Hochschule für Angewandte Wissenschaften Hamburg
*Hamburg University of Applied Sciences*

# Bachelorarbeit

Horst Wendelmuth

## Agile Vorgehensmodelle im qualitativen Vergleich anhand des Agile Software Development Model (ASDM)

*Fakultät Technik und Informatik*
*Department Informatik*

*Faculty of Engineering and Computer Science*
*Department of Computer Science*

**Horst Wendelmuth**

# Agile Vorgehensmodelle im qualitativen Vergleich anhand des Agile Software Development Model (ASDM)

Bachelorarbeit eingereicht im Rahmen Bachelorprüfung

im Studiengang 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. Bettina Buth

Abgegeben am 14. April 2015

**Thema der Arbeit**

Agile Vorgehensmodelle im qualitativen Vergleich anhand des Agile Software Development Model (ASDM)

**Stichworte**

Agile Softwareentwicklung, Softwarequalität, Scrum, XP, Kanban, Softwareengineering, Vorgehensmodell,

**Kurzzusammenfassung**

Agile Vorgehensmodelle werden textuell und damit ungenau beschrieben. Wenn die Anforderung besteht, ein Vorgehensmodell zu wählen, welches ein Höchstmaß an Softwarequalität sicherstellt, müssen Vorgehensmodelle miteinander vergleichbar sein.

Das Agile Software Development Model (ASDM) liefert eine formale Beschreibung eines agilen Vorgehensmodells. In dieser Arbeit wird untersucht, ob mit dem ASDM Vorgehensmodelle hinsichtlich der Softwarequalität verglichen werden können.

**Title of the paper**

Agile Process-Models in a quality comparison concerning the Agile Software Development Model (ASDM).

**Keywords**

Agile Process-Model, Softwarequality, Scrum, XP, Kanban, Softwareengineering, Process-Model

**Abstract**

Agile process models are usually described textually, this method entails the risk of inaccuracy. If the requirement is made to choose a process model that ensures the highest level of software quality, process models have to be comparable.

There is a formal description of an agile process model in the Agile Software Development Model (ASDM). Using the ASDM, this paper investigates the comparability of process models in terms of the quality of software.

# Inhaltsverzeichnis

# 1 Einführung

## 1.1 Motivation

In der Vergangenheit wurden große Softwareprojekte meist nach einem Vorgehensmodell mit “wasserfallartigen” Zügen abgewickelt. Selbst in großen Konzernen weiß man jedoch mehr und mehr die Vorzüge von Agilen Methoden zu schätzen. Insbesondere die direkte Einbeziehung des Kunden in den Entwicklungsprozess wird zunehmend als qualitativen Vorteil wahrgenommen.

Trotz des zunehmenden Einsatzes von agilen Entwicklungsmethoden fehlt oft ein Bewertungskonzept, um das für die Anforderungen passende Vorgehensmodell auszuwählen. Vor allem in großen Konzernen mit vielen Beteiligten und verteilten Prozessen müssen viele Einzelaspekte berücksichtigt werden. Nicht jedes Vorgehensmodell ist gleichermaßen leicht einzusetzen. Zwar ist es möglich ein einzelnes Vorgehensmodell isoliert zu betrachten, dabei fehlt aber in der Regel die Möglichkeit unterschiedliche Vorgehensmodelle zu vergleichen.

Für diese Anforderung will diese Abschlussarbeit anhand des Agile Software Development Modells (ASDM) von André Janus drei agile Vorgehensmodelle miteinander vergleichen. Dabei liefert das ASDM eine formale Beschreibung, die genutzt wird, um die unterschiedlichen Vorgehensmodelle nach einheitlichen Kriterien zu klassifizieren. Bei dieser Betrachtung werden hinsichtlich der Qualität anerkannter Qualitätskriterien (nach ISTQB) berücksichtigt.

Ziel dieser Arbeit ist es herauszufinden, ob das ASDM genutzt werden kann, ein bestimmtes agiles Vorgehensmodell für die Erstellung von Software mit bestimmten Qualitätskriterien einzusetzen. Dabei müssen die beschriebenen Werte eines Vorgehensmodells in Prinzipien und Praktiken umgesetzt werden.

## 1.2 Abgrenzung

In der Praxis werden fast immer die einzelnen Aspekte aus verschiedenen agilen Vorgehensmodellen gemischt. Zum Beispiel ist die Mischform „Scrumban“ weit verbreitet.

Diese Arbeit untersucht ausschließlich die beschriebenen Formen von agilen Vorgehensmodellen. Sie trifft Aussagen über einen qualitativen Vergleich, wenn die beschriebenen agilen Vorgehensmodelle so eingesetzt werden wie sie beschrieben werden.

Die Erkenntnisse in dieser Arbeit können nicht beliebig gemischt werden. Die Aussagen haben nur dann Gültigkeit, wenn alle genannten Teilaspekte in der Einordnung der Vorgehensmodelle im Rahmen des ASDM betrachtet werden. Ein Rückschluss auf Einzelaspekte ist nicht zulässig.

# 2 Agile Vorgehensmodelle

Verglichen mit klassischen Vorgehensmodellen werden bei agilen Methoden besondere Werte in den Mittelpunkt gestellt. Diese sind davon bestimmt ein gemeinsames Arbeiten bei der Zielerreichung zu gewährleisten. Zwei Werte stehen dabei im Mittelpunkt: Kommunikation und Einfachheit. [vgl. Agile Softwareentwicklung von Wolf-Gideon]

**Kommunikation**

Bei klassischen Vorgehensmodellen werden die einzelnen Teilaufgaben im Rahmen der Entwicklung von unterschiedlichen Personen und Abteilungen, die organisatorisch und oft auch räumlich weit von anderen entfernt sind, erledigt. Die notwendigen Informationen werden über eine Vielzahl von Dokumenten (zum Beispiel Pflichtenheft) übermittelt.

