Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Eike-Christian Ramcke

Kompetenzaufbau für die agile Softwareentwicklung mit Hilfe eines didaktischen Methodenbaukastens zur Entwicklung individueller Lehrkonzepte

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Eike-Christian Ramcke

Kompetenzaufbau für die agile Softwareentwicklung mit Hilfe eines didaktischen Methodenbaukastens zur Entwicklung individueller Lehrkonzepte

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Ulrike Steffens
Zweitgutachter: Prof. Dr. Stefan Sarstedt

Eingereicht am: 23. Oktober 2014

Eike-Christian Ramcke

Thema der Arbeit
Kompetenzaufbau für die agile Softwareentwicklung mit Hilfe eines didaktischen Methodenbaukastens zur Entwicklung individueller Lehrkonzepte

Stichworte
Agile Softwareentwicklung, Scrum, Kanban, Lehre, Lehrmethode, Lehrkonzept, Kompetenzorientierung, Kompetenz, Lernziel

Kurzzusammenfassung
Die agile Softwareentwicklung wird in der Praxis von vielen Unternehmen eingesetzt, um Produktqualität und Kundenzufriedenheit zu steigern, sowie Entwicklungszeiten zu verringern. Die kompetenzorientierte Lehre ist eine Möglichkeit, um Projektteams auf diesen Anspruch vorzubereiten. Diese Bachelorarbeit zeigt einen Weg, wie von Kompetenzen und Lernzielen ausgehend, ein individuelles Lehrkonzept mit aktivierenden Lehrmethoden zusammengestellt werden kann. Dazu wurden Lehrinhalte für die verbreiteten agilen Methoden, Scrum und Kanban, zusammengefasst und existierende Lehrmethoden der agilen Softwareentwicklung bewertet. Darauf basierend wird ein Grundgerüst mit Beispiel für ein Lehrkonzept vorgeschlagen, das bestehende Unterrichtskonzepte der Pädagogik, Aufwand-Nutzen-Verhältnis und Erfahrungen des Verfassers einbezieht.

Eike-Christian Ramcke

Title of the paper
Building competence of developing individual educational concepts for the agile software development using a didactical method kit

Keywords
Agile software development, Scrum, Kanban, teaching, teaching method, educational concept, skills-based learning, competence, learning target

Abstract
Agile software development was introduced from many companies to improve the product quality, increase customer satisfaction and decrease development periods. Skills-based teaching is one opportunity to dispose project teams regarding this requirement. This bachelor thesis shows a way to compile an individual educational concept with activating teaching methods. For the agile methods Scrum and Kanban, content of teaching has been summarised and existing teaching methods have been evaluated. A matrix with an example for the educational concept is given which includes existing training concepts of educational theory, effort-benefit ratio and experiences of the author.

Danksagung

Zunächst möchte ich bei meiner Betreuerin *Prof. Dr. Ulrike Steffens* bedanken, die es ermöglichte, das Studentenprojekt „myHAW“ im Rahmen der Pflichtveranstaltung „Projekt“ mit der Leitung durch meine Lerngruppe durchzuführen. Ohne dieses Projekt und den daraus resultierenden Erfahrungen wäre das Thema dieser Arbeit nicht zustande gekommen. Außerdem bedanke ich mich bei ihr für die umfangreiche Unterstützung und Motivation während der Bearbeitungszeit, die unter anderem dazu führte, dass ich meine Arbeit auf dem Alumnitreffen der Hochschule für Angewandte Wissenschaften Hamburg (HAW) im September 2014 präsentieren durfte.

Ich möchte mich bei meinen *Eltern* bedanken, die mich nicht nur finanziell unterstützten, sondern überhaupt das Studieren möglich machten. Sie halfen mir stets einen Weg zu finden, das Studium an der HAW zu beginnen, das mir dann so viel Freude bereitete. Meinem *Vater* danke ich auch für das schnelle Korrekturlesen.

Ein besonderer Dank gilt meiner Freundin *Louisa Spahl*, die mich seelisch unterstützte und gerade im Endspurt auf viel gemeinsame Zeit verzichten musste.

Ein großes Dankeschön geht an meinen Lernpartner *Thorsten Graf*, mit dem ich gemeinsam für alle Klausuren die Lerninhalte erarbeitete. Unsere gemeinsamen Diskussionen haben zu viel Verständnis und Wissen geführt, so dass keine einzige Klausur während des Studiums misslungen ist. Ein Dank gilt auch meinem engsten Praktikumspartner *Andreas Kamenz*, mit dem ich viel Zeit, vor allem an Wochenenden, für die Praktikumsaufgaben verbrachte, die allesamt immer rechtzeitig fertig wurden.

Meiner gesamten Lerngruppe, bestehend aus *Andreas Kamenz*, *Björn Baltbardis*, *Thorsten Graf* und *Victoria Bibaeva* möchte ich für die gemeinsame Zeit während des Studiums danken. Der Dank gilt auch meinen Kommilitonen, mit denen ich das Studium begonnen habe. Es existierte in unserem Semester immer ein Zusammenhalt und Austausch, der große Auswirkungen auf meinen Studienerfolg hatte. Ich freue mich, dass es mit vielen von Euch nun im Master weiter geht.

Inhaltsverzeichnis

1 Einleitung **1**
- 1.1 Motivation . . . 2
- 1.2 Zielsetzung . . . 2
- 1.3 Aufbau der Arbeit . . . 3

2 Lehrinhalte **4**
- 2.1 Agile Softwareentwicklung . . . 4
  - 2.1.1 Werte . . . 4
  - 2.1.2 Prinzipien . . . 6
  - 2.1.3 Verbreitung agiler Methoden . . . 10
  - 2.1.4 Abgrenzung . . . 11
- 2.2 Scrum . . . 11
  - 2.2.1 Empirische Prozesssteuerung . . . 12
  - 2.2.2 Team . . . 12
    - 2.2.2.1 Product Owner . . . 12
    - 2.2.2.2 Scrum Master . . . 14
    - 2.2.2.3 Entwicklungsteam . . . 15
  - 2.2.3 Ereignisse . . . 16
    - 2.2.3.1 Sprint . . . 16
    - 2.2.3.2 Sprint Planning . . . 18
    - 2.2.3.3 Daily Scrum . . . 20
    - 2.2.3.4 Sprint Review . . . 21
    - 2.2.3.5 Sprint Retrospektive . . . 21
  - 2.2.4 Artefakte . . . 23
    - 2.2.4.1 Product Backlog . . . 24
    - 2.2.4.2 Sprint Backlog . . . 27
    - 2.2.4.3 Produkt-Inkrement . . . 28
  - 2.2.5 Definition of Done . . . 29
  - 2.2.6 Skalierung . . . 29
    - 2.2.6.1 Abgrenzung . . . 30
    - 2.2.6.2 Chief Product Owner . . . 30
    - 2.2.6.3 Sichten auf ein Product Backlog . . . 31
    - 2.2.6.4 Neue Ereignisse . . . 32
    - 2.2.6.5 Impediment Backlog . . . 37
    - 2.2.6.6 Teamübergreifende Mitglieder . . . 38

2.2.6.7 Koordinierende Teams . . . 38
2.3 Kanban . . . 39
2.3.1 Abgrenzung zu agilen Methoden . . . 40
2.3.2 Kernpraktiken . . . 40
2.3.3 Visualisierung . . . 42
2.3.3.1 Kanban-Board . . . 43
2.3.3.2 Aufgaben . . . 44
2.3.3.3 Nebenläufigkeit . . . 45
2.3.3.4 Spalten ohne Reihenfolge . . . 45
2.3.3.5 Aufgabentypen . . . 45
2.3.4 Work in Progress-Limits . . . 46
2.3.4.1 Probleme . . . 47
2.3.4.2 Engpässe . . . 47
2.3.4.3 Queues und Puffer . . . 47
2.3.4.4 Aufgabentypen limitieren . . . 48
2.3.5 Serviceklassen . . . 48
2.3.5.1 Beschleunigt . . . 49
2.3.5.2 Fester Liefertermin . . . 49
2.3.5.3 Standardklasse . . . 49
2.3.5.4 Unbestimmbare Kosten . . . 50
2.3.5.5 Kapazitäten . . . 50
2.3.6 Koordinierung . . . 50
2.3.6.1 Daily Standup Meeting . . . 50
2.3.6.2 Queue Replenishment Meeting . . . 51
2.3.6.3 Release-Planungsmeeting . . . 51
2.3.6.4 Triage . . . 51
2.3.6.5 Operations Review . . . 52
2.3.6.6 Teamretrospektive . . . 52
2.3.7 Messungen . . . 52

3 Lehrmethoden für agile Softwareentwicklung 53
3.1 Ball Point Game . . . 53
3.2 Coding Dojo . . . 55
3.3 Daily Standup . . . 58
3.4 Kanban Pizza Game . . . 59
3.5 Lego Scrum Simulation . . . 63
3.6 Marshmallow Challenge . . . 66
3.7 Magic Estimation . . . 69
3.8 Pair Storytelling . . . 71
3.9 Planning Poker . . . 73

4 Erstellung von individuellen Lehrkonzepten 76
4.1 Kompetenzen . . . 76

4.2 Lernziele . . . 77
4.3 Verlaufsplanung . . . 79
4.3.1 Rahmenbedingungen . . . 80
4.3.2 Einsatz von Frontalunterricht . . . 80
4.3.3 Einsatz themenunabhängiger Lehrmethoden . . . 81
4.3.4 Verlaufsplan . . . 84
4.3.4.1 Beispiel-Verlaufsplan . . . 85
4.3.4.2 Ergebnisspeicher . . . 91
4.3.4.3 Ablauf der Veranstaltung . . . 91
4.3.4.4 Vereinbarungen . . . 91
4.3.4.5 Kennenlernen . . . 92
4.3.4.6 Feedback . . . 93
4.3.5 Zeitmanagement . . . 93
4.4 Projektarbeit . . . 94
4.5 Umgang mit schwierigen Situationen . . . 95

5 Fazit **98**
5.1 Zusammenfassung . . . 98
5.2 Bewertung . . . 99
5.3 Ausblick . . . 100

Abbildungsverzeichnis **101**

Abkürzungsverzeichnis **102**

Glossar **103**

Quellenverzeichnis **111**

Index **112**

1 Einleitung

Nach einer weltweiten Studie setzen immer mehr Unternehmen auf die agile Softwareentwicklung (vgl. Komus, 2014, S. 22). Sie versprechen sich damit eine höhere Qualität ihrer Produkte, zufriedenere Kunden bei kürzerer Entwicklungszeit und geringeren Kosten. Jedoch ist die erfolgreiche Umsetzung von agilen Methoden trotz weniger und einfacher Regeln herausfordernd, denn sie enthalten nicht nur andere Werte als die herkömmliche Softwareentwicklung, sondern auch eine andere Denkweise.

Studierende der Informatik lernen schon früh Probleme anhand der Spezifikation zu analysieren und eine optimale Lösung zu entwerfen, bevor sie mit der Programmierung beginnen. Erst mit dem vollständigen Entwurf beginnt die Implementation. Diese Denkweise entspricht dem prinzipiellen Vorgehen, dem gesunden Menschenverstand, und wird schnell akzeptiert, denn auch im Bauwesen kann kein Haus ohne den entsprechend vollständigen Entwurf gebaut werden. Dies und noch vieles andere Gelernte steht jedoch im Widerspruch zur agilen Softwareentwicklung, in der ein vollständiger Entwurf nicht notwendig und oft sogar unerwünscht ist (vgl. Cohn, 2010, S. 36). Es müssen nicht einmal alle Anforderungen bekannt sein. Software ist immateriell und daher bei flexibler Architektur leichter zu ändern als ein Haus. Die agilen Arbeitsweisen sind vielen Menschen nicht vertraut und widersprechen den eigenen Erfahrungen, wodurch es gar zu einem ablehnenden Verhalten kommen kann (vgl. Cohn, 2010, S. 37).

In dieser Arbeit soll ein individuelles Lehrkonzept in Form eines Methodenbaukastens für die agile Softwareentwicklung strukturiert werden, dass einen kompetenzorientierten Unterricht mit aktivierenden Lehrmethoden unterstützt. Diese Art des Unterrichts soll Lernende auf das Arbeiten mit agilen Methoden vorbereiten und ausgewählte Aspekte erlebbar machen. Das erste Kapitel dient als Einstieg in das Thema. Es gibt einen Überblick über Motivation, Zielsetzung und Aufbau dieser Arbeit.

1.1 Motivation

Der Ursprung für die Wahl des Themas war ein Projekt an der HAW, in dem das Ziel festgelegt wurde, eine Applikation für die mobile Plattform Android agil zu entwickeln. Die Teilnehmer[1] hatten, bis auf wenige Ausnahmen, keine Erfahrung mit der agilen Softwareentwicklung. Daher wurden die Teilnehmer im ersten Schritt über die eingesetzte agile Methode Scrum mit Skalierung unterrichtet. Zunächst wurde Frontalunterricht eingesetzt und später, als die Entwicklung des Produktes begann, verschiedene existierende Lehrmethoden aus der agilen Softwareentwicklung. Der Lernprozess wurde bei der Produktentwicklung ersichtlich, denn erst mit dem Einsatz der aktivierenden Lehrmethoden begannen die Teilnehmer Scrum zu verstehen und kompetent einzusetzen. Der Abschluss des Projektes erbrachte kein Produkt, das veröffentlicht werden konnte, weil in der verfügbaren Zeit von 15 Terminen zu je 4,5 Stunden die Anwendung von skaliertem Scrum einen Großteil vereinnahmte.

Diese Beobachtung zeigte, dass sich der ganzheitliche Einsatz von Scrum zur Entwicklung eines Produktes im Rahmen einer Lehrveranstaltung nicht unbedingt eignet. Aus dieser Einsicht entwickelte sich die Idee, agile Softwareentwicklung mit Hilfe eines individuellen Methodenbaukastens und aktivierenden Lehrmethoden zu lehren, die für Lehrveranstaltungen mit und ohne Projektbezug geeignet sind.

1.2 Zielsetzung

Diese Arbeit soll einen Weg aufzeigen, wie von Kompetenzen und Lernzielen ausgehend, ein individuelles Lehrkonzept mit aktivierenden Lehrmethoden strukturiert werden kann. Um dies zu erreichen, sind zunächst Lehrinhalte verbreiteter agiler Methoden zusammenzustellen, die die Festlegung von Kompetenzen und Lernzielen unterstützen. Anschließend werden relevante Lehrmethoden aus dem Bereich der agilen Softwareentwicklung beschrieben und bewertet. Die Lehrmethoden sollen sich mittels eines didaktischen Methodenbaukastens so anordnen lassen, dass der Verlauf einer Unterrichtseinheit beschrieben wird. Bei der Gestaltung des Lehrkonzeptes werden dem Planenden alle Freiheiten ermöglicht. Als Beispiel soll eine konkrete Unterrichtseinheit anhand des Methodenbaukastens konzipiert werden. Abschließend wird auf Aspekte, wie den Umgang mit Problemen und Zeitmanagement, die während der Durchführung von Bedeutung sind, eingegangen.

1.3 Aufbau der Arbeit

Diese Arbeit ist in fünf Kapitel gegliedert, die folgendermaßen aufgebaut sind:

Kapitel 1, Einleitung, führt in die Thematik ein und gibt einen Überblick über das Thema.

Kapitel 2, Lehrinhalte, geht auf mögliche Lehrinhalte näher ein, die die Festlegung von Kompetenzen und Lernziele für ein individuelles Lehrkonzept unterstützen. Dabei werden die verbreiteten agilen Methoden und skalierte Ansätze betrachtet.

Kapitel 3, Lehrmethoden für agile Softwareentwicklung, untersucht und beurteilt bestehende Lehrmethoden aus dem Bereich der agilen Softwareentwicklung.

Kapitel 4, Erstellung von individuellen Lehrkonzepten, geht auf Kompetenzen und Lernziele ein, die eine Basis zur Auswahl von Lehrmethoden in einem individuellen Lehrkonzept darstellen. Anhand eines konkreten Beispiel-Verlaufsplans (dem Ergebnis der Anwendung des Methodenbaukastens) wird erläutert, wie sich Lehrmethoden zielgerichtet und systematisch anordnen lassen. Abschließend werden weitere Aspekte, die bei der Durchführung einer Unterrichtseinheit von Bedeutung sind, besprochen und mögliche Lösungen vorgeschlagen.

Kapitel 5, Fazit, fasst die Inhalte dieser Arbeit zusammen und bewertet das Ergebnis. Der Ausblick erläutert offene Fragestellungen und schließt die Arbeit ab.

[1] Aus Gründen der besseren Lesbarkeit wird in dieser Bachelorarbeit die Sprachform des generischen Maskulinums angewendet. Es wird an dieser Stelle darauf hingewiesen, dass die ausschließliche Verwendung der männlichen Form geschlechtsunabhängig verstanden werden soll.

2 Lehrinhalte

Dieses Kapitel gibt einen Überblick über Lehrinhalte, die die Festlegung von Kompetenzen und Lernziele für ein individuelles Lehrkonzept unterstützen. Ebenso dient dieses Kapitel als Nachschlagewerk für Lehrende, um sich über die relevante Themen aus dem Bereich Agile Softwareentwicklung zu informieren. Auch einem Leser ohne Vorkenntnisse wird der Einstieg ermöglicht.

Ausführlichere und ebenfalls wichtige Informationen für die Lehre finden sich zum Thema *Agile Softwareentwicklung* und *Scrum* im Scrum-Guide (Schwaber u. Sutherland, 2013) und in den Büchern Cohn (2010), Pichler (2008), Bleek u. Wolf (2011) und Andresen (2014), sowie zu Kanban (für die Softwareentwicklung) in den Büchern Anderson (2012) und Leopold u. Kaltenecker (2013). Für größere Teilnehmergruppen, die mit skalierten Verfahren arbeiten, finden sich weitere Informationen in den Büchern Eckstein (2012) und Larman u. Vodde (2009).

2.1 Agile Softwareentwicklung

Im Februar 2001 trafen sich 17 Vertreter verschiedener agiler Methoden in Utah, um eine Grundlage für leichtgewichtige Methoden und Prozesse in der Entwicklung von Software zu formulieren. Zu den vertretenen Methoden gehörten unter anderem Scrum und Extreme Programming (XP). Bei diesem Treffen einigten sich die Teilnehmer auf das *Manifest für Agile Softwareentwicklung*, das übergreifende Werte und Grundsätze ihrer Methoden zusammenfasst. Diese Werte werden von zwölf Prinzipien unterstützt.

2.1.1 Werte

Im *Manifest für Agile Softwareentwicklung* werden die Werte wie folgt beschrieben (Beck u. a., 2001):

> Wir erschließen bessere Wege, Software zu entwickeln, indem wir es selbst tun und anderen dabei helfen. Durch diese Tätigkeit haben wir diese Werte zu schätzen

gelernt:

Individuen und Interaktionen mehr als Prozesse und Werkzeuge
Funktionierende Software mehr als umfassende Dokumentation
Zusammenarbeit mit dem Kunden mehr als Vertragsverhandlung
Reagieren auf Veränderung mehr als das Befolgen eines Plans

Das heißt, obwohl wir die Werte auf der rechten Seite wichtig finden, schätzen wir die Werte auf der linken Seite höher ein.

Die erste Aussage „*Individuen und Interaktionen* mehr als Prozesse und Werkzeuge“ drückt aus, dass Softwareentwickler darauf achten müssen, nicht abhängig von ihren Prozessen und eingesetzten Werkzeugen zu werden. Dennoch bieten auch die Prozesse und Werkzeuge einen Mehrwert. Nur ist der Wert von Individuen und Interaktionen höher zu bewerten. Im Mittelpunkt steht also der Mensch, der ein Softwareprojekt ausmacht und die Interaktion miteinander. Unter *Prozesse* sind auch agile Prozesse zu verstehen und zu den *Werkzeugen* zählen auch Taskboards, wie sie zum Beispiel häufig in Scrum und Kanban eingesetzt werden. Die Prozesse und Werkzeuge sollen ein unterstützendes Hilfsmittel sein, an das sich nicht stur gehalten werden muss.

„*Funktionierende Software* mehr als umfassende Dokumentation“ drückt aus, dass der Projektfortschritt anhand der laufenden Software gemessen wird. Nur laufende Software kann aufzeigen, ob der angestrebte Zweck erfüllt wird. Da die laufende Software im Vordergrund steht, ist es notwendig, diese im Entwicklungsprozess möglichst früh und möglichst häufig zur Verfügung zu stellen, damit sie ausprobiert werden kann. So lässt sich feststellen, ob die implementierten Funktionen nutzbar sind und den Vorstellungen der Stakeholder entsprechen. Mit dem dadurch entstehenden Feedback können neue Anforderungen entstehen oder andere wegfallen. Eine umfassende Dokumentation, die *alle* Funktionalitäten einer Software beschreibt, wird nicht benötigt. Dennoch ist Dokumentation wichtig und sie wird für die unmittelbar umzusetzenden Funktionalitäten erstellt, jedoch immer so viel Dokumentation wie es sinnvoll und nötig ist, denn nur benötigter Dokumentation wird ein Wert zugesprochen. Es ist nicht einfach, den richtigen Umfang zu finden, denn dieser variiert nach Kunde und Projekt. Es gibt zum Beispiel regulierte Umfelder, in denen der Umfang der Dokumentation gesetzlich vorgeschrieben ist.

„*Zusammenarbeit mit dem Kunden* mehr als Vertragsverhandlung“ drückt aus, dass nur gemeinsam mit dem Kunden nutzbringende Software entwickelt werden kann. Die Zufriedenheit

des Kunden ist ein wichtiger Maßstab, um ein Projekt erfolgreich abzuschließen. Denn wenn der Kunde das Projekt ansehen und verwenden kann, wird er detailliert nennen können, wo noch Anpassungen nötig sind. Solche Anpassungen haben Einfluss auf den weiteren Projektverlauf und stehen im Widerspruch zu Festpreisverträgen mit festgelegtem Leistungsumfang und Auslieferungszeitpunkt. Der Nutzen ist in Frage gestellt, wenn ein Projekt nach Vertrag ausgeliefert wird, aber die entstandene Software am Ende für den Kunden unbrauchbar ist. Vertragsverhandlungen haben einen höheren Wert, wenn bestimmte Freiheiten, besonders im Leistungsumfang, gegeben sind. Die Festlegung auf einen endgültigen aber realistischen Auslieferungszeitpunkt muss hingegen nicht im Widerspruch stehen, denn beim erfolgreichen Einsatz agiler Softwareentwicklung ist eine lauffähige Software schon früh vorhanden und die wertvollsten Funktionalitäten sind zuerst umgesetzt.

„*Reagieren auf Veränderung* mehr als das Befolgen eines Plans“ drückt aus, dass ständige Veränderungen, seien es Anforderungen, Prozesse, Randbedingungen oder das Verständnis des Problemfeldes, begrüßt werden. In den meisten Fällen liegen die Veränderungen in einem Erkenntnisgewinn oder Lernergebnis begründet. Aber auch personelle Veränderungen im Projektteam oder wegfallende Geschäftsprozesse beim Kunden können der Grund sein. Treten solche außerplanmäßigen Veränderungen auf, hilft es nicht, stur den ursprünglichen Plan zu verfolgen, der nicht mehr eingehalten werden kann. Vielmehr muss sich der Plan an die neue Situation anpassen und notfalls neu entwickelt werden, um weiterhin wertvoll zu bleiben. Dafür ist es sinnvoll, nur die Planung für die nahe Zukunft detaillierter zu beschreiben und spätere Anforderungen höchstens grob zu formulieren. Der Plan lässt sich zum gegebenen Zeitpunkt, wenn es wichtig wird, detaillieren.

Diese Aussagen basieren auf den Projekterfahrungen der Teilnehmer am Treffen in Utah, die von über tausend weiteren Unterzeichnern des Manifests geteilt werden (Beck u. a., 2001).

2.1.2 Prinzipien

Die 12 Prinzipien des agilen Manifests sind Grundlage vieler agiler Methoden und verdeutlichen die Werte und Denkweisen. Sie sollen daher kurz beschrieben werden (Beck u. a., 2001):

> Unsere höchste Priorität ist es, den Kunden durch frühe und kontinuierliche Auslieferung wertvoller Software zufrieden zu stellen.

Im Verlauf der Entwicklung erhält der Kunde frühzeitig und kontinuierlich ein lauffähiges und wertvolles Produkt-Inkrement. Das Produkt-Inkrement enthält nur vollständige Funktionalitäten und repräsentiert die aktuelle Ausbaustufe im Projektverlauf. Mit diesem Inkrement

kann der Kunde ausprobieren, testen und es sogar, wenn es Sinn macht, produktiv einsetzen. So kann der Kunde und das Projektteam herausfinden, wie gut die realisierten Anforderungen verstanden wurden und welche Software eigentlich benötigt wird. Mit dem Feedback können Fehlentwicklungen erkannt werden. Es kann gegengesteuert und fehlende Funktionalitäten können anders priorisiert werden. Der Kunde soll mit diesem Vorgehen eine realistische Sicht auf seine Anforderungen erhalten und am Ende genau das Softwareprodukt bekommen, das er braucht und einsetzen kann.

> Heisse [sic] Anforderungsänderungen selbst spät in der Entwicklung willkommen. Agile Prozesse nutzen Veränderungen zum Wettbewerbsvorteil des Kunden.

In der Praxis spiegeln die zu Anfang zusammengestellten Anforderungen nicht den endgültigen Stand des Softwareproduktes wider. Bei längeren Projekten ist die Wahrscheinlichkeit groß, das sich Anforderungen im Laufe der Zeit ändern. Dabei muss nicht unbedingt am Ende ein anderes Produkt entstehen als ursprünglich geplant. Vielmehr entstehen im Entwicklungsprozess, auch gefördert durch kontinuierlich verfügbare Produkt-Inkremente, neue Ideen oder ein neues Verständnis, was nötig und möglich ist. Änderungen der Anforderungen sollten nicht geblockt werden, denn sie können für den Kunden von großem Nutzen sein und ihm einen Wettbewerbsvorteil geben. Das Projektteam soll Anforderungsänderungen begrüßen und sich mit ihnen auseinandersetzen. Das Team soll mit dem Kunden kommunizieren, was möglich und sinnvoll ist und wie sich die Änderungen auf den Aufwand und die Kosten auswirken. Der Kunde hat so die Möglichkeit zu entscheiden, was er wann haben möchte. Dabei können auch Anforderungen, die nur einen geringen Nutzen und hohen Aufwand haben, komplett entfallen.

> Liefere funktionierende Software regelmäßig innerhalb weniger Wochen oder Monate und bevorzuge dabei die kürzere Zeitspanne.

Funktionierende Software soll nicht nur einmal mit dem Ende eines Projektes ausgeliefert werden, sondern in regelmäßigen verlässlichen Zeitintervallen. Je kürzer die Zeitintervalle liegen, desto öfter und besser kann der Kunde Feedback und Anforderungsänderungen einbringen. Zusätzlich erhält das Projektteam selbst auch öfter Feedback über nicht-funktionale Anforderungen. Die Zeitspanne, in der ein Kunde ein neues Produkt-Inkrement erhält, kann beliebig niedrig sein. Es ist jedoch nicht damit gemeint, dass der Kunde bei sehr kurzen Zeitintervallen mit wenigen neuen Funktionalitäten zur Vorversion, für jedes Inkrement aufgefordert ist Feedback zu geben, denn sonst vergeht ihm das Interesse am Ausprobieren und Testen der Software und das Feedback verliert seinen Nutzen.

> Fachexperten und Entwickler müssen während des Projektes täglich zusammenarbeiten.

Im Projektverlauf tauchen regelmäßig Fragen und Probleme auf, sei es auf fachlicher oder technischer Ebene. Die Fachexperten des Kunden kennen ihre fachliche Domäne und die Entwickler im Projektteam die technische. Beide müssen über die Projektdauer möglichst effektiv zusammenarbeiten, indem sie regelmäßig kommunizieren, sich vertrauen und sich verstehen. Dies ist wichtig um Feedback der jeweils anderen Position im Prozess einfließen lassen zu können.

> Errichte Projekte rund um motivierte Individuen. Gib ihnen das Umfeld und die Unterstützung, die sie benötigen und vertraue darauf, dass sie die Aufgabe erledigen.

Programmieren ist eine kreative Tätigkeit, da sich jede Aufgabe auf unterschiedliche Art implementieren lässt. Dieses Prinzip unterstützt die Ansicht, dass ein günstiges Umfeld die Kreativität von motivierten Entwicklern steigert, um ihre Aufgaben bestmöglich zu realisieren. Einem so engagierten Team kann auch Vertrauen entgegengebracht werden, dass es seine Aufgaben optimal lösen wird. Werden ihm hingegen Räume und Möglichkeiten beschränkt, dann wird die Moral und Leistungsfähigkeit des Individuums und damit auch des Teams gemindert. Typische Einschränkungen, die auf ein Team wirken können, sind unter anderem Rahmenbedingungen, die aus höherer Hierarchiestufe herab erlassen wurden oder ein verstreut arbeitendes Team. Jedoch bewertet jedes Individuum sein Umfeld unterschiedlich und es können unterschiedliche Gegebenheiten eine Einschränkung bedeuten.

> Die effizienteste und effektivste Methode, Informationen an und innerhalb eines Entwicklungsteams zu übermitteln, ist im Gespräch von Angesicht zu Angesicht.

Für die Entwicklung von Software ist permanente Abstimmung durch Informationen nötig, seien sie durch den Kunden an das Team gerichtet und innerhalb des Teams. Je aufwändiger der Austausch von Informationen ist, desto mehr Zeit nimmt die Kommunikation in Anspruch. Hinzu kommt, dass menschliche Kommunikation nicht eindeutig sein muss und Interpretationsspielräume bietet. Diese können nur durch weitere Kommunikation ausgeräumt werden. Bei langen Kommunikationswegen, die nicht permanent zur Verfügung stehen, kommt es zu Verzögerungen und zu Fehlinterpretationen. Die Produktivität der Entwicklung kann dadurch stark eingeschränkt werden. Die effektivste Form der Kommunikation ist das Gespräch von Angesicht zu Angesicht, denn dann wird neben der verbalen Sprache auch mit Körpersprache, Mimik und Gestik kommuniziert und die Informationswege sind kurz. In räumlich oder global

verteilten Ansätzen agiler Methoden muss dieses Prinzip gebrochen werden. Es müssen mehr Dinge schriftlich erledigt werden - seien es schriftliche Statusberichte, Designvorschläge oder E-Mails (vgl. Cohn, 2010, S. 402).

> Funktionierende Software ist das wichtigste Fortschrittsmaß.

Das wertvollste Maß im Projektfortschritt ist, was von den Anforderungen vollständig umgesetzt wurde und einwandfrei funktioniert. Das bedeutet nicht, dass nur die lauffähige Software zählt und Qualität keine Bedeutung hat. Es ist eine projektspezifische Definitionsfrage, welche Qualitätsmerkmale die Software erfüllen muss. Anforderungen zählen erst als fertig, wenn sie dieser Definition entsprechen. Dies ist wichtig, da sonst unvorhersagbare Nacharbeiten an der Software entstehen können.

> Agile Prozesse fördern nachhaltige Entwicklung. Die Auftraggeber, Entwickler und Benutzer sollten ein gleichmäßiges Tempo auf unbegrenzte Zeit halten können.

Nachhaltige Entwicklung bedeutet für das Projektteam, dass die Produktivität und Auslastung keinen starken Schwankungen unterliegt. Leerlaufzeiten sollen sich nicht mit Überlastung abwechseln. Das Ziel ist das höchstmögliche Entwicklungstempo, das dauerhaft und gleichmäßig erbracht werden kann. Nachhaltigkeit in Projektteams fördert zudem auch die Zufriedenheit der Mitarbeiter, da sich die Überstunden reduzieren (vgl. Cohn, 2010, S. 42). Für den Auftraggeber und Benutzer bedeutet es, dass das endgültige Produkt im produktiven Betrieb nicht nur eine kurze Zeit nutzbar ist, sondern einen langfristigen Wert bietet.

> Ständiges Augenmerk auf technische Exzellenz und gutes Design fördert Agilität.

Um vom agilen Ansatz maximal zu profitieren, muss ein Projektteam aus Personen bestehen, die über Teamfähigkeit verfügen. Es sollten Personen mit allen notwendigen Fähigkeiten vorhanden sein, um die Software realisieren zu können (vgl. Cohn, 2010, S. 220). Auch müssen sie bereit sein, ihren technischen und fachlichen Horizont zu erweitern und befähigt sein, ihre Umsetzung gut zu durchdenken, um qualitativ hochwertige Software abliefern zu können. Sie müssen Fähigkeiten entwickeln können, die sie möglicherweise bisher nicht brauchten. Zum Beispiel muss ein Programmierer auch Aufgaben eines Testers übernehmen können und umgekehrt. Durch den Wissensaustausch und ihren gestreuten Fähigkeiten lernen alle im Team voneinander. Als Team zu arbeiten bedeutet auch, nicht über *meine Aufgabe* oder *deine Aufgabe* nachzudenken, sondern über *unsere Aufgabe*. Das gesamte Team muss sich für Qualität und ein gutes Design verantwortlich fühlen (vgl. Cohn, 2010, S. 59).

> Einfachheit - die Kunst, die Menge nicht getaner Arbeit zu maximieren - ist essenziell.

Die Komplexität von Software ist hoch, daher kann Einfachheit auf verschiedenen Ebenen als wertvoll angesehen werden. Bei technischen Lösungen sollen nur benötigte Funktionalitäten umgesetzt werden und keine unverlangten Extras. Sie sollen einfach sein, damit sie einfach zu verstehen und zu warten sind. Es kann so vermeidbarer Aufwand vollständig eingespart werden (vgl. Bleek u. Wolf, 2011, S. 11). Die Schwierigkeit liegt darin, Einfachheit von Minimalismus zu trennen. Beispielsweise sollte aus vermeintlicher Vereinfachung nicht auf notwendige Tests oder Dokumentation verzichtet werden, denn dadurch werden unnötige Risiken eingegangen.

> Die besten Architekturen, Anforderungen und Entwürfe entstehen durch selbstorganisierte Teams.

Selbstorganisierte Teams fühlen sich verantwortlicher für das Lösen bestehender Probleme und das Erreichen gegebener Ziele. Während in einem Team alle Entscheidungen von einer Person getroffen werden, kann in einem anderen Team die Entscheidung der Entwickler oder die Entwicklerin tragen, die gerade an der betroffenen Funktionalität arbeitet. Nicht jedes Team organisiert sich also auf die gleiche Weise. Es führt zu einem besseren Ergebnis, wenn sich das Team nicht allein auf seinen Vorgesetzten verlässt, sondern auf die gesammelten Einsichten des Teams (vgl. Cohn, 2010, S. 219 ff.). Selbstorganisation bedeutet nicht, dass sich ein Team ohne Regeln bewegt und beliebig agieren kann. Zum einen verfolgt ein Team vereinbarte Ziele und ist zum anderen an das leichtgewichtige Regelwerk der ausgewählten agilen Methode gebunden.

> In regelmäßigen Abständen reflektiert das Team, wie es effektiver werden kann und passt sein Verhalten entsprechend an.

Das Projektteam analysiert in regelmäßigen Abständen den Projektverlauf, das Ergebnis, identifiziert Schwachstellen und Verbesserungspotentiale. Es beschließt konkrete Verbesserungsmaßnahmen, die möglichst direkt umgesetzt werden sollen. Das Ziel ist die Verbesserung der Zusammenarbeit innerhalb des Teams und die Optimierung des Prozesses.

2.1.3 Verbreitung agiler Methoden

Nach einer internationalen Studie der Hochschule Koblenz, in der etwa 600 Personen befragt wurden, ist Scrum mit 86% die meistgenutzte agile Methode. Danach folgen Kanban, Extreme Programming und Feature Driven Development (vgl. Komus, 2014, S. 39 ff.). Die Studie fand außerdem heraus, dass 39% der befragten Anwender Mischformen agiler Methoden anwenden und nur 21% arbeiten durchgehend agil.

Weitere 25% arbeiten in einzelnen Projekten selektiv mit klassischen Methoden und 15% durchgehend klassisch. Fast zwei Drittel der Teilnehmer nutzen agile Methoden erst seit 4 Jahren (vgl. Komus, 2014, S. 17 f.). Es lässt sich also ein starker Zuwachs in der Verbreitung agiler Methoden feststellen, der die Wichtigkeit der Lehre in diesem Bereich unterstreicht. Jedoch muss die steigende Verbreitung nicht bedeuten, dass jedes Unternehmen die agile Softwareentwicklung auch als Wertsystem, als Kultur und Geisteshaltung verstanden hat. Agilität wirkt sich weiter auf ein Unternehmen aus, als nur auf die Entwicklungsabteilung und das macht einen Übergang schwer (vgl. Cohn, 2010, S. 36).

