet werden oder sofort zu neuen Aufgaben übergegangen wird? Wie ist die Handhabung mit alten Artefakten, wie wird dieses Wissen gesichert. Im aktuellen Beispiel bei der SEM Scrum ist das Hauptartefakt das Product Backlog. Da dieses immer erweitert wird oder Einträge gelöscht oder ausgetauscht werden, findet keine direkte Änderung des Artefakts statt. Das heißt, das Artefakt wird weder gelöscht noch ausgetauscht. Ein einfa-

cher Konflikt bezüglich Aufgaben wurde im letzten Beispiel gezeigt. Dort wurden zwei Varianten erstellt, die je nach Ausführung gegriffen haben. Es wäre zu untersuchen, wie dies bei anderen Konflikten aussehen kann und es wäre zu beurteilen, wann eine Aufgaben, Aktivität etc. beendet wird und wann die neuen greifen.

8.2.5 Übertragung des Ansatzes auf andere Bereiche

Der Ansatz MAPE-K4SEM ist von verschiedenen Bereichen und Ansätzen beeinflusst und hat Vorteile beispielsweise der Projektkontrolle, aber auch die Phasen des Change Managements verinnerlicht. Eine interessante weiterführende Frage wäre, ob und wie sich der vorgestellte Ansatz in diese Bereiche zurückübertragen lässt.

Der Fokus des vorgestellten Ansatzes liegt hier auf der Software-Engineering-Methode. Es müsste untersucht werden, wie sich der Ansatz auf den Kontext eines Projektes, also die Projektkontrolle, übertragen lässt. Dafür müssten die Ziele aus dem magischen Dreieck wie in Abschnitt 2.2.1 beschrieben betrachtet und als entsprechende Ziele für den Ansatz definiert werden. Sind die Ziele definiert, können die weiteren Schritte entsprechend aus dem Ansatz abgeleitet werden. Es muss aber zunächst überprüft werden, ob und wie sich der beschriebene Zyklus der Projektkontrolle abändern und an die MAPE-Feedbackschleife anpassen lässt.

Eine andere Möglichkeit wäre zu untersuchen, ob sich der Ansatz in einen ganz anderen Bereich, wie Geschäftsprozesse und deren Überwachung, oder in die der Produktion übertragen lässt. Gerade die 8D-Methodik zeigt, dass für das Reklamationsmanagement ein Zyklus ähnlich dem Six Sigma eingesetzt wird. Die Frage ist: Wäre es möglich eine kontinuierliche Überwachung und schnelle Anpassung mit Hilfe des MAPE-K4SEM in diesem Bereich einzusetzen?

Die Qualitätsziele für die Produkte sind bekannt und es müsste bekannt sein, welche Eigenschaften der Produktionsprozess erfüllen muss, um eine gute Qualität zu liefern. Ist es möglich, aus diesen Zielen die entsprechenden Regeln und Metriken sowie Planungsmaßnahmen im Vorfeld abzuleiten? Wären die Definition der Ziele und entsprechenden Ableitungen der weiteren Schritte in diesem Umfeld möglich, so sollte auch die anschließende Durchführung des MAPE-K mit diesen entsprechenden Werten möglich sein.

Literaturverzeichnis

[AE13] Ahlemann, F., Eckl, C. (Hrsg.): Strategisches Projektmanagement, Springer-Verlag Berlin Heidelberg 2013.

[AM01] Agile Manifesto, www.agilemanifesto.org. (zuletzt besucht: 06.11. 2014).

[Am06] Ambler, S.: The Agile Unified Process (AUP), Ambysoft, http://www.ambysoft.com/unifiedprocess/agileUP.html, latest Version 2006 (zuletzt besucht: 10.11. 2014).

[At14] Atlassian: JIRA. https://www.atlassian.com/de/software/jira (zuletzt besucht: 10.11. 2014).

[BCR94] Basili, V., Caldiera, G., Rombach, H.D.: The Goal Question Metric Approach. In: Encyclopedia of Software Engineering. John Wiley & Sons, S. 528–532, 1994.

[BK13] Broy, M., Kurhmann,M.: Projektorganisation und Management im Software Engineering. Xpert.press, Springer Vieweg, 2013.

[Bo81] Boehm, B.: Software Engineering Economics. Englewood Cliffs: Prentice Hall, 1981.

[Bo13] Borger, S.: Resilienz im Projektmanagement. SpringerGabler, Wiesbaden 2013.

[Br96] Brinkkemper, S.: Method engineering: engineering of Information systems development methods and tools. In: Information and Software Technology, Number 38, pp. 275-280, 2996.

[Br09] Brun et. Al.: Engineering Self-Adaptive Systems through Feedback Loops, In: Software Engineering for Self-Adaptive Systems, Lecture Notes in Computer Science, Volume 5525/2009, pp. 48-70, Springer Verlag, Heidelberg 2009.

[Ch09] Cheng et. al.: Software Engineering for Self-Adaptive Systems: A Research Roadmap, In: Software Engineering for Self-Adaptive Systems, Lecture Notes in Computer Science, Volume 5525/2009, pp. 1-26, Springer Verlag, Heidelberg 2009.

[CGL05] Crespi, V., Galstyan, A., Lerman, K.: Comparative Analysis of Top-Down and Bottom-Up Methodologies for Multy-Agent

System Design. In Proceedings of AAMAS'05, Utrecht, Niederlande, 2005.

[Co02] Cockburn, A.: Agile Software Development. The Agile Software Development Series; Pearson Education, Inc. 2002

[De86] Deming, W.E.: Out of the crisis. Center for Advanced Engineering Study, MIT, Cambridge, MA, 1986.

[DGG95] Dittrich, K.R., Gatiu, S. Geppert, A.: The Active Database Management System Manifesto: A Rulebase of ADBMS Features. In: Rules in Database Systems, LNCS, Volume 985, pp. 1-17, 1995.

[Do06] Dobson, S., Denazis, S., Fernandez, A., Gaıti, D., Gelenbe, E., Massacci, F., Nixon, P., Saffre, F., Schmidt, N., Zambonelli, F.: A survey of autonomic communications. In: ACM Transactions Autonomous Adaptive Systems (TAAS) 1(2), pp. 223–259, 2006.

[DW99] Dröschel, W., Wiemers, M.: Das V-Modell 97. Der Standard für die Entwicklung von IT-Systemen mit Anleitung für den Praxiseinsatz. Oldenbourg, München 1999

[EG09] Engels, G., Geisen, S., Sauer, S., Port, O.: Sicherstellen der Betrachtung von nicht-funktionalen Anforderungen in SCRUM-Prozessen durch Etablierung von Feedback. In S. Fischer, E. Maehle, R. Reischuk (eds.): Informatik 2009 - Im Focus das Leben. Gesellschaft für Informatik (GI) (Bonn), Lecture Notes in Informatics, vol. 154, pp. 3537-3551, 2009.

[ES10] Engels, G., Sauer, S.: A Meta-Method for Defining Software Engineering Methods. In: Nagl Festschrift, LNCS 5765, Springer-Verlag Berlin Heidelberg, pp. 411-440, 2010.

[FLE13] Fazal-Baquaie, M., Luckey,M., Engels, G.: Assembly-based Method Engineering with Method Patterns. Software Engineering 2013 Workshopband, GI, Köllen Druck+Verlag GmbH, Bonn, pp. 435-444, 2013.

[Fi10] Fiedler, R.: Controlling von Projekten. Vieweg + Teubner, GWV Fachverlage GmbH, Wiesbaden 2010.

[GA02] Gernert, C., Ahrend, N.: IT-Management: System statt Chaos – Ein praxisorientiertes Vorgehensmodell. Oldenbourg Wissenschaftsverlag GmbH, München, 2002.

[Ga14] Gabler Wirtschaftslexikon, Stichwort: Requirements Engineering, Springer Gabler Verlag (Herausgeber), http://wirtschaftslexikon.gabler.de/Archiv/75983/requirements-engineering-v8.html (zuletzt besucht: 08.11.2014).

[GCS03] Garlan, D., Cheng, S.W., Schmerl, B.: Increasing system dependability through architecture-based self-repair. In: de Lemos, R., Gacek, C., Romanovsky, A. (eds.) Architecting Dependable Systems. LNCS, vol. 2677. Springer, Heidelberg, 2003.