2.1.4 Abgrenzung

Diese Arbeit beschäftigt sich primär mit den verbreiteten agilen Methoden Scrum und Kanban. Darüber hinaus werden einige Entwicklungspraktiken beschrieben, die mit Extreme Programming und Feature Driven Development bekannt wurden und auch im Zusammenhang mit Scrum oder Kanban eingesetzt werden können, zum Beispiel Pair Programming oder Continuous Integration.

2.2 Scrum

Scrum ist ein agiles Prozessframework zur Entwicklung und Erhaltung von komplexer Software. Es beruht auf den Werten des *Manifests für Agile Softwareentwicklung* (2.1.1). Scrum besteht aus Rollen, Ereignissen und Artefakten, die mit wenigen leicht verständlichen Regeln verbunden sind. Zu den Rollen gehört der Product Owner, der das Team zum richtigen Ziel führt, der Scrum Master, der als Team-Trainer für gute Zusammenarbeit sorgt und das Entwicklungsteam, das die technische Kompetenz besitzt und die technischen Entscheidungen trifft. Zu den Ereignissen zählt der Sprint als Arbeitszyklus über wenige Wochen mit definiertem Ziel, das Sprint Planning zur Festlegung des Sprint-Ziels, das Daily Scrum zur täglichen Synchronisierung der Team-Aktivitäten, das Sprint Review als Ermittlung fertiggestellter Ergebnisse und die Sprint Retrospektive für die Team- und Prozessoptimierung. Die Artefakte umfassen das Product Backlog als priorisierte Sammlung aller bekannten Anforderungen, das Sprint Backlog, bestehend aus den Arbeitseinheiten des laufenden Sprints, und dem Produkt-Inkrement, dem zu jeder Zeit funktionsfähigen und auslieferbaren Produkt. (vgl. Schwaber u. Sutherland, 2013)

2.2.1 Empirische Prozesssteuerung

Scrum basiert auf der Theorie der empirischen Prozesssteuerung. Diese fußt auf drei Säulen.

Die erste Säule beschreibt die *Transparenz* (Sichtbarkeit). Es geht darum ein gemeinsames Verständnis über den Prozess, somit über wichtige Aktionen, Beobachtungen und Ereignisse, zu bekommen. Unterstützt wird dies durch eine Prozesssprache, die gemeinsam im Team gebildet und geteilt wird. Durch Transparenz sollen Prozesse verfolgbar und mögliche Fehlinterpretationen ausgeräumt werden.

Die zweite Säule beschreibt die *Überprüfung*. Der Fortschritt des Prozesses soll kontinuierlich überprüft werden. Durch diese Kontrolle sollen Abweichungen festgestellt werden.

Die dritte Säule beschreibt die *Anpassung*. Der Prozess wird angepasst, wenn sich nicht tolerierbare Abweichungen ergeben haben. Diese Anpassungen sollen möglichst schnell durchgeführt werden.

2.2.2 Team

Ein Scrum-Team besteht aus dem Product Owner, dem Scrum Master und dem Entwicklungsteam. Teams, die mit Scrum arbeiten, organisieren sich selbst. Das Team entscheidet somit eigenständig darüber, wie die zu erledigende Arbeit am besten bewältigt werden kann. Das bedeutet auch, dass niemand von außen vorgibt, wie die Arbeit zu erledigen ist. Scrum-Teams liefern Software regelmäßig (iterativ) und stufenweise erweiternd (inkrementell) aus, um Gelegenheiten für Feedback zu maximieren und nützliche Versionen der Software zur Verfügung zu stellen.

2.2.2.1 Product Owner

„Der Product Owner ist für die Wertmaximierung des Produkts sowie die Arbeit des Entwicklungsteams verantwortlich“ (Schwaber u. Sutherland, 2013, S. 5). Er sorgt dafür, dass das Team das richtige Ziel verfolgt. Wie der Product Owner dies umsetzt ist vom Team, Projekt und Organisation abhängig. Einhergehend mit seiner Verantwortung trifft der Product Owner Entscheidungen über die Bearbeitung von Anforderungen durch das Entwicklungsteam, die im Team respektiert werden müssen. Diese Entscheidungen fließen in das Product Backlog (2.2.4.1) ein.

2.2.2.1.1 Aufgaben

Die primäre Aufgabe des Product Owners ist die Verwaltung und Priorisierung des Product Backlogs, in der die Anforderungen des Kunden in Form von User Stories (2.2.4.1.2) dokumentiert sind. Die Priorisierung entspricht der Reihenfolge, in der das Entwicklungsteam die Aufgaben nach dem Pull-Prinzip abholen und bearbeiten kann (nicht muss). Er formuliert neue Einträge im Product Backlog, entfernt wertlose, detailliert bestehende, wenn sie weiter oben in der Priorität stehen. Zusammen mit dem Entwicklungsteam und dem Kunden entwickelt er eine Produktvision, die sich in den Backlog-Einträgen widerspiegelt. Mit diesem Vorgehen geht der Product Owner auf die Bedürfnisse des Kunden ein. Nach Pichler (2008, S. 9) ist die Rolle des Product Owners weit mehr als die eines traditionellen Programm-, Produkt- oder Projektmanagers. Die Rolle vereint Produkt- und Projektmanagementaufgaben in sich und ist zugleich fest in die Softwareentwicklung integriert. Das bedeutet auch, dass der Product Owner für das Team verfügbar ist. Er verbringt ausreichend Zeit mit dem Team; Nach Pichler (2008, S. 11) mindestens eine Stunde täglich. Er nimmt an den Scrum-Ereignissen teil, beantwortet offene Fragen zu Product Backlog-Einträgen sowie Produktvision und gibt Feedback zu Arbeitsergebnissen im laufenden Sprint. Er hat immer eine klare Vorstellung von dem Produkt, die er jederzeit mit dem Team teilt. Er ist ebenso in der Lage und bevollmächtigt, nach der Produktvision Entscheidungen zu treffen.

Eine detaillierte Aufgabenliste lässt sich für die Rolle des Product Owners nicht aufstellen, weil sich seine Aufgaben je nach Organisation, Team und Projekt unterscheiden. Zum Beispiel kann es sein, dass er an Rahmenbedingungen im Unternehmen gebunden ist, die ihm in bestimmten Bereichen keine freie Entscheidungsgewalt überlassen. Auch können vertragliche Bedingungen gesetzt sein, durch die er vermehrt Bedingungen an das Team stellen muss.

2.2.2.1.2 Zeitaufwand

Lernt ein Team die Praktiken von Scrum gerade erst kennen, ist die Produktivität noch gering. Das liegt daran, dass das Team erst lernen muss, wie detailliert Product Backlog-Einträge (2.2.4.1.2) formuliert werden müssen, wie viel in einem Sprint an Arbeit erledigt werden kann und wie sich die Zusammenarbeit im Team organisieren lässt. Im Laufe des Lernprozesses wird das Team produktiver, da es sich kontinuierlich verbessert und Hindernisse beseitigt. Die Aufgabenlast des Product Owners kann daher anfangs gering sein und sich erst im Laufe der Zeit erhöhen, wenn sich mehr Fragen vom Entwicklungsteam ergeben. Der Project Owner kann daher anfangs weitere Aufgaben wie beispielsweise die Entwicklung von Akzeptanztests übernehmen. Erfahrungen in einem Projekt im Rahmen des Informatik-Studiums an der HAW haben gezeigt, dass ein Team auch gemeinsam zusätzliche Aufgaben für den Product Owner

absprechen kann, um die Entwicklung zu unterstützen. So hatte sich ein Product Owner im Konsens mit dem Team zur Aufgabe gemacht, die bestehende Architektur einer Android-Applikation einzelnen Teammitgliedern zu erklären und Fragen dazu zu beantworten.

Durch die undefinierbare Last an Aufgaben des Product Owners kann es auch dazu kommen, das er seine Aufgaben im Rahmen der verfügbaren Zeit nicht mehr erledigen kann. In dem Fall kann er beispielsweise das Formulieren von User Stories an das Entwicklungsteam abgeben (vgl. Schwaber u. Sutherland, 2013, S. 5). Eine weitere Möglichkeit ist ein Product Owner-Team zu bilden und die Aufgaben innerhalb dieses Teams zu verteilen (vgl. Cohn, 2010, S. 158). Wichtig dabei ist, dass die Verantwortung und Autorität weiterhin von nur einer Person ausgeht, denn der „Product Owner ist eine einzelne Person, kein Komitee“ (Schwaber u. Sutherland, 2013, S. 5). Ein Komitee würde die Gefahr entstehen lassen, dass ein Entwicklungsteam auf Entscheidungen warten muss und das soll vermieden werden.

2.2.2.2 Scrum Master

Der Scrum Master agiert als Trainer für das Team und ist „für das Verständnis und die Durchführung von Scrum verantwortlich“ (Schwaber u. Sutherland, 2013, S. 6). Er kümmert sich um die Einhaltung des Scrum-Prozesses bei der Anwendung im Team. Der Scrum Master hilft dem Product Owner das richtige Ziel so effizient wie möglich zu erreichen (vgl. Cohn, 2010, S. 153) indem er Hindernisse, die sich dem Team stellen, aus dem Weg räumt. Er motiviert das Team unter anderem mit Hilfe von Gruppenübungen wie es beispielsweise auch Trainer in Sportvereinen tun. Er setzt dafür Übungen für das bessere Verständnis von Scrum ein oder trainiert Eigenschaften im Team, die beispielsweise zu einer besseren Kommunikation untereinander führen sollen oder führt zusammen mit dem Team neue Entwicklungspraktiken ein. Auch beim Scrum Master unterscheiden sich somit seine Aktivitäten je nach Team und Projekt. Der Scrum Master arbeitet durchgehend eng mit dem Team zusammen und hat seinen Schreibtisch daher im Regelfall im Teambereich (vgl. Pichler, 2008, S. 20).
Die Autorität des Scrum Masters ist jedoch begrenzt. Er kann das Team nicht dazu zwingen bestimmte Übungen durchzuführen. Er erinnert das Team stattdessen an seine Ziele und lässt sich durch seine Unterstützung die Autorität vom Team selbst übertragen. (vgl. Cohn, 2010, S. 145 f.)

Ein *guter Scrum Master* übernimmt die Verantwortung, den Durchsatz eines Teams zu erhöhen und bei der Einführung von Scrum zu helfen. Er erkennt die Werte der Teammitglieder an und etabliert Konsens im Team. Er fördert die Zusammenarbeit durch Schaffung einer angenehmen Atmosphäre, in der sich das Team traut über Probleme offen zu sprechen. Er ist engagiert,

die Probleme und Hindernisse zeitnah zu beseitigen, auch wenn er manchmal später in den Feierabend gehen muss. Er nimmt Einfluss auf das Team, indem er zum Beispiel neue Entwicklungspraktiken mit dem Team ausprobiert. Er schützt das Team vor Überlastung, indem er sich gegen Product Owner, die immer mehr in immer kürzerer Zeit wollen, zur Wehr setzt. Neben dem Wissen und Erfahrungen mit Scrum hat der Scrum Master auch technisches, betriebswirtschaftliches und sonstiges Fachwissen, das dem Team hilft die technischen Probleme zu erkennen und zu beheben. (vgl. Cohn, 2010, S. 147 f.)

Bei einem Team, das gerade Scrum lernt und damit das erste Projekt durchführt, ist der Scrum Master durchaus ein Vollzeit-Job. Anfangs gibt es erfahrungsgemäß noch viele Hindernisse und Widerstände. Mit der Zeit ändert sich dies und der Scrum Master beseitigt nur noch wiederkehrende Hindernisse, während das Team durch eigene Verbesserung immer schneller wird und mehr Zeit vom Product Owner für Fragen beansprucht. Mit der gewonnenen Zeit und Erfahrung kann sich der Scrum Master um weitere Teams im Unternehmen kümmern. (vgl. Cohn, 2010, S. 156 f.; Pichler, 2008, S. 19 f.)

2.2.2.3 Entwicklungsteam

Das Entwicklungsteam setzt Anforderungen um und liefert zum Ende eines Sprints (2.2.3.1) ein auslieferbares Produkt-Inkrement. Wie schon das Scrum-Team ist auch das Entwicklungsteam für sich selbstorganisiert. Sie haben die technische Kompetenz und entscheiden, wie aus den Anforderungen ein potentiell auslieferbares Inkrement entstehen soll. Entwicklungsteams arbeiten außerdem interdisziplinär. Das Entwicklungsteam verfügt somit von der Zusammenstellung an über alle Kompetenzen um ihr Arbeitsergebnis zu erreichen ohne dabei von anderen abhängig zu sein.

Ihre Fähigkeiten und Herangehensweisen an Probleme sollten unter den Teammitgliedern möglichst gestreut sein, damit Kreativität und Lernbereitschaft gefördert wird. Denn wenn alle im Team die selben Herangehensweisen gewohnt sind, wird die Umsetzung nicht mehr hinterfragt. Es wird niemand die Problematik aus einem anderen Blickwinkel durchdenken. (vgl. Cohn, 2010, S. 239 f.)

Das Entwicklungsteam ist außerdem bevollmächtigt zu entscheiden, welche und wie viele Anforderungen innerhalb des nächsten Sprints für ein Produkt-Inkrement umgesetzt werden können. Es legt für sich die Menge an Arbeit fest, die es im Sprint zuverlässig erledigen kann (Pull-Prinzip). Das Entwicklungsteam muss auch *Nein* zu schwierigen Anforderungen sagen können, die sich nicht mehr im Sprint umsetzen lassen.

Nach Schwaber u. Sutherland (vgl. 2013, S. 6) trägt ausnahmslos jedes Mitglied des Entwicklungsteams den Titel *Entwickler*. Es dürfen auch keine weiteren Unterteilungen innerhalb des Entwicklungsteams etabliert werden, wie beispielsweise *Backend* oder *Frontend*. Entwickler können zwar in bestimmten Gebieten über spezielle Fähigkeiten verfügen, jedoch soll die Rechenschaftspflicht beim gesamten Entwicklungsteam liegen. Es sollen keine Verantwortlichkeiten anhand von Titeln oder Unterteilungen erkennbar gemacht werden.

Die Mitglieder des Entwicklungsteams sollten ihre Arbeitsplätze in räumlicher Nähe zueinander haben, damit enge und spontane Kommunikation möglich ist. Dies erhöht die Produktivität und begünstigt die Gruppendynamik (vgl. Pichler, 2008, S. 17).

Das Entwicklungsteam sollte nach Scrum-Guide (Schwaber u. Sutherland, 2013, S. 6) aus drei bis neun Entwicklern bestehen. Bei weniger als drei Personen soll sich die Interaktion reduzieren und die Produktivität zu gering steigern lassen. Außerdem kann es vorkommen, dass zu kleine Entwicklungsteams kein Produkt-Inkrement liefern können. Unter Umständen auch deswegen, weil nicht genug benötigte Fähigkeiten zur Verfügung stehen. Bei zu großen Teams wird hingegen die Koordination zu aufwendig und komplex. Zusammen mit den anderen beiden Rollen besteht ein Scrum-Team somit aus fünf bis elf Personen.

2.2.3 Ereignisse

Scrum besteht aus einem Kern-Ereignis, dem *Sprint*, der eine Iteration über maximal 4 Wochen darstellt. Jeder Sprint enthält vier Meetings, die zeitlich begrenzt sind. Das *Daily Scrum* wird an jedem Arbeitstag durchgeführt. Das *Sprint Planning* wird am Anfang eines Sprints abgehalten und startet gleichzeitig den Sprint. Das *Sprint Review* und die *Sprint Retrospektive* werden am Ende eines Sprints abgehalten und schließen den Sprint ab. Der nächste Sprint beginnt sofort im Anschluss an die Sprint Retrospektive - siehe dazu Abbildung 2.1.

Die Ereignisse innerhalb eines Sprints sind an die empirische Prozesssteuerung gekoppelt und sollen an den kritischen Stellen *Transparenz* und *Überprüfung* ermöglichen. Ergebnisse stellen somit die Gelegenheit dar, den Ist-Zustand zu überprüfen und entsprechend anzupassen (vgl. Schwaber u. Sutherland, 2013, S. 8).

2.2.3.1 Sprint

Der Sprint stellt eine maximal vierwöchige *Iteration* dar, in der Anforderungen in ein Produkt-Inkrement umgewandelt werden. Anders als das Wort *Sprint* vermuten lässt, soll die Durchführung ohne Stress der Teammitglieder passieren, sondern entspannt, fokussiert und produktiv

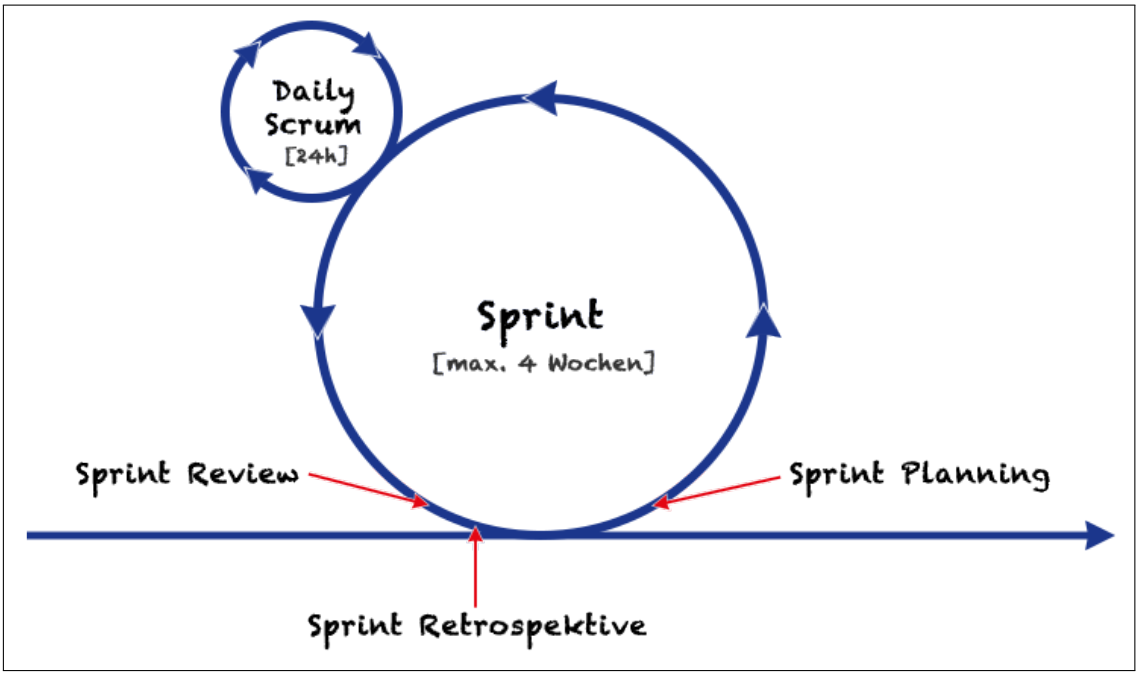

Abbildung 2.1: Übersicht der Scrum-Ereignisse

(vgl. Pichler, 2008, S. 81). Nach dem Scrum-Guide (Schwaber u. Sutherland, 2013, S. 6) sollte jeder Sprint von Beginn eines Projektes bis zum Abschluss die gleiche Dauer haben. Außerdem kann jeder Sprint wie ein eigenständiges Projekt verstanden werden. Es gibt daher immer ein festgelegtes Ziel - ein Produkt-Inkrement mit definiertem Leistungsumfang. In einem Sprint darf keine Änderung vorgenommen werden, die dieses Ziel gefährdet oder den Qualitätsanspruch verringert. Auch personelle Änderungen am Team sind nur an den Grenzen des Sprints erlaubt. So können neue Mitglieder des Teams an den drei Ereignissen mit Meeting-Charakter teilnehmen und erhalten damit ein Projekt-Verständnis. Das Team kann sich so auf die neue Situation einstellen und Einarbeitungszeiten einplanen.

Anforderungen, die zu Beginn im *Sprint Planning* für die Umsetzung vorgesehen wurden, können während des Sprints noch mit dem Product Owner neu ausgehandelt werden, wenn sich neue Erkenntnisse ergeben. Zum Beispiel wenn das Entwicklungsteam merkt, dass einige Anforderungen nicht mehr umgesetzt werden können, kann das Team diese in Absprache mit dem Product Owner auch während des Sprints wieder abgeben. Das erlaubt dem Product Owner, die abgegebenen Anforderung(en) erneut zu priorisieren oder in mehrere Teile zu splitten. Sollte das Entwicklungsteam die eingeplanten Anforderungen des Sprints verfrüht abgearbeitet

haben, können auch während des Sprints neue Anforderungen durch das Entwicklungsteam nach dem Pull-Prinzip in den laufenden Sprint geholt werden.

Ein Sprint kann auch abgebrochen werden, wenn die Fortführung keinen Sinn mehr macht. Dies ist der Fall wenn beispielsweise ein Team nicht in der Lage ist das Sprint-Ziel zu erreichen oder wenn sich die Zielrichtung des Unternehmens ändert. Der Abbruch des Sprints kann vom Product Owner und vom Entwicklungsteam eingeleitet werden. Es wird im Anschluss das Sprint Review durchgeführt, um die Arbeitsergebnisse zu dokumentieren. Außerdem findet anschließend die Sprint Retrospektive statt, in der die Gründe für den Sprint-Abbruch beseitigt werden. Sprint-Abbrüche sollten vermieden werden, da sie Ressourcen verbrauchen und sich demotivierend auf das Team auswirken. (vgl. Schwaber u. Sutherland, 2013, S. 8 f.)

2.2.3.2 Sprint Planning

Im Sprint Planning wird die Arbeit für den kommenden Sprint geplant und im Sprint Backlog (2.2.4.2) dokumentiert. Am Sprint Planning nimmt das gesamte Scrum-Team teil. Es ist zeitlich auf 2 Stunden für jede Woche des Sprints begrenzt. Zum Beispiel dürfte bei einem dreiwöchigen Sprint das Sprint Planning maximal 6 Stunden dauern. Der Scrum Master kümmert sich um das Stattfinden des Ereignisses, um die Einhaltung der zeitlichen Begrenzung und moderiert die Sitzung (vgl. Pichler, 2008, S. 93). Das Sprint Planning ist in zwei Phasen geteilt, die jeweils einer Leitfrage folgen:

2.2.3.2.1 Phase 1 *Was kann in diesem Sprint fertiggestellt werden?*

Der Product Owner hat vor Beginn des Sprint Plannings das Product Backlog (2.2.4.1) für dieses Ereignis vorbereitet. Es liegen fein granulierte User Stories priorisiert, detailliert und abgeschätzt im Backlog bereit, die vom Entwicklungsteam abgeholt werden können. Für die Abschätzung und Detaillierung kann der Product Owner auch das Entwicklungsteam um Hilfe bitten. Die vom Product Owner für diesen Sprint vorgesehenen User Stories richten sich nach einem Sprint-Ziel, einer Vorstellung, wie das Produkt-Inkrement am Ende des Sprint aussehen sollte und welchen Nutzen es mit sich bringt. Der Nutzen muss nicht nur aus beobachtbaren Features bestehen. Auch eine gemeinsame Datenzugriffsschicht kann beispielsweise einen Nutzen darstellen (vgl. Cohn, 2010, S. 294). Nach Pichler (2008, S. 88) sollte dieses Sprint-Ziel mit dem gesamten Scrum-Team gemeinsam formuliert werden. Auf diese Weise etabliert sich das Ziel bei allen Beteiligten und es wird realistischer und besser verständlich. Das vereinbarte Sprint-Ziel sollte gut sichtbar festgehalten werden, beispielsweise auf einem Flipchart (vgl. Pichler, 2008, S. 94).

Gemeinsam mit dem Sprint-Ziel stellt der Product Owner jede User Story vor und erarbeitet ein Verständnis über den Arbeitsinhalt mit dem Team. Es sollten mehr User Stories vorbereitet sein, als das Entwicklungsteam in dem Sprint umsetzen kann, damit es genug Auswahl beim Abholen gibt (vgl. Pichler, 2008, S. 89). Auf diese Weise kann die Kapazität des Entwicklungsteams besser ausgeschöpft werden, wenn beispielsweise größere User Stories nicht mehr in den Sprint passen. Die Kapazität beziehungsweise die Entwicklungsgeschwindigkeit, auch *Velocity* genannt, ist auf einen Sprint bezogen und wird aus voran gegangenen Sprints errechnet. Sie beschreibt den Durchsatz an Anforderungen pro Sprint. Für den ersten Sprint muss die Kapazität noch geschätzt werden. Das letzte Wort für das Abholen an User Stories in den Sprint hat das Entwicklungsteam, da es beurteilen kann, was in der verfügbaren Zeit machbar ist.

2.2.3.2.2 Phase 2 *Wie wird die ausgewählte Arbeit erledigt?*

Mit Beginn der 2. Phase ist eine Auswahl an User Stories, die im aktuellen Sprint umgesetzt werden sollen, aus dem Product Backlog in den Sprint Backlog des Entwicklungsteams übertragen worden. Für jede User Story im Sprint Backlog werden allein durch das Entwicklungsteam alle Aktivitäten beziehungsweise *Tasks* ermittelt, die für die Umsetzung erforderlich sind, typischerweise Design-, Implementierungs-, Test- und Dokumentationsaufgaben. Das Team muss auch berücksichtigen, wenn in Paaren entwickelt wird und wenn Integrationsaufgaben zum Ende des Sprints notwendig sind. Auch während der 2. Phase können in Zusammenarbeit mit dem Product Owner weitere User Stories aus dem Backlog geholt oder zurück gelegt werden, wenn sich neue Erkenntnisse bezüglich der Kapazität des Teams ergeben. Generell steht der Product Owner auch in der 2. Phase für Fragen und Kompromisse zur Verfügung. Bei Bedarf dürfen auch andere Teilnehmer am Sprint Planning teilnehmen, wenn das Entwicklungsteam technische oder fachliche Unterstützung benötigt. (vgl. Schwaber u. Sutherland, 2013, S. 10)

Am Ende des Sprint Plannings enthält der Sprint Backlog neben der Funktionalität, die im Sprint fertiggestellt werden sollte, idealerweise auch den Umsetzungsplan mit geschätztem Aufwand. Hilfreich ist es im Sprint Planning, die Tasks auf Moderationskarten (Story Cards) zu notieren und auf dem Tisch oder Wand für alle sichtbar zu ordnen und zu ergänzen (vgl. Pichler, 2008, S. 95).

Das Sprint Planning endet wenn alle User Stories in Tasks zerlegt wurden oder wenn die Zeit abgelaufen ist. Jeder Task sollte einen maximalen Aufwand von einem Tag enthalten. Es müssen nicht alle elementaren Tasks vorliegen. Wichtig ist, wenn das Team abschätzen kann, welche Aufgaben im Sprint erledigt werden können (vgl. Schwaber u. Sutherland, 2013, S. 10). Das Entwicklungsteam gibt anhand der Aufwandsschätzungen eine Zusage für die Umsetzung

der Aufgaben im Sprint Backlog (vgl. Cohn, 2010, S. 331). Rückgaben von User Stories an das Product Backlog sind im Sprint nur über Absprache mit dem Product Owner möglich.

2.2.3.3 Daily Scrum

Das Daily Scrum ist ein Ereignis, das an jedem Arbeitstag zur selben Zeit am selben Ort in maximal 15 Minuten durchgeführt wird. Es dient dem Entwicklungsteam seine Aktivitäten zu synchronisieren indem die Arbeit seit dem letzten Daily Scrum überprüft und die 24 Stunden bis zum nächsten Daily Scrum geplant werden. Das Daily Scrum soll die Kommunikation verbessern, Hindernisse identifizieren, eine schnelle Entscheidungsfindung fördern und den Wissensstand des Entwicklungsteams erhöhen. Um dies in der kurzen Zeit zu gewährleisten ist die Besprechung in drei Fragen strukturiert, die jeder Teilnehmer aus dem Team beantwortet:

1. Was habe ich seit dem letzten Daily Scrum erreicht, das dem Entwicklungsteam hilft, das Sprint-Ziel zu erreichen?
2. Was werde ich heute erledigen, um dem Entwicklungsteam bei der Erreichung des Sprint-Ziels zu helfen?
3. Sehe ich irgendwelche Hindernisse, die mich oder das Entwicklungsteam vom Erreichen des Ziels abhalten?

(Schwaber u. Sutherland, 2013, S. 11)

Am Daily Scrum nimmt das Entwicklungsteam, der Scrum Master und idealerweise der Product Owner teil. Als Zuhörer dürfen auch andere Stakeholder dabei sein. Aufgabe des Scrum Masters ist die Einhaltung der 15 minütigen Begrenzung und die Moderation der Besprechung. Im Daily Scrum sollen keine Probleme gelöst, Anpassungen oder Umplanungen im Detail besprochen sondern nur aufgezeigt werden. Für umfangreichere Absprachen kann ein separates Treffen mit den betroffenen Personen vereinbart werden. (vgl. Pichler, 2008, S. 105)

In der Praxis wird das Daily Scrum häufig als Stand-up Meeting durchgeführt. Dabei stehen die Teilnehmer in einem Kreis zueinander oder im Halbkreis nahe einer (Stell-)Wand mit dem Sprint Backlog (2.2.4.2). Die Besprechung im Stehen durchzuführen soll die Teilnehmern dabei unterstützen ihre Antworten kürzer anzusprechen als sie es bei Meetings im Sitzen tun würden. (vgl. Bleek u. Wolf, 2011, S. 88; Pichler, 2008, S. 106)

2.2.3.4 Sprint Review

Das Sprint Review findet zum Ende eines Sprints statt und dient der Überprüfung des Produkt-Inkrements. Anhand dessen wird der Projektfortschritt im Product Backlog festgehalten. Zeitlich ist das Ereignis, ebenso wie das Sprint Planning, auf 2 Stunden für jede Woche des Sprints begrenzt.

Am Sprint Review nimmt das gesamte Scrum Team und weitere durch den Product Owner eingeladene Stakeholder teil, um zu prüfen, ob das Team alle im Sprint Planning festgelegten Anforderungen erfolgreich umgesetzt hat. Der Scrum Master ist verantwortlich für das Stattfinden, die Einhaltung der zeitlichen Begrenzung und die Moderation des Ereignisses. Das Entwicklungsteam präsentiert alle *fertigen* Anforderungen, spricht aufgetauchte Probleme mit ihrer gefundenen Lösung an und beantwortet Fragen zum Produkt-Inkrement. Der Product Owner sollte aktiv teilnehmen und zur Überprüfung der Anforderungen auch eigene Tests auf einer Testumgebung mit laufendem Produkt-Inkrement durchführen (vgl. Pichler, 2008, S. 108).

Entsprechen Anforderungen nicht der *Definition of Done* (2.2.5), dann werden diese wieder in das Product Backlog übernommen, damit sie in einem Sprint Planning erneut abgeholt werden können. Die Entscheidung, ob Ergebnisse akzeptiert und abgeschlossen werden oder zurück ins Product Backlog wandern, trifft allein der Product Owner. Verantwortlichkeiten für fehlerhafte oder unfertige Anforderungen trägt das gesamte Team kollektiv, unabhängig von den Personen, die an einzelnen Features gearbeitet haben (vgl. Pichler, 2008, S. 108).

Sind alle Anforderungen vorgestellt und das Product Backlog aktualisiert, wird gemeinsam auf das nächste Sprint Planning hingearbeitet. Dazu stellt der Product Owner den aktuellen Stand des Product Backlogs vor. Alle Teilnehmer erarbeiten gemeinsam, welche User Stories als nächstes relevant sind und von dem Product Owner für das nächste Sprint Planning vorbereitet werden sollten. Nach Pichler (2008, S. 108) sollten fehlerhafte und unfertige Anforderungen stets für den nächsten Sprint vorbereitet und später auch bevorzugt abgearbeitet werden, damit möglichst wenige unfertige Funktionalitäten im Produkt-Inkrement enthalten sind. Das Ergebnis des Sprint Reviews ist das überarbeitete Product Backlog mit allen User Stories, die im nächsten Sprint Planning durch das Entwicklungsteam nach dem Pull-Prinzip abgeholt werden können. (vgl. Schwaber u. Sutherland, 2013, S. 12)

2.2.3.5 Sprint Retrospektive

Die Sprint Retrospektive ist das letzte Ereignis in einem Sprint und schließt diesen ab. Die Sprint Retrospektive findet zwischen dem Sprint Review und dem Sprint Planning statt. Sie

dient der stetigen Optimierung der Zusammenarbeit des Scrum Teams und der verbesserten Anwendung des Prozesses. Die zeitliche Begrenzung liegt bei 45 Minuten für jede Woche des Sprints. Damit dürfte bei einem zweiwöchigen Sprint die Retrospektive maximal 90 Minuten dauern. (vgl. Schwaber u. Sutherland, 2013, S. 12)

An der Sprint Retrospektive nimmt das gesamte Team teil. Der Scrum Master kümmert sich um das Stattfinden, die Einhaltung der zeitlichen Begrenzung, die Vorbereitung der Inhalte, das Verständnis der Teilnehmer und die Moderation des Ereignisses. (vgl. Pichler, 2008, S. 112)

Das Ziel der Sprint Retrospektive ist eine Definition von Veränderungen im Team für die nächsten Sprints. Dies wird durch Überprüfung des vergangenen Sprints in Bezug auf die beteiligten Personen, Beziehungen, Prozesse und Werkzeuge erreicht. Dafür werden Elemente, die verbesserungswürdig oder besonders gut gelaufen sind identifiziert und priorisiert. Positive Elemente sollen verstärkt und negative verändert werden. Darüber hinaus wird die *Definition of Done* (2.2.5) zur Verbesserung der Produktqualität angepasst. (vgl. Schwaber u. Sutherland, 2013, S. 12)

In der Sprint Retrospektive muss das Team offen und ehrlich über seine Zusammenarbeit sprechen. Das fällt nicht immer leicht, weil die Teilnehmer eventuell auch Dinge ansprechen müssen, die sie angreifbar und verletzlich machen, um die Zusammenarbeit zu verbessern. Die Offenheit der Teilnehmer muss durch gute Vorbereitung, der Schaffung einer harmonischen Atmosphäre und guter Moderation herbeigeführt werden. Derby u. Larsen (2006) etablierten aus diesem Wissen heraus eine Struktur aus fünf Phasen für Retrospektiven:

1. Set the Stage
: Das Stimmungsbild der Teilnehmer wird abgefragt und jeder Teilnehmer kommt zu Wort. Ziel ist dafür zu sorgen, dass sich alle Beteiligten der Sprint Retrospektive öffnen und die Projektarbeit hinter sich lassen.

2. Gather Data
: Beobachtungen, nicht erklärbare Phänomene, offensichtliche Fakten und Symptome auf der Prozess- und Gefühlsebene werden vom Team benannt und im Anschluss reduziert und gegebenenfalls priorisiert.

3. Generate Insights
: Verständnis für Ursache und Umstände von Beobachtungen aus der *Gather Data*-Phase werden erarbeitet.

4. Decide What to Do

Handlungsoptionen auf Basis des erarbeiteten Verständnisses werden diskutiert und in möglichst wenigen Vereinbarungen formuliert.

5. Closing the Retrospective

Das Stimmungsbild der Teilnehmer in Bezug auf die Retrospektive wird abgefragt und Feedback eingeholt um die nächsten Retrospektiven zu verbessern.

(vgl. Derby u. Larsen, 2006, S. 5-14; Andresen, 2014, S. 38-51; Pichler, 2008, S. 113 ff.)

Andresen (2014, S. 37 f.) empfiehlt die Vorschaltung einer weiteren kurzen Phase als Einstimmung auf die Sprint Retrospektive: Die *Intro*-Phase. Das Intro besteht aus der Agenda, der Vegas-Regel, der obersten Direktive, der Beschlusskontrolle und weiteren Teamregeln. Mit der Vegas-Regel (*„What happens in Vegas, stays in Vegas"*) verpflichten sich alle Beteiligten nicht über die Inhalte der Sprint Retrospektive mit Außenstehenden zu sprechen. Mit der *obersten Direktive* bekunden alle gemeinsam die Überzeugung, dass jeder Teilnehmer zu jedem Zeitpunkt gewissenhaft handelte, um die Lösungsfindung zu fördern und Schuldzuweisungen zu minimieren. Die Beschlusskontrolle zeigt nochmal die Beschlüsse der letzten Sprint Retrospektive, die im stillen reflektiert aber nicht diskutiert werden.

Mit dieser Phasen-Struktur lässt sich eine Sprint Retrospektive mit Methodenbausteinen zusammenstellen. Eine Sammlung von erprobten Methodenbausteinen für jede Phase findet sich in Baldauf (2014) und Andresen (2014). Bei der richtigen Auswahl der Methoden sollte die Entwicklungsphase des Teams nach Bruce Tuckman (1965) beachtet werden. Tuckman beschrieb die Entwicklung eines Teams mit fünf Phasen: *Forming* ist geprägt von Unsicherheiten, *Storming* von Konflikten, *Norming* von Gesprächen und Lösungsfindungen, *Performing* von effizientem Arbeiten und *Adjourning* von Trauer und Unruhe wenn der Projektabschluss naht. (vgl. Andresen, 2014, S. 15-20)

Am Ende der Sprint Retrospektive sollten Verbesserungen für Zusammenarbeit und Produktqualität festgehalten sein, die im nächsten Sprint umgesetzt werden.

2.2.4 Artefakte

Artefakte von Scrum dokumentieren Arbeit und Wert und sollen die Transparenz über Schlüsselinformationen maximieren (vgl. Schwaber u. Sutherland, 2013, S. 13).

2.2.4.1 Product Backlog

Das Product Backlog ist eine priorisierte, sich ändernde Liste von bekannten Anforderungen und Arbeitsergebnissen. Es existiert so lange wie auch das Produkt existiert. Die Verantwortlichkeit über das Product Backlog besitzt der Product Owner, der Inhalte, Zugriffe und Prioritäten von Einträgen verwaltet. Jedes Scrum Projekt hat sein eigenes Product Backlog, das auch *Living Document* genannt wird, da es sich über die gesamte Projektdauer verändert.

Alle Einträge werden als *User Story* (2.2.4.1.2) beschrieben und enthalten als Attribute zumindest Priorität (oder Reihenfolge), Schätzwert, einen weiteren Wert für den Nutzen und wenn mehrere Teams am Projekt beteiligt sind ein weiteres Team-Attribut (vgl. Schwaber u. Sutherland, 2013, S. 13). Der Schätzwert gibt die Schätzung der Größe einer User Story an - nicht die des Aufwands. Für die Schätzung von Einträgen wird in der Regel eine möglichst kleine User Story als Referenz ausgewählt. Die Größe dieser festgelegten Referenz-User Story dient als Maßeinheit für alle anderen User Stories, die noch abzuschätzen sind. Üblicherweise wird die Maßeinheit *Storypoint* genannt. (vgl. Opelt u. a., 2012, S. 18 ff.)

Als Wert für den Nutzen kann der Geschäftswert (*Business Value*) eingesetzt werden, der sich durch den Kunden und dessen Kosteneinsparungen über die beschriebene Funktionalität ergibt (vgl. Opelt u. a., 2012, S. 52).

Zu Beginn eines Projektes enthält das Product Backlog nur die ersten Entwicklungsschritte und Anforderungen, die zumindest gerade genug Einzelheiten enthalten um sie zu schätzen und zu priorisieren. Solche User Stories, die noch sehr grob formuliert und für gewöhnlich noch sehr umfangreich sind, werden auch *Epic* oder *Epos* genannt (vgl. Cohn, 2010, S. 275).

Einträge im Product Backlog, die in naher Zukunft bearbeitet werden sollen, sind möglichst kurz und detailliert beschrieben, damit sie innerhalb eines Sprints vollständig implementiert werden können. Je weiter die angedachte Bearbeitung eines Eintrags in der Zukunft liegt und damit auch die Priorität niedriger ist, desto größer und ungenauer wird der Detailgrad und die Schätzung. Beim Prozess der Verfeinerung von umfangreichen User Stories werden Ergänzungen zu Einträgen hinzugefügt oder entfernt, Schätzungen erstellt und Prioritäten geändert. Außerdem müssen umfangreiche User Stories und vor allem Epen gegebenenfalls mehrfach aufgeteilt werden. (vgl. Cohn, 2010, S. 275 ff.)

Zur Aufteilung der User Stories können unter anderem die von Lawrence (2009) veröffentlichten *Patterns for Splitting User Stories* genutzt werden. Die Verfeinerung findet kontinuierlich statt und kann vom Product Owner selbst, von ihm weiter delegiert oder vom gesamten Team gemeinsam durchgeführt werden. Einträge im Product Backlog, die detailliert und klein sind,

so dass sie in einem Sprint Planning abgeholt werden können, erhalten den Status *ready*. User Stories müssen nicht vollständig verstanden sein um sie als *ready* für einen Sprint zu bezeichnen. Ausreichend verstandene User Stories reichen, um sie in einen Sprint zu übernehmen. Die Teammitglieder führen während des Sprints weitere Gespräche, wenn sich noch Unklarheiten ergeben. (vgl. Cohn, 2010, S. 297 f.)

Schätzungen werden vom Entwicklungsteam durchgeführt - immer von denen, die die Arbeit erledigen. Gern *gespielte* Methoden für Schätzungen sind *Planning Poker* (vgl. Opelt u. a., 2012, S. 20) oder bei großen Teams *Magic Estimation* (vgl. Opelt u. a., 2012, S. 49), die im Kapitel Lehrmethoden für agile Softwareentwicklung im Detail beschrieben sind.

Aufwände für Verfeinerungen sollten beim Entwicklungsteam nicht mehr als 10% der verfügbaren Zeit beanspruchen (vgl. Cohn, 2010, S. 296; Schwaber u. Sutherland, 2013, S. 14). Wie unter 2.2.3.1 Sprint schon angesprochen, kann anhand der Menge an User Stories, die ein Entwicklungsteam pro Sprint umsetzt, auf den Durchsatz beziehungsweise die *Velocity* geschlossen werden. Die Storypoints der im Sprint abgeschlossenen User Stories werden dabei aufsummiert und ergeben den gemessenen Wert der Velocity. Haben alle Einträge im Product Backlog eine geschätzte Größe in Storypoints, dann ist die gemessene Velocity eine genaue Möglichkeit um zu bestimmen, wann zukünftige Sprint-Ziele erreicht werden. (vgl. Opelt u. a., 2012, S. 18)

In jedem Sprint Review wird diese Rechnung durch den Product Owner durchgeführt, um den Fortschritt der Arbeiten festzustellen und im Verhältnis zur restlichen Zeit zu begutachten (vgl. Schwaber u. Sutherland, 2013, S. 14).

2.2.4.1.1 DEEP

Das Akronym *DEEP* beschreibt zusammenfassend die Eigenschaften eines guten Product Backlogs.

Detailed Appropriately
: User Stories im Product Backlog, die in Kürze bearbeitet werden, sind *angemessen detailliert* und ausreichend verstanden.

Estimated
: Das Product Backlog ist auch ein Planungswerkzeug, daher sind alle Einträge *geschätzt*.

Emergent
: Das Product Backlog *wächst* kontinuierlich, da sich mit der Zeit neue Kenntnisse ergeben, sich User Stories aufteilen und Prioritäten ändern.

Prioritized

Das Product Backlog ist *priorisiert* - die wertvolleren Einträge stehen weiter oben. Das Abholen von User Stories durch das Entwicklungsteam geschieht von oben nach unten. Dadurch entsteht ein maximal wertvolles Produkt.

(vgl. Cohn, 2010, S. 285)

2.2.4.1.2 User Story

Eine User Story ist eine kurze und einfache Beschreibung einer Funktionalität aus der Sicht einer Person, beispielsweise eines Anwenders oder Kunden, der sich die Funktionalität wünscht. Die Formulierung einer User Story folgt einem Muster. Zum Beispiel:

> *Als <Art des Anwenders> wünsche ich <ein Ziel>, damit <ein Grund>*
>
> Als vergesslicher Benutzer wünsche ich in der Lage zu sein, ein neues Kennwort anzufordern, so dass ich nicht endgültig ausgesperrt werde, wenn ich mein altes vergesse. (Cohn, 2010, S. 270)

User Stories sind keine vollständigen Beschreibungen für Funktionalitäten und verpflichten daher zu einem Gespräch mit dem Product Owner, der sie formulierte. Die Kürze erlaubt das Notieren auf „low-tech“ Medien wie Moderationskarten oder Haftnotizen, die sich einfach auf Wänden und Tischen ausbreiten lassen und so die Planung, Diskussion und Aufteilung vereinfachen. (vgl. Cohn, 2010, S. 270 f.)

Ist eine User Story ausreichend verstanden und so klein wie es noch sinnvoll ist, dann erhält sie im letzten Schritt des Verfeinerungsprozesses sogenannte *Akzeptanzkriterien* (auch *Zufriedenheitsbedingungen* genannt). Akzeptanzkriterien entsprechen einem Akzeptanztest auf hoher Ebene und enthalten Erwartungen über Funktionen, die enthalten oder auch nicht enthalten sein sollen. Auf Moderationskarten (Story Cards) werden sie auf die Rückseite geschrieben. (vgl. Cohn, 2010, S. 280)

Auch für die Eigenschaften einer guten User Story existiert ein Akronym: *INVEST*.

Independent

User Stories sind möglichst *unabhängig* voneinander in Bezug auf das Produkt-Inkrement und können im Idealfall einzeln umgesetzt werden. Sind Abhängigkeiten vorhanden, können möglicherweise weitere Aufteilungen Abhilfe schaffen - siehe dazu Lawrence (2009).

Negotiable

Eine User Story bleibt *verhandelbar*, so dass sie selbst im Sprint Planning, kurz vor Beginn der Bearbeitung, noch gemeinsam präzisiert werden kann.

Valueable

Eine User Story sollte *nützlich* sein. Dieser Nutzen beziehungsweise Mehrwert wird auch in der Beschreibung angegeben, denn sonst wäre eine Umsetzung nicht nötig. Im User Story-Muster taucht der Wert im „ein Grund"-Teil auf.

Estimatable

Nur überschaubare, nachvollziehbar formulierte und vom Team verstandene User Stories können auch *abgeschätzt* werden.

Small

Eine *kleine* User Story kann schneller umgesetzt werden und findet damit auch früher im Produkt-Inkrement wieder. Außerdem lassen sich kleinere User Stories besser abschätzen und effizienter in einem Sprint unterbringen.

Testable

Eine User Story besitzt präzise und *testbare* Akzeptanzkriterien, über die sich eine erfolgreiche Realisierung später im Sprint Review beurteilen lässt.

Es sollten zumindest die hoch priorisierten User Stories über diese Eigenschaften verfügen, um mögliche Probleme bei der Abarbeitung zu vermeiden. (vgl. Pichler, 2008, S. 45 f.)

2.2.4.1.3 Epic

Unter einem *Epic* oder *Epos* wird eine umfangreiche User Story verstanden, die durchaus ein bis zwei Sprints zur Abarbeitung umfassen kann. Es gibt keine Größe, ab der eine User Story ein Epic darstellt. Sie verpflichtet nur zu einer Aufteilung in kleinere User Stories, bevor die Bearbeitung stattfinden kann. (vgl. Cohn, 2010, S. 277 f.)

2.2.4.2 Sprint Backlog

Das Sprint Backlog enthält alle für den Sprint ausgewählten Product Backlog-Einträge, für dessen Umsetzung sich das Entwicklungsteam im Sprint Planning (2.2.3.2) verpflichtet hat. Außerdem enthält es zu jeder User Story die untergeordneten Aufgaben (oder *Tasks*), die in Personenstunden geschätzt werden. Ein Task sollte den Aufwand von einem Arbeitstag nicht überschreiten. (vgl. Pichler, 2008, S. 102)

Das Sprint Backlog ist somit auch das Planungswerkzeug des Entwicklungsteams, an dem nur das Entwicklungsteam Änderungen vornehmen darf. Gleichzeitig dient es der Transparenz über die Organisation und erforderliche Arbeit auf dem Weg zum Sprint-Ziel. (vgl. Schwaber u. Sutherland, 2013, S. 15)

Genauso wie das Product Backlog ist auch das Sprint Backlog ein *Living Document*, weil es täglich angepasst wird. Es werden jeden Tag Tasks abgeschlossen, aber auch ergänzt, aktualisiert und erneut abgeschätzt, um den Plan immer ausreichend detailliert zu halten. Im Daily Scrum (2.2.3.3) dient das Sprint Backlog als Angabe für den Fortschritt des Sprints, die hilfreich für die Synchronisierung der anstehenden Arbeit ist. (vg. Schwaber u. Sutherland, 2013, S. 15)

Das Sprint-Backlog sollte jederzeit sichtbar sein, damit alle Teammitglieder den Fortschritt verfolgen können und Verantwortungsbewusstsein gefördert wird. Arbeitet das Entwicklungsteam im selben Raum oder Teambereich, dann empfieht sich eine (Stell-)Wand, die für alle erreichbar und sichtbar ist. Auf dieser kann der Plan mit Moderationskarten strukturiert und von mehreren Teammitgliedern gleichzeitig angepasst werden. (vgl. Pichler, 2008, S. 102 f.)

Zu jedem Daily Scrum wird der restliche Aufwand nicht abgeschlossener Tasks aufsummiert und mit der noch verfügbaren Zeit verglichen. Dadurch kann das Entwicklungsteam den Fortschritt ablesen und bei Abweichungen reagieren. (vgl. Schwaber u. Sutherland, 2013, S. 15 f.)

2.2.4.3 Produkt-Inkrement

Das Produkt-Inkrement ist eine potentiell auslieferbare Software, die am Ende eines Sprints zur Verfügung steht. Sie enthält alle bis zu dem Zeitpunkt durch das Entwicklungsteam fertiggestellten und getesteten Anforderungen (vgl. Cohn, 2010, S. 290). Für die Prüfung, ob die Anforderungen tatsächlich vollständig erledigt wurden, legt der Product Owner und das Entwicklungsteam Kriterien, die *Definition of Done* (2.2.5), fest. Diese Vereinbarung wird im ersten *Sprint Planning* des Projektes vorgenommen und für die anschließenden Sprints in der *Sprint Retrospektive*.

Verlässlich funktionierende Produkt-Inkremente an jedem Sprintende zur Verfügung zu stellen ist eine große Herausforderung für das Team. Es ist gleichzeitig eine Verpflichtung, wenn agil gearbeitet wird, denn funktionierende Software stellt eines der Grundwerte im agilen Manifest dar: „*Funktionierende Software* mehr als umfassende Dokumentation“. Um dies zu erreichen muss die Software im Sprint Planning gut geplant werden und zum Ende des Sprints bereits

über eine ausreichende Menge an Tests (besonders Integrationstests) verfügen. Die Software muss nach dem ersten Sprint noch nicht unbedingt zusammenhängend sein, dies lässt sich je nach Projekt erst nach 2 bis 3 Sprints realisieren. Jedoch sollte nach Nutzbarkeit gestrebt werden und nur eine Codebasis vorhanden sein (vgl. Cohn, 2010, S. 291).

Zurückblickend auf das Projekt an der HAW ist ein nutzbares Produkt-Inkrement beispielsweise zu keinem Sprintende gelungen. Es konnten in allen vier Sprints nur einzelne Komponenten zur Verfügung gestellt werden, die nicht allen gemeinsam festgelegten Kriterien entsprachen um als *fertig* bezeichnet zu werden. In der Zusammenarbeit von zwei Scrum-Teams sind außerdem mehrere Versionen der Codebasis entstanden, die sich nur erschwert zusammenführen lassen.

2.2.5 Definition of Done

Die *Definition of Done* ist eine gemeinsame Vereinbarung des Scrum Teams, unter welchen Bedingungen ein Product Backlog-Eintrag oder das Produkt-Inkrement als *fertig* angesehen werden darf (vgl. Schwaber u. Sutherland, 2013, S. 16). Die Vereinbarungen der Definition of Done sind Qualitätsmerkmale, die das Team für das Produkt im Einvernehmen festlegt. Sie ergänzen die Akzeptanzkriterien der User Story insofern, dass sie für jeden Eintrag gleichermaßen gültig sind. Durch die wechselhaften Anforderungen in jedem Sprint ist es auch mit einer Definition of Done nicht immer klar, wann Aufgaben abgeschlossen sind und ob die Vereinbarungen ausreichen. Daher wird die Definition of Done regelmäßig überprüft und angepasst. Dies geschieht in der Sprint Retrospektive (2.2.3.5). (vgl. Andresen, 2014, S. 34)

Die Definition of Done hilft dem Team dabei ein festgelegtes Qualitätsniveau zu erreichen als auch bei der Schätzung der Größe von User Stories. Arbeiten mehrere Scrum Teams an einem Produkt, dann müssen sich alle Teams auf eine gemeinsame Definition of Done einigen, um ein Chaos von Konventionen und Standards zu vermeiden. Die Vereinbarungen bedeuten nicht, dass der Product Owner erledigte Aufgaben abnehmen muss. Im Sprint Review entscheidet er auch bei Einhaltung der Definiton of Done nochmals, ob die fertiggestellten User Stories auch „abgenommen fertig“ sind. (vgl. Eckstein, 2012, S. 90 f.)

2.2.6 Skalierung

Auch bei der Skalierung von Scrum wird das Scrum Framework konsequent eingehalten. Das bedeutet, dass ein Team aus maximal elf Personen bestehen darf, um den Koordinationsaufwand für Kommunikation innerhalb eines Team möglichst gering zu halten (vgl. Eckstein, 2012, S. 41). Daher wird in skalierten Ansätzen von Scrum mit mehreren kleinen Teams gearbeitet. Aber

auch hier können mit der steigenden Anzahl an Teams immer neue Schwierigkeiten auftreten (vgl. Eckstein, 2012, S. 8). Wenn mehrere Scrum Teams gemeinsam an einem Projekt arbeiten, müssen somit die Ereignisse, Artefakte und Rollen darauf angepasst werden.

In der Lehre kann ein skalierter Ansatz von Scrum bei Gruppen von mehr als elf Teilnehmern wahlweise eingebracht werden. Besonders relevant ist so ein Ansatz, wenn Aufgaben gestellt werden, in denen mehrere Teams zusammenarbeiten müssen. Wie eine skalierte Form von Scrum aussehen kann, soll in den folgenden Abschnitten kurz vorgestellt werden. Als Grundlage dienen primär die Werke von Larman u. Vodde (2009) und Cohn (2010).

2.2.6.1 Abgrenzung

Skalierte Ansätze mit verteilten Teams werden in diesem Kapitel nicht betrachtet, weil damit weitere Aspekte und Schwierigkeiten angesprochen werden müssten, die den Rahmen dieser Arbeit sprengen würden. Außerdem wird davon ausgegangen, dass die Lehrmethoden in unweit voneinander getrennten Räumen stattfinden. Informative Literatur zum Thema *Verteilte Entwicklung mit Scrum* findet sich in Cohn (2010, S. 385-431).

2.2.6.2 Chief Product Owner

Die Aufgaben des Product Owners (2.2.2.1.1) in einem klassischen Scrum-Team sind bereits auf einen Vollzeit-Job ausgelegt. Seine zeitintensiven Tätigkeiten beziehen sich auf die Verwaltung von Product Backlog-Einträgen (definieren neuer Features, priorisieren, aktualisieren und entscheiden) und auf seine Verfügbarkeit für das Team zur Beantwortung von Fragen. Bei mehreren Teams werden entsprechend mehr Einträge im Product Backlog entstehen und mehr Zeit durch die Teams beansprucht. Gleichzeitig muss er eine Produktvision über das *gesamte Projekt* vertreten. Dieser Konflikt lässt sich dadurch auflösen, dass jedes Team zumindest ein unterstützendes Teammitglied hat, das die Produktvision teilt und die Aufgaben für das Eintragen von Anforderungen sowie die Verfügbarkeit in den Teams übernimmt. Nach Larman u. Vodde (2009, S. 293 f.) gibt es bis zu einer Größe von 10 Teams weiterhin nur *einen einzigen Product Owner*, der sich auf die Vermittlung der Produktvision und die damit verbundene Priorisierung konzentriert.

Im Modell von Cohn (2010, S. 357 f.) wird die Rolle des *Chief Product Owner* stellvertretend für den *einzigen Product Owner* eingeführt. Den Teams sind *Produktsegmente* zugeordnet, die festgelegte Aufgabenbereiche innerhalb des Produktes übernehmen. Die Aufgaben des unterstützenden Teammitglieds übernimmt hier ein vollwertiger Product Owner, der für diese

Produktsegemente verantwortlich ist und sich mit dem Chief Product Owner im Bezug auf die Produktvision synchronisiert. Der Chief Product Owner lässt sich darüber hinaus hin und wieder bei Team Meetings blicken. Der klassische Product Owner innerhalb der Teams sollte aufgrund seiner Verantwortung und Aufgaben je nach Auslastung *maximal* zwei Teams zugeteilt sein.

Ab einer bestimmen Menge an Teams (bei Larman u. Vodde (2009) die benannten 10) wird eine Zwischenstufe hinzugefügt und so eine Hierarchie von *Product Ownern* eingeführt - siehe Abbildung 2.2.

Larman u. Vodde (2009, S. 298 f.) empfehlen in dieser Größenordnung die Einteilung von *„major requirement areas“*. Eine solche *Area* wird durch einen *Area Product Owner* abgedeckt, der mehrere *Feature Teams* mit jeweils einem unterstützenden Teammitglied verwaltet. Cohn (2010, S. 358 f.) führt in seinem Modell auf den Zwischenstufen den *Product Line Owner* ein. Trotz der Hierarchie sind in beiden Modellen alle Product Owner gemeinsam verantwortlich für das Produkt (vgl. Cohn, 2010, S. 359) und bilden so ein *Product Owner Team* (vgl. Larman u. Vodde, 2009, S. 298). Bei Larman u. Vodde (2009) gehört das unterstützende Teammitglied nicht zum Product Owner Team.

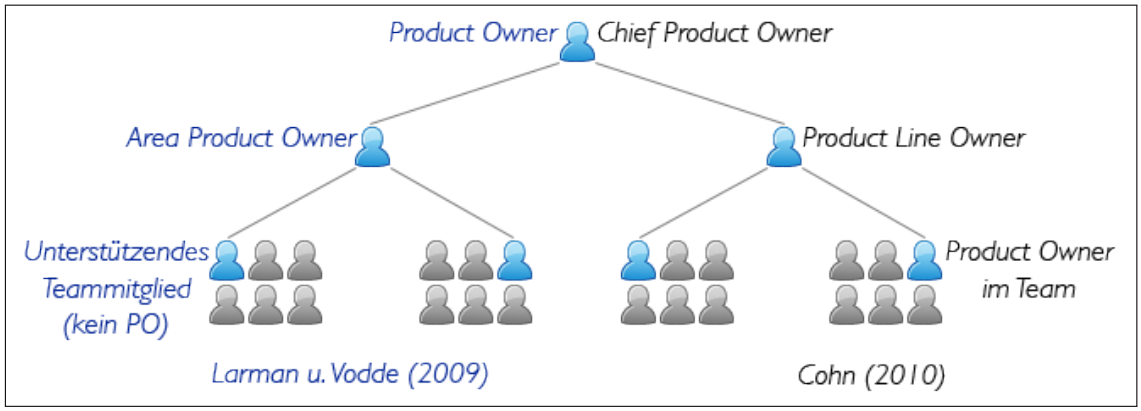

Abbildung 2.2: Modell des *Product Owner Teams* in Hierarchie mit Bezeichnungen (nach Larman u. Vodde (2009) links und Cohn (2010) rechts)

2.2.6.3 Sichten auf ein Product Backlog

Ein Produkt sollte nur *ein Product Backlog* besitzen. Sind mehrere Product Backlogs vorhanden, dann kann es zu abweichend gesetzten Prioritäten zwischen den Teams und Produktsegmenten

kommen. Damit können Wartezeiten durch teamübergreifende Abhängigkeiten benötigter Funktionalitäten entstehen. Außerdem existieren in größeren Projekten auch team- oder produktsegmentübergreifende User Stories, die von den betreffenden Teams gemeinsam im selben Sprint abgearbeitet werden müssen. Auch hier wäre es fatal, wenn die Teams abweichende Prioritäten festlegen. Es ist möglich die Prioritäten zu synchronisieren, jedoch entsteht dadurch je nach Anzahl der Teams und Produktsegemente ein erheblicher Mehraufwand. (vgl. Cohn, 2010, S. 360 f.)

Larman u. Vodde (vgl. 2009, S. 298 f.) und Cohn (vgl. 2010, S. 360 f.) empfehlen daher Sichten auf ein einziges Product Backlog mit globaler Priorisierung. Für jedes Produktsegment (oder bei Larman u. Vodde (2009) *Area* genannt), in denen ein oder mehrere Teams festgelegten Aufgabenbereichen zugeordnet sind, wird eine Sicht auf das Product Backlog angelegt. Eine solche Sicht nennen Larman u. Vodde (vgl. 2009, S. 298) *Area Backlog*. Jeder Eintrag im Product Backlog erhält eine Zuordnung zu ein oder mehreren Sichten, die zu den Aufgabenbereichen der entsprechenden Teams passen. Einträge, die mehreren Teams zugeordnet sind, können die Bedeutung haben, dass sie entweder von einem der Teams oder gemeinsam bearbeitet werden sollen.

Im Umgang mit einem Product Backlog in großen Projekten soll noch erwähnt werden, dass der *Chief Product Owner* je nach Anzahl an Teams über Hunderte von Backlog-Einträgen den Überblick behalten muss. Ab einer bestimmten Menge ist dies unrealistisch. Auch aus diesem Grund unterstützt die Zuordnung zu Sichten und damit zu Produktsegmenten, den Überblick über das Gesamtprojekt zu behalten. Gerade wenn es viele Überschneidungen von Einträgen über mehrere Sichten gibt, kann es außerdem hilfreich sein, die Einträge nach *Themen* zu ordnen. Ein Thema kann ein Schlagwort oder auch das zuerst formulierte *Epic* sein. (vgl. Cohn, 2010, S. 362 f.)

2.2.6.4 Neue Ereignisse

Arbeiten mehrere Teams zusammen, kann das Problem entstehen, die Arbeit dieser Teams zu koordinieren. Das klassische Scrum-Framework soll dabei unverändert bleiben. Schwaber u. Sutherland (2013, S. 17) schreiben dazu im Schlusswort: „Es ist zwar möglich, nur Teile von Scrum einzusetzen - das Ergebnis ist dann aber nicht Scrum“.

Die im Folgenden beschriebenen Ereignisse unterstützen die Koordination von Teams. Sie bieten Gelegenheiten für Transparenz, Überprüfung und Anpassung für die Zusammenarbeit über Teamgrenzen hinaus. Diese Ereignisse sind aber nicht Teil des von Schwaber u. Sutherland

(2013) entwickelten *Scrum Guides* und müssen nicht eingesetzt werden um im Rahmen von Scrum zu bleiben.

2.2.6.4.1 Meta-Scrum (Scrum of Scrums)

Im *Meta-Scrum* Ereignis, auch *Scrum of Scrums* genannt, sollen die Teams untereinander ihre Arbeit besprechen, Entscheidungen treffen und Probleme auf Teamebene lösen. Cohn (vgl. 2010, S. 372) empfiehlt zwei bis drei Meta-Scrum Ereignisse pro Woche, weil die zu treffenden Entscheidungen und zu lösenden Probleme viele der Projektbeteiligten betreffen können. Ausstehende Entscheidungen sollen möglichst zeitnah getroffen werden, um Wartezeiten zu vermeiden. Probleme, die in dieser Besprechung angesprochen werden, sollen so schnell wie möglich gelöst werden. (vgl. Cohn, 2010, S. 370 ff.)

Im klassischen Daily Scrum (2.2.3.3) der einzelnen Teams wird jeweils eine Person durch das Entwicklungsteam ausgewählt, die an den nächsten Meta-Scrum Besprechungen teilnehmen soll. Die ausgewählte Person, die das Team im Meta-Scrum vertritt, kann im Projektverlauf variieren. Wenn wenige Teams am Projekt arbeiten, können auch zwei Personen pro Team für das Meta-Scrum ausgewählt werden. Arbeiten viele Teams am Projekt, dann können rekursiv weitere Ebenen des Meta-Scrum Ereignisses eingeführt werden - siehe Abbildung 2.3. Existieren

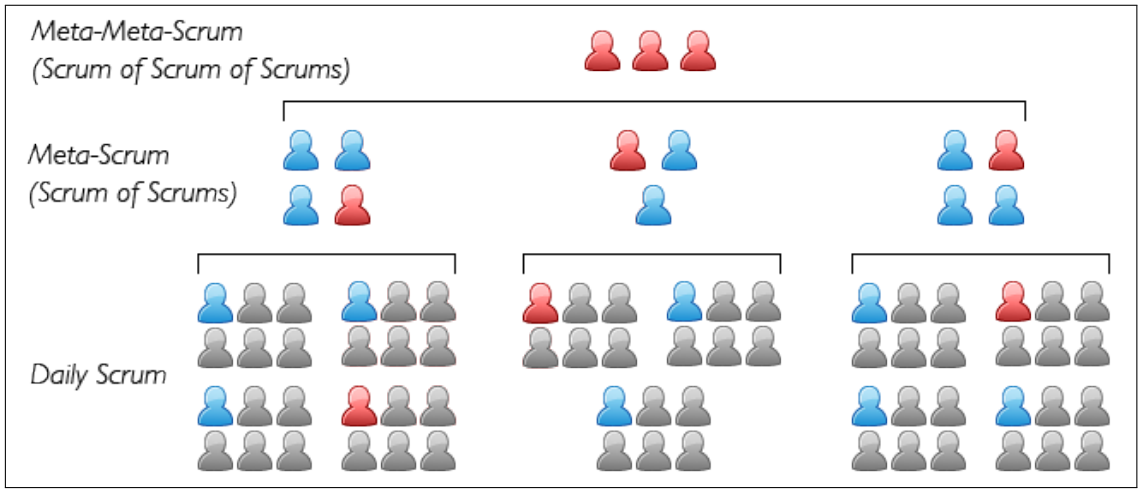

Abbildung 2.3: Beispielhafte Teamzusammenstellung für Meta-Scrum Ereignisse

mehrere Meta-Scrum-Ebenen, dann wird in den jeweiligen Meta-Scrum Besprechungen ein Vertreter für die nächsthöhere Stufe ausgewählt. Das Meta-Scrum kann pro Ebene parallel durchgeführt werden. (Cohn, 2010, S. 370 f.)

Das Ereignis besteht im Ablauf aus zwei Phasen. Die erste Phase ist auf 15 Minuten begrenzt und strukturiert sich mit drei Fragen ähnlich wie das Daily Scrum:

1. Was hat mein Team seit der letzten Besprechung getan, das Einfluss auf die Arbeit anderer Teams nehmen kann?
2. Was wird mein Team bis zur nächsten Besprechung tun, das Einfluss auf die Arbeit anderer Teams nehmen kann?
3. Welchen Problemen sieht sich mein Team gegenüber, bei denen es die Hilfe anderer Teams braucht?

(Cohn, 2010, S. 373)

Die zweite Phase ist optional, nicht begrenzt und beginnt direkt im Anschluss. Probleme, die in der ersten Phase identifiziert wurden beziehungsweise im *Impediment Backlog* (2.2.6.5) der beteiligten Teams hinterlegt sind, werden besprochen und nach Möglichkeit gelöst. Einige der identifizierten Probleme benötigen gegebenenfalls nicht anwesende Teammitglieder und werden daher im Impediment Backlog eingetragen um sie innerhalb der Teams zu lösen. Zu komplexe Probleme, die sich nicht während einer Meta-Scrum Besprechung lösen lassen, werden ebenfalls im Impediment Backlog verzeichnet, um sie erneut aufzugreifen. Die zweite Phase ist nicht begrenzt, damit Probleme, die viele Teammitglieder betreffen, nicht wegen einer zeitlichen Begrenzung aufgeschoben werden müssen. (Cohn, 2010, S. 372 f.)

Nicht in jedem skalierten Projekt ist eine Etablierung von Meta-Scrum nötig. Voraussetzung dafür ist zum einen, dass sich Product Backlog-Einträge eindeutig auf Feature Teams mit abgrenzbaren Aufgabenbereichen zuordnen lassen, die diese Aufgaben unabhängig voneinander bearbeiten können. Zum anderen sollten Integrationaufgaben über Konzepte wie Continuous Integration koordiniert werden. (vgl. Larman u. Vodde, 2009, S. 301).

2.2.6.4.2 Pre-Sprint Planning

Das *Pre-Sprint Planning*, oder nach Eckstein (2012, S. 102) *Vorplanung*, ist ein Ereignis, das vor jedem Sprint Planning vom *Product Owner Team* (2.2.6.2) durchgeführt wird, um die Prioritäten des umfassenden Product Backlogs (inklusive der Sichten) zu koordinieren. Das Treffen findet gewöhnlich in der Mitte eines Sprints, spätestens in der Woche vor dem folgenden Sprint Planning statt. Die Product Owner besprechen die wichtigen Funktionalitäten und Prioritäten jeder Sicht auf das Product Backlog. Sie tauschen Ideen sowie Feedback aus und synchronisieren Anpassungen der umfassenden Produktvision anhand neuer Erkenntnisse aus dem laufenden Sprint. (vgl. Larman u. Vodde, 2009, S. 300; Eckstein, 2012, S. 102 f.)

Nach Eckstein (vgl. 2012, S. 102 f.) werden häufig technisch versierte Personen, wie beispielsweise ein verantwortlicher Architekt des Produktes, zu diesem Ereignis eingeladen, damit dieser auf technische Abhängigkeiten der Funktionalitäten hinweist oder diese auflöst.

2.2.6.4.3 Sprint Planning im Großraumverfahren

Beim Sprint Planning Ereignis innerhalb der Teams kann es zu einigen Problemen bei großen Projekten kommen. Liegt zum Beispiel der Sprint Planning-Termin mehrerer Teams auf dem selben Datum, dann können wichtige Personen wie Softwarearchitekten oder Product Owner fehlen, die in mehreren Sprint Planning Besprechungen gleichzeitig gebraucht werden. Liegen die Termine von Sprint Planning Besprechungen auseinander, dann lässt sich ein vernünftiger Termin für das Pre-Sprint Planning (2.2.6.4.2) vielleicht gar nicht finden, um Prioritäten für die umfassende Produktvision zu setzen. Außerdem können im Sprint Planning entdeckte Abhängigkeiten zu anderen *Feature Teams* dazu führen, dass bestimmte Aufgaben für einen Sprint nicht zugesagt werden können, weil das andere Team ihre Planung bereits abgeschlossen hat. (vgl. Cohn, 2010, S. 375 f.)

Abhängig von der Projektgröße können diese Probleme unterschiedlich gelöst werden. Zunächst lassen sich die Sprints synchronisieren, so dass die möglichen Sprint Planning-Zeitpunkte der Teams auf identische oder um einen Tag versetzte Termine fallen. Identische Termine sind bei weniger großen Projekten hilfreich, in denen alle Teams noch in einen Raum passen, so dass ein Sprint Planning im *Großraumverfahren* durchgeführt werden kann. Bei größeren Projekten, in denen die Raumgrößen dafür nicht ausreichen, sind leicht versetzte Sprint Planning-Termine sinnvoll. Dann können benötigte Personen trotzdem an allen Planungsbesprechungen teilnehmen. Auch mit versetzten Sprint Planning-Terminen lassen sich die Raumgrößen ausnutzen und mehrere Teams, beispielsweise alle eines Produktsegments (vgl. Larman u. Vodde, 2009, S. 300), führen gleichzeitig die Planungsbesprechung durch. (vgl. Cohn, 2010, S. 375 ff.)

In beiden Lösungsmöglichkeiten wird das Sprint Planning im *Großraumverfahren* angewendet. Bei diesem Ereignis zieht sich jedes Team in einem Bereich des Raums zurück und führt ein klassisches Sprint Planning durch. Werden im Verlauf der einzelnen Sprint Planning Besprechungen Abhängigkeiten in Product Backlog-Einträgen zu anderen Teams erkannt, treffen sich einige Personen beider Teams und klären, wie die Aufgaben aufgeteilt und gelöst werden können. Häufig benötigte Personen wie Softwarearchitekten oder Product Owner können über sichtbar platzierte *Signalflaggen* angefordert werden, ohne dabei andere Teams durch *laute Zurufe* zu stören (vgl. Cohn, 2010, S. 376 ff.). Siehe dazu Abbildung 2.4 mit den originalen Signalflaggen aus dem HAW Projekt.