[Ge12] Geisen, S.: Ein Ansatz zur Anpassung von Software-Engineering-Methoden im laufenden Projekt. In Proceedings of Software Engineering 2012 (SE 2012) – Doktorandensymposium, Petra Hofstedt, Claus Lewerentz (BTU Cottbus), vol. Report 01/12, pp. 7-12, 2012.

[GIAK] Gesellschaft für Informatik – Arbeitskreis Begriffssammlungen, „Begriffe und Konzepte der Vorgehensmodellierung“, http://public.beuth-hochschule.de/~giak/ (zuletzt besucht: 10.11.2014).

[GG12] Geisen, S., Güldali, G.: „Agiles Testen in Scrum“, OnlineSpecial des ObjektSpektrums, 18. Oktober 2012.

[GH08] Gonzalez-Perez, C., Henderson-Sellers,B.: Metamodelling of Software Engineering. John Wiley & Sons, Ltd. 2008.

[Gl08] Gloger, B.: SCRUM. Hanser Fachbuchverlag, München, 2008.

[GLE12] Geisen, S., Luckey,M., Engels, G.: Ein Ansatz zur dynamischen Qualitätsmessung, -bewertung und Anpassung von Software-Engineering-Methoden. In: 19. GI-WIVM-Workshop der Fachgruppe Vorgehensmodelle im Fachgebiet Wirtschaftsinformatik, Shaker Verlag, pp. 111-120, 2012.

[He13] Herrmann, A. et al., Requirements Engineering und Projektmanagement. Xpert.press, Springer Vieweg, 2013.

[HP14] Hewlett-Packard Development Company: Quality Center Enterprise http://www8.hp.com/de/de/software-solutions/quality-center-quality-management/ (zuletzt besucht: 10.11.2014).

[HS06] Henderson-Sellers, Method Engineering: Theory and Practice. In: Information Systems Technology and Its Applications. 5th International Conference ISTA 2006, D. Karagiannis, H.C.

Mayr, Eds. Lecture Notes in Informatics (LNI) – Proceedings, Volume P-84, pp. 13-23, , Gesellschaft Für Informatik, Bonn, 2006.

[HSR10] Henderson-Sellers, B.; Ralyté, J.: Situational Method Engineering: State-of-the-Art Review. Journal of Universal Computer Science,vol. 16, no. 3, pp. 424-478, 2010.

[IBM06] IBM Corporation: An architectural blueprint for autonomic computing. White Paper 4th edn., IBM Corporation 2006.

[ISO07] ISO/ IEC 24744. Software Engineering – Metamodel for Devolopment Methodologies. Geneva: International Organization for Standardization/ International Electronical Comission, 2007.

[ISO08] ISO 9001:2008. Qualitätsmanagementsysteme – Anforderungen. 2008.

[ITA14] it-agile: Was ist agile Softwareentwicklung? http://www.it-agile.de/wissen/methoden/agilitaet/ (zuletzt besucht: 06.11. 2014).

[JSW11] Jung, B., Schweißer, S., Wappis, J.: 8D und 7STEP – Systematisch Probleme lösen. Carl Hanser, München 2011.

[KA06] Kwak, Y.H.; Anbari, F.T.: Benefits, obstacles, and future of six sigma approach. In: Technovation, Volume 26, Issues 5-6, May-June 2006; pp. 708-715.

[KC03] Kephart, J.O., Chess, D.M.: The vision of autonomic computing. In: IEEE Computer 36(1), 2003, pp. 41–50.

[Kr03] Kruchten, P.:The Rational Unified Process. An Introduction. Addison-Wesley Longman, Amsterdam, 2003.

[La09] Ladas, C.: Scrumban – Essays on Kanban Systems for Lean Software Development. Bertrams Print on Demand, 2009.

[La10] Lauer, T.: Change Management – Grundlagen und Erfolgsfaktoren, Springer Verlag Berlin Heidelberg, 2010.

[LL10] Ludewig, J.; Lichter, H.: Software Engineering – Grundlagen, Menschen, Prozesse, Techniken, dpunkt.verlag, Heidelberg, 2010.

[MD89] McCarthy, D.R., Dayal, U.: The Architecture Of An Active Data

Base Management System. In: SIGMOD'89, Proceedings of the 1989 ACM SIGMOD international conference on Management of data, pp.215-224, 1989.