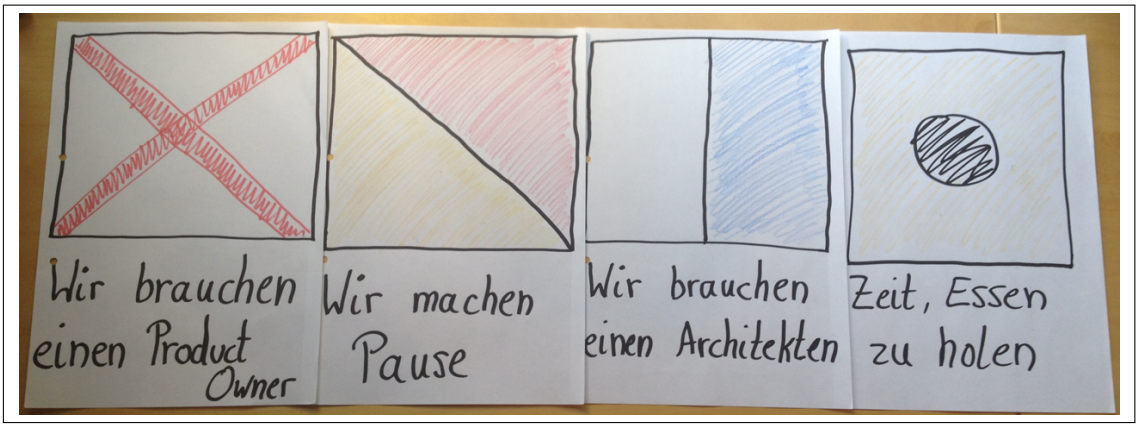

Abbildung 2.4: Über Signalflaggen benötigte Personen anfordern und Pausen kommunizieren

Alternativ zu dem Ablauf von Cohn (2010) können Sprint Planning Ereignisse, die alle Teams eines Produkts oder Produktsegments betreffen, als ein gemeinsames Sprint Planning im klassischen Stil durchgeführt werden, ohne das sich die Teams in Bereiche im Raum zurück ziehen. Während der *ersten Phase* der Besprechung präsentiert der leitende Product Owner die abholbaren Product Backlog-Einträge nach Reihenfolge und die Teams entscheiden, ob sie die aktuelle User Story in ihren Sprint übernehmen möchten und können. Mit Absprachen und Klärungen von Fragen übernimmt ein Team (bei Abhängigkeiten auch mehrere) die entsprechende User Story in ihren Sprint. Ist nicht genug Platz in dem Raum, kann jedes Team zwei Entwickler entsenden. Für die *zweite Phase* begeben sich alle Personen in ihre Teamräume und führen das Sprint Planning wie in Abschnitt 2.2.3.2 Sprint Planning beschrieben zuende. (vgl. Larman u. Vodde, 2009, S. 296)

2.2.6.4.4 Product Backlog Refinement

Verfeinerungen am Product Backlog (Einträge hinzufügen, ergänzen, entfernen, schätzen und priorisieren) werden im nicht skalierten Scrum kontinuierlich durch Product Owner und mittels Delegation auch vom Entwicklungsteam durchgeführt. Larman u. Vodde (vgl. 2009, S. 297) empfehlen in großen Projekten mit mehreren Teams den Prozess der Verfeinerung in ein Ereignis zu verlagern, dem *Product Backlog Refinement*. Dazu treffen sich je nach Projektgröße alle Product Owner mit Teamvertretern oder gar alle Teams (bei genügend Platz) zu einer vierstündigen Besprechung in jedem Sprint.

2.2.6.4.5 Joint Review

Wenn es sinnvoll ist, die Arbeit mehrerer Teams zu begutachten, kann ein *Joint Review* durchgeführt werden. Dies kann beispielsweise über ein Produktsegment (2.2.6.3) geschehen. Dabei kommt das *Product Owner Team* und die Teams des Produktsegments (oder zumindest Vertreter der Teams) zusammen. Im Fokus steht die Präsentation von Features, die für die Mehrheit der Product Owner von Interesse sind oder die für das nächste Release eine entscheidende Bedeutung haben. Darüber hinaus werden die Ergebnisse diskutiert und Verbesserungen abgesprochen. (vgl. Larman u. Vodde, 2009, S. 300)

Das *Joint Review* ist ein separates Ereignis und eignet sich für Projekte mit mehreren Produktsegmenten - für Larman u. Vodde (vgl. 2009, S. 298) ab etwa zehn Teams. Für kleinere Projekte kann auch das bestehende klassische Sprint Review (2.2.3.4) mit mehreren Teams und identischem Ablauf durchgeführt werden. Voraussetzung dafür sind *synchrone Sprints*, wie unter Sprint Planning im Großraumverfahren (2.2.6.4.3) beschrieben.

2.2.6.4.6 Joint Retrospektive

Die *Joint Retrospektive* findet zeitlich nach der klassischen *Sprint Retrospektive* mit wenigen Vertretern der Teams statt. Dabei sollen für den effektiveren Einsatz von Scrum systemrelevante Hindernisse identifiziert und behoben werden. Aus dem Grunde sind besonders Scrum Master als Team-Vertreter am Ereignis beteiligt. Je nach Projektgröße und Nutzen findet die Joint Retrospektive über alle Teams, auf Produktsegment-Ebene und/oder mit verschiedenen Vertretern aus Produktsegmenten statt. Da nur wenige Team-Vertreter an der Joint Retrospektive teilnehmen, kann das Ereignis auch in der ersten Woche des neuen Sprints durchgeführt werden. (vgl. Larman u. Vodde, 2009, S. 297 ff.)

2.2.6.5 Impediment Backlog

Das *Impediment Backlog* ist eine priorisierte Liste von ungelösten Probleme und Hindernissen, die ein Team am effizienten Arbeiten behindern. Ein Impediment Backlog wird pro Team geführt. Es reicht dafür gewöhnlich ein einfacher, technisch anspruchsloser Mechanismus wie beispielsweise ein Flipchart-Papier sichtbar im Teamraum oder ein Tabellenblatt. Das Impediment Backlog ist sinnvoll, wenn sich Probleme sammeln oder nicht unmittelbar lösen lassen. (vgl. Cohn, 2010, S. 372)

Die Anhäufung von Hindernissen muss nicht nur in skalierten Scrum-Projekten der Fall sein. Die Verwendung des Impediment Backlogs findet sich auch im klassischen Scrum wieder (vgl. Komus, 2014, S. 5).

2.2.6.6 Teamübergreifende Mitglieder

Ergeben sich aller Voraussicht nach zwischen mehreren Teams Abhängigkeiten, dann kann es hilfreich sein, einzelne Personen mehreren Teams zuzuordnen, um die Kommunikation

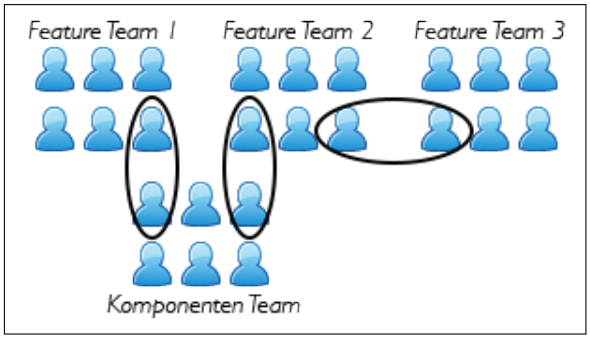

Abbildung 2.5: Beispiel für den Einsatz teamübergreifender Mitglieder bei wahrscheinlichen Abhängigkeiten

beider Teams zu fördern. Dies kann beispielsweise der Fall sein, wenn ein Komponenten Team Schnittstellen für mehrere Feature Teams zur Verfügung stellt - siehe dazu Abbildung 2.5 - oder wenn mehrere Features gleichzeitig umgesetzt werden, die stark ineinander verzahnt sind. Nachteile gibt es, wenn ein Team in viele Richtungen Abhängigkeiten besitzt, so dass mehrere Teammitglieder gleichzeitig mehreren anderen Teams zugeordnet sind. (vgl. Cohn, 2010, S. 367)

In dem Fall ist das zentrale Team mit mehreren Teilzeitmitgliedern weniger produktiv, weil die geteilten Mitglieder zwischen mehreren Aufgaben wechseln müssen (vgl. Cohn, 2010, S. 367 f.).

2.2.6.7 Koordinierende Teams

Bei großen Projekten entstehen Teams, die koordinierende und kommunizierende Aufgaben übernehmen. Darunter fällt auch das unter 2.2.6.2 Chief Product Owner beschriebene *Product Owner Team*, das gemeinsam die Verwaltung des Product Backlogs koordiniert und die umfassende Produktvision kommuniziert. Die Aufgaben weiterer möglicher Teams sind nach den Werken von Cohn (2010), Larman u. Vodde (2009) und Eckstein (2012) nicht einheitlich beschrieben und können sich überschneiden oder zusammengefasst werden. Im Folgenden wird das breite Spektrum an möglichen Teams dargestellt:

Integrationsteam
: Übernimmt Integrationsaufgaben und kommuniziert die Verwendung mit den Projekt-

Teams. Zu den Aufgaben gehören zum Beispiel die Etablierung eines Continuous Integration Systems oder das Lösen von Problemen (Konflike, fehlgeschlagene Tests) bei der Integration. (vgl. Cohn, 2010, S. 368 f.; Eckstein, 2012, S. 108-111; vgl. Larman u. Vodde, 2009, S. 180-185)

Architekturteam

Übernimmt die Etablierung und Kommunikation einer einheitlichen Architektur. Zu den Aufgaben gehören auch die Überprüfung der bestehenden Architektur auf falsche Verwendung und die Aufdeckung nicht erkannter oder vernachlässigter Schnittstellen. (Eckstein, 2012, S. 138 ff.; vgl. Cohn, 2010, S. 368)

Kommunikationsteam

Bildet sich nach Eckstein (vgl. 2012, S. 67 ff.) aus den Scrum Mastern aller Teams mit der Aufgabe sich gegenseitig auszutauschen. Einige Vorteile dieses Austausches können beispielsweise das Vermitteln von Kontakten und Lösungen, das Zusammenführen von Teams zur Lösungsfindung gemeinsamer Probleme oder die Etablierung erfolgreich eingesetzter Aspekte im Projekt sein.

Refactoringteam

Führt notwendige (globale) Refactorings mit Werkzeugunterstützung durch. Diese Durchführung findet gelegentlich zu ungewöhnlichen Zeiten statt, um Integrationskonflikte zu vermeiden. (vgl. Eckstein, 2012, S. 160 ff.)

2.3 Kanban

Kanban kommt ursprünglich aus der Produktion und ist ein adaptiertes Vorgehensmodell für die Softwareentwicklung. Kanban (übersetzt *Signalkarte*) setzt Karten für die Repräsentation von Aufgaben ein, die den zugehörigen visualisierten Prozess aus gegliederten Prozessschritten durchlaufen. Mit der Einführung von Kanban wird dabei ein bereits bestehender Prozess übernommen.

Nur eine limitierte Anzahl an Aufgaben dürfen in einem Prozessschritt gleichzeitig bearbeitet werden. Die Visualisierung und die Limitierung der Aufgaben offenbaren im Zusammenspiel Schwachstellen innerhalb des angewendeten Prozesses. Zusammen mit der Offenlegung von Schwachstellen unterstützt Kanban die evolutionäre Weiterentwicklung und Verbesserung von Prozessen. (vgl. Anderson, 2012, S. 13-17)

Die *evolutionäre Verbesserung* wird bei Kanban mit der sogenannten *Kaizen-Kultur* etabliert werden. *Kaizen* kommt aus dem japanischen und ist die „Idee der stetigen, inkrementellen

Optimierung von Arbeitsabläufen, welche das Management und alle Mitarbeiter in den Veränderungsprozess einbezieht“ (Andresen, 2014, S. 12).

Bei Kanban wird kein Soll-Zustand definiert und Prozesse revolutionär umgestellt, wie es beispielsweise bei Scrum der Fall ist. Kanban geht davon aus, dass bestehende Prozesse nicht ohne Grund geschaffen wurden (vgl. Leopold u. Kaltenecker, 2013, S. 14 f.).

2.3.1 Abgrenzung zu agilen Methoden

Kanban basiert durch seinen Ursprung nicht auf dem *Manifest für Agile Softwareentwicklung* (2.1), sondern auf den Prinzipien des *Lean Managements*. Lean Management ist auf die Optimierung von Fertigungsprozessen mit hoher Stückzahl ausgerichtet. Agile Methoden unterstützen hingegen die Erstellung eines Individualproduktes. (vgl. Komus u. Kamlowski, 2014)

Somit ist Kanban genau genommen *keine* agile Methode, da sie nicht auf den Werten des Manifests für Agile Softwareentwicklung (2.1.1) beruht. Dennoch teilen sich die Agile Softwareentwicklung und Lean Management ähnliche Werte und Prinzipien, auch wenn sich das Vorgehen unterscheidet (vgl. Komus u. Kamlowski, 2014, S. 25 f.). Aus diesem Grund wird Kanban für die Softwareentwicklung in Literatur und Studien oft als agile Methode bezeichnet. Ebenso wird auch im Rahmen dieser Arbeit Kanban zu den agilen Methoden im weiteren Sinne hinzugezählt.

2.3.2 Kernpraktiken

Kanban besteht nach David J. Anderson (Anderson, 2012), dem Erfinder von Kanban für die Softwareentwicklung, aus *sechs* Kernpraktiken, die für einen erfolgreichen Einsatz nötig sind:

1. Visualisiere den Arbeitsfluss

Mit Kanban sollen Probleme, die den Arbeitsfluss behindern, sichtbar gemacht werden. Dazu wird eine Visualisierung gewählt, die den Arbeitsfluss beschreibt. Der Arbeitsfluss wird in Arbeitsschritte geteilt, durch die Aufgaben in eine Richtung (niemals rückwärts) *fließen*. Eine beliebte Möglichkeit der Visualisierung ist das *Kanban-Board* (2.3.3.1), in dem die Arbeitsschritte als Spalten dargestellt sind und der Arbeitsfluss von links nach rechts verläuft. Die Art der Visualisierung wird von Kanban jedoch nicht vorgegeben.

2. Limitiere den Work in Progress (WiP)

Probleme und Engpässe werden erst durch die Limitierung von Arbeitsaufgaben innerhalb der einzelnen Bearbeitungsschritte sichtbar. Wie dies geschieht kann an Abbildung 2.6 abgelesen werden.

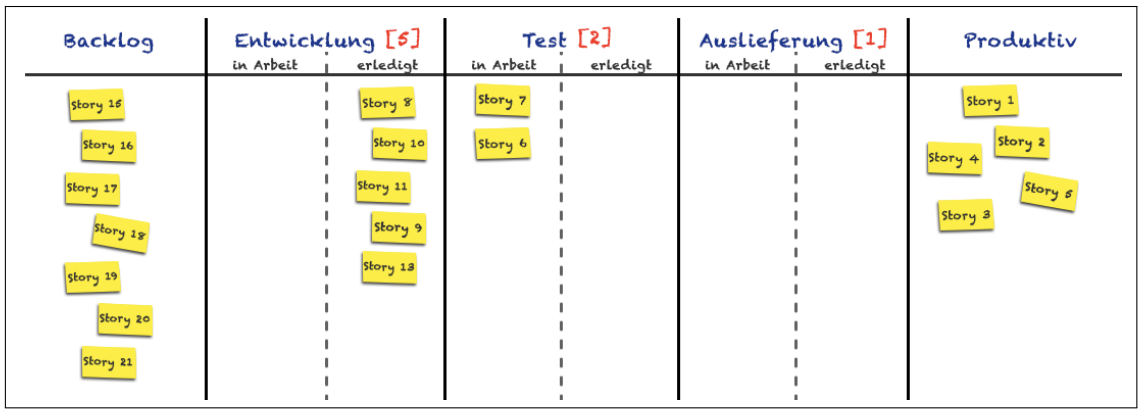

Abbildung 2.6: Blockade macht Probleme sichtbar (mittels Kanban-Board)

Die roten Zahlen stellen die WiP-Limitierung dar. Auf den gelben Moderationskarten sind die Aufgaben, im Umfeld von Kanban üblicherweise als *Ticket* bezeichnet, notiert. Diese bewegen sich je nach Zustand von links nach rechts entlang des Arbeitsflusses. In den Arbeitsschritten *Entwicklung* und *Test* ist die WiP-Limitierung bereits erreicht. Es kommt zu einer Blockade, weil die Test-Tickets noch in Bearbeitung sind. Sowohl die Entwicklung als auch die Auslieferung können und dürfen sich keine Tickets aus dem vorherigen Bearbeitungsschritt abholen (Pull-Prinzip). Ist das Kanban-Board größer, setzt sich dies als Stau bis ganz zum ersten Bearbeitungsschritt fort. Aus dieser Situation wird somit das Problem schnell sichtbar und das Team ist gezwungen sich eine Lösung zu überlegen. Beispielsweise könnten sich die Entwickler um die Abarbeitung der im Test befindlichen Tickets kümmern.

Die Idee dieser Vorgehensweise kommt aus dem Produktmanagement, in dem unfertige Produkte gebundenes Kapital bedeuten und daher möglichst gering gehalten werden sollen (vgl. Leopold u. Kaltenecker, 2013, S. 18). Auf die Softwareentwicklung bezogen sind unfertige Features ebenso gebundenes Kapital und sind zu vermeiden, da sie für den Kunden keinen Wert bieten und das Team dafür kein Feedback von den Anwendern erhält.

3. Messe und steuere den Arbeitsfluss

Der Arbeitsfluss wird kontinuierlich überprüft um Probleme, die den Arbeitsfluss behindern, zu identifizieren. Um die Probleme zu lösen, werden Änderungen am Prozess vorgenommen.

Zur Einschätzung der Auswirkungen dieser Änderungen wird der Arbeitsfluss gemessen, ob sich die Änderungen bewähren und erneut geprüft, ob sich neue Probleme ergeben. Auf diese Weise helfen die Messungen den Prozess kontinuierlich zu verbessern. Die Messungen helfen außerdem dabei, feste Zusagen und Vereinbarungen mit Auftraggebern zu treffen, wenn sich beispielsweise der Durchsatz an Aufgaben über längere Zeit vorhersagen lässt. (vgl. Leopold u. Kaltenecker, 2013, S. 19 f.)

4. Mache Prozessregeln explizit

Der Prozess, den ein Kanban-Team anwendet, kann als eine Menge von veränderbaren Regeln verstanden werden. Diese Regeln sollen für alle Beteiligten sichtbar und nachvollziehbar sein, damit der Prozess eingehalten und verbessert werden kann. (vgl. Leopold u. Kaltenecker, 2013, S. 20 f.)

5. Implementiere Feedback-Mechanismen

Damit sich ein Kanban-Team kontinuierlich verbessert, benötigt es Feedback, um zu lernen wie es sich verbessern kann. Unternehmen setzen dabei gerne auf bewährte Feedback-Mechanismen, die auch in Scrum angewendet werden. Darunter das *Daily Standup-Meeting*, das mit dem unter 2.2.3.3 beschriebenen Daily Scrum identisch ist und im Stehen durchgeführt wird. Oder die *Retrospektive* (2.2.3.5), um die Zusammenarbeit zu verbessern. Kanban gibt keine Feedback-Mechanismen zum Einsatz vor und überlässt die Wahl dem Kanban-Team. (vgl. Leopold u. Kaltenecker, 2013, S. 21 f.)

6. Führe gemeinschaftliche Verbesserungen durch (basierend auf Modellen und wissenschaftliche Methoden)

Für die Lösung von bekannten Problemen können bewährte Ansätze und Modelle adaptiert werden. Sogar Kanban selbst ist ein adaptiertes Modell aus der Automobilindustrie. (vgl. Leopold u. Kaltenecker, 2013, S. 21 f.)

2.3.3 Visualisierung

Da sich Kanban nicht nur auf die Softwareentwicklung anwenden lässt, kann die Visualisierung des Arbeitsflusses auf verschiedenen Skalen durchgeführt werden. Auf ein ganzes Unternehmen, auf eine gesamte Wertschöpfungskette oder auf Teilabschnitte einer Wertschöpfungskette, die wenige Teams betreffen. Um die Grenzen dieses Teilabschnitts zu bestimmen sollte festgestellt werden, welche Aufgaben dieser Teilabschnitt umfasst (und welche nicht), welche

Prozessschritte in Reihenfolge existieren und wie die Übergänge zu anderen Teilabschnitten beziehungsweise Verantwortlichkeiten aussehen. Bei der Definition der Prozessschritte sollte das Team hinzugezogen werden, das den bisherigen Prozess erlebt, da Prozessbeschreibungen nicht unbedingt der realen Arbeitsweise entsprechen müssen.

2.3.3.1 Kanban-Board

Die Art der Visualisierung wird von Kanban nicht vorgegeben. In diesem Kapitel wird das *Kanban-Board* als Beispiel für die Möglichkeiten der Visualisierung verwendet. Auch in der

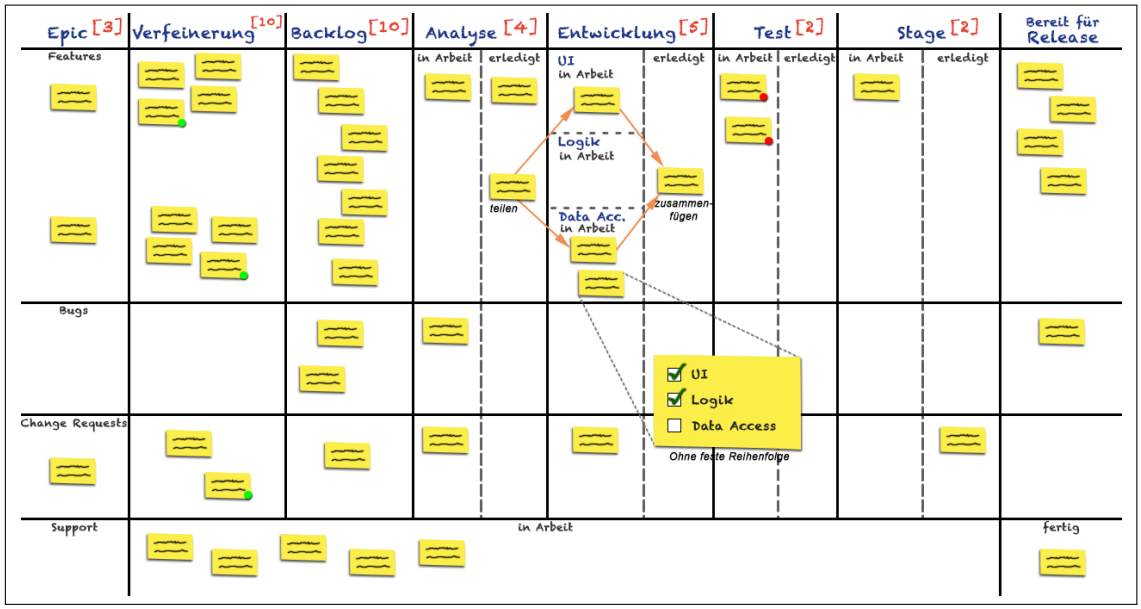

Abbildung 2.7: Mehrere Prozessabläufe am Beispiel eines Kanban-Boards

Verwendung des Kanban-Boards sind der freien Gestaltung keine Grenzen gesetzt. Dennoch sollen im Folgenden einige konkrete und sinnvolle Situationen aus der Praxis gezeigt werden, die anhand eines komplexeren Boards wie in Abbildung 2.7 zu verdeutlichen sind. (vgl. Leopold u. Kaltenecker, 2013, S. 25 f.)

Alle identifizierten Prozessschritte werden zunächst nach Ablauffolge in Spalten auf das Kanban-Board gebracht. Für jede Spalte können Kriterien definiert werden, unter welchen Bedingungen Aufgaben beziehungsweise *Tickets* als *fertig* bezeichnet werden - vergleichbar mit der 2.2.5 Definition of Done von Scrum. In der Spalte ganz links befindet sich die *Input Queue*, in der die zu bearbeitenden Aufgaben für den Teilabschnitt landen. Im Beispiel aus Abbildung 2.7 handelt

es sich um *Epics* (2.2.4.1.3), umfangreichen Anforderungen aus Kundensicht. Die Spalte ganz rechts ist analog der Übergabepunkt für den nächsten Teilabschnitt (auf der Abbildung 2.7 „Bereit für Release“). (vgl. Leopold u. Kaltenecker, 2013, S. 26 f.)

2.3.3.2 Aufgaben

Arbeitsaufgaben, im Rahmen von Kanban auch *Tickets* genannt, werden üblicherweise auf Haftnotizen oder Moderationskarten notiert und wandern in Ablauffolge der Prozessschritte über das Kanban-Board. Auf dem Ticket werden nötige Informationen der Aufgabe und für Messungen notiert. Zum Beispiel:

- ID im elektronischen Ticketsystem, in dem gegebenenfalls ausführlichere Informationen zur Aufgabe stehen (vgl. Anderson, 2012, S. 79)
- Titel der Aufgabe
- Kurzbeschreibung der Aufgabe
- Datum der Aufnahme auf das Kanban-Board
- Datum der Fertigstellung
- Datum der Deadline, falls vorhanden
- Ersteller des Tickets
- Aktueller Bearbeiter des Tickets

Für den Spaltenwechsel von Aufgaben wird das Pull-Prinzip angewendet. Es holen sich die Verantwortlichen eines Prozessschritts ihre Aufgaben aus der Vorgängerspalte. Ein Abholen von Tickets ist nur möglich, wenn Aufgaben zur Verfügung stehen, die bereit zur Abholung sind. Dies lässt sich zum Beispiel durch einen *grünen Sticker* am Ticket anzeigen oder durch eine *weitere Aufteilung* der Spalte in „in Arbeit“ und „erledigt“. In Abbildung 2.7 werden beide Techniken angewendet. Die Sticker-Technik wird beispielsweise in Spalte „Aufgeteilt“ deutlich. (vgl. Leopold u. Kaltenecker, 2013, S. 27)

Es kann vorkommen, dass Tickets aufgrund fehlender Information oder Infrastruktur innerhalb eines beliebigen Ablaufschritts nicht (weiter) bearbeitet werden können. Das Ticket wird dann als *blockiert* gekennzeichnet. Beispielsweise mittels eines *roten Stickers* (siehe Abbildung 2.7, Spalte „Test“).

2.3.3.3 Nebenläufigkeit

Die nebenläufige Abarbeitung von Tickets lässt sich auf verschiedene Arten visualisieren. Die einfachste Möglichkeit ist, mehrere Tätigkeiten, die nebenläufig abgearbeitet werden sollen, auf ein Ticket zu schreiben. (vgl. Leopold u. Kaltenecker, 2013, S. 30)

Die aufwendigere Möglichkeit geschieht durch horizontale Unterteilung innerhalb eines Prozessschritts des „in Arbeit"-Abschnitts (vgl. Anderson, 2012, S. 82 f.). Diese Möglichkeit wird in Abbildung 2.7 (Spalte „Entwicklung") mit orangen Pfeilen verdeutlicht. Fertige Tickets aus dem vorherigen Ablaufschritt werden nach Bereitschaft in die Abschnitte „UI" und „Data Access" geholt und dabei kopiert. Sobald alle erstellten Ticketkopien vollständig umgesetzt sind, können sie erst im „erledigt"-Abschnitt wieder zusammengeführt werden. In diesem Beispiel musste das Ticket nicht in das Segment „Logik" kopiert werden, weil es keine Logik umzusetzen gab.

2.3.3.4 Spalten ohne Reihenfolge

Tickets, die Ablaufschritte ohne Reihenfolge durchlaufen sollen, erhalten diese als abhakbare Liste. Das Beispiel in Abbildung 2.7 (Spalte „Entwicklung") verdeutlicht dies durch ein vergrößertes Ticket unter „Data Access". Das Ticket kann nun durch die horizontalen Unterteilungen von „Entwicklung" ohne Beachtung der Reihenfolge wandern. Ist eine Unterteilung abgeschlossen, wird ein Haken im Ticket an die entsprechende Tätigkeit gesetzt. Dieses Verfahren funktioniert auch ohne horizontale Unterteilungen und einer variierenden Liste von Tätigkeiten pro Ticket. (vgl. Anderson, 2012, S. 84 f.)

2.3.3.5 Aufgabentypen

In der Softwareentwicklung treten mehrere *Aufgabentypen* auf, beispielsweise Anforderungen, Change Requests oder Bugs. Aufgabentypen können nach Art, Quelle, Größe und Ankunftsrate der Aufgabe unterschieden werden. (vgl. Leopold u. Kaltenecker, 2013, S. 33)

Die Visualisierung von Aufgabentypen geschieht auf einem Kanban-Board mit horizontalen Unterteilungen über das gesamte Board, sogenannte *Swim Lanes*. Um sinnvolle und *nötige Swim Lanes* auf das Board zu bringen, sollten ähnliche Aufgabentypen zusammengefasst werden. Beispielsweise können alle Arten von Fehlern in einer Swim Lane „Bugs" zusammengefasst werden ohne zu unterscheiden, ob sie im Produktiv-System oder beim Test auftraten.

Beim Zusammenfassen der *Größe* von Aufgaben ist es üblich, diese in *small* (wenige Stunden/Tage), *medium* (bis zu einem Monat) und *large* (ein Monat oder länger) aufzuteilen. (vgl. Anderson, 2012, S. 72 f.; Leopold u. Kaltenecker, 2013, S. 32 ff.)

Die *Swim Lane* „Support“ in Abbildung 2.7 ist ein *Subboard*, weil es einen abweichenden Arbeitsfluss zu den anderen Aufgabentypen aufweist. Das kann verschiedene Gründe haben. Zum Beispiel könnten die Support-Tickets so wenig Aufwand bedeuten, dass sich ein Umhängen der Tickets im Verhältnis nicht lohnt. (vgl. Leopold u. Kaltenecker, 2013, S. 34)

Das Kanban-Board aus Abbildung 2.7 visualisiert einen Beispielprozess von der Anforderungssammlung in grob formulierten *Epics* bis zur fertigen Realisierung. Ausgenommen ist dabei der Aufgabentyp „Support“. Um die Größe von Aufgaben zu vereinheitlichen, wurde im Beispiel der Prozessschritt der *Verfeinerung* gewählt, der auch in Scrum eingesetzt wird (2.2.4.1).

2.3.4 Work in Progress-Limits

Mit den WiP-Limits werden Begrenzungen auf Abschnitte gesetzt (siehe rote Zahlen in Abbildung 2.7). Unbeschränkt sind idealerweise nur Prozessschritte, die möglichen Engpässen nachgelagert sind. Dies trifft auf Ergebnisse oder Ziele eines Prozesses zu, beispielsweise der letzten Spalte aus Abbildung 2.7 (vgl. Anderson, 2012, S. 125 ff.). Die Limits gelten über alle Unterteilungen eines Prozessschritts (horizontale oder Swim Lanes) hinweg. Damit wird gleichzeitig eine Begrenzung der gesamten im Kanban-Prozess befindlichen Arbeit erreicht.

WiP-Limits haben einige Vorteile. Ein Vorteil ist, dass mit einer limitierten Menge gleichzeitiger Aufgaben weniger zwischen begonnener Arbeit gewechselt wird. Dadurch werden weniger Aufgaben schneller fertig. Darüber hinaus ist die Durchlaufzeit von Aufgaben mit Limitierung über den gesamten Prozess geringer, weil Menschen sich besser konzentrieren können, wenn sie wenige gleichzeitige Aufgaben bearbeiten. Bei hoher Konzentration wird die Sorgfalt in der Bearbeitung besser, die Aufgabe wird insgesamt schneller fertig und es kommt somit schneller Feedback vom Anwender des Produktes. (vgl. Leopold u. Kaltenecker, 2013, S. 38 f.)

WiP-Limits können angepasst werden, daher lassen sich anfangs die Prozessspalten mit einer ausgesuchten Zahl belegen. Soll die Zusammenarbeit mehrerer Personen unterstützt werden, können die Zahlen pro Prozessspalte niedriger angesetzt werden, als verantwortliche Personen diesen Schritten zugeteilt sind. Blockierte Tickets können auftreten und sollten in der Bestimmung des Limits bedacht werden. (vgl. Anderson, 2012, S. 121 f.)

Eine wichtige Eigenschaft von WiP-Limits in Verbindung mit einer Visualisierung, wie beispielsweise das Kanban-Board, ist die Sichtbarkeit von Problemen und Engpässen:

2.3.4.1 Probleme

Probleme äußern sich häufig durch Tickets, die als *blockiert* markiert wurden. Bei blockierten Tickets kann die Arbeit nicht fortgeführt werden, weil Informationen oder die nötige Infrastruktur fehlt. Eine Blockade kann beispielsweise durch rote Sticker, wie in Abbildung 2.7, dargestellt werden. Solche Tickets erzeugen *Engpässe* und sollten daher eine höhere Priorität erhalten, damit sie weiter bearbeitet beziehungsweise abgeschlossen werden. (vgl. Leopold u. Kaltenecker, 2013, S. 40 f.)

2.3.4.2 Engpässe

Engpässe äußern sich durch ausgelastete „erledigt"-Queues. Ist dies der Fall, können keine neuen Tickets zur Bearbeitung aus dem Vorgängerschritt abgeholt werden, weil die erledigten Tickets vom Nachfolger noch nicht abgeholt wurden. Einer der Nachfolger stellt den Verursacher des Engpasses dar, weil in seinem Prozessschritt die Aufgaben nicht *schnell* genug abgearbeitet werden. (vgl. Leopold u. Kaltenecker, 2013, S. 41 f.)

An diesen Stellen offenbart sich das Verbesserungspotential für den Prozess und die verantwortlichen Personen der wartenden Prozessschritte unterstützen idealerweise die Person am Engpass. Der Durchsatz des gesamten Prozesses wird durch diesen Engpass bestimmt, denn es spielt keine Rolle wie viele Tickets nachrücken, wenn am Engpass nur eine bestimmte Menge pro Zeiteinheit durchfließen kann. Kanban profitiert somit von Teams, die sich gegenseitig helfen und die Kompetenz besitzen Aufgabenschritte von Teammitgliedern zu übernehmen.

2.3.4.3 Queues und Puffer

In Kanban wird versucht nicht mehr Aufgaben in das System zu lassen, als der Engpass verarbeiten kann. Daher werden die WiP-Limits der Prozessspalten so angepasst, dass gerade so viel Arbeit durch das System fließt, um den Engpass nicht zu überlasten. Um dies zu erreichen wird vor jedem neu identifizierten Engpass eine neue Spalte als *Puffer* hinzugefügt. In Abbildung 2.8 wurde ein solcher Puffer für den Engpass in „Test" eingefügt, der ebenfalls so klein wie möglich gehalten wird, aber groß genug, um einen gleichmäßigen Arbeitsfluss zu erhalten. Ein Puffer stellt sicher, dass weitergearbeitet werden kann, wenn Prozessschritte in Verzug geraten, beispielsweise wenn eine verantwortliche Person ausfällt. (vgl. Anderson, 2012, S. 123 f.; vgl. Leopold u. Kaltenecker, 2013, S. 44)

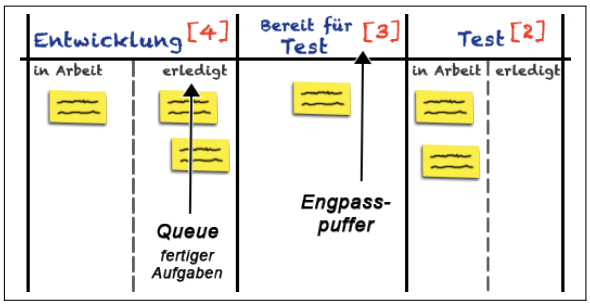

Abbildung 2.8: Queue für erledigte Aufgaben und ein Puffer vor dem Engpass „Test“

Erledigte Arbeit wird in *Queues* abgelegt, aus denen die Tickets für den nächsten Arbeitsschritt abgeholt werden. Queues werden, wie in Abbildungen 2.8 zu sehen, gewöhnlich mit dem Arbeitsschritt zusammengefasst. Folgt wie in Abbildung 2.8 nach einer Queue direkt ein Puffer, so kann „in Arbeit“ und „erledigt“ auch zusammengefasst werden, weil die Menge wartender Tickets unter „Entwicklung“ quasi null ist. (vgl. Anderson, 2012, S. 122 f.)

2.3.4.4 Aufgabentypen limitieren

Neben der Limitierung von Prozessspalten können auch Aufgabentypen auf den *Swim Lanes* limitiert werden. Damit lässt sich garantieren, dass für bestimmte Aufgabentypen Kapazitäten übrig bleiben. (vgl. Anderson, 2012, S. 129)

2.3.5 Serviceklassen

In Kanban können Tickets in sogenannte Serviceklassen unterteilt werden. Die Einteilung in verschiedene Serviceklassen basiert auf den Auswirkungen und Risiken für das Geschäft und sind vergleichbar mit Prioritäten, denen Risikoinformationen hinterlegt sind. Serviceklassen sind darüber hinaus mit bestimmten Zusagen gegenüber dem Kunden verbunden, sogenannten *Service Level Agreements*, die beispielsweise besagen können, dass ein Prozentsatz von Tickets einer Serviceklasse innerhalb einer Zeitspanne erledigt ist. In der Visualisierung werden für Serviceklassen verschiedene Hintergrundfarben, Sticker oder eigene Swim Lanes für Tickets verwendet. Die Anzahl unterschiedlicher Serviceklassen sollte gering gehalten werden, damit sich der Verwaltungsaufwand in Grenzen hält und sich die Beteiligten die Zuordnungen (Bedeutung und Farbe) merken können. Die Zuordnung von Aufgaben zu Serviceklassen geschieht mit Ankunft in der *Input Queue*. (vgl. Anderson, 2012, S. 132-144; vgl. Leopold u. Kaltenecker, 2013, S. 53-63)

Für jede Serviceklasse existieren mehrere Regeln, die definieren, wie die Tickets durch das Kanban-System fließen. Ändern sich die Rahmenbedingungen eines Tickets, kann die Serviceklasse gewechselt werden. (vgl. Anderson, 2012, S. 138 f.)