[MN06] Moen, N., Norman, C.: Evolution of the PDCA-Cycle. Erhältlich unter: http://pkpinc.com/files/NA01MoenNormanFullpaper.pdf (zuletzt besucht: 10.11. 2014).

[MPS08] Müller, H.A., Pezzé, M., Shaw, M.: Visibility of control in adaptive systems. In: Second International Workshop on Ultra-Large-Scale Software-Intensive Systems (ULSSIS 2008), ICSE Workshop, 2008.

[MS02] Mar, K.; Schwaber, K.: Scrum with XP, 2002. Erhältlich unter http://faculty.salisbury.edu/~xswang/research/papers/serelated/scrum/scrumxp.pdf (zuletzt besucht: 05.11.2014)

[No14] Noé, M., Change-Prozesse effizient durchführen – Mit Projektmanagement den Unternehmenswandel gestalten, SpringerGabler, Wiesbaden 2014.

[OMG08] Object Management Group: Software & Systems Process Engineering Meta-Model Specification, Version 2.0, 2008.

[PD99] Paton, N.W., Diaz, O.: Active Database Systems. In: ACM Computing Surveys, Vol. 31, No. 1, März 1999.

[Qi07] Qiao,Y. et al.: Developing Event-condition-action Rules in Real-time Active Database. In Proceedings of SAC'07, ACM, pp 511-516, Korea, 2007.

[RB07] Rausch, A., Broy, M.: Das V-Modell XT – Grundlagen, Erfahrungen und Werkzeuge. dpunkt.verlag, Heidelberg 2007.

[RR01] Ralyté, J., Roland,C.: An Assembly Process Model for Method Engineering. In: Proceedings of CAiSE 2001, LNCS 2068, pp. 267-283, Springer-Verlag Berlin Heidelberg, 2001.

[Sa11] Sauer, S.: Systematic Development of Model-based Software Engineering Methods, Dissertation, 2011.

[SB99] van Solingen, R., Berghout, E.: The Goal/Question/Metric Method: a practical guide for quality improvement of software development. McGraw-Hill Publishing Company, Berkshire, England, 1999.

[SB02] Schwaber, K.; Beedle, M.: Agile Development with Scrum. Prentice Hall, 2002.

[Sc13] Schreckeneder, B.,C.: Projektcontrolling. Haufe-Lexer GmbH & Co KG, Freiburg, 2013.

[SH13] Stolzenberg, K., Heberle, K.: Change Management – Veränderungsprozesse erfolgreich gestalten, Springer Verlag Berlin Heidelberg, 2013.

[Sn05] Sneed, H. M.: Software-Projektkalkulation, Hanser Fachbuchverlag, 2005.

[So07] Ian Sommerville, "Software Engineering", 8., aktualisierte Auflage, Pearson Education Limited, 2007.

[SS13] Schwaber, K., Sutherland, J.: The Scrum Guide, the official rule book. Aktuelle Version: Juli 2013. http://www.scrumguides.org/ (zuletzt besucht: 10.11. 2014)

[ST09] Salehie, M.; Tahvildari, L.. Self-adaptive software: Landscape and research challenges. ACM Trans. Autonom. Adapt. Syst. 4, 2, Article 14 , 2009.

[TM94] Thomas, M.; McGarry, F.: Top-down vs. Bottom-up process improvement. IEEE Software, Volume 11, Nr. 4, S. 12-13, 1994.

[Vo13] Voller, R.: Why Projects Fail – Projekte scheitern lassen… aber richtig. Whitepaper, Trivadis AG, Mai 2013. http://www.trivadis.com/uploads/tx_cabagdownloadarea/WP_Why_Projects_fail_V1.0.pdf (zuletzt besucht: 10.11. 2014)

[Vr03] Vries, C.: Certifying for CMM Level 2 and ISO9001 with XP@Scrum. In: Proceedings of Agile Development Conference (ADC'03), pp. 120-124, IEEE, 2003.

[Wa07] Wallmüller, E.: SPI – Software Process Improvement mit CMMI, PSP/ TSP und ISO 15504, Hanser Verlag, 2007.

Abbildungsverzeichnis