Vier gängige Serviceklassen sollen im Folgenden betrachtet werden: *Beschleunigt*, *Fester Liefertermin*, *Standardklasse*, *Unbestimmbare Kosten.* (vgl. Anderson, 2012, S. 133-137)

2.3.5.1 Beschleunigt

In der *Beschleunigt*-Serviceklasse (auch *Silver Bullet* genannt) befinden sich alle Arbeiten die sofort erledigt werden müssen. Einerseits wegen möglicher hoher Folgeschäden des Kunden bei Verzögerungen der Arbeit oder andererseits durch mögliche Neugeschäfte des eigenen Unternehmens bei schneller Bearbeitung. *Beschleunigt*-Tickets können das ursprüngliche WiP-Limit mittels Regel überschreiten. Damit jedoch nicht alle Tickets einfach als *beschleunigt* gelten, muss die Anzahl ebenfalls mittels Regel limitiert werden, zum Beispiel: „Maximal ein *Beschleunigt*-Ticket befindet sich in Bearbeitung." (vgl. Anderson, 2012, S. 133 f.; vgl. Leopold u. Kaltenecker, 2013, S. 56 f.)

2.3.5.2 Fester Liefertermin

Die *Fester Liefertermin*-Serviceklasse erhält Arbeit, die zeitlich eingegrenzt wird, also eine Deadline besitzt. Wird der Zeitraum überschritten folgen häufig hohe Kosten, Strafen, Kundenbeschwerden und in seltenen Fällen sogar Geschäftsunfähigkeit. Im Gegensatz zu *Beschleunigt*-Tickets muss bei *Fester Liefertermin*-Tickets das WiP-Limit eingehalten werden. Eine Regel dieser Serviceklasse kann beispielsweise lauten: „Ist die Deadline überschritten, erhält das Ticket die Serviceklasse *Beschleunigt*". (vgl. Leopold u. Kaltenecker, 2013, S. 57 f.)

2.3.5.3 Standardklasse

In diese Serviceklasse sollten die meisten Arbeiten fallen. Werden diese Arbeiten nicht erledigt oder erst später, sind es lediglich entgangene Einnahmen aber keine großen Schäden für das Unternehmen. Eine Regel dieser Serviceklasse könnte beispielsweise lauten: „Standardtickets werden nach dem FIFO-Mechanismus bearbeitet". (vgl. Leopold u. Kaltenecker, 2013, S. 59)

2.3.5.4 Unbestimmbare Kosten

In die *Unbestimmbare Kosten*-Serviceklasse sollten Aufgaben kategorisiert werden, die noch nicht sehr wichtig sind, aber in der Zukunft dringend werden können. Tickets aus dieser Klasse sollten gezogen werden, wenn es keine anderen Tickets der übergeordneten Serviceklassen gibt. Eine Regel könnte lauten: „Trifft eine Aufgabe aus der *Beschleunigt*-Serviceklasse ein, werden alle Tickets der *Unbestimmbare Kosten*-Serviceklasse eingestellt und zu einem anderen Zeitpunkt wieder aufgenommen". (vgl. Leopold u. Kaltenecker, 2013, S. 60 f.)

2.3.5.5 Kapazitäten

Serviceklassen können auch Kapazitäten beziehungsweise Limitierungen zugeordnet werden. Sind zum Beispiel 20 Tickets über WiP-Limits im Kanban-System, können 5% *Beschleunigt* (1 Ticket), 20% *Fester Liefertermin* (4 Tickets), 50% *Standardklasse* (10 Tickets) und 30% *unbestimmte Kosten* (6 Tickets) also insgesamt 21 Tickets über Serviceklassen aufgeteilt sein. Ein Ticket mehr, weil sich das *Beschleunigt*-Ticket über das WiP-Limit hinweg setzt. (vgl. Anderson, 2012, S. 143 ff.)

2.3.6 Koordinierung

Primär koordiniert sich Kanban über die besprochene Visualisierung. Für die weitere Koordinierung werden bekannte Feedback-Mechanismen verwendet. Die Praxis bedient sich an dieser Stelle unter anderem bei den aus Scrum bekannten Ereignissen. Die gängigen sollen kurz im Vergleich zu Scrum angesprochen werden:

2.3.6.1 Daily Standup Meeting

Die Durchführung ist identisch mit dem in 2.2.3.3 beschriebenen Daily Scrum nur das es definitiv im Stehen durchgeführt wird. Besondere Aufmerksamkeit erhalten dabei blockierte Tickets oder Tickets, die blockiert werden müssen. Nach einem *Daily Standup Meeting* kann es zu spontanen *Anschlussmeetings* kommen, in denen sich die Verantwortlichen eines Tickets für die Bearbeitung koordinieren. (vgl. Anderson, 2012, S. 90 f.)

2.3.6.2 Queue Replenishment Meeting

Das *Queue Replenishment Meeting*, oder auch *Nachschubmeeting* genannt, ist vergleichbar mit dem in 2.2.3.2 beschriebenen Sprint Planning, das regelmäßig durchgeführt werden sollte. Es geht darum die *Input Queue* eines Kanban-Systems zu befüllen und zu priorisieren, ähnlich wie Sprint Backlog im Sprint Planning gefüllt wird. Die Abstände des Meetings sind abhängig vom Limit der *Input Queue*. In der Praxis ist der Abstand von einer Woche üblich. Geübte Teams priorisieren nach Bedarf und nutzen dieses Meeting nur zu Beginn. (vgl. Anderson, 2012, S. 91 ff.)

2.3.6.3 Release-Planungsmeeting

Das *Release-Planungsmeeting* dient der Planung der Auslieferung am Ende des Kanban-Systems, der Ergebnis-Spalte. An dem Meeting nehmen neben dem Projektmanager auch das Team und weitere Stakeholder teil. Es wird geklärt, welche Tickets bereit sind für ein Release, welche noch folgen, welche Tests noch erforderlich sind, welche Risiken bestehen und was noch für ein Release benötigt wird. Fortgeschrittene Teams setzen hier auf Continuous Integration. (vgl. Anderson, 2012, S. 93 f.)

2.3.6.4 Triage

Beim *Triage* (der Begriff kommt aus der Medizin) handelt es sich um ein Meeting zur *Verfeinerung* von Tickets, ähnlich wie das 2.2.6.4.4 Product Backlog Refinement im skalierten Ansatz von Scrum. Das Meeting wird in eher längeren Rhythmen durchgeführt, beispielsweise monatlich oder gar quartalsweise. Es wird zusammen mit den Projektmanagern oder Product Ownern und Stakeholdern durchgeführt, die bereits an den Queue Replenishment Meetings teilgenommen haben. Bestehende Tickets, besonders in der *Input Queue*, werden dabei durchgegangen und geprüft, ob diese noch einen Wert haben oder entfernt werden können. Grund dafür ist, dass Tickets den Durchsatz behindern können, wenn sie sich sehr lange in der *Input Queue* aufhalten. Gegebenenfalls sind diese Tickets aktuell noch nicht umsetzbar oder zu vernachlässigen. Eine *Triage* wird auch durchgeführt, wenn das Limit der *Input Queue* verringert werden soll. (vgl. Anderson, 2012, S. 94 f.)

2.3.6.5 Operations Review

Operations Reviews setzen den Fokus auf den Gesamtprozess oder auf den Prozess der gesamten Organisation. Wenn mehrere Teams mit Kanban arbeiten, dann stellen ihre Visualisierungen nur Teilabschnitte einer Wertschöpfungskette dar. Innerhalb ihres Abschnitts machen sie wertvolle Erfahrungen, die in *Operations Reviews* ausgetauscht werden sollen, um die Arbeitsweise der Organisation zu verbessern. Das Operations Review kann beispielsweise monatlich für zwei Stunden durchgeführt werden. Jedes Team kann dabei seine Messungen über den Arbeitsfluss präsentieren und gemeinsam mit anderen Teams Schlüsse daraus ziehen. (vgl. Leopold u. Kaltenecker, 2013, S. 72 f.)

2.3.6.6 Teamretrospektive

Die *Teamretrospektive* ist identisch mit der unter 2.2.3.5 beschriebenen Sprint Retrospektive. (vgl. Leopold u. Kaltenecker, 2013, S. 72)

2.3.7 Messungen

Mit Kanban werden Messungen durchgeführt um den Prozess zu verbessern und um Zusagen für Stakeholder zu treffen. Gemessen wird:

- Tickets der *Input Queue* eines Zeitintervalls
- Tickets innerhalb von Arbeitsschritten eines Zeitintervalls
- Tickets im Ergebnis-Puffer am Ende der Wertschöpfungskette eines Zeitintervalls
- Durchlaufzeiten der Tickets in zum Beispiel Tagen (auch nach Aufgabentypen oder Serviceklasse trennbar)
- Anzahl Blockaden von Tickets eines Zeitintervalls (auch nach Prozessschritten trennbar)

(vgl. Anderson, 2012, S. 147-155)

Anhand dieser Messungen können verschiedene *Metriken* erstellt werden, die hier nicht im einzelnen betrachtet werden sollen. In den Lehrmethoden werden einfache Metriken (anhand von Punkteverteilungen) vorgestellt, die für den Einsatz auf Tafel oder Flipchart geeignet sind.

3 Lehrmethoden für agile Softwareentwicklung

Dieses Kapitel umfasst verschiedene aktivierende Lehrmethoden für den Kompetenzerwerb agiler Softwareentwicklung, die in ein Lehrkonzept schließlich einfließen können. Es werden existierende Lehrmethoden gesammelt und Teilaspekte agiler Methoden zu Lehrmethoden umgewandelt, diese beschrieben und bewertet. Die Bewertungen beruhen auf Erfahrungen des Autors dieser Arbeit, die bei Teilnahmen an Lehrveranstaltungen oder Durchführungen gemacht wurden.

3.1 Ball Point Game

The Ball Point Game simuliert den Scrum-Prozess in einer komprimierten Weise. Die Aufgabe ist, in Teams möglichst viele Bälle zu *produzieren*. Ein Ball gilt als produziert, sobald jede Person im Team ihn mindestens einmal berührt hat. Dabei darf ein Ball nicht direkt zum Nachbarn gegeben werden und muss sich zeitweise in der Luft befunden haben. Die Lehrmethode arbeitet mit Iterationen (2.2.3.1), vor denen jedes Team die Anzahl Bälle schätzen muss, die sie produzieren werden.

Durchführung

Benötigte Zeit
: 30 Minuten

Anzahl Personen
: Es werden 5 bis 11 Personen für ein Team benötigt. Die maximale Anzahl Personen wird erst durch den Platz im Raum und Material (Bälle) begrenzt.

Benötigtes Material
: 60 Bälle pro Team

Beschreibung

Die Lehrmethode beginnt, indem sich die Teams zu 5 bis 11 Personen zusammenfinden. Danach folgt eine zweiminütige Einleitung, in der kurz die Aufgabe erläutert wird: Produziere im Team so viele Bälle wie möglich (siehe Einleitung dieser Lehrmethode).

Anschließend werden die Regeln erläutert (optional gleichzeitig auf Beamer oder Flipchart-Papier präsentiert): Jeder Teilnehmer spielt in einem oder mehreren Teams. Jedes Teammitglied muss einen Ball mindestens einmal berühren, damit er als produziert gilt. Die Person, die einen Ball zur Produktion aufnimmt, erhält diesen zur Fertigstellung wieder zurück - muss den Ball somit mindestens zweimal berühren. Der Ball darf nicht zu einem direkten Nachbarn gespielt werden, es muss sich eine andere Person dazwischen befinden. Bei Übergabe muss der Ball zeitweise unberührt in der Luft sein. Fällt ein Ball herunter, gilt dieser als Ausschuss und kann nicht mehr produziert werden. Für eine Iteration stehen zwei Minuten zur Verfügung und es werden fünf Iterationen gespielt. Zwischen jeder Iteration erhält das Team eine Minute für Absprachen zur Verbesserung und Schätzung.

Nachdem die Regeln geklärt sind, dürfen sich die Teams zwei Minuten auf ihre erste Iteration vorbereiten. Anschließend nimmt die Moderation die Schätzungen jedes Teams entgegen. Geschätzt wird, wie viele Bälle in der zweiminütigen Iteration produziert werden können. Die Schätzungen jedes Teams und jeder Iteration werden auf Tafel oder Flipchart dokumentiert. Die erste Iteration kann anschließend beginnen. Nach jeder Iteration werden die tatsächlich produzierten Bälle gezählt und ebenfalls dokumentiert - inklusive der Abweichung.

Nach den fünf Iterationen wird die Durchführung ausgewertet und reflektiert. Für die Auswertung wird das dokumentierte Ergebnis diskutiert. Mögliche Fragestellungen: Welche Iteration war für die Mehrheit der Teams die erfolgreichste? Was fällt bei den Abweichungen der Schätzungen auf? An welcher Stelle wurde von den Teams der Scrum-Prozess bewusst wahrgenommen? Welche Elemente von Scrum (2.2) wurden wiedererkannt?

Quelle

Gloger (vgl. 2008)

Bewertung

The Ball Point Game eignet sich besonders in der Einführung von Scrum, nachdem die Theorie und vor allem die Ereignisse von Scrum (2.2.3) zumindest im Ansatz vermittelt wurden. Die Lehrmethode simuliert letztlich die Ereignisse: Den Sprint (2.2.3.1) über die Iterationen mit der Produktion von Bällen, das Sprint Planning (2.2.3.2) über die Schätzung und die Sprint Retrospektive (2.2.3.5) über die Absprachen zur Verbesserung. Dadurch sind auch die Elemente *Überprüfung* und *Anpassung* der *Empirische Prozesssteuerung* (2.2.1) erkennbar.

In der Regel sollten die Teams in den ersten Iterationen eine stärkere Verbesserung in der Produktion von Bällen feststellen, die sich zum Ende immer mehr angleicht. Gleichzeitig passt sich die Schätzung dieser Kurve an und wird genauer. An dieser Stelle wird die *Velocity* (die Geschwindigkeit des Teams bei der Produktion) erkennbar (2.2.3.2.1) und lässt sich ebenfalls thematisieren.

Die aufgeführten und weiteren Fragestellungen können einen Übergang zu den nächsten Lehrmethoden darstellen: Welche Bedeutung haben Schätzverfahren in Scrum? Wie könnten sich komplexere Aufgaben, die weniger machbar erscheinen, auf den Scrum-Prozess auswirken?

3.2 Coding Dojo

Bei *Coding Dojo* werden gemeinsam kleine Programmieraufgaben umgesetzt, sogenannte *Coding Katas* oder kurz *Katas*. Katas können auf unterschiedliche Art gelöst werden. Im Internet lassen sich viele verschiedene Katas finden, die unterschiedliche Probleme im Fokus haben. Darunter gibt es algorithmische Katas, wie beispielsweise die Abbildung von arabischen in römische Zahlen, Architekturkatas, die Modellierungen oder komplexere Implementierungen beinhalten und agile Katas, die eine evolutionäre Entwicklung unterstützen.

Coding Dojo kann in verschiedenen Variationen durchgeführt werden. In der Durchführung wird eine für die Lehre geeignete Variante beschrieben, die nach den Erfahrungen des Verfassers viele Teilnehmer einbezieht und das unvorbereitete Programmieren von Teams im Plenum vermeidet. In der Durchführung der Lehrmethode wird grundsätzlich mit Test Driven Development (TDD) entwickelt und jede Übung wird mit einer kleinen Retrospektive abgeschlossen.

Um den Ablauf der Lehrmethode für die Lehrenden zu vereinfachen, empfiehlt der Autor dieser Arbeit den Einsatz von *Cyber-Dojo* (Jagger, 2010). Dabei handelt es sich um eine Web-Applikation, die bereits *Katas* in mehreren Sprachen und eine Test- sowie Entwicklungsumge-

bung zur Verfügung stellt. Ein Kata kann dabei für alle Teilnehmer über einen Identifikator festgelegt werden, über den die Durchführung vom Lehrenden überwacht und ausgewertet werden kann.

Durchführung

Benötigte Zeit

Die benötigte Zeit ist sehr davon abhängig, welches Kata ausgewählt wurde. Dauert ein Kata beispielsweise 40 Minuten und wird in zwei Runden gespielt (jeweils ein Kata), dann können 150 Minuten geplant werden (15 Minuten für die Einführung, 80 Minuten für die Durchführung beider Runden, 20 Minuten für den Übergang zur zweiten Runde und 30 Minuten für die Retrospektive).

Anzahl Personen

Mindestens 2 Personen für ein Paar werden benötigt. Die maximale Anzahl Personen wird durch die verfügbaren Computer begrenzt.

Benötigtes Material

Ein Computer mit Internetzugang für jeweils zwei Teilnehmer. Außerdem einen Beamer im Raum.

Beschreibung

Für die Einführung ins *Coding Dojo* können zunächst die Entwicklungspraktiken TDD, Refactoring und Pair Programming vorgestellt werden. Für TDD sollte der Ablauf im Umgang mit Unit-Tests erläutert oder mit *Cyber-Dojo* vorgeführt werden. Der Ablauf gestaltet sich in Mikroiterationen, die sich wiederholen, bis das *Kata* abgeschlossen ist:

1. Neuen Test schreiben und prüfen, ob dieser tatsächlich fehlschlägt
2. Implementierung mit geringem Aufwand anpassen, so dass der Test erfüllt wird
3. Quellcode mit Refactoring aufräumen und erneut prüfen, ob der Test noch erfüllt wird

Nach jeder kleinen, aber sinnvollen, Änderung am Quellcode werden die Unit-Tests erneut ausgeführt. Dieses Verfahren ist ebenfalls eine Entwicklungspraktik, die *Baby Steps* genannt wird.

Die Teilnehmer finden sich in Paaren zusammen und belegen einen Computer pro Paar. Jedes Paar öffnet die *Cyber-Dojo* Web-Applikation und gibt den Identifikator des

Lehrenden ein. Anschließend kann eine Programmiersprache und ein *Kata* ausgewählt werden. Beides wird durch den Lehrenden oder in der Gruppe vorher festgelegt.

Für die Durchführung der ersten Runde stehen typischerweise 40 Minuten zur Verfügung. Diese Zeit wird durch das *Kata* bestimmt oder kann von dem Lehrenden vorher durch ausprobieren festgelegt werden. Während der Zeit arbeiten die Teams im *Pair Programming* zusammen: Alle 5 bis 7 Minuten wechselt eine Person jedes Teams den Partner und Computer. Die andere Person bleibt am Platz und wird in der nächsten Runde wechseln. Dabei arbeiten während des *Katas* immer zwei unterschiedliche Teilnehmer im Paar zusammen.

Ist die erste Runde beendet, dürfen 1 bis 2 Teams ihr Vorgehen zur Lösung des *Katas* über den Beamer präsentieren. Die Lösung kann anschließend im Plenum diskutiert werden.

Die Übung kann in beliebig vielen Runden gespielt werden, je nach verfügbarer Zeit und Interesse der Teilnehmer. In jeder Runde wird ein anderes *Kata* gespielt.

Zum Abschluss der Lehrmethode wird eine Retrospektive (2.2.3.5), idealerweise mit allen fünf Phasen, durchgeführt. Für die Auswahl der Methodenbausteine für die Phasen kann die *Retr-O-Mat Webseite* (Baldauf, 2014) sehr hilfreich sein.

Quelle

Gaillot (vgl. 2010)

Bewertung

Coding Dojo als Lehrmethode ist nicht nur für die Vermittlung von Entwicklungspraktiken und Retrospektiven geeignet, sondern kann einem Thema zugeordnet sein, zum Beispiel: Entwurfsmuster, Vergleich mit und ohne TDD, Umgang mit der Entwicklungsumgebung ohne Maus (dafür nur mit Tastaturkürzeln arbeiten), und so weiter.

Um die Fokussierung von Perfektionismus auf KISS (*keep it simple and stupid*) zu wechseln, kann der Lehrende weitere Bedingungen an das Ergebnis einer Runde knüpfen, zum Beispiel: Alle arabischen Zahlen bis Eintausend sind auf römische Zahlen abbildbar und alle Tests laufen erfolgreich durch.

Eine Alternative zu *Coding Dojo* ist die Lehrmethode *Pair Storytelling* (3.8), die ebenfalls mit einer Retrospektive verknüpft werden kann.

3.3 Daily Standup

Das *Daily Standup* ist eine direkte Umwandlung zur Simulation des Daily Scrum (2.2.3.3) Ereignisses mit angepassten Leitfragen für die Lehre. Die Lehrmethode wird im Stehen durchgeführt und kann als Ritual für den Einstieg in jede Unterrichtseinheit eingesetzt werden.

Durchführung

Benötigte Zeit

15 bis 30 Minuten bei großen Gruppen

Anzahl Personen

Die Lehrmethode ist ausgelegt für 4 bis 12 Personen bei Einhaltung der zeitlichen Begrenzung von 15 Minuten des Daily Scrum (2.2.3.3) Ereignisses. Stehen mehr Personen zur Verfügung, dann kann die Gruppe in Teams von bis zu 12 Personen aufgeteilt werden. Alternativ darf die Zeitbegrenzung auch ignoriert werden und alle Teilnehmer bilden ein einziges Team. In beiden Fällen wird auch mehr Zeit für Teambildung beziehungsweise Durchführung benötigt.

Benötigtes Material

Ein Moderationsball pro Team

Beschreibung

Die Lehrmethode beginnt bei großen Gruppen mit der Teamaufteilung. Abschließend stellen sich die Teams kreisförmig auf. Dafür sollte im Raum genügend Platz zur Verfügung stehen. Jeder Teilnehmer sollte die gesamte Durchführung im Stehen verbringen - sich nicht anlehnen oder gar sitzen.

Für die Beantwortung der folgenden Leitfragen wird ein Moderationsball in einer Richtung durch jeden Kreis gereicht:

1. Was habe ich in der letzten Veranstaltung gelernt?
2. Was erwarte ich in der heutigen Veranstaltung zu lernen?
3. Was könnte mich heute am Lernen hindern?

Wird ein Ergebnisspeicher (4.3.4.2) verwendet, kann dieser zur Unterstützung der Antworten in der Nähe ausgebreitet werden. Die Durchführung ist beendet, sobald jeder Teilnehmer die Leitfragen beantwortet hat.

Bewertung

Nach den Erfahrungen des Verfassers mit dieser Lehrmethode wird die letzte Frage von Teilnehmern häufig übergangen. Daher sollte darauf geachtet werden, dass diese von den Teilnehmern ebenfalls durchdacht wird. Mit der Frage lassen sich Störungen und Konflikte (4.5) in der Gruppe und bei Einzelpersonen finden, die im Anschluss noch besprochen werden können - durchaus auch im Einzelgespräch.

Das *Daily Standup* profitiert von einer engeren Bindung zwischen den Teilnehmern und dem Lehrenden. Eine anonyme Gruppe wird Probleme in der letzten Frage nicht ansprechen wollen, wenn keine vertraute Umgebung existiert. Daher ist es sinnvoll, vor dem Einsatz dieser Lehrmethode schon einige Lehrmethoden zum Kennenlernen (4.3.4.5) eingesetzt und Vereinbarungen (4.3.4.4) abgesprochen zu haben.

3.4 Kanban Pizza Game

Das *Kanban Pizza Game* bildet einen gesamten Prozess zur Produktion von Pizza mit Hilfe von Kanban (2.3) ab. Dabei werden besonders die Kernpraktiken (2.3.2) Kanbans angesprochen und spielerisch erlebbar gemacht. Aufgabe ist es in Teams zunächst einen eigenen Prozess für die Produktion von Pizza zu entwickeln, der im Anschluss durch Kanban verbessert wird. Die Verbesserung wird durch Vergabe von Punkten gemessen. Zum Schluss wird der finale Prozess in einem Kanban-Board (2.3.3.1) visualisiert.

Durchführung

Benötigte Zeit
: 120 bis 150 Minuten

Anzahl Personen
: Es werden mindestens vier Personen für ein Team benötigt. Die obere Grenze wird durch die Größe des Raumes, die verfügbaren Tische und Materialien gesetzt.

Benötigtes Material
: Für jedes Team werden folgende Materialien benötigt:

  - 1 Block Haftnotizen jeweils in gelb, pink und grün - übliche Größen in 76x76 Millimeter oder 127x76 Millimeter, jedoch nicht die Streifenform verwenden

- 1 Block Moderationskarten in weiß, gelb oder einer anderen helleren Farbe
- 1 roter Stift
- 1 Stoppuhr
- Kleinere Schere zum Schneiden der Haftnotizen
- Größere Schere zum Schneiden der Moderationskarten
- Klebeband
- 1 DIN-A4 Blatt mit dem Inhalt „Backofen: 30 Sekunden Backzeit, maximal 3 Pizzastücke gleichzeitig“
- 1 Satz mit etwa 25 Pizza-Bestellungen: Jede Bestellung besteht aus 1 bis 5 Stück Pizza, die unterschiedlich auf Pizza Hawaii und Pizza Rucola verteilt sind. Beispiel: 3 Stück Pizza Hawaii, 1 Stück Pizza Hawaii und 3 Stück Pizza Rucola, 4 Stück Pizza Rucola, und so weiter.
- 1 leeres Flipchart-Papier für die Visualisierung der Kanban-Boards
- 1 freier Tisch mit genügend Platz für alle Teammitglieder

Für die allgemeine Durchführung wird ein vorbereitetes Stück *Pizza Hawaii* (siehe dazu Abbildung 3.1) und ebenfalls eine Stoppuhr (oder ein Stoppuhr-Programm) benötigt.

Beschreibung

Zunächst werden Teams aus 4 bis 8 Personen gebildet, die jeweils einem Tisch zugeordnet sind. An jedem Tisch befinden sich bereits alle beschriebenen Materialien.

Die *erste Aufgabe* für die Teams besteht darin, sich einen *eigenen Prozess* zur Produktion von Pizza zu überlegen.

Die Moderation zeigt den Teams, wie ein fertiges Stück Hawaii Pizza auszusehen hat (siehe dazu Abbildung 3.1) und beschreibt die Zutaten: Eine dreieckig geschnittene Moderationskarte für den Pizzaboden. Die rote Farbe stellt die Tomatensoße dar. Gelbe, klein geschnittene Haftnotizen für die Ananas und pinke für den Schinken. Es ist immer ordentlich Tomatensoße auf den Pizzastücken vorhanden, Ananas und Schinken sind ordentlich geschnitten und gleichmäßig verteilt. Außerdem erhält jedes Team ein vorbereitetes DIN-A4 Blatt mit dem symbolischen Backofen. In diesen passen maximal 3 Stück Pizza, die nach 30 Sekunden fertig sind. Während der Backzeit darf kein Stück Pizza hineingeschoben oder herausgenommen werden. Ist die Pizza fertig gebacken, kann dies beispielsweise durch Klatschen kommuniziert werden.

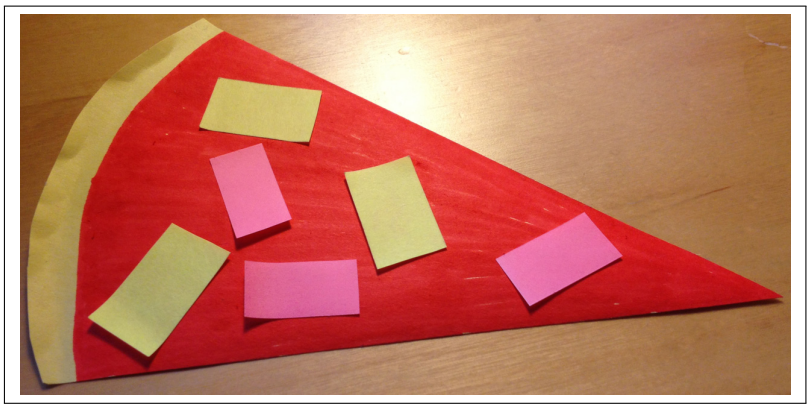

Abbildung 3.1: Ein Stück *Pizza Hawaii* aus dem Kanban Pizza Game

Die Teams erhalten noch ein wenig Zeit, um ihren eigenen Prozess abzusprechen. Anschließend haben sie 5 bis 7 Minuten Zeit so viel *Pizza Hawaii* zu produzieren wie möglich. Die genaue Laufzeit der Runde legt die Moderation fest und gibt sie nicht an Teilnehmer weiter. Zu jeder Runde sollte eine andere Rundenzeit zwischen 5 und 7 Minuten gewählt werden.

Für die *zweite Aufgabe* wird nun Kanban in den Prozess integriert. Der Lehrende kann an dieser Stelle die Kernpraktiken von Kanban (2.3.2) allen Teilnehmern vorstellen. Außerdem wird ein Punktesystem eingeführt, das die Einhaltung der Kernpraktiken belohnt: Jedes fertige Stück Pizza erhält +10 Punkte. Jedes unfertige Teilprodukt gibt Minuspunkte: Pizzaboden (mit oder ohne Tomatensoße) -4 Punkte und Belegungen der Pizza je -1 Punkt.

Jedes Team muss den eigenen Prozess analysieren und Stationen für jeden Arbeitsschritt auf den Tisch mit Klebeband markieren. Die Stationen müssen ausreichend groß sein, damit Zwischenprodukte und Material darin Platz haben. Für jede Station muss außerdem ein WiP-Limit (2.3.4) festgelegt werden. Die Moderation sollte nochmals auf den Qualitätsanspruch der Pizzastücke hinweisen: Genügend Tomatensoße, ordentlich geschnittene und gleichmäßig verteilte Belegung.

Für das Anlegen der Stationen und Setzen der WiP-Limits hat jedes Team 5 Minuten Zeit. Zu Beginn der zweiten Runde müssen alle unfertigen Materialien der letzten Runde vom Tisch entfernt werden. Ist dies geschehen, dann startet die zweite Runde.

Ist die zufällig gewählte Zeit von 5 bis 7 Minuten vorbei, dann werden die Punkte zusammengezählt und von jedem Team auf Tafel oder Flipchart-Papier gesammelt. Die

Teams haben anschließend eine Minute Zeit für Überprüfungen, welche Arbeitsschritte gut oder schlecht liefen. Sie erhalten eine weitere Minute um den Prozess zu verändern und neue WiP-Limits zu setzen. Nach dem Wegräumen aller Teilprodukte der letzten Produktion beginnt die *dritte Runde* - wieder mit einer zufälligen Laufzeit von 5 bis 7 Minuten und erneut werden im Anschluss die Punkte gesammelt.

Für die *vierte und letzte Runde* wird die Aufgabe erweitert: Die neue Sorte Pizza Rucola wird eingeführt und jedes Team erhält einen Satz mit Bestellungen. Rucola wird mit grünen Haftnotiz-Streifen, die etwa die halbe Breite und doppelte Länge der anderen Belegungen haben, dargestellt. Es müssen 7 Rucola-Streifen *nach* dem Backen im Backofen auf ein Pizzastück gelegt werden. Das Punktesystem verändert sich ebenfalls: Es werden nur Pizzastücke gezählt, die zu einer fertigen Bestellung gehören. Nicht zugeordnete, aber fertige Pizzastücke geben Minuspunkte - Pizzaboden und Belegung werden gezählt, wie bei Teilprodukten. Eine neue Bestellungen muss von einer zentralen Station abgeholt werden und fertig bearbeitete Bestellungen in eine zentrale Station gelegt werden. Bevor die Runde startet sind erneut alle alten Teilprodukte zu entfernen und nach den 5 bis 7 Minuten erneut die Punkte zu sammeln.

Die *letzte Aufgabe* für die Teams besteht darin, den Prozess des Spiels auf einem Kanban-Board zu visualisieren. Dafür kann der Lehrende die Möglichkeiten zur Visualisierung aus den Lehrinhalten (2.3.3.1) präsentieren. Ziel der Aufgabe ist, nochmals über das Geschehen und Gelernte im Spiel nachzudenken. Als Kanban-Board wird ein Flipchart-Papier verwendet und statt der Tickets kann übrig gebliebenes Material aus der Durchführung verwendet werden.

Zum Reflektieren können schließlich alle Kernpraktiken mit dem Geschehen im Spiel, den Messungen anhand des Punktesystems und den Kanban-Boards verglichen werden.

Quelle

Kruse u. Ivancsich (vgl. 2011)

Bewertung

Das *Kanban Pizza Game* ist als Lehrmethode besonders für den Einstieg in Kanban geeignet, weil der Fokus auf den Kernpraktiken (2.3.2) liegt. Die Lehrmethode endet mit der Visualisierung der Pizza-Produktion auf Kanban-Boards, daher bietet sich der Übergang zu den Möglichkeiten für Visualisierungen auf Kanban-Boards (2.3.3.1) an.

Wenn mehr Zeit zur Verfügung steht, lässt sich das *Kanban Pizza Game* durch eine weitere Retrospektive (2.2.3.5) und eine fünfte Runde, die identisch abläuft wie die vierte, erweitern. Dadurch kann der Lehrende die Durchführung von Retrospektiven vermitteln und die Wirksamkeit auf den Prozess verdeutlichen.

3.5 Lego Scrum Simulation

Das *Lego Scrum* simuliert den gesamten Scrum-Prozess und lässt dabei den skaliertem Ansatz mit einfließen.

Durchführung

Benötigte Zeit
: 120 bis 180 Minuten

Anzahl Personen
: 4 bis 25 Personen

Benötigtes Material
: Für jedes Team, bestehend aus 4 bis 6 Personen, wird eine Lego Box mit Grundbausteinen benötigt. Außerdem Moderationskarten oder große Haftnotizen, Flipchart Papier, Permanent Marker, Buntstifte (blau, grün und grau) und ein Planning Poker Kartenset für jeden Teilnehmer. Das Kartenset kann auch selbst aus Moderationskarten hergestellt werden (pro Karte eine Zahl: 0, 1, 2, 3, 5, 8, 13, 20, 40 und 100). Für jedes Team und das Produkt-Inkrement wird ein Tisch benötigt. Außerdem eine Stoppuhr-Applikation, die über einen Beamer für alle sichtbar gemacht wird.

  Der Lehrende sollte die Produktvision einer Stadt vorbereitet haben. Die Stadt kann beliebige Eigenschaften haben, zum Beispiel: Geschäfte, Parks, Flüsse, Kirchen, Krankenhäuser, Industrie, Brücken, Bahn- und Buslinien. Idealerweise werden auch einige Charakteristiken einer Stadt formuliert, zum Beispiel: Asiatische Architektur oder Stadt mit hoher Bevölkerungsdichte (dadurch wird symbolisiert, dass mehr Hochhäuser benötigt werden).

  Optional kann der Lehrende auch das Product Backlog mit den User Stories (2.2.4.1.2) vorbereiten. Dadurch werden in der Durchführung etwa 15 bis 20 Minuten gespart, aber auch das Entwickeln von User Stories nicht vermittelt.

Beschreibung

Zu Beginn findet sich die Gruppe in Teams aus 4 bis 6 Personen zusammen. Jedes Team stellt ein Scrum-Team dar und wird einem Tisch zugeordnet, auf dem sich eine Lego Box befindet. Für das Produkt-Inkrement wird ebenfalls ein Tisch vorbereitet, der mit Flipchart Papier bedeckt ist, damit Wiesen, Straßen und Flüsse aufgemalt werden können. Die Rolle des Chief Product Owners (2.2.6.2) übernimmt der Lehrende und stellt die vorbereitete Produktvision vor.

Im nächsten Schritt wird das Product Backlog mit User Stories (2.2.4.1.2) anhand der Produktvision erstellt (falls nicht vorhanden) und geschätzt. Dazu wird an dieser Stelle die Lehrmethode Planning Poker (3.9) oder alternativ Magic Estimation (3.7) verwendet. Die User Stories werden auf Moderationskarten notiert (inklusive Akzeptanzkriterien) und an Wand oder Tafel geklebt.

Es werden nun 3 bis 4 Sprints simuliert: 3 Minuten für *Sprint Planning* (2.2.3.2), 5 Minuten für den *Sprint* selbst (2.2.3.1), 5 Minuten für das *Sprint Review* (2.2.3.4) und 3 Minuten für die *Sprint Retrospektive* (2.2.3.5). Für alle Ereignisse wird die Stoppuhr-Applikation auf dem Beamer gezeigt und sich strikt an die Zeitbegrenzung gehalten.

Für das *Sprint Planning* (2.2.3.2) wird ein gemeinsames Sprint Backlog (2.2.4.2) für alle Scrum-Teams direkt neben dem Product Backlog aufgebaut. Das Sprint Backlog ist eine Matrix aus den Teams (Spalten) und Sprint-Iterationen (Zeilen). Das Sprint Planning wird von den Product Ownern der Teams moderiert, die jeweils User Stories aus dem Product Backlog pullen.

In der *Sprint* (2.2.3.1) Durchführung werden User Stories durch die Teams umgesetzt. Optional kann ein Scrum Master (2.2.2.2) vom Team gewählt werden, der die Umsetzung für das Team moderiert und sich Notizen für Verbesserungen macht. Die Product Owner beteiligen sich nicht an der Umsetzung, sondern verfeinern User Stories für den nächsten Sprint, sprechen Details der Produktvision ab und beantworten Fragen der Teams. Mit Ablauf der Zeit müssen die Ergebnisse im Produkt-Inkrement integriert sein (inklusive Straßen, Wiesen und Flüsse malen).

Im *Sprint Review* (2.2.3.4) fragt der Lehrende (Chief Product Owner) als erstes: „Wo ist die Stadt?“ Damit soll signalisiert werden, dass das Produkt-Inkrement fertig integriert sein muss. Die Product Owner entscheiden, ob die User Stories vollständig abgeschlossen sind, ansonsten fließend sie zurück ins Product Backlog. Dabei achten sie auf einige Details und begründen ihre Entscheidungen. Zum Beispiel: Passen alle Häuser proportional zusammen? Sind die Farben der Steine für die Gebäude chaotisch gewählt (nicht

symmetrisch)? Haben die Gebäude Fenster und Türen?
Die Schätzungen für User Stories, die zurück ins Product Backlog fließen, werden nur grob beim Zurückhängen neu geschätzt. Dabei wird innerhalb der Zeile für die Sprint-Iteration der geplante Schätzwert, der sich aus den ursprünglichen User Stories ergibt, und der tatsächliche Wert, der sich aus dem ursprünglichen Wert abzüglich der neuen Schätzungen zurückgehängter User Stories errechnet, notiert. Diese beiden Werte ergeben ein *Burndown Chart*, somit eine Angabe für die restliche Arbeit für die übrigen Sprints und außerdem ein Verhältnis zwischen Schätzung und tatsächlicher Umsetzung.

In der *Sprint Retrospektive* (2.2.3.5) besprechen sich die Teams, wie der nächste Sprint in der Zusammenarbeit verbessert werden kann. Der Scrum Master bringt dazu seine notierten Verbesserungsvorschläge ein.

Im *Abschluss* wird das Geschehen zusammengefasst. Dazu kann im Plenum diskutiert oder Lehrmethoden wie beispielsweise *Kartenabfrage* eingesetzt werden. Die Autoren der *Lego Scrum Simulation* empfehlen dafür die folgenden Leitfragen:

- Was haben die Teilnehmer beobachtet?
- Wie fühlten sich die Teilnehmer, in einem Scrum Team zu sein?
- Wie liefen die kurzen Iterationen?
- Wie genau waren die Schätzungen?
- Was würden die Teilnehmer von Anfang an anders machen, wenn sie die Chance hätten, das Spiel ein zweites Mal zu spielen?
- Was waren die Aufgaben des Product Owner?
- Wie fühlte es sich für die Teilnehmer nach dem ersten Sprint an, als nahezu alle Elemente nachbearbeitet werden mussten?
- Was haben die Scrum Master getan?
- Wie würde sich die Vorgehensweise der Teilnehmer ohne Product Owner ändern?
- Wie lief die Kommunikation zwischen den Teams? Gab es Abhängigkeiten? Wie wurden diese aufgelöst?
- Was haben die Teilnehmer gelernt?

Quelle

Krivitsky (vgl. 2011)

Bewertung

Aufgrund der Komplexität der *Lego Scrum Simulation* ist die Moderation für den Lehrenden herausfordernd. Aus diesem Grund ergibt sich die Begrenzung auf 25 Teilnehmer. Mit Unterstützung eines Kollegen, denen die Durchführung bereits bekannt ist, kann die Simulation auch von mehr Personen gespielt werden. Bei größeren Gruppen ist als Alternative das *Ball Point Game* (3.1) besser geeignet, auch wenn es weniger Aspekte von Scrum betrachtet.

Als Hilfestellung bei der Moderation kann der Lehrende außerdem Aufgaben an bestimmte Rollen in den Scrum-Teams *delegieren*. So kann es beispielsweise passieren, dass der Lehrende in seiner Doppelrolle die Stoppuhr aus den Augen verliert. Ist dies der Fall, kann er das Ansagen der Zeit und das Zurücksetzten der Uhr an einen Product Owner oder Scrum Master weitergeben. Da die Scrum-Teams selbstorganisiert sind, sollte die Delegation nicht als Verpflichtung, sondern als Wunsch formuliert sein.

3.6 Marshmallow Challenge

Die *Marshmallow Challenge* ist eine Lehrmethode zur Förderung von Teamarbeit, Innovation und Kreativität. Die Lehrmethode soll die Beteiligten zu der Einsicht bringen, dass die vollständige Planung eines Projektes mit (versteckten) unklaren Faktoren und Problemen nicht so zielführend ist wie die iterative und inkrementelle Erstellung von Prototypen mit Verfeinerungen, die solche Unklarheiten beseitigen. Für die (agile) Softwareentwicklung ist diese Lehrmethode daher sehr geeignet, weil Softwareprojekte Teamarbeit, Innovation und Kreativität erfordern und in der Praxis neue Kundenbedürfnisse decken sollen, die erst im Laufe eines Projektes über regelmäßiges Feedback durch den Kunden verstanden werden (vgl. Pichler, 2008, S. 3 f.).

Die Aufgabe der *Marshmallow Challenge* besteht darin aus 20 Spaghetti, ein Meter Klebeband, ein Meter Schnur und einen Marshmallow innerhalb von 18 Minuten in Teamarbeit eine stabile Konstruktion zu entwickeln, die den Marshmallow trägt. Dabei wird eine Wettkampfsituation inszeniert, in der das Team gewinnt, das den Mashmallow am höchsten mit einer stabilen freistehenden Konstruktion trägt. (vgl. Durchführung)

Durchführung

Benötigte Zeit
: 45 bis 60 Minuten

Anzahl Personen

Es sollten mindestens vier Teilnehmer zur Bildung eines Teams vorhanden sein. Die Lehrmethode lässt sich auf beliebig viele Teilnehmer skalieren, da alle Teams parallel arbeiten.

Benötigtes Material

Für jedes Team werden folgende Materialien benötigt:

- 1 Papierbeutel, in dem das Material „versteckt“ wird
- 20 ungekochte, nicht zu dünne und brechbare Spaghetti
- 1 Meter händisch reißbare Schnur (alternativ eine Schere beifügen)
- 1 Meter händisch reißbares Klebeband
- 1 Marshmallow in „Standardgröße“ (etwa drei Zentimeter Länge)

Für die allgemeine Durchführung wird außerdem ein Maßband benötigt, um die Höhe der Konstruktion zu messen. Außerdem sollte für alle Teams die verbleibende Zeit während der Durchführung sichtbar sein. Dazu eignet sich beispielsweise ein Beamer mit einem Stoppuhr-Programm. Alternativ oder zusätzlich kann über eine Soundanlage eine Playlist von exakt 18 Minuten abgespielt werden.

Beschreibung

Zu Beginn der *Marshmallow Challenge* sollten die Ziele und Regeln klar vermittelt werden: Es ist unter allen teilnehmenden Teams die größte freistehende Konstruktion mithilfe der beschriebenen Materialien zu entwickeln, an dessen Spitze der Marshmallow angebracht ist. Gemessen wird vom unteren Ende der Konstruktion bis zum oberen Ende des Marshmallows. Stühle, Tische oder andere Gegenstände, auf denen die Konstruktion steht werden nicht gemessen. Wird der Marshmallow teilweise oder ganz gegessen beziehungsweise geteilt, wird das entsprechende Team vom Spiel disqualifiziert. Die Materialien dürfen teilweise oder ganz für die Konstruktion aufgebraucht werden. Das Spaghetti, Schnur und Klebeband dürfen beliebig zerteilt werden. Die Papiertüte darf nicht als Teil der Konstruktion verwendet werden. Nach Ablauf der 18 Minuten dürfen die Konstruktionen nicht mehr berührt werden, sonst werden entsprechende Teams ebenfalls disqualifiziert.

Sind die Regeln dargestellt, idealerweise mit Verständnis-Nachfrage und Wiederholung, dann kann die *Challenge* beginnen, indem die Stoppuhr gestartet wird. Von der Moderation sollte hin und wieder die restliche Zeit angesagt werden, beispielsweise bei 12, 9, 7,

5, 2 und 1 Minuten, sowie 30 und 10 Sekunden. Um den Wettbewerb zu unterstützen, kann der Status von einzelnen Teams für die gesamte Gruppe kommentiert werden, zum Beispiel wenn ein Team bereits eine stabile Konstruktion entwickelt hat. Sind die 18 Minuten fast abgelaufen, sollte nochmals daran erinnert werden, dass Teams ihre Konstruktion nicht festhalten oder stützen dürfen, sobald die Zeit abgelaufen ist. Viele werden es versuchen, wenn ihre Konstruktion mit der Platzierung des Marshmallows teilweise langsam umknickt und direkt zerfällt. Nur Teams mit stabilen Konstruktionen können gewinnen.

Nachdem die 18 Minuten abgelaufen sind, werden die Ergebnisse der Teams von der augenscheinlich kleinsten bis höchsten Konstruktion gemessen. Jede vermessene Höhe wird der gesamten Gruppe genannt und optional für weitere Auswertungen dokumentiert. Das Sieger-Team erhält eine kurze Ehrung, eventuell einen Preis (falls ausgeschrieben). Zum Abschluss der Übung werden die Geschehnisse reflektiert.

Es kann zum Reflektieren eine Präsentation oder eine Beschreibung mit folgenden Inhalten verwendet werden, die auf statistischen Ergebnissen aus vergangenen Durchführungen von Wujec (2010) basieren. Die Kernaussage der Präsentation ist, dass Kinder in jeder Messung größere und interessantere Konstruktionen entwickelten als Studenten der Betriebswirtschaftslehre. Kinder investieren mehr Zeit in die Entwicklung von Prototypen, arbeiten iterativ und inkrementell und beginnen ihre Konstruktion beim Marshmallow. Die Studenten hingegen arbeiten sequentiell, investieren mehr Zeit in die Planung der Konstruktion, führen ihren Plan anschließend aus und haben meistens keine Zeit mehr übrig, um fehlerhafte Designentscheidungen zu beheben, sobald der Marshmallow auf die Spitze gesetzt wird.

Quelle

Wujec (vgl. 2010)

Bewertung

Nach Wujec (2010) ist die *Marshmallow Challenge* eine Metapher für die versteckten Annahmen eines Projektes: Der Marshmallow ist leicht und fluffig und kann von Spaghetti leicht getragen werden. In Verbindung mit einer solchen Konstruktion wirkt der Mashmallow plötzlich nicht mehr so leicht, denn er kann das gesamte Produkt zum Einsturz bringen. Die Erkenntnis, die daraus gezogen werden soll ist, dass ein reales Projekt ebenfalls aus vielen Annahmen besteht. Es bietet sich daher an, im Rahmen des Reflektierens solche möglichen Annahmen

und Möglichkeiten zur Aufdeckung von den Teilnehmern zu sammeln. Beispielsweise durch frühes und häufiges Testen oder durch iterative und inkrementelle Vorgehensweisen.

Die *Marshmallow Challenge* bietet sich für den Einstieg in die agile Softwareentwicklung an, weil ein Bezug zu Erfolgsfaktoren von Projekten hergestellt wird. So lassen sich Aspekte sammeln, die in der weiteren Bearbeitung des Themas erneut aufgegriffen werden können. Eine Möglichkeit wäre daher, das Manifest für agile Softwareentwicklung im Anschluss an diese Lehrmethode zu thematisieren.

3.7 Magic Estimation

Magic Estimation ist ein Schätzverfahren, das für ein umfangreiches Product Backlog (2.2.4.1) und große Teams geeignet ist. Die Schätzung wird schweigsam durchgeführt. Ähnlich wie bei *Planning Poker* (3.9) können im Rahmen einer Lehrmethode beliebige Objekte geschätzt werden. Je nach Gruppengröße sollte das Schätzobjekt jedoch groß genug sein, dass bei einer Aufteilung auf jeden Teilnehmer zumindest drei Aufgaben fallen.

Durchführung

Benötigte Zeit
: 20 bis 30 Minuten

Anzahl Personen
: Es sollten zumindest 4 Personen zur Verfügung stehen. Nach oben ist die Lehrmethode durch die Raumgröße und der Anzahl Teilaufgaben des Schätzobjektes begrenzt.

Benötigtes Material
: Moderationskarten beliebiger Farbe, Klebeband und Permanent Marker zum Beschreiben. Bei größeren Gruppen werden mehrere Permanent Marker benötigt: Etwa einer auf 10 Teilnehmer.

Beschreibung
: Zur Vorbereitung wird eine lange Skala der unreinen Fibonacci-Reihe an eine Wand des Raumes angebracht, beispielsweise mit Moderationskarten. Die Zahlen der Skala stellen *Storypoints* (2.2.4.1) dar. Außerdem stehen die Zahlen mit einem Abstand an der Wand, der zum Verhältnis der Zahlen zueinander in etwa passt. Zum Beispiel:

1, 2, 3, . 5, .. 8, ... 13, ... 20, ... 40, ... 100

Ist die Gruppe so groß, dass fast ein ganzer Raum gefüllt ist, dann sollte die Skala an der längeren Wand des Raumes angebracht werden und die gesamte Breite ausnutzen. Die Schrift der Skala sollte so groß sein, dass die Zahlen von der anderen Seite des Raumes lesbar sind.

Ist das Schätzobjekt im Rahmen der Lehre noch nicht in Aufgaben aufgeteilt, ist dies der nächste Schritt. Dazu wird die Gruppe in Teams, bestehend aus 4 bis 10 Personen, geteilt, die jeweils einen unterschiedlichen Aspekt des Schätzobjektes zerlegen. Die Aufgaben sollten von den Teams mit einem Schlagwort in Großbuchstaben auf der Vorderseite der Moderationskarte beschrieben werden. Die genauere Beschreibung kommt auf die Rückseite. Nach der Aufteilung des Schätzobjektes sollten zumindest 3 Aufgaben für jeden Teilnehmer verfügbar sein.

Der Lehrende sammelt die Aufgaben ein, mischt sie und verteilt sie auf alle Teilnehmer, so dass jeder etwa gleich viele Aufgaben erhält. Ab jetzt wird der Rest der Lehrmethode vollkommen schweigend durchgeführt: Kein Austausch unter den Teilnehmer ist erlaubt.

Die Teilnehmer lesen sich ihre Aufgaben durch und kleben die Karten unterhalb der Zahlen der Skala an die Wand, die ihrer Meinung nach in etwa zum Verhältnis der ihnen bekannten Aufgaben passen. Jede Aufgabe ist dabei genau einer Zahl der Skala zugeordnet. Zwischenwerte gibt es nicht. Außerdem haben die Storypoints keinen Maßstab - das ist in diesem Fall so gewollt, denn dieser entsteht im Verlauf der Schätzung. Je schwieriger das Verhältnis der Aufgabe zu den Storypoints durch einen Teilnehmer bestimmt werden kann, desto höher sollte er diese Aufgabe bewerten.

Hat ein Teilnehmer alle seine Aufgaben bewertet, liest er sich die anderen Karten auf der Skala nacheinander durch und verändert die Position der Aufgaben, die seiner Meinung nach an der falschen Position kleben. Versteht ein Teilnehmer eine Aufgabe nicht, dann klebt er diese zur 100. Das Lesen und Verändern der Position führen im Verlauf alle Teilnehmer parallel durch.

Der Lehrende beobachtet die Durchführung und markiert Karten, die auffallend oft auf der Skala springen. Daraus lässt sich eine Meinungsverschiedenheit unter den Teilnehmern erkennen. Das Spiel wird beendet, sobald sich keine Karte mehr bewegt, es nur markierte Karten gibt oder sich die Teilnehmer von der Skala abwenden.

Wurde für die Schätzung ein Product Backlog (2.2.4.1) verwendet, dann können die Zahlen in dieses übernommen werden.

Quelle
: Opelt u. a. (vgl. 2012, S. 49 f.)

Bewertung

Magic Estimation ist besonders in Projektunterricht (4.4) geeignet, wenn eine große Menge an Aufgaben schon verfügbar ist und in kurzer Zeit geschätzt werden soll. Nach Opelt u. a. (vgl. 2012, S. 49) lassen sich mit dieser Schätzmethode in etwa 70 Aufgaben mit einer Gruppe von 10 Personen in 20 Minuten schätzen.

Ein Nachteil dieses Schätzverfahrens als Lehrmethode ist die Schweigsamkeit der Teilnehmer. In Verbindung mit anderen Lehrmethoden sind die Teilaufgaben häufig auf Moderationskarten geschrieben und damit nicht ausreichend ausführlich, um von jedem Teilnehmer verstanden zu werden. Daher ist die Kommunikation sehr hoch zu bewerten. Bei der Auswahl des Schätzverfahrens kommt es darauf an, ob Geschwindigkeit oder Kommunikation wichtiger sind. Ist die Kommunikation wichtiger, ist *Planning Poker* (3.9) als Lehrmethode zum Schätzen besser geeignet.

3.8 Pair Storytelling

Die Lehrmethode *Pair Storytelling* ist eine analoge Version der Entwicklungspraktik Pair Programming, die primär in der agilen Methode Extreme Programming verwendet wird, weil sie dort zu den Schlüsselpraktiken zählt. Die Aufgabe dieser Lehrmethode besteht darin, in Paaren ein Kindermärchen, wie beispielsweise *Hänsel und Gretel*, auf das 21. Jahrhundert zu übertragen. Dabei dürfen die beiden Autoren beliebig kreativ werden.

Durchführung

Benötigte Zeit
: 90 Minuten

Anzahl Personen
: 2 bis beliebig

Benötigtes Material
: Vorbereitete Moderationskarten mit den Titeln von Kindergeschichten und optional

weiteren Aufgaben auf der Rückseite der Karte, zum Beispiel: „Schreibe nicht mehr als zwei DIN-A4 Seiten“.

Außerdem können Bücher mit Kindergeschichten oder Computer mit Internetzugang zum Nachschlagen der Geschichten hilfreich sein.

Beschreibung

Zunächst sollte der Lehrende das Prinzip der Praktik Pair Programming kurz vorstellen, damit die Teilnehmer einen Bezug aus dem Ablauf von *Pair Storytelling* herstellen können. Anschließend erhält jeder Teilnehmer eine der vorbereiteten Moderationskarten und hat 20 Minuten Zeit die darauf genannte Geschichte auf das 21. Jahrhundert zu adaptieren. Nachdem die Zeit abgelaufen ist, dürfen 3 bis 4 Teilnehmer ihre Geschichten vorlesen.

Im zweiten Schritt finden sich die Teilnehmer in Paaren zusammen. Jedes Paar erhält wieder eine der vorbereiteten Moderationskarten, die idealerweise eine neue Geschichte benennt, die die beiden noch nicht im ersten Teil der Lehrmethode umgeschrieben haben. Jedes Paar hat nochmals 20 Minuten Zeit die Geschichte *gemeinsam* auf das 21. Jahrhundert zu übertragen.

Optional kann während der Bearbeitung, alle 7 Minuten, eine Person jedes Teams den Partner wechseln. Zunächst die linke und beim zweiten Wechsel die rechte Person, so dass beide Personen mit drei unterschiedlichen Partnern und idealerweise an zwei unterschiedlichen Geschichten arbeiten.

Nach der zweiten Runde dürfen die Paare ihre Geschichten vorlesen, beziehungsweise bei großen Gruppen nur einige. In dem meisten Fällen sollte das Vorlesen der Geschichten die Teilnehmer zum Lachen bringen.

Zur *Auswertung* wird die Frage gestellt: *Wie war es für Dich?* Die Frage ist offen, damit jeder Teilnehmer, wenn er möchte, dazu etwas sagen kann. Werden Aspekte nicht genannt, können diese direkt nachgefragt werden:

- Wie hat sich die gemeinsame Arbeit im Gegensatz zur Einzelarbeit angefühlt?
- Was war in beiden Arbeitsformen identisch?
- Waren die Ergebnisse der gemeinsamen Arbeit kreativer, künstlerischer oder vollständiger als es allein möglich wäre?
- Wurde die gemeinsame Aufgabe konzentrierter oder weniger konzentriert umgesetzt?

- Hat es allein oder im Paar mehr Spaß gemacht?
- Wie war es, nicht zu wissen, was der Partner als nächstes schreibt?
- Hat der Partner für Kreativität gesorgt?
- Hat der Partner dabei geholfen Fehler in der Geschichte zu finden?
- Wurde die Produktivität durch die Teamarbeit erhöht oder verringert?
- Könnte sich diese Simulation vom *Pair Programming* unterscheiden?

Quelle

Grossman u. Bergin (vgl. 2005)

Bewertung

Die Lehrmethode *Pair Storytelling* verlangt von den Teilnehmer viel Kreativität. Besonders in der ersten Runde, in der die Teilnehmer unter Zeitdruck noch alleine eine Geschichte umschreiben müssen, kann dies zur Überforderungen und Frustration bei Einzelpersonen führen. Diese äußern sich besonders bei anonymen Gruppen durch Störungen, wie beispielsweise Arbeitsverweigerungen oder Widerständen (4.5). Für den Lehrenden ist es daher wichtig, darauf vorbereitet zu sein und mit ruhiger Art, Verständnis und positivem Zuspruch zu intervenieren.

Nachdem die ersten Teilnehmer ihre Geschichten vorgelesen haben, wird die Stimmung schließlich ins positive umschwenken, weil sich erfahrungsgemäß sehr lustige und unterhaltsame Geschichten ergeben.

3.9 Planning Poker

Planning Poker ist ein Schätzverfahren, das auch in der Praxis zum Schätzen von User Stories (2.2.4.1.2) angewendet wird und verfolgt dabei den Ansatz der Expertenschätzung. Im Rahmen einer Lehrmethode können beliebige Objekte geschätzt werden, beispielsweise Funktionalitäten bekannter Applikationen wie Facebook oder Twitter, aber auch Unternehmungen wie eine Weltreise, ein Umzug ins Ausland oder die Neueinrichtung des Lehrgebäudes.

Durchführung

Benötigte Zeit

Es werden 20 Minuten zum Aufteilen des Schätzobjektes in Teilaufgaben benötigt und

20 Minuten zum Schätzen selbst. Ist ein Product Backlog (2.2.4.1) verfügbar, werden somit nur 20 Minuten für die Lehrmethode benötigt.

Anzahl Personen

4 bis 10 Personen

Benötigtes Material

Jeder Teilnehmer benötigt einen Satz Spielkarten, die eine unreine Fibonacci-Reihe beschreiben. Die Karten können mit selbst beschrifteten kleineren Moderationskarten ersetzt (pro Karte eine Zahl: 0, 1, 2, 3, 5, 8, 13, 20, 40 und 100) oder erworben werden. Zur Dokumentation der Aufgaben und Schätzwerte wird außerdem eine Tafel oder 1 bis 2 Flipchart-Papiere benötigt.

Beschreibung

Für die Lehrmethode ist es nicht nötig, dass das Schätzobjekt in User Stories oder Use Cases aufgeteilt ist. Zum Schätzen ist jedoch zumindest eine beliebige Aufteilung notwendig. Daher sollte das Schätzobjekt im ersten Schritt von den Teilnehmern in Aufgaben zerteilt werden, die an Tafel oder Flipchart dokumentiert werden, wenn kein Product Backlog (2.2.4.1) zur Verfügung steht.

Optional kann auch ein Eintrag erstellt werden, der das gesamte Schätzobjekt umschreibt. Dieser sollte in der eigentlichen Schätzung von der Reihenfolge als zweites geschätzt werden. Der geschätzte Eintrag kann später zum Vergleich herangezogen werden.

Zunächst weist die Moderation darauf hin, dass mit *Storypoints* (2.2.4.1) gearbeitet wird, nicht mit dem Aufwand in Stunden oder Tagen. Die Moderation kann auch von einem Teilnehmer geführt werden, der sich so in die Rolle eines Product Owners (2.2.2.1) begibt. Das Team wählt für die Bestimmung der Maßeinheit *Storypoint* eine möglichst kleine Aufgabe aus. Diese Aufgabe erhält den Schätzwert von einem Storypoint.

Im Anschluss werden die Aufgaben in Reihenfolge nach folgendem Ablauf geschätzt: Die Aufgabe wird kurz von dem Product Owner erläutert und kann bei Verständnisproblemen diskutiert werden. Sind alle Fragen geklärt, wählt jeder Teilnehmer eine Karte aus seinem Deck aus, die seiner Meinung nach einen korrekten Schätzwert darstellt. Dabei lässt sich keiner in seine Karte schauen. Aufgedeckt wird von allen Teilnehmern gleichzeitig, sobald jeder eine Karte gewählt hat. Die beiden Teilnehmer mit dem höchsten und niedrigsten Schätzwert erläutern kurz, wie sie auf ihren Schätzwert gekommen sind. Die Erläuterungen enthalten eventuell neue Informationen, die für eine weitere Schätzung hilfreich sind. Aus diesem Grund wird die Schätzung wiederholt. In den

meisten Fällen gleichen sich nun die Schätzwerte an. Können sich die Teilnehmer nicht auf einen Schätzwert einigen, wird eine weitere Runde gespielt. Ist diese Runde ebenfalls unentschieden, wird der Wert verwendet, der am meisten vorkam beziehungsweise zumindest sinnvoll erscheint, denn ein sinnvoller Wert ist das eigentliche Ziel. Das Ergebnis wird dokumentiert.

Quelle

Opelt u. a. (vgl. 2012, S. 20 f.)

Bewertung

Planning Poker eignet sich für kleinere Gruppen mit bis zu 10 Personen. Stehen mehr Teilnehmer zur Verfügung, kann auf die Schätzmethode *Magic Estimation* (3.7) ausgewichen werden. Alternativ kann *Planning Poker* auch skaliert werden, indem nur die erste Teilaufgabe, die zur Bestimmung eines *Storypoints* dient, im Plenum geschätzt wird. Alle weiteren Teilaufgaben können anschließend unter den Teams aufgeteilt und geschätzt werden.

Planning Poker hat als Lehrmethode neben der Schätzung selbst primär das Ziel ein Schätzverfahren vorzustellen und erlebbar zu machen. Steht der Lehrende zeitlich bereits im Verzug und wird die Schätzung selbst im Anschluss nicht weiter verarbeitet, dann lässt sich die Methode beenden, bevor alle Aufgaben geschätzt sind. Ist dies vor Beginn erkennbar, dann kann vorher auf den optionalen Fall der ganzheitlichen Schätzung verzichtet werden. Eine Auswertung der Ergebnisse ist nicht zeitintensiv, weil die Vergleichswerte ohne Verknüpfung mit einer praktischen Lehrmethode fehlen.

Diese Lehrmethode kann in einer Variante als *Business Value Poker* auch zum Schätzen des *Business Values* (2.2.4.1) verwendet werden. Die Durchführung ist identisch, nur das Kartenset besteht aus den folgenden Zahlen: 100, 200, 300, 500, 800, 1200, 2000 und 3000.

4 Erstellung von individuellen Lehrkonzepten

Dieses Kapitel beschreibt einen Weg zur Erstellung von individuellen Lehrkonzepten. Dazu wird zunächst auf Kompetenzen und Lernziele eingegangen, die eine Basis zur Auswahl von Lehrmethoden darstellen. Der Methodenbaukasten wird anhand einer konkreten Verlaufsplanung als Beispiel für eine Unterrichtseinheit erläutert, in der sich die Lehrmethoden zielgerichtet anordnen. Um den Lehrenden bei der Planung alle gestalterischen Freiheiten zu lassen, wird außerdem auf den Einsatz von Lehrmethoden hingewiesen, die unter 3 Lehrmethoden für agile Softwareentwicklung nicht beschrieben wurden. Abschließend werden weitere Aspekte, die bei der Durchführung einer Unterrichtseinheit von Bedeutung sind, besprochen und mögliche Lösungen vorgeschlagen.

4.1 Kompetenzen

Wird der Erwerb von Kompetenzen angestrebt, dann muss der Begriff „Kompetenz“ zunächst definiert und abgegrenzt werden. In der Literatur, besonders im Zusammenhang mit der Diskussion über Konflikte und Kontroversen, kommt der Begriff in unterschiedlichen Varianten vor und die Definitionen sind unterschiedlich und widersprüchlich (vgl. Hahn, 2011, S. 47-50). Für diese Arbeit wird die *Standarddefinition* verwendet, die von Weinert (2001, S. 27 f.) stammt:

> Dabei versteht man unter Kompetenzen die bei Individuen verfügbaren oder durch sie erlernbaren kognitiven Fähigkeiten und Fertigkeiten, um bestimmte Probleme zu lösen, sowie die damit verbundenen motivationalen, volitionalen[2] und sozialen Bereitschaften und Fähigkeiten um die Problemlösungen in variablen Situationen erfolgreich und verantwortungsvoll nutzen zu können.

Kompetenzen konzentrieren sich somit zentral auf kognitive Fähigkeiten, die ihre Wirksamkeit über den Erwerbskontext hinaus in variablen Situationen zur Lösung neuer Probleme

[2] *Volition* ist die willentliche Steuerung von Handlungen und Handlungsabsichten.

besitzen. Kompetenzen sind im Lernprozess noch nicht vorhanden, sie werden erst über den Lernprozess erworben. Ihre Überprüfbarkeit und Leistungsfähigkeit kommt erst nach dem gesamten Lernprozess zum tragen, zum Beispiel in beruflichen Situationen.

Angestrebte Kompetenzen orientieren sich in erster Linie an einer Domäne - ein Lernbereich oder Fachbereich - und nicht an fachübergreifenden Schlüsselkompetenzen, wie beispielsweise Sozialkompetenz (vgl. Klieme u. a., 2007, S. 22). Nach Weinert (2001) sollte bei einem systematischen Aufbau von Kompetenz bei domänenspezifischen Kompetenzen begonnen werden. Durch zunehmende Vernetzung dieser bereichsbezogenen Kompetenzen können sich auch Schlüsselkompetenzen entwickeln. Zielsetzungen, die über domänenspezifische Kompetenzen hinausgehen, werden in *Kompetenzmodellen* zusammengefasst. Diese Kompetenzmodelle legen nicht nur Fachkompetenzen fest, sondern auch *Methodenkompetenzen*, *Sozialkompetenzen* und *Selbstkompetenzen* (vgl. Cursio u. Jahn, 2013, S. 2).

Ein Beispiel für Kompetenz in der Domäne der Mathematik, die jeder Informatiker im Laufe seines Studiums erworben haben sollte, kann lauten: „Fähigkeit, reale Situationen in die Sprache der Mathematik zu übersetzen, zu lösen und das Ergebnis für reale Situationen zu interpretieren.“ (vgl. Hasemann u. Gasteiger, 2014, S. 72). Es handelt sich um eine grundlegende Kompetenz einer Domäne. Ihre Formulierung beschreibt im Kern Tätigkeiten, die über das Abarbeiten von gegebenen Algorithmen hinaus geht und das *Können* in den Vordergrund stellt. Im Bereich der agilen Softwareentwicklung könnte eine Kompetenz beispielsweise lauten: „Fähigkeit, ein Projekt mit Einsatz einer geeigneten agilen Methode zu gestalten“.

4.2 Lernziele

Während des Lernprozesses im Rahmen einer Lehrveranstaltung wird von *Lernziel* gesprochen, da die angestrebten Kompetenzen in der Regel noch nicht vorhanden sind. Für jeden Unterricht[3] auf allen Ebenen des Bildungssystems, auch betriebliche Weiterbildung, sind Lernziele zu benennen, damit der Unterricht zielorientiert und beurteilbar ist. Die Lernziele werden gezielt darauf ausgerichtet, dass die ausgewählten Kompetenzen erreicht werden können. Für die Formulierung eines Lernziels sind, nach Bloom (1956), Inhaltsbereiche zu spezifizieren und das angestrebte Kompetenzniveau im Umgang mit dem Inhalt zu beschreiben. Inhaltsbereiche (zum Beispiel Kapitel 2 Lehrinhalte) können auf vier *Inhaltsdomänen* vermittelt werden:

[3] *Unterricht* und *unterrichten* werden im Zusammenhang allgemeiner Vorgänge zur Aneignung von Fertigkeiten und Wissen verwendet. Dabei wird nicht zwischen Schule, Studium oder Erwachsenenbildung unterschieden.

1. *Sachwissen* meint das Vermitteln von Fakten wie beispielsweise „Ein Sprint Planning in Scrum besteht aus zwei Phasen“.
2. *Konzeptwissen* setzt Strukturen, Grundgedanken und Modelle in den Mittelpunkt, beispielsweise „Warum wird Scrum in der Softwareentwicklung eingesetzt?“
3. *Prozesswissen* meint die methodische Kenntnis, mit der Prozesse durch den Lernenden künftig ohne fremde Anleitung durchgeführt und weiterentwickelt werden können.
4. *Metakognitives Wissen* umfasst auch Kenntnisse im eigenen Denken, beispielsweise wenn der Lernende durch Selbstreflexion erkennen kann, dass eine Weiterentwicklung eines Prozesses nötig ist.

Das angestrebte Kompetenzniveau im Umgang mit dem Inhalt gliedert sich nach Schwierigkeit:

1. *Erinnern*, somit Wissen lediglich wiedergeben können, beispielsweise „Ereignisse von Scrum nennen können“.
2. *Verstehen*, die sinngemäße Kenntnis über das Erinnerte.
3. *Anwenden*, das Wissen aktiv nutzen können, ohne alle Zusammenhänge zu kennen. Zusammen mit *Konzeptwissen* wäre das beispielsweise „Ein Kanban-Board für einen gegebenen Prozess erstellen können“.
4. *Analysieren*, somit Gemeinsamkeiten und Unterschiede erarbeiten können, beispielsweise zwischen Scrum und Kanban.
5. *Bewerten*, somit Inhalte kritisch einschätzen können.
6. *Entwickeln*, entwickeln von neuem Wissen, beispielsweise ein neues Vorgehensmodell erfinden.

(vgl. Wiechmann, 2003, S. 79-83)

Bei der Formulierung der Lernziele kann es hilfreich sein, die Inhaltsdomänen und Kompetenzniveaus als Matrix darzustellen (vgl. Wiechmann, 2003, S. 83). Werden Kompetenzen angestrebt, dann reicht es nicht aus das Kompetenzniveau „verstanden“ als Ziel zu vereinbaren. Das vermittelte Wissen muss zumindest zur (selbstständigen) Anwendung gebracht werden. (vgl. Lersch, 2010, S. 10)

Lehrmethoden stellen den Weg zu den Lernzielen dar. Informationsvermittelnde Lehrmethoden, wie beispielsweise das Gruppenpuzzle, sowie Frontalunterricht befinden sich in der Matrix im Bereich der Inhaltsdomänen bei *Sachwissen* und *Konzeptwissen* und im Bereich des Kompetenzniveaus bei *Erinnern* bis *Anwenden* (vgl. Wiechmann, 2003, S. 84-89). Entdeckende

Lehrmethoden, bei denen ein Phänomen oder eine Beobachtung in den Mittelpunkt gestellt wird, die von den Lernenden empirisch und selbstständig untersucht werden, befinden sich in der Matrix im Bereich der *Inhaltsdomänen* bei *Prozesswissen* und *Metakognitives Wissen*, sowie im Bereich des Kompetenzniveaus bei *Analysieren* bis *Entwickeln* (vgl. Wiechmann, 2003, S. 90-93).

Die folgenden zwei beispielhaften Lernziele unterstützen die Kompetenz „Fähigkeit, ein Projekt mit Einsatz einer geeigneten agilen Methode zu gestalten“:

- Lernende kennen die Werte und Prinzipien des Manifests für Agile Softwareentwicklung, stellen ihre Bedeutung heraus und erklären ihre Vorkommnisse innerhalb von agilen Methoden.
- Lernende organisieren und wenden Ereignisse von Scrum eigenständig an, strukturieren ihre anstehende Arbeit und lösen mit deren Hilfe auftretende Probleme und Hindernisse, die sich in der Zusammenarbeit (im Team oder in der Lerngruppe) ergeben.

4.3 Verlaufsplanung

Nach der Festlegung der Lernziele lässt sich der konkrete Verlauf einer *Unterrichtseinheit*, einer Veranstaltung oder eines Seminars planen. In der Pädagogik wird dazu eine ausführliche Unterrichtsplanung angewendet (vgl. Gonschorek u. Schneider, 2010, S. 396-405), die nach den Erfahrungen des Verfassers aus dem „myHAW“ Projekt und weiteren Tutorien an der HAW einige nützliche Aspekte enthält. Ziel ist es, in diesem Abschnitt die Verlaufsplanung anhand bestehender Konzepte und Erfahrungen so zu gestalten, dass das möglichst beste Verhältnis aus Aufwand und Nutzen geboten wird. Dieses Verhältnis ist von Bedeutung, weil Unterrichtseinheiten nicht perfekt geplant werden können (vgl. Gonschorek u. Schneider, 2010, S. 299). Es gibt für eine exakte Planung zu viele nicht planbare Faktoren. Dazu gehören zum Beispiel: Anzahl, Motivation oder Stimmung der Teilnehmer, Raumänderungen oder verfügbare Medien. Es ist daher sinnvoll einen Verlaufsplan zu erstellen, der nicht zu detailliert ist und Freiräume sowie Alternativen zulässt.

Eine Verlaufsplanung in kompakter Form hat darüber hinaus den Vorteil, dass der Plan während der Unterrichtseinheit mitgeführt werden kann und dabei überschaubar bleibt. Der Lehrende kann einzelne Details nachschauen, um nicht „ins Schwimmen“ zu geraten. (vgl. Gonschorek u. Schneider, 2010, S. 325)

4.3.1 Rahmenbedingungen

Für die Verlaufsplanung ist es hilfreich, wenn die Rahmenbedingungen der Unterrichtseinheit geklärt sind. Dazu gehört beispielsweise:

- Wie viel Zeit steht zur Verfügung?
- Wie groß ist die Gruppe?
- Findet der Unterricht in einem Hörsaal mit festmontierter Einrichtung statt oder sind zumindest alle Tische verschiebbar?
- Welche Medien wie Tafel, Flipchart, Pinnwand oder Beamer stehen zur Verfügung?
- Dürfen die Wände mit Moderationskarten oder Flipchart-Papier beklebt werden?
- Welches Vorwissen haben die Teilnehmer?
- Kennen die Teilnehmer bereits den Umgang mit aktivierenden Lehrmethoden oder sind sie nur mit Frontalunterricht vertraut?
- Welche Sprachen sprechen die Teilnehmer?
- Kennen sich die Teilnehmer bereits untereinander?

Sind einige der Rahmenbedingungen nicht geklärt, müssen entsprechend alternative Lehrmethoden in die Verlaufsplanung mit einfließen.

4.3.2 Einsatz von Frontalunterricht

Frontalunterricht zählt in den Grundlagen der Pädagogik nicht als Lehrmethode (vgl. Wiechmann, 2003, S. 83), es ist vielmehr ein Lehrverfahren beziehungsweise eine Sozialform des Unterrichts, in der vorgetragenes Wissen dem Lernenden präsentiert wird. Kommunikationswege und Interaktionen werden dabei weitestgehend von der lehrenden Person bestimmt.

Der Frontalunterricht wird oft sehr kritisch gesehen. Das liegt an den Nachteilen, die mit dieser Sozialform verbunden sind - um einige zu nennen: Was der Lehrende präsentiert, wird der Lernende in dem Moment nicht unbedingt lernen. Lernende verfügen über unterschiedliche Motivationen und Kompetenzen, wenn sie am Unterricht teilnehmen. Zum Beispiel sind einige Teilnehmer in einer Vorlesung dem Thema gegenüber aufgeschlossen und interessiert, andere täuschen Aufmerksamkeit bei Anwesenheitspflicht vor oder bleiben ansonsten der Vorlesung fern. Einige finden Aufgaben spannend, andere bemühen sich um Verständnis, um die angestrebten Prüfungen zu bestehen. Außerdem fördert Frontalunterricht keine sozialen

Kompetenzen, da er schon per Definition keine Gruppenarbeit einschließt. Vielmehr wird bei Frontalunterricht eine vorgegebene gegliederte Struktur diktiert, die bei Gruppenarbeit auch parallel und freier erarbeitet werden könnte. (vgl. Gudjons, 2011, S. 27-36)

Nach Terhart (1997, S. 142-144) bietet der Frontalunterricht auch Vorteile, denn er ist „ökonomisch, und zwar deswegen, weil er Zeit sparen hilft, mittelbar auch Geld“. Außerdem ist Frontalunterricht ein didaktisch einfaches Lehrverfahren und erleichtert dem Lehrenden, die Kontrolle über die Teilnehmer zu halten, weil er alle im Blick hat (vgl. Terhart, 1997, S. 142-144).

Gudjons (vgl. 2011, S. 36) stellt eine für das Lehrkonzept dieser Arbeit vereinbare These auf: „Frontalunterricht ist unverzichtbar als Unterrichtsphase mit relativem Stellenwert“. Gudjons (2011) löst sich damit von dem Einsatz als einzige Sozialform und schlägt ein *integriertes Konzept* vor, in dem der Frontalunterricht didaktische Funktionen in einem methodischen Unterricht abdeckt, die nur im Plenum sinnvoll sind. Dabei darf der Frontalunterricht einen relativen Schwellenwert zu anderen Sozialformen wie Gruppenarbeit und Einzelarbeit nicht überschreiten. Terhart (1997, S. 144) gibt außerdem an, dass Frontalunterricht besonders geeignet sei für Unterrichtsinhalte „mit geringem Schwierigkeitsgrad“.

Frontalunterricht kann somit im Lehrkonzept für einfache Zusammenhänge von Inhalten, für Einführung und Abschluss einer Unterrichtseinheit, für Übergänge zwischen Lehrmethoden oder für Zusammenfassungen aus Erkenntnissen sehr nützlich sein.

4.3.3 Einsatz themenunabhängiger Lehrmethoden

Die in dieser Arbeit aufgezeigten Lehrmethoden für agile Softwareentwicklung sind speziell für den Themenbereich *Agile Softwareentwicklung* konzipiert und reichen nicht in jedem Fall aus, um alle Aspekte in eine aktivierende Lehrveranstaltung zu bringen. Damit nicht alle Lücken mit Frontalunterricht gefüllt werden müssen, lassen sich weitere Lehrmethoden einsetzen, die nicht an einen Themenbereich gekoppelt sind.

Im Projekt „myHAW“ und in Tutorien an der HAW wurden durch den Verfasser dieser Arbeit einige allgemeine Lehrmethoden angewendet, die auf positives Feedback von den Teilnehmern stießen. Drei der bereits angewendeten themenunabhängigen Lehrmethoden sollen hier kurz beschrieben werden:

Gruppenarbeit in Stationen

Diese Methode enthält Aspekte aus der Lehrmethode *Stationenlernen* und *Pair Storytelling*. Die Teilnehmer bilden Teams aus zwei bis vier Personen und beginnen mit einer

vom Lehrenden gestellten Aufgabe. Die zu lösende Aufgabe ist für alle Teams die gleiche. Nach einer bestimmten Bearbeitungszeit wechselt jedes Team zu der begonnenen Ausarbeitung eines anderen Teams - wechselt somit die Station.

Die Teams arbeiten nun mit der Ausarbeitung eines anderen Teams weiter und lassen sich durch die vorgefundenen Ideen inspirieren. Gleichzeitig integrieren sie in das Zwischenergebnis ihre eigenen Lösungen und führen die Ausarbeitung fort. Optional kann eine Person jedes Teams bei seiner Station bleiben, um bisherige Lösungsansätze zu begründen und zu verteidigen. Dieser Ablauf verläuft so weiter, bis jedes Team jede Station mindestens einmal besuchte. Je nach Komplexität der gestellten Aufgaben durch den Lehrenden, können auch mehrere Runden durchlaufen werden.

Ist das Spiel beendet, kann es weiter gehen mit der *Präsentation* von Lösungen eines oder mehrerer zufällig ausgewählter Teams. Alternativ oder zusätzlich kann ein *Entscheidungsspiel* durchgeführt werden, wenn eine Lösung zum Weiterarbeiten benötigt wird.

Entscheidungsspiel „Chaos Cocktail Party“

Das Entscheidungsspiel *Chaos Cocktail Party* wurde vor dem „myHAW“-Projektstart mehrfach durchgeführt - erstmals im Sommer 2013. Die ursprüngliche Quelle dafür stammt aus Baldauf (2014). Das Spiel ist sehr gut einsetzbar in Retrospektiven, um zu entscheiden, welche User Stories als nächstes bearbeitet werden sollten.

Die Ausgangssituation für den Einsatz des Spiels sind mehrere Artefakte oder Positionen, die von Teilnehmern vertreten oder verteidigt werden. Ziel ist es diese Artefakte nach dem Konsens der Gruppe zu priorisieren. Für die Durchführung sollte genügend Freiraum zur Verfügung stehen, so dass alle Teilnehmer sich darin bewegen können.

Jeder Teilnehmer beschreibt zunächst auf einer Seite einer Moderationskarte sein Artefakt als Schlagwort oder Zeichnung. Danach bewegen sich alle auf der freien Fläche (wie auf einer Cocktail Party). Sobald die Moderation ein Zeichen gibt, finden sich zwei nahestehende Personen zusammen und diskutieren für eine Minute über ihre Artefakte. Nach Ablauf der Minute stehen jedem Teilnehmer fünf Punkte zur Verfügung, die auf beide diskutierten Moderationskarten verteilt werden müssen. Notiert werden die Punkte auf der Rückseite der Karte. Sind die Punkte verteilt, bewegen sich alle wieder im Raum und treffen sich erneut mit anderen Diskussionspartnern.

Haben sich alle Teilnehmer einmal getroffen, dann werden die Punkte summiert und nach der Summe in Reihenfolge gebracht.

Gruppenpuzzle mit Visualisierung und Quiz

Diese Methode ist eine Abwandlung vom *Gruppenpuzzle* (vgl. Brauer, 2011, S. 82) und eignet sich zum Vermitteln von Lehrinhalten. Sie ist umfangreich, so dass sie gegebenenfalls auf mehrere Unterrichtseinheiten aufgeteilt werden muss.

In der Durchführung wird zunächst ein umfangreiches Thema in Teilthemen unterteilt und auf Teams bestehend aus vier bis fünf Personen verteilt. Die Bearbeitung des Themas muss durch die Teams auf einem Flipchart-Papier zusammenhängend visualisiert werden. Das Flipchart-Papier darf den Titel des Teilthemas enthalten, ansonsten aber keine beschreibenden Worte. Jedes Team muss dadurch das Teilthema verstehen und abstrahieren können, um es visualisieren zu können.

Die fertigen Visualisierungen werden mit etwas Abstand an die Wände gehängt und bilden so Stationen. Jedes Mitglied eines Teams gilt nach Fertigstellung der Visualisierung als *Experte* des Teilthemas.

Im nächsten Schritt werden erneut Teams gebildet, so dass jedes neu zusammengestellte Team aus jeweils einem Experten jedes Teilthemas besteht. Jedes neue Team positioniert sich an jeweils einer Station mit einer Visualisierung. Der Experte, der an der vorgefundenen Visualisierung beteiligt war, unterrichtet sein Team über die Inhalte des Teilthemas. Nach einer bestimmten Zeit wechseln die Teams die Stationen und bringen schließlich das gesamte Thema in Zusammenhang.

Hat jedes Team alle Stationen einmal besucht, finden sich erneut die ursprünglichen Experten-Teams zusammen, die gemeinsam ein Teilthema visualisiert hatten. Sie tauschen sich über die neuen Kenntnisse aus, die während der Stationen-Phase erarbeitet wurden. Zum Abschluss wird das Wissen über die Inhalte auf die Probe gestellt, indem ein Quiz durchgeführt wird. Quiz-Methoden gibt es mehrere und sie können wahlweise an dieser Stelle ausgewechselt werden.

Der Verfasser verwendete ein Gruppen-Quiz mit gegenseitiger Überprüfung durch die Lernenden. Jedes Experten-Team überlegt sich fünf Fragen für sein Teilthema und verteilt auf diese je nach Schwierigkeit 50 Punkte. Die Fragen werden ebenfalls auf Flipchart-Papier geschrieben und zu den Visualisierungen geklebt. Im Anschluss werden die Fragen durch die anderen Teams gemeinsam und schriftlich beantwortet.

Die Expertenteams bewerten die Antworten der anderen Teams und küren so ein Sieger-Team. Wird diese Methode öfter eingesetzt, können Belohnungen durch den Lehrenden, beispielsweise Medaillen, vergeben werden.

In der Literatur und im Internet finden sich darüber hinaus *Methodensammlungen*, die ebenfalls in einem Lehrkonzept eingesetzt werden können:

Bildungsportal (2013)
: Beschreibt über hundert Methoden und geht dabei unter anderem auf Einsatzmöglichkeiten und Gruppengröße ein.

bpb (2014)
: Beschreibt über 270 Methoden für alle Altersgruppen.

Reich (2014)
: Der Methodenbaukasten der *Uni Köln* beschreibt ausführlich über hundert Methoden, die an Fachhochschulen und Universitäten eingesetzt werden können.

Gugel (2011)
: Das Buch beschreibt 2000 Lehrmethoden für Schule und Lehrerbildung. Die Lehrmethoden sind anhand von Themen aus Geschichte und Politik beschrieben, können aber größtenteils für andere Themenbereiche umgestellt werden.

Die oben beschriebenen Methoden, sowie die Methodensammlungen, sollen als Beispiel und Inspiration für die Entwicklung eigener angepasster Lehrmethoden dienen.

4.3.4 Verlaufsplan

Der grundlegende Verlaufsplan, der mit dieser Arbeit vorgeschlagen wird, stellt den eigentlichen *Methodenbaukasten* dar. Dieser gliedert sich in vier Spalten: Methode und Beschreibung, Ziele, Material und Medien, sowie Aufwand. Außerdem ist der Ablauf in Phasen (Einstieg, Problemorientierung, Problembearbeitung, Ergebnisorientierung und Abschluss) nach Gonschorek u. Schneider (vgl. 2010, S. 320) unterteilt, die den Planenden bei der Strukturierung unterstützen und den Einsatz verschiedener Sozialformen fördern. Zu jeder Phase ordnen sich die ausgesuchten Lehrmethoden ein.

Der Verlaufsplan ist nicht in Stein gemeißelt, sondern sollte kontinuierlich an die Bedürfnisse des Lehrenden und seines Unterrichts angepasst werden. Hilfreich ist es, die Planung mittels Software durchzuführen, um Einträge während der Bearbeitung verschieben und detaillieren zu können.

Nach Erfahrungen des Verfassers ist es sinnvoll zunächst die Spalte „Ziele“ mit prototypischen Stichworten auszufüllen, um sich häufiges Umstellen der Lehrmethoden im Nachhinein zu ersparen. Durch die vorherige Festlegung der groben Ziele erhält der Planende bereits einen

„roten Faden“, der anschließend die sinnvolle Zuordnung von Lehrmethoden erleichtert. Dies könnte beispielsweise wie folgt aussehen:

Phase 1: Einstieg

- Zur Begrüßung stelle ich mich den Teilnehmern vor und verdeutliche meine Motivation am Thema
- Allen Teilnehmern den Ablauf der heutigen Veranstaltung vorstellen
- Gegenseitiges Kennenlernen fördern

Phase 2: Problemorientierung

- Versteckte Annahmen und Erfolgsfaktoren von Projekten aufzeigen

Phase 3: Problembearbeitung

- Historie und Inhalte des Manifest für Agile Softwareentwicklung vermitteln
- Bezug zu Erfolgsfaktoren von Projekten herstellen
- Scrum als agile Methode vorstellen

Phase 4: Ergebnisorientierung

- Den Scrum-Prozess erleben und bewerten

Phase 5: Abschluss

- Zusammenfassung der bearbeiteten Inhalte und Erkenntnisse
- Feedback von Teilnehmern einholen

Steht eine grobe Struktur anhand der Ziele fest, können iterativ Lehrverfahren und Lehrmethoden ausgewählt und der Verlaufsplan vervollständigt werden. Der Verlaufsplan einer ersten Unterrichtseinheit könnte beispielsweise folgendermaßen aussehen:

4.3.4.1 Beispiel-Verlaufsplan

Verlaufsplan: Einstieg in die Agile Softwareentwicklung (Veranstaltung #1)			
Methode und Beschreibung	**Ziele**	**Material und Medien**	**Aufwand**
Ergebnisspeicher Flipchart-Papier mit allen bisher erarbeiteten Ergebnisse zu jeder Veranstaltung sichtbar in den Raum hängen. Neue Ergebnisse können durch die TNs jederzeit hinzugefügt werden.	– TN: Ergebnisse zusammenfassen und formulieren. – TN: Ergebnisse bisheriger Veranstaltungen regelmäßig in Erinnerung rufen.	1 Flipchart-Papier mit Überschrift „Ergebnisspeicher“	-
Phase 1: Einstieg			
Begrüßung – Vorstellung meiner Person gegenüber den TNs. Persönliche Seite zeigen: Besondere Momente meines Lebenslaufs und wie ich zur Agilen Softwareentwicklung kam. – Erläutern: Wie bin ich zu erreichen? Wann antworte ich auf E-Mail? ... – Für gute Zusammenarbeit das „Du“ anbieten.	– TN: Kennenlernen der lehrenden Person – L: Beginnen, eine gute Beziehung mit den TNs aufzubauen	Tafel	10 Minuten
Ablauf der laufenden Veranstaltung – Flipchart-Papier mit dem Ablauf der heutigen Veranstaltung sichtbar aufhängen. – Kurz erklären - nicht jeden Eintrag - nicht zu detailliert. – Abgeschlossene Einträge im Laufe der Veranstaltung mit grünem Flipchart Marker abhaken und die Bearbeitungszeit daneben schreiben.	– TN: Sicherheit für Ablauf der Veranstaltung – TN: Einschätzen von Verlauf und Gestaltung – TN: Transparenz für Zielsetzung erreichen – L: Bei Zeitüberschreitungen gemeinsame Alternativen bestimmen können – L: Orientierung am eigenen Zeitplan schaffen	– 1 Flipchart-Papier mit Ablauf – 1 Flipchart Marker	5 Minuten
Vereinbarungen – Vegas-Regel: „What happens in Vegas, stays in Vegas“ – Mobiltelefone stumm schalten – Präsentationen und Skripte werden durch den Lehrenden zur Verfügung gestellt – Lehrender steht in Pausen für Gespräche und Fragen zur Verfügung – Belohnung vereinbaren, wenn sich keiner zur Veranstaltung verspätet Offen für weitere Vorschläge der TNs.	– TN: Sicherheit für offene Kommunikation und eine vertraute Atmosphäre schaffen – TN: Eventuellen zukünftigen Störfaktoren entgegenwirken – TN: Präsenz des Lehrenden wahrnehmen	Tafel (und/oder Beamer)	5 Minuten

Methode und Beschreibung	Ziele	Material und Medien	Aufwand
„Alle, die...“ – TNs bilden Stuhlkreis mit einem fehlenden Stuhl – Ein freiwilliger TN stellt sich in die Mitte – Die Person in der Mitte denkt sich einen Satz aus, der mit „Alle, die...“ beginnt und fordert damit die sitzenden TNs zum Platztausch auf. Zum Beispiel: „Alle, die gerne programmieren.“ – Die Person in der Mitte sucht sich ebenfalls einen Platz und schließlich befindet sich erneut eine Person in der Mitte, die eine neue „Alle, die...“-Aufforderung stellt...	– TN: Gegenseitiges Kennenlernen der Gruppe – TN: Aufnahmefähigkeit für die anschließenden Inhalte durch Bewegung aktivieren	-	15 Minuten
Phase 2: Problemorientierung			
The Marshmallow Challenge – Gruppe in 4er Teams aufteilen – Jedes Team erhält eine Papiertüte mit 20 Spaghetti, 1 Meter Bindfaden, 1 Schere, 1 Marshmallow – 1 Meter Klebeband an den Enden leicht an den Tisch kleben – Ziel und Regeln mittels Folien vorstellen: In 18 Minuten die größte freistehende Konstruktion unter den Teams erbauen, die einen Marshmallow trägt. Disqualifikation, wenn Marshmallow gegessen oder geteilt wird und wenn nach Ablauf der Zeit die Konstruktion noch berührt wird. Verbaut werden darf Klebeband und alles in der Tüte bis auf die Schere. Gemessen wird vom unteren Ende der Konstruktion bis zum oberen Ende des Marshmallows. – Nach Durchführung: Präsentation mit Projektbezug und Auswertung vorstellen	– TN: Eigene Erfahrungen und Einsichten über versteckte Annahmen in Projekten entwickeln – TN: Einbringung eigener Sachkompetenz, Erfahrungen und schöpferischer Kreativität – TN: Relevanz für die Agile Softwareentwicklung entdecken – TN: Aspekte und Erfolgsfaktoren von Projekten wahrnehmen - zum Beispiel: Den Umgang mit begrenzten Ressourcen erfahren – TN: Probleme kooperativ lösen (Zusammenarbeit, Gruppendynamik) – TN: Interesse am weiteren Verlauf der Veranstaltung wecken – TN: Gegenseitiges Kennenlernen im Team	– Vorbereiteten Foliensatz mit Ziel, Regeln – 1 Packung Marshmallows – 1 Rolle Bindfaden – 1 Rolle Klebeband – 10 Din-A4 Papiertüten – 10 Scheren – 2 Packungen Spaghetti – 1 Maßband – Stoppuhr-App	45 Minuten
Pause (15 Minuten)			
Phase 3: Problembearbeitung			

Methode und Beschreibung	Ziele	Material und Medien	Aufwand
Kopfstand Negative Frage stellen und im Anschluss umkehren. Fragestellung: „Wie müsste ein Projekt durchgeführt werden, damit es auf jeden Fall fehlschlägt?“ – Beantwortung in Kleingruppen (10 Minuten; Größe der Kleingruppen nach Anzahl anwesender TN bestimmen - maximal 10 Gruppen) – Gemeinsames Sammeln und Gruppieren der Antworten an der Wand. Dazu wählt jedes Team einen Sprecher, der die Antworten kurz vorstellt, die sich noch nicht an der Wand befinden. – Frage und Antworten umkehren und so gemeinsam die Erfolgsfaktoren von Projekten erarbeiten	– TN: Auseinandersetzung mit (Miss-)Erfolgsfaktoren von Projekten – TN: Erfahrungen und Beobachtungen aus der Marshmallow Challenge reflektieren und mitteilen – TN: Freies Sprechen vor einer größeren Gruppe – TN: Andere Blickwinkel auf Projekte erzeugen – TN: Kooperatives Arbeiten in Teams	– Idealerweise 1 Pinnwand pro Team – Alternativ: Tafel oder Wand bekleben	25 Minuten
Scrum-Einleitung (frontal) – Empirische Prozesssteuerung als Zyklus erläutern – Iteratives-Inkrementelles Modell am Beispiel eines Sprints vorstellen und daran die Ereignisse kurz erläutern – Bezug zu den erarbeiteten Erfolgsfaktoren herstellen	– TN: Einleitenden Überblick über Scrum erhalten – TN: Erfolgsfaktoren in Scrum erkennen	Beamer + vorbereiteter Foliensatz	15 Minuten

Methode und Beschreibung	Ziele	Material und Medien	Aufwand
The Ball Point Game Gruppe in Teams aus 5-11 TNs bilden. Spiel zunächst erläutern: Aufgabe ist es Bälle zu "produzieren", dazu muss jedes Teammitglied den Ball mindestens einmal berührt haben. Anschließend die Regeln vorstellen: – Jeder TN gehört zu einem oder mehreren Teams. – Startpunkt (Person) jedes Balles ist auch der Endpunkt. – Ball darf nicht dem direkten Nachbarn übergeben werden. – Ball muss bei Übergabe unberührt in der Luft gewesen sein. – Heruntergefallene Bälle können nicht mehr produziert werden. – 2 Minuten pro Iteration + 1 Minute für Verbesserungen und Schätzungen (Wie viele Bälle kommen durch den Prozess) in insgesamt 5 Iterationen. Zeiten und Phasenübergänge laut ansagen und Schätzungen, sowie Ergebnisse in Tafelbild eintragen. Gemeinsam im Plenum reflektieren: Welche Iteration war die beste? Welche Auswirkungen hatte die kontinuierliche Verbesserung? Wie wurde der Scrum-Prozess wahrgenommen?	– TN: Scrum-Prozess erleben und bewerten – TN: Relevanz von Schätzungen wahrnehmen – TN: Erkennen, dass jedes Team seine eigene Prozess-Geschwindigkeit entwickelt und diese abschätzen können – TN: Den Sinn von Iterationen bewerten – TN: Kreative Ideen entwickeln, Erfahrungen einbringen – TN: Wünsche, Ideen anderer TN (an-)erkennen – TN: Kooperatives Arbeiten und Probleme lösen in Teams	– 60 Moderationsbälle pro Team – Vorbereitetes Tafelbild für Schätzverfahren – Stoppuhr-App (auf Beamer zeigen)	30 Minuten
Phase 4: Ergebnisorientierung			
Frontal + Kartenabfrage – Die heutigen Aktivitäten mit dem Ablaufplan abgleichen und in Erinnerung rufen – Kartenabfrage in Kleingruppen: „Welche Ergebnisse nehme ich aus der heutigen Veranstaltung mit?“. Ergebnisse an der Wand sammeln und gruppieren. – Ergebnisse gegebenenfalls ergänzen. – Ergänzungen und stellvertretende Ergebnisse aus Gruppierungen in den Ergebnisspeicher eintragen.	– TN: Heutige Veranstaltung memorieren und Schwerpunkte erkennen – TN: Eigene Erfahrungen mit denen anderer TNs abgleichen	– Moderationskarten – 20 Permanent Marker	25 Minuten
Phase 5: Abschluss			

Methode und Beschreibung	Ziele	Material und Medien	Aufwand

Ausblick
Einen Ausblick auf die nächste Veranstaltung geben:
– Wir starten mit einer Simulation des Daily Scum Ereignisses von Scrum mit den Leitfragen:
  – Was habe ich in der letzten Vorlesung gelernt?
  – Was erwarte ich in dieser Vorlesung zu lernen?
  – Was könnte mich heute am Lernen hindern?
– Themenbehandlung: Manifest für Agile Softwareentwicklung + die Werte und Prinzipien
– Vergleichen der Aspekte im Manifest und in Scrum
– Bearbeiten der Rollen von Scrum
	– TN: Möglichkeit zur Vorbereitung wahrnehmen
	– L: TNs zum weiteren Besuchen der Veranstaltung motivieren
	– L: Interessen der TNs wecken
		-	5 Minuten

4-Karten-Feedback In die Mitte des Raumes, oder alternativ an jeden Tisch, werden Moderationskarten in vier verschiedenen Farben mit ausreichend Permanent Markern verteilt. Jeder Teilnehmer hat zumindest eine Karte zu für Feedback zu beschreiben. – Was war gut? (Grüne Karte mit „+“) – Was war verbesserungswürdig? (Rote Karte mit „-“) – Was habe ich mitgenommen? (Gelbe Karte mit einem gezeichneten Koffer) – Was ist offen geblieben? (Blaue Karte mit „???“) Diese vier Karten werden zu den Titel-Karten gelegt.	– TN: Inhalte memorieren, reflektieren und bewerten – TN: Eigene Stimmungen wahrnehmen und ausdrücken – L: Verbesserungsvorschläge für nächste Veranstaltungen sammeln – L: Zeigen, dass die Lehre mir am Herzen liegt	– Grüne, rote, gelbe und blaue Moderationskarten – 20 Permanent Marker	5 Minuten
			180 Minuten
Legende: TN: Teilnehmer; L: Lehrender			

4.3.4.2 Ergebnisspeicher

Die erste ausgewählte Lehrmethode „Ergebnisspeicher“ ist vor der ersten Phase eingetragen, da sie begleitend zu allen Veranstaltungen[4] durchgeführt wird. Im Beispiel-Verlaufsplan wird der Ergebnisspeicher erst in der vierten Phase *Ergebnisorientierung* mit Inhalten gefüllt. Das muss aber nicht immer so sein, denn der Ergebnisspeicher ist dazu da, dass zu jeder Zeit, wenn Ergebnisse erarbeitet wurden, diese dort eingetragen werden können. Alternativ oder zusätzlich zum Ergebnisspeicher kann auch ein *Themenspeicher* angeboten werden, in dem offene Wünsche und Themen der Teilnehmer formuliert und eingetragen werden, die nicht immer sofort bearbeitet werden können. Nach den Erfahrungen des Verfassers mit dieser Methode werden Themen- und Ergebnisspeicher nicht aktiv von Lernenden angewendet, wenn nicht weitere Lehrmethoden diesen einbeziehen.

4.3.4.3 Ablauf der Veranstaltung

Der Ablauf der laufenden Veranstaltung wird vor Beginn sichtbar in den Raum gehängt - siehe dazu Abbildung 4.1.

Neben den im Beispiel-Verlaufsplan erwähnten Zielen ist der „Ablaufplan“ auch eine Stütze für den Lehrenden, denn auch er hat damit eine weitere Möglichkeit - neben dem Verlaufsplan in der Hosentasche - die Zeitplanung zu überprüfen. Dazu trägt er neben den Einträgen nach Abschluss einer Lehrmethode die verbrauchte Zeit ein. Verbrauchen Lehrmethoden deutlich mehr Zeit, als vorher geplant, dann kann der Ablaufplan eine Möglichkeit sein, Alternativen mit den Teilnehmern abzusprechen oder Methoden zu streichen beziehungsweise zu verschieben. Dabei bleibt die Transparenz über den Ablauf gegenüber den Teilnehmern gewahrt.

4.3.4.4 Vereinbarungen

Vereinbarungen eignen sich dazu, bestimmte Störungen in einer Veranstaltung zu vermeiden. Jeder Teilnehmer kommt mit anderen Verhaltensmustern und Erwartungen in eine Veranstaltung. Einige der Teilnehmer verspäten sich oft, schalten ihr Notebook mit angeschaltetem Sound ein oder lassen Ihr Mobiltelefon klingeln. Wenn es den Lehrenden stört, kann er gemeinsam mit den Teilnehmern Regeln vereinbaren. Diese Regeln sollten dabei auf Gegenseitigkeit beruhen, daher sollten auch Regeln aufgestellt werden, die den Erwartungen der Teilnehmer

[4] *Veranstaltung* wird im Rahmen des Beispiel-Verlaufplans synonym für Unterricht verwendet, weil sich die Planungen an den Rahmenbedingungen eines Moduls im Studiengang *Angewandte Informatik* der HAW orientieren.

Abbildung 4.1: Vorbereiteter Ablauf zum Beispiel-Verlaufsplan auf Flipchart-Papier

entsprechen, wie beispielsweise: „Der Lehrende steht in den Pausen für Gespräche und Fragen zur Verfügung.“ (vgl. Brauer, 2011, S. 47 f.)

Um Vertrauen und eine offene Kommunikation mit Teilnehmern zu fördern, eignen sich weitere Vereinbarungen, wie die in der Beispiel-Veranstaltung aufgeführte *Vegas-Regel* (2.2.3.5).

4.3.4.5 Kennenlernen

In einer aktivierenden Lehrveranstaltung lernen und arbeiten die Teilnehmer häufig zusammen, daher ist es hilfreich Lehrmethoden anzuwenden, die das persönliche Kennenlernen der Teilnehmer untereinander fördern. Außerdem wird nach Brauer (vgl. 2011, S. 41) „ein gutes Klima [...] die Effizienz Ihres Unterrichts“ erhöhen. Im Beispiel-Verlaufsplan wird die Lehrmethode „Alle, die...“ angewendet, die dafür sorgen soll, dass sich die Teilnehmer auf persönlicher Ebene kennenlernen und gleichzeitig nach einem langen Lerntag in Bewegung gebracht werden. Da auch die Beziehung zwischen Lehrenden und Lernenden für effektives Lernen wichtig ist Schumacher (2011, S. 37), kann der Lehrende bei der ausgewählten Kennlern-Methode auch aktiv teilnehmen.

Die anschließenden Lehrmethoden unterstützen ebenfalls das Kennenlernen der Teilnehmer, weil es sich um kooperative Gruppenübungen handelt. Diese Methoden fordern jedoch nicht zum Offenbaren von persönlichen Interessen und Erfahrungen auf.

4.3.4.6 Feedback

Die im Verlaufsplan ausgewählte Lehrmethode *„4-Karten-Feedback“* ist eine Möglichkeit, um anonymes Feedback zu erhalten, das helfen kann, die Veranstaltung für die Teilnehmer zu verbessern. Nach Erfahrungen des Verfassers ist es sinnvoll, am Ende jeder Veranstaltung eine Feedback-Methode einzusetzen, wenn für die Teilnehmer das Unterrichtsgeschehen noch präsent ist. Bei der Abfrage nach Feedback müssen sich die Teilnehmer nochmal erinnern und das Erlebte formulieren, um Feedback geben zu können. Daher kann es hilfreich sein, zu erfragen, welche Lehrmethoden oder Inhalte für die Teilnehmer wichtig waren, zum Beispiel durch die Frage: „Was habe ich mitgenommen?“ (symbolisiert durch einen Koffer in der Durchführung des *4-Karten-Feedbacks*). Dadurch lässt sich erkennen, welche Methoden positiv aufgenommen wurden, präsenter sind, und welche durch Nichtnennung weniger in Erinnerung blieben.

Feedback einzuholen zeigt den Teilnehmern auch, dass dem Lehrenden die Qualität des Unterrichts wichtig ist (vgl. Brauer, 2011, S. 66).

4.3.5 Zeitmanagement

Der Verlaufsplan enthält mit der rechten Spalte die Möglichkeit, eine Unterrichtseinheit zeitlich zu planen. Jedoch gestaltet sich die genaue Planung des Zeitaufwands der einzelnen Lehrmethoden als schwierig, denn der Planende kann vorher nicht wissen, wie aktiv sich die Teilnehmer am Unterricht beteiligen (vgl. Gonschorek u. Schneider, 2010, S. 299). Stellen die Lernenden viele oder wenige Fragen? Finden sie sich schnell in Teams zusammen? Klappen die Übergänge zwischen den Lehrmethoden reibungslos? Es gibt viele Faktoren, die den Zeitplan durcheinander bringen können.

Nach den Erfahrungen des Autors dieser Arbeit ist es sinnvoll Abweichungen der Zeitplanung zu dokumentieren. Entweder, wie schon erwähnt, im Ablaufplan (4.3.4.3) oder auf dem Verlaufsplan selbst. Zur Dokumentation reicht es aus, die verbrauchte Zeit oder die Abweichung neben die Einträge im Verlaufsplan oder Ablaufplan zu notieren. Kommt es im Verlauf der Unterrichtseinheit zu Verzögerungen im Zeitplan, kann eine Lehrmethode weggelassen werden. Dies betrifft eher die weiter hinten liegenden Methoden. Im Beispiel-Verlaufsplan wäre am ehesten die Methode „The Ball Point Game“ betroffen, denn sie liegt spät genug, um

die entstandene Verzögerung einschätzen zu können und befindet sich noch vor der Phase „Ergebnisorientierung“, die das Unterrichtsgeschehen zusammenfasst und somit eine höhere Bedeutung in der Unterrichtsstruktur hat.

Bleibt im Verlauf des Unterrichts Zeit übrig, dann ist es einfacher, darauf zu reagieren. Entweder es wird die übrige Zeit in den nachfolgenden Lehrmethoden ausgenutzt, der Unterricht früher beendet, oder es wird spontan eine weitere Lehrmethode hinzugefügt. Letztere Entscheidung kann sich auch wieder negativ auswirken, wenn dadurch die Zeit knapp wird. Gestaltet der Lehrende den Unterricht anhand des Ablaufplans transparent, dann kann es den Teilnehmern unvorbereitet vorkommen, wenn zunächst eine Lehrmethode hinzukommt und anschließend eine wegfällt.

4.4 Projektarbeit

Ein Projekt in der Lehre durchzuführen kann unter anderem als Projektunterricht geschehen. Die Lehrmethoden im Projektunterricht werden so ausgewählt, dass sie thematisch ein konkretes Projekt verfolgen. Ein verwertbares Produkt am Ende zu produzieren, steht dabei nicht im Vordergrund. Das Ziel ist das Verständnis über den Projektverlauf, die Zusammenarbeit und die Selbstständigkeit im Team (vgl. Frey, 2002).

Eine andere Möglichkeit ist, ein Projekt durchzuführen, dass tatsächlich zu einem verwertbaren Produkt führen soll. Im Rahmen des Projektes „myHAW“, das eine Pflichtveranstaltung im 5. Semester im Studiengang „Angewandte Informatik“ darstellt, hat der Verfasser dieser Arbeit gemeinsam mit 17 Studenten ein solches Projekt durchgeführt. Die Pflichtveranstaltung bestand aus 15 Terminen mit wöchentlich je 4,5 Stunden mit Anwesenheitspflicht. Das erste Ziel war eine mobile Applikation für das Android Betriebssystem, die mit Projektabschluss veröffentlicht werden sollte. Das zweite Ziel war die Durchführung des Projektes mit Scrum und Skalierung.

Während der Projektdurchführung nahmen die Ereignisse von Scrum einen Großteil der Zeit mit Anwesenheitspflicht ein, so dass innerhalb der 15 Termine zu wenig Zeit in die Umsetzung investiert wurde. Daraus resultierte, dass beide gesetzten Ziele so nicht bis zum Projektende umsetzbar waren, denn Rollen, Artefakte, Ereignisse und Regeln von Scrum sind unveränderlich (vgl. Schwaber u. Sutherland, 2013, S. 17). In der zweiten Hälfte des Semesters wurde durch die Scrum Master entschieden, einige Aspekte von Scrum dennoch zu reduzieren beziehungsweise auszulassen, damit mehr Zeit für die Umsetzung übrig bleibt.

Im Projektabschluss bewerteten die Teilnehmer die Durchführung des Projektes. Dabei wurde besonders der zeitliche Aufwand von Scrum in einer wöchentlichen Veranstaltung von 4,5

Stunden kritisiert, der Einsatz und das Erleben von Scrum in einem Projekt hingegen positiv bewertet. Im Nachhinein lässt sich vermuten, dass Kanban als wesentlich offenere agile Methode für Projekte mit wenigen Wochenstunden besser geeignet ist.

4.5 Umgang mit schwierigen Situationen

Während der Durchführung einer Unterrichtseinheit kann es zu Störungen, Konflikten und Lernwiderständen kommen. Störungen können beispielsweise Geräusche im Unterricht, Essen und Trinken oder Verspätungen sein. Wie schwerwiegend die Störung wahrgenommen wird, hängt unter anderem vom Lehrenden ab. Konflikte äußern sich durch höhere emotionale Brisanz unter den Beteiligten. Eine spezielle Form eines Konfliktes ist der *Lernwiderstand*, bei dem „Lernende [...] bewusst oder unbewusst nicht das [tun], worauf der Lehrende gerade hinaus will“ (Schumacher, 2011, S. 17).

Schumacher (vgl. 2011, S. 99-105) hat für den Umgang mit Widerständen bewährte Interventionen und die darin enthaltenen „besten Strategien“ zusammengefasst, auf die hier situationsbezogen eingegangen werden soll:

Wertschätzendes Verstehen und konkretisierendes lösungsorientiertes Nachfragen
: Der Lehrende kann beim Lernenden inhaltlich und auf Beziehungsebene nachfragen, wenn ein Problem oder ein Widerstand vorliegt. Zum Beispiel: „Was genau fehlt Ihnen zur weiteren Bearbeitung des Themas?“ oder „Wie müsste eine Gruppenarbeit aussehen, die gelingt?“ Dabei erhält der Lehrende konkretere Informationen zum vorliegenden Problem aus Sicht des Teilnehmers und kann gemeinsam mit ihm eine Lösung finden.

Selbstoffenbarung
: Die Selbstoffenbarung versucht Transparenz zu schaffen und Verständnis beim Teilnehmer zu wecken. Zum Beispiel: „Ihre Äußerung über ... löst in mir ... aus.“

Metakommunikation
: Durch Metakommunikation (Verlagerung der Kommunikation auf eine „höhere Ebene“ der Betrachtung) eine Distanz zum eigentlichen Problem aufbauen, um dadurch die Gesprächsrichtung zu ändern. Dabei eignen sich auch die Ziele des Unterrichts oder die ausgemachten Vereinbarungen (4.3.4.4). Zum Beispiel: „Sprechen wir nochmal über unsere Vereinbarungen. Wir hatten ausgemacht, dass...“

Vorwurf-Wunsch-Regel
: Vorwürfe sollten gegenüber Teilnehmern in einen Wunsch verwandelt werden. Zum

Beispiel: „Sie haben die Aufgabe nicht erfüllt“ zu „Ich wünsche mir, dass Sie die Aufgabe gemeinsam mit Ihrem Team bis ... erarbeiten“.

Umdeuten/Reframing

Gelangen Teilnehmer in eine Sackgasse, kann die Perspektive der Lernenden konstruktiv umgedeutet werden. Finden Teilnehmer in der Bearbeitung einer Aufgabe beispielsweise keinen Praxisbezug, dann lässt sich dies positiv bewerten und anschließend ein Bezug herstellen: „Ihr kritischer Blick ist eine gute Voraussetzung für die weitere Bearbeitung. Versetzen Sie sich in die Lage von Person...“

Vergleich/Bilder

Um Verständnis und Einsicht bei den Lernenden zu fördern, kann ein Prozess oder eine Situation mit einem Vergleich oder Bild erklärt werden. „Der Lernprozess/Gruppenarbeit ist vergleichbar mit einer Bergtour, bei der es auch Blasen und Muskelkater gibt.“ Für die Verdeutlichung von Störungen eignen sich Bilder sehr gut.

Humorvoll-provokatives Verhalten

Stimmt die Beziehungsebene mit den Teilnehmern, können Sachverhalte auf humorvoll-provokative Weise erklärt werden. Humorvolle Aussagen können auch zum Wecken von Verständnis eingesetzt werden. Zum Beispiel wenn eine Gruppe den Lehrenden zum schnelleren Vermitteln von Inhalten auffordert und versucht, die Gruppenarbeit zu vermeiden: „Sie sollen das selbst machen, damit Sie mich später, wenn Sie im Job sind, nicht ständig anrufen.“ Wichtig ist dabei, dass die Persönlichkeit der Teilnehmer nicht im Zentrum der Aussage steht, sondern maximal ein konkretes Verhalten.

Delegieren/Ignorieren

Manche Einwände und Störungen können ignoriert werden, beispielsweise Nebengespräche oder ein Zwischenruf. Andere können delegiert werden, zum Beispiel durch den Einsatz des Themenspeichers (4.3.4.2). Delegation kann auch beinhalten, dass bestimmte Fragen durch andere Teilnehmer beantwortet werden oder andere Teilnehmer zu Einwänden Stellung beziehen.

Nach Ansicht des Verfassers können Interventionen auch kombiniert werden. Fällt beispielsweise ein Teilnehmer durch lange Monologe auf, kann eine Moderationskarte mit der Aufschrift „Noch 30 Sekunden“ hochgehalten werden (humorvoll-provokant und Bild).

Häufig entscheidet der Anfang einer Veranstaltung, wie gut der Umgang mit Störungen, Konflikten und Prozessen der Gruppendynamik in der Zusammenarbeit gelingt. Es ist ratsam die Teilnehmer zu vernetzen und das Kennenlernen zu fördern, denn eine anonyme Gruppe

ist untereinander anfälliger für Störungen und Konflikte, als eine Gruppe, die bereits enger zusammengewachsen ist (vgl. Schumacher, 2011, S. 37).

Die aktivierende Lehre hat den Vorteil, dass passives Verhalten und Arbeitsverweigerungen auffallen, wenn Sachverhalte in Gruppenarbeit gelöst werden sollen. Der Lehrende kann mit den entsprechenden Teilnehmern sprechen und die beschriebenen Interventionen anwenden. Richtet sich die ganze Gruppe gegen den Lehrenden, sollte dieser Widerstand im Plenum besprochen werden, um neue Vereinbarungen zu treffen, wie auf beiden Seiten weitergearbeitet werden kann (vgl. Schumacher, 2011, S. 18).

5 Fazit

Dieses Kapitel fasst zunächst die Inhalte dieser Arbeit zusammen und bewertet das Ergebnis. Der Abschnitt Ausblick erläutert weitere Aspekte, die im Umgang mit dem Lehrkonzept offen blieben.

5.1 Zusammenfassung

Zunächst befasste sich diese Arbeit mit den Lehrinhalten (2), die Einfluss auf die Zielsetzungen eines Lehrkonzeptes haben. Die Lehrinhalte gehen auf das Manifest für Agile Softwareentwicklung ein, das auf Basis praxisorientierter Erfahrung eine Grundlage für agile Methoden festlegt. Zu den verbreitetsten agilen Methoden gehören Scrum (2.2) und Kanban (2.3), die anhand unterschiedlicher Literatur detaillierter beschrieben wurden. Ein Schwerpunkt bei der Beschreibung lag auf der Skalierung (2.2.6), da Produkte häufig von mehr als zwölf Personen entwickelt werden und auch Lehrmethoden wie *Lego City* oder *Kanban Pizza Game* diesen Aspekt beachten.

Anschließend wurden existierende Lehrmethoden der agilen Softwareentwicklung (3) beschrieben und nach Einsetzbarkeit in Lehrkonzepten bewertet.

Schließlich konnte ein Weg zur Erstellung von individuellen Lehrkonzepten (4) aufgezeigt werden, der die Formulierung von Kompetenzen (4.1) und Lernzielen (4.2) erläuterte. Diese Zielsetzungen flossen in ein Methodenbaukasten ein, der mit einer Verlaufsplanung (4.3) realisiert wurde. Anhand eines Beispiel-Verlaufsplans (4.3.4) wurde verdeutlicht, wie sich Lehrmethoden systematisch und zielorientiert anordnen lassen. Verwendet wurden dabei Lehrmethoden der agilen Softwareentwicklung, aber auch themenunabhängige Lehrmethoden (4.3.3) und verschiedene Sozialformen wie Frontalunterricht (4.3.2) und Gruppenarbeit. Abschließend wurden Aspekte in der Unterrichtsdurchführung angesprochen, darunter Zeitmanagement (4.3.5) und der Umgang mit schwierigen Situationen (4.5), die beim Einsatz von aktivierenden Lehrmethoden von Bedeutung sind.

5.2 Bewertung

Die agile Softwareentwicklung legt viel Wert auf Teamarbeit und Kommunikation von Angesicht zu Angesicht, daher ist sie für kompetenzorientierten und aktivierenden Unterricht geeigneter als beispielsweise die Automatentheorie, wo Praxisbezug nicht so naheliegend ist. Außerdem lassen sich Teilaspekte agiler Methoden stellvertretend für Lehrmethoden in einen aktivierenden Unterricht integrieren. In dieser Arbeit wird dies ersichtlich, denn beispielsweise das Daily Scrum (2.2.3.3) ist zu einer Lehrmethode (3.3) umgewandelt worden. Dies soll den Leser dazu ermutigen, eigene Lehrmethoden zu entwickeln, um Abläufe zu simulieren und sie in ein Lehrkonzept zu integrieren. Lehrmethoden, die Aspekte der agilen Softwareentwicklung simulieren, harmonieren auch mit Projektunterricht (4.4), weil ein Bezug zu einem beliebigen Projekt hergestellt werden kann.

Die *Verlaufsplanung* ist das Schlüsselergebnis dieser Arbeit, denn der strukturelle Aufbau des Verlaufsplans stellt das angestrebte Ziel, den Methodenbaukasten, dar. Die Verlaufsplanung hat ihre Wurzeln in der Pädagogik und damit liegt der Bezug zur Informatik gegebenenfalls fern. Ein Lehrender, der den Leitfaden für die Erstellung eines Lehrkonzeptes aus dieser Arbeit anwendet, wird die Relevanz der Lehrinhalte (2) feststellen. Die Ziele lassen sich besser bestimmen und Lehrmethoden entwerfen, wenn ein zusammengefasstes Nachschlagewerk zur Verfügung steht, das zudem für Detailfragen von Lernenden hilfreich sein kann.

Die Einträge des Verlaufsplans enthalten Stichpunkte und Schlagworte, die vom Planenden formuliert werden und nicht unbedingt von Dritten verstanden werden müssen. Der Beispiel-Verlaufsplan dieser Arbeit ist ausformulierter, als er es in der Praxis sein muss, damit die Leser dieser Arbeit die Einträge nachvollziehen können. Letztlich wird die Idee vertreten, die Einträge (Lehrmethoden) wie *Tickets* zu behandeln, die wie in einem *Product Backlog* in eine Reihenfolge gebracht werden können. Der Aufwand liegt primär in der eigentlichen Planungsarbeit und weniger in der Formulierung. Die Parallelen zu einem *Product Backlog* setzen sich fort, denn weicht die Durchführung von der Planung ab, kann dies Auswirkungen auf den Verlauf der folgenden Unterrichtseinheiten haben. Auch hier kann eine zunächst grobe Planung für spätere Unterrichtseinheiten hilfreich sein, die es erst mit näher rückenden Unterrichtsdatum zu detaillieren gilt.

5.3 Ausblick

Das Ergebnis dieser Arbeit, der Methodenbaukasten, ist in dieser Form noch nicht erprobt. Die Struktur basiert letztlich auf Unterrichtsskizzen aus Literatur und vier Semestern Erfahrung des Verfassers, die in Tutorien und dem „myHAW“-Projekt gesammelt wurden. Diese Erfahrungen formten die Struktur und die Literatur ließ weitere Ideen einfließen. Jedoch bleibt offen, wie erfahrenere Dozenten den Methodenbaukasten bewerten und diesen in der Praxis einsetzen.

Abbildungsverzeichnis

1 Einleitung **1**

2 Lehrinhalte **4**

2.1 Übersicht der Scrum-Ereignisse 17

2.2 Modell des *Product Owner Teams* in Hierarchie mit Bezeichnungen (nach Larman u. Vodde (2009) links und Cohn (2010) rechts) 31

2.3 Beispielhafte Teamzusammenstellung für Meta-Scrum Ereignisse 33

2.4 Über Signalflaggen benötigte Personen anfordern und Pausen kommunizieren 36

2.5 Beispiel für den Einsatz teamübergreifender Mitglieder bei wahrscheinlichen Abhängigkeiten 38

2.6 Blockade macht Probleme sichtbar (mittels Kanban-Board) 41

2.7 Mehrere Prozessabläufe am Beispiel eines Kanban-Boards 43

2.8 Queue für erledigte Aufgaben und ein Puffer vor dem Engpass „Test“ 48

3 Lehrmethoden für agile Softwareentwicklung **53**

3.1 Ein Stück *Pizza Hawaii* aus dem Kanban Pizza Game 61

4 Erstellung von individuellen Lehrkonzepten **76**

4.1 Vorbereiteter Ablauf zum Beispiel-Verlaufsplan auf Flipchart-Papier 92

5 Fazit **98**

Abkürzungsverzeichnis

DEEP	Detailed Appropriately (Angemessen detailliert), Estimated (Geschätzt), Emergent (Gewachsen), Prioritized(Priorisiert)
FDD	Feature Driven Development
HAW	Hochschule für Angewandte Wissenschaften Hamburg
INVEST	Independent (Unabhängig), Negotiable (Verhandelbar), Valuable (Nützlich), Estimatable (Schätzbar), Small (Klein), Testable (Testbar)
PO	Product Owner
TDD	Test Driven Development
WiP	Work in Progress
XP	Extreme Programming

Glossar

Agile Methode

Konkrete Zusammenstellung von Praktiken, die ein agiles Vorgehen in der Softwareentwicklung beschreiben und die agilen Werte berücksichtigen. Dazu zählen unter anderem Scrum oder Extreme Programming.

Akzeptanztest

Softwaretest, der überprüft, ob die funktionalen Erwartungen und Anforderungen im Gebrauch erfüllt werden.

Continuous Integration

Eine Anwendung wird zusammen mit allen Teilkomponenten mit jedem Einchecken in ein Versionskontrollsystem zusammengefügt und zur Ausführung gebracht. Die Ausführung wird in Form von automatischen Softwaretests realisiert. Üblicherweise wird die Anwendung außerdem analysiert und eine Menge von Metriken erzeugt. Die Entwickler erhalten zeitnahes Feedback über den Zustand einzelner Komponenten, beispielsweise über einen Monitor im Team-Bereich.

Extreme Programming

Agile Methode und Vorgehensmodell, das auf den drei Hauptbestandteilen beruht: Werte, Prinzipien und Techniken. Diese wurden von Kent Beck 2001 und 2004 (Neuauflage) in einem Zusammenspiel beschrieben, das heute als Extreme Programming bekannt ist. Als die bekannteste Praktik zählt *Pair Programming*.

Feature Driven Development

Agile Methode, die aus einer Ansammlung bewährter Praktiken besteht. Im Mittelpunkt steht die iterative zeitnahe Umsetzung von geforderten Features. Ein Projekt durchläuft typischerweise 5 definierte Prozesse, in denen zunächst alle Features gesammelt und

im stetigen Wechsel zwischen Entwurf und Konstruktion priorisiert nach Reihenfolge umgesetzt werden.

Feature Team

Eine Bezeichnung für ein Team in skalierten Ansätzen, das sich um die Umsetzung und Auslieferung vollständiger, getesteter Features kümmert. Das Team ist auf allen Schichten einer Softwarearchitektur tätig und liefert ihre Ergebnisse an Endanwender aus.

Gruppenpuzzle

Eine aktive Lehrmethode, in der sich Lernende gegenseitig unterrichten. Zunächst erarbeiten mehrere Lernende Grundlageninformationen zu einem Teilbereich. Anschließend diskutieren die Lernenden innerhalb von Kleingruppen die aufgenommenen Inhalte und erarbeiten selbstständig einen Plan, wie sie diese Inhalte anderen vermitteln wollen. Im nächsten Schritt bilden sich neue Gruppen mit jeweils einem Mitglied jeder kleinen Arbeitsgruppe und unterrichten sich gegenseitig über die Inhalte. Zum Abschluss wird der Lernerfolg in der Gesamtgruppe überprüft.

Inkrement

Siehe Produkt-Inkrement.

Kanban

Vorgehensmodell, in dem bestehende Prozesse in kleine Schritte geteilt und stetig verbessert werden. Die parallelen Aufgaben werden als Workflow visualisiert und limitiert, um Engpässe im Durchlauf sichtbar zu machen. Der Ursprung von Kanban liegt in der Produktion und wurde für die Softwareentwicklung angepasst.

Komponenten Team

Eine Bezeichnung für ein Team in skalierten Ansätzen, das sich um die Umsetzung und Auslieferung vollständiger, getesteter Komponenten kümmert. Das Team ist nur auf der Schicht mit der befindlichen Komponente einer Softwarearchitektur tätig und liefert ihre Ergebnisse an andere Teams aus.

Pair Programming

Zwei Personen programmieren gemeinsam an einem Rechner. Ein Entwickler übernimmt die Rolle des „Fahrers“, hat somit die Kontrolle über den Rechner und behält die nahe

Implementierung im Fokus. Der andere Entwickler nimmt die Rolle des „Navigators“ an, der die Gesamtheit der Lösung fokussiert, Korrekturen sofort anspricht und den Lösungsansatz prüft. Nach wenigen Minuten oder wenn einer der Entwickler Lösungen der jeweils andere Rolle gedanklich ausgearbeitet hat, wechseln beide die Rolle. Üblich ist auch das Wechseln der Paarungen innerhalb eines Team. Die Rollenbezeichnungen sind nicht festgelegt und können sich unterscheiden. Ziel ist die Steigerung der Softwarequalität.

Produkt-Inkrement

Produkt-Inkrement ist die dokumentierte, lauffähige und potentiell auslieferbare Software, die bereits umgesetzte Anforderungen enthält. Sie ist das Ergebnis eines Scrum-Sprints.

Pull-Prinzip

Arbeit wird aus dem vorherigen Prozessschritt abgeholt. Das Team entscheiden sich auf diese Weise selbst für Aufgaben, die sie erledigen werden. Dieses Vorgehen soll Überlastung von Teammitgliedern verhindern. Das Gegenteil entspricht dem Push-Prinzip. (vlg. Bleek u. Wolf, 2011, S. 170 f.)

Push-Prinzip

Arbeit wird in den nachfolgenden Prozessschritt geschoben. Auf diese Weise werden Aufgaben dem Team zugewiesen.

Refactoring

Refactoring beschreibt die Strukturverbesserung von Quellcode ohne Veränderung des Programmverhaltens. Das Ziel des Refactorings ist das Verringern von Aufwand bei einer späteren Fehleranalyse. Daher werden beim Refactoring die Lesbarkeit, Übersichtlichkeit, Verständlichkeit, Erweiterbarkeit und Testbarkeit verbessert und Redundanzen vermieden.

Scrum

Agile Methode und Framework, das ein iteratives und inkrementelles Vorgehen beschreibt. Scrum besteht aus einfachen Regeln, die ein selbst-organisierendes Team mit wenigen Rollen in flacher Hierarchie vorgeben. Außerdem beschreibt es Rollen, Ereignisse (Meetings) mit fester Zielsetzung und Artefakte (Dokumente) mit Prioritäten. Die Vorgaben bieten viele Freiheiten in der individuellen Gestaltung des Prozesses.

Sozialform

Die Sozialform regelt in der Didaktik die Beziehungsstruktur des Unterrichts. Sie wird untergliedert in die äußere Sozialform, die die räumliche Sitzordnung der Teilnehmer beschreibt und in die innere Sozialform, die die Kommunikations- und Interaktionsstruktur beschreibt. Nach Hilbert Meyers (vorherrschender) Definition gibt es vier Sozialformen: Einzelarbeit, Partnerarbeit, Gruppenarbeit und Frontalunterricht.

Stakeholder

Interne und externe Personengruppen (auch Einzelpersonen), die von unternehmerischen Tätigkeiten gegenwärtig oder zukünftig betroffen sind oder ein berechtigtes Interesse am Verlauf, Projekten, Prozessen oder Ergebnissen haben.

Test Driven Development

Test Driven Development ist eine Entwicklungspraktik, bei der die Testfälle konsequent vor der Implementierung einer zu testenden Komponente erstellt werden.

Quellenverzeichnis

[Anderson 2012] Anderson, David J. ; Roock, Arne (Hrsg.) ; Wolf, Henning (Hrsg.): *Kanban - Evolutionäres Change Management für IT-Organisationen.* 1. korrigierter Nachdruck. Heidelberg : Dpunkt.Verlag GmbH, 2012. – ISBN 978–0–9845214–0–1

[Andresen 2014] Andresen, Judith: *Retrospektiven in agilen Projekten - Ablauf, Regeln und Methodenbausteine.* 1. Auflage. München : Carl Hanser Verlag GmbH & Co. KG, 2014. – ISBN 978–3–446–43908–5

[Baldauf 2014] Baldauf, Corinna: *Retr-O-Mat.* http://www.plans-for-retrospectives.com/. Version: August 2014. – Zuletzt besucht am 29. August 2014

[Beck u. a. 2001] Beck, Kent ; Beedle, Mike ; Bennekum, Arie van ; Cockburn, Alistair ; Cunningham, Ward ; Fowler, Martin ; Grenning, James ; Highsmith, Jim ; Hunt, Andrew ; Jeffries, Ron ; Kern, Jon ; Marick, Brian ; Martin, Robert C. ; Mellor, Steve ; Schwaber, Ken ; Sutherland, Jeff ; Thomas, Dave: *Manifest für Agile Softwareentwicklung.* http://www.agilemanifesto.org/iso/de/. Version: 2001. – Zuletzt besucht am 29. August 2014

[Bildungsportal 2013] Qualitäts- und Unterstützungsagentur - Landesinstitut für Schule: *Methodensammlung - Anregungen und Beispiele für Moderatoren.* Bildungsportal Nordrein-Westfalen. http://www.standardsicherung.schulministerium.nrw.de/methodensammlung/. Version: November 2013. – Zuletzt besucht am 5. Oktober 2014

[Bleek u. Wolf 2011] Bleek, Wolf-Gideon ; Wolf, Henning: *Agile Softwareentwicklung - Werte, Konzepte und Methoden.* 2. Auflage. Köln : Dpunkt.Verlag GmbH, 2011. – ISBN 978–3–898–64701–4

[bpb 2014] Bundeszentrale für politischen Bildung: *Methodenkoffer*. http://www.bpb.de/lernen/unterrichten/methodik-didaktik/227/methodenkoffer. Version: Oktober 2014. – Zuletzt besucht am 5. Oktober 2014

[Brauer 2011] Brauer, Markus ; Fausel, Andrea (Hrsg.) ; Ahrens, Ruth (Hrsg.): *An der Hochschule lehren - Praktische Ratschläge, Tricks und Lehrmethoden*. Springer-Verlag Berlin Heidelberg, 2011. – ISBN 978–3–542–42005–4

[Cohn 2010] Cohn, Mike: *Agile Softwareentwicklung - Mit Scrum zum Erfolg!* 1. Auflage. München : Pearson Deutschland GmbH, 2010. – ISBN 978–3–827–32987–5

[Cursio u. Jahn 2013] Cursio, Michael ; Jahn, Dirk: *Leitfaden zur Formulierung kompetenzorientierter Lernziele auf Modulebene an der Philosophischen Fakultät und Fachbereich Theologie*. Februar 2013

[Derby u. Larsen 2006] Derby, Esther ; Larsen, Diana: *Agile Retrospectives - Making Good Teams Great*. 1. Auflage. The Pragmatic Bookshelf, 2006. – ISBN 0–9776166–4–9

[Eckstein 2012] Eckstein, Jutta: *Agile Softwareentwicklung in großen Projekten - Teams, Prozesse und Technologien - Strategien für den Wandel im Unternehmen*. 2. Auflage. Heidelberg : Dpunkt.Verlag GmbH, 2012. – ISBN 978–3–898–64790–8

[Frey 2002] Frey, Karl: *Die Projektmethode - Der Weg zum bildenden Tun*. 9. überarbeitete Auflage. Beltz Verlag, 2002. – ISBN 978–3–4072–5274–6

[Gaillot 2010] Gaillot, Emmanuel: *Coding Dojo*. http://codingdojo.org/. Version: 2010. – Zuletzt besucht am 20. Oktober 2014

[Gloger 2008] Gloger, Boris: *Ball Point Game*. http://borisgloger.com/2008/03/15/the-scrum-ball-point-game/. Version: März 2008. – Zuletzt besucht am 19. September 2014

[Gonschorek u. Schneider 2010] Gonschorek, Gernot ; Schneider, Susanne ; Petersen, Jörg (Hrsg.) ; Reinert, Gerd-Bodo (Hrsg.): *Einführung in die Schulpädagogik und die Unterrichtsplanung*. 7. überarbeitete und aktualisierte Auflage. AAP Lehrerfachverlage GmbH, Donauwörth : Auer Verlag, 2010. – ISBN 978–3–403–03216–8

[Grossman u. Bergin 2005] Grossman, Fred ; Bergin, Joe: *Pair Storytelling*. http://csis.pace.edu/~bergin/xp/pairstorytelling.html. Version: Oktober 2005. – Zuletzt besucht am 19. Oktober 2014

[Gudjons 2011] Gudjons, Herbert ; Klinkhardt, Julius (Hrsg.): *Frontalunterricht - neu entdeckt; Integration in offene Unterrichtsformen.* 3. Auflage. UTB, 2011. – ISBN 978–3–8252–3611–3

[Gugel 2011] Gugel, Günther: *2000 Methoden für Schule und Lehrerbildung.* 1. überarbeitete und neu ausgestattete Auflage. Weinheim und Basel : Beltz Verlag, 2011. – ISBN 978–3–407–25555–6

[Hahn 2011] Hahn, Gabriela: *Selbstkompetenz im Wandel - Eine kritische Analyse des Begriffes Selbstkompetenz und seiner Bedeutung(en) im Wandel der letzten 40 Jahre.* Mai 2011

[Hasemann u. Gasteiger 2014] Hasemann, Klaus ; Gasteiger, Hedwig: *Anfangsunterricht Mathematik.* 3. Auflage. Berlin Heidelberg : Springer Spektrum, 2014. – ISBN 978–3–642–39312–9

[Jagger 2010] Jagger, Jon: *cyber-dojo.org: the place to practice programming.* http://cyber-dojo.org/. Version: 2010. – Zuletzt besucht am 20. Oktober 2014

[Klieme u. a. 2007] Klieme, Eckhard ; Avenarius, Hermann ; Blum, Werner ; Döbrich, Peter ; Gruber, Hans ; Prenzel, Manfred ; Reiss, Kristina ; Riquarts, Kurt ; Rost, Jürgen ; Tenorth, Heinz-Elmar ; Vollmer, Helmut J.: *Zur Entwicklung nationaler Bildungsstandards - Expertise.* 2007

[Komus 2014] Komus, Ayelt: *Studie Status Quo Agile - Verbreitung und Nutzen agiler Methoden.* Version: Langfassung, Juli 2014. http://www.status-quo-agile.de/. – Zuletzt besucht am 11. August 2014

[Komus u. Kamlowski 2014] Komus, Ayelt ; Kamlowski, Waldemar: *Gemeinsamkeiten und Unterschiede von Lean Management und agilen Methoden.* Mai 2014

[Krivitsky 2011] Krivitsky, Alexey: *Scrum Simulation with LEGO.* http://www.lego4scrum.com/. Version: 2.0, 2011. – Zuletzt besucht am 21. Oktober 2014

[Kruse u. Ivancsich 2011] Kruse, Ralf ; Ivancsich, Franz N.: *Kanban Pizza Game.* http://www.agile42.com/en/training/kanban-pizza-game/. Version: September 2011. – Zuletzt besucht am 18. Oktober 2014

[Larman u. Vodde 2009] Larman, Craig ; Vodde, Bas: *Scaling Lean & Agile Development - Thinking and Organizational Tools for Large-Scale Scrum.* Boston : Pearson Education Inc., 2009. – ISBN 978–0–321–48096–5

[Lawrence 2009] Lawrence, Richard: *Patterns for Splitting User Stories.* http://www.agileforall.com/patterns-for-splitting-user-stories/. Version: Oktober 2009. – Zuletzt besucht am 31. August 2014

[Leopold u. Kaltenecker 2013] Leopold, Klaus ; Kaltenecker, Siegfried: *Kanban in der IT - Eine Kultur der kontinuierlichen Verbesserung schaffen.* 2. überarbeitete Auflage. München : Carl Hanser Verlag GmbH & Co. KG, 2013. – ISBN 978–3–446–43826–2

[Lersch 2010] Lersch, Rainer: *Wie unterrichtet man Kompetenzen? Didaktik und Praxis kompetenzfördernden Unterrichts.* Mai 2010

[Opelt u. a. 2012] Opelt, Andreas ; Gloger, Boris ; Pfarl, Wolfgang ; Mittermayr, Ralf: *Der agile Festpreis - Leitfaden für wirklich erfolgreiche IT-Projekt-Verträge.* München : Carl Hanser Verlag GmbH & Co. KG, 2012. – ISBN 978–3–446–43226–0

[Pichler 2008] Pichler, Roman: *Scrum - Agiles Projektmanagement erfolgreich einsetzen.* Dpunkt.Verlag GmbH, 2008. – ISBN 978–3–89864–478–5

[Reich 2014] Reich, Kersten: *Unterrichtsmethoden im konstruktiven und systemischen Methodenpool.* http://www.uni-koeln.de/hf/konstrukt/didaktik/. Version: März 2014. – Zuletzt besucht am 5. Oktober 2014

[Schumacher 2011] Schumacher, Eva-Maria: *Schwierige Situationen in der Lehre - Methoden der Kommunikation für die Lehrpraxis.* Verlag Barbara Budrich, 2011. – ISBN 978–3–8252–3507–9

[Schwaber u. Sutherland 2013] Schwaber, Ken ; Sutherland, Jeff: *Der Scrum Guide - Der gültige Leitfaden für Scrum: Die Spielregeln.* https://www.scrum.org/scrum-guide/. Version: 2013. – Zuletzt besucht am 29. August 2014

[Schüler 2014] Schüler, Jonas: *Unterrichtsmethoden Methodensammlung.* http://schuelerecke.net/schule/unterrichtsmethoden-%E2%80%93-methodensammlung/. Version: April 2014. – Zuletzt besucht am 5. Oktober 2014

[Terhart 1997] Terhart, Ewald: *Lehr-Lern-Methoden - Eine Einführung in Probleme der methodischen Organisation von Lehren und Lernen.* 2. Auflage. Weinheim und München : Juventa Verlag, 1997

[Weinert 2001] Weinert, Franz E.: *Leistungsmessungen in Schulen.* 2. Auflage. Weinheim und Basel : Beltz Verlag, 2001

[Wiechmann 2003] Wiechmann, Jürgen: *Schulpädagogik*. Band 2. Wilhelmstr. 13, D-73666 Baltmannsweiler : Schneider Verlag Hohengehren, 2003. – ISBN 3–89676–705–4

[Wujec 2010] Wujec, Tom: *The Marshmallow Challenge*. http://marshmallowchallenge.com/. Version: Januar 2010. – Zuletzt besucht am 17. September 2014

Index

4-Karten-Feedback, 93

Ablaufplan, 91
Agile Methode, 103
Agile Softwareentwicklung, 1, 4
    Prinzipien, 6
    Verbreitung, 10
    Werte, 4
Akzeptanzkriterien, 26
Analysieren, 78
Anwenden, 78
Area, 32
Area Backlog, 32

Ball Point Game, 53
Bewerten, 78
Bloomsche Taxonomie, 77
Business Value, 24, 75
Business Value Poker, 75

Chaos Cocktail Party, 82
Chief Product Owner, 30, 32, 64
Coding Dojo, 55
Continuous Integration, 103

Daily Scrum, 20, 33
Daily Standup, 58
Daily Standup Meeting, 50
Definition of Done, 29

Empirische Prozesssteuerung, 12
Entscheidungsspiel, 82
Entwickeln, 78
Entwicklungsteam, 15
Epic, 24, 27, 32
Epos, 24, 27
Ergebnisspeicher, 86, 91
Erinnern, 78
Expertenschätzung, 69, 73
Extreme Programming, 10, 103

Feature Driven Development, 10, 103
Feature Team, 104
Feedback, 90, 93
Frontalunterricht, 80

Gruppenpuzzle, 83, 104

Impediment Backlog, 34, 37
Input Queue, 43
Iteration, 16

Joint Retrospektive, 37
Joint Review, 36

Kanban, 10, 39, 59, 104
    Aufgaben, 44

Aufgabentypen limitieren, 48
Kernpraktiken, 40
Koordinierung, 50
Messungen, 52
Nachschubmeeting, 51
Prinzipien, 40
Puffer, 47
Queue, 47
Service Level Agreements, 48
Serviceklassen, 48
Beschleunigt, 49
Fester Liefertermin, 49
Kapazitäten, 50
Standardklasse, 49
Unbestimmbare Kosten, 50
Tickets, 44
Visualisierung, 42
Kanban Pizza Game, 59
Kanban-Board, 40, 43
Aufgabentypen, 45
Nebenläufigkeit, 45
Puffer, 47
Queue, 47
Spalten ohne Reihenfolge, 45
Ticket, 44
Kartenabfrage, 89
Kennenlernen, 92
Kompetenz, 76
Kompetenzmodell, 77
Komponenten Team, 104
Konflikt, 95
Konzeptwissen, 78
Koordinierende Teams, 38
Kopfstand, 88

Lego Scrum Simulation, 63
Lehrinhalte, 4
Lehrkonzept, 76
Lehrmethode, 53
Themenunabhängig, 81
Lernwiderstand, 95
Lernziele, 77

Magic Estimation, 69
Manifest für Agile Softwareentwicklung, 4
Marshmallow Challenge, 66
Meta-Scrum, 33
Metakognitives Wissen, 78
Methodenbaukasten, 84
Methodensammlung, 84

Oberste Direktive, 23
Operations Review, 52

Pair Programming, 71, 104
Pair Storytelling, 71, 81
Planning Poker, 73
Pre-Sprint Planning, 34
Product Backlog, 24, 31, 36
DEEP, 25
Sichten, 31
Product Backlog Refinement, 36
Product Line Owner, 31
Product Owner, 12, 30
Aufgaben, 13
Zeitaufwand, 13
Product Owner Hierarchie, 31
Product Owner Team, 31
Produkt-Inkrement, 28
Projektarbeit, 94
Projektunterricht, 94
Prozesswissen, 78

Queue Replenishment Meeting, 51

Refactoring, 105
Release-Planungsmeeting, 51

Sachwissen, 78
Schlüsselkompetenz, 77
Schwierige Situationen, 95
Scrum, 10, 11, 63, 105
    Artefakte, 23
    Ereignisse, 16, 32
    Produkt-Inkrement, 105
    Skalierung, 29
    Team, 12
    Vorplanung, 34
Scrum Master, 14
    Aufgaben, 14
    Zeitaufwand, 15
Scrum of Scrums, 33
Sozialform, 106
Sprint, 16
Sprint Backlog, 27
Sprint Planning, 18, 34, 35
    Phase 1, 18
    Phase 2, 19
Sprint Planning im Großraumverfahren, 35
Sprint Retrospektive, 21, 37
    Phasen, 22
Sprint Review, 21, 36
Störung, 91, 95
Stationenlernen, 81
Storypoint, 24, 25, 70
Synchronisierte Sprints, 35

Teamübergreifende Mitglieder, 38
Teamretrospektive, 52
Test Driven Development, 106
The Ball Point Game, 89
The Marshmallow Challenge, 87
Themenspeicher, 91
Triage, 51

Unterrichtsskizze, 85
User Story, 24, 26
    INVEST, 26
    Splitten, 24

Vegas-Regel, 23
Velocity, 19, 25
Vereinbarungen, 91
Verlaufsplanung, 79
    Beispiel, 84
    Rahmenbedingungen, 80
Verstehen, 78

Widerstand, 95
WIP-Limits, 46
    Anpassung, 47
    Engpässe, 47
    Probleme, 47

Zeitmanagement, 93
Zeitplanung, 93
Zufriedenheitsbedingungen, 26

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 23. Oktober 2014 Eike-Christian Ramcke