Agile Vorgehensmodelle sehen hingegen einen großen Gewinn in der direkten Kommunikation aller Projektbeteiligten. Insbesondere bei kritischen Betrachtungen erwartet agile Softwareentwicklung von der direkten Kommunikation einen erheblichen Vorteil. Dabei ist entscheidend, dass die Kommunikation offen und mutig geführt ist. Wenn alle Beteiligten mit Respekt behandelt werden und ohne Vorbehalte ihre Auffassung vertreten können, kann Kommunikation den echten Mehrwert für das Projekt erfüllen.

**Einfachheit**

Softwareentwicklung muss bei der Umsetzung von Anforderungen fast immer sehr unterschiedliche Interessengruppen (Stakeholder) berücksichtigen. Zum einen möchte der Auftraggeber oder Benutzer die erwünschten Funktionen umgesetzt wissen, zum anderen müssen auch der Betrieb der Software und mögliche Weiterentwicklung berücksichtigt werden. Hinzu kommt, dass bei der Entwicklung unterschiedliche Berufsgruppen in Projekten zusammen arbeiten müssen. Diese Aspekte zusammengenommen, erklären den Grad der Komplexität in der Softwareentwicklung.

In agilen Vorgehensmodellen wird angestrebt, Software so einfach wie möglich zu entwickeln. Technologisch soll zu jeden Zeitpunkt nur das implementiert werden, was auch tatsächlich benötigt wird. Funktionen „auf Halde“ werden nicht realisiert. Auch werden nur Technologien eingesetzt, die leicht in die vorhandene Infrastruktur integriert werden können. Dabei werden

auch Ablauforganisationen berücksichtigt. Zum Beispiel werden oft sehr flache Hierarchien angestrebt und im Idealfall verwalten sich kleine Teams selbst.

## 2.1 Das Agile Manifest

Im Februar 2001 trafen sich siebzehn Vertreter unterschiedlicher agiler Vorgehensmodellen in einem Skigebiet in den Bergen von Utah. In einer sehr entspannten Umgebung hatten die Repräsentanten von eXtreme Programming, Scrum, Crystal und anderen Modellen die Möglichkeit, sich über Gemeinsamkeiten zu verständigen. Als Ergebnis dieses Treffen entstand das sogenannte Agile Manifest [vgl. http://agilemanifesto.org/] mit folgenden Grundsätzen:

> *We are uncovering better ways of developing software by doing it and helping others do it.*
>
> Through this work we have come to value:
>
> **Individuals and interactions** over processes and tools
>
> **Working software** over comprehensive documentation
>
> **Customer collaboration** over contract negotiation
>
> **Responding to change** over following a plan
>
> *That is, while there is value in the items on the right, we value the items on the left more.*

Deutsche Übersetzung unter: [http://agilemanifesto.org/iso/de/]

Die Unterzeichner wurden von der Idee getrieben, Grundsätze aus der praktischen Erfahrung heraus zu entwickeln. Im Mittelpunkt stand hierbei nicht eine akademische Sichtweise. Die entwickelten Grundsätze wurden klar formuliert und mit erreichbaren Zielen versehen. Jeder Grundsatz sollte aus der Praxis und für die Praxis sein.

**Individuals and interactions** over processes and tools

Speziell in großen Organisationen und Projekten werden Prozesse und Werkzeuge in den Mittelpunkt gestellt. Die Verantwortlichen erwarten durch striktes Einhalten der Prozesse eine höhere Effizienz aller Beteiligten. Das agile Manifest stellt eine Abkehr von dieser Sichtweise dar. Dies bedeutet nicht, dass Prozesse als überflüssig eingeschätzt werden. Auch in einem agilen Umfeld werden Prozesse gepflegt und gelebt. Allerdings stehen sich Prozesse und die Personen, die diese befolgen sollen, oft als gegensätzliche Kontrahenten gegenüber. Fühlen sich Personen durch Prozesse nicht unterstützt und empfinden diese überwiegend als störend, entscheidet sich das agile Manifest eindeutig für den Menschen. Dies soll der Einsicht Rechnung tragen, dass einzig die Menschen Software schlussendlich entwickeln. Dadurch müssen deren Interessen immer höher bewertet werden als organisatorische Belange. In der Praxis werden Prozesse regelmäßig von allen Beteiligten kritisch bewertet. Damit gehört die ständige Überprüfung von Ablauforganisationen zu der täglichen Arbeit. Dabei wird stets überprüft ob der einzelne Prozess einen echten Mehrwert erbringt. Ist dies nicht der Fall wird der Prozess verändert oder gestrichen. Mit diesem Vorgehen wird der Mensch in den Vordergrund gestellt und die Prozesse an dem Handeln der Menschen abgestimmt.

**Working software** over comprehensive documentation

Der Fortschritt von agilen Projekten wird einzig an dem Funktionsumfang der laufenden Software gemessen. Die Zeitspanne von dem Entwurf bis zur Implementierung wird dabei so kurz wie möglich gehalten. Dabei werden die Arbeitspakte so gewählt, dass jedes für sich zügig und autark implementiert werden kann. Das Ziel ist einen Status von „fast fertig“ eines einzelnen Arbeitspaketes möglichst nie aufkommen zu lassen. Anhand von einer lauffähigen Software können Projektmanager, Kunden und Tester das Softwareprodukt bewerten und gemeinsam mit Entwicklern weiter vorantreiben. Durch den Fokus auf fertige Features wird die Dokumentation bewusst in den Hintergrund verlagert. Fertige Software wird in dem Umfang dokumentiert, wie dies nötig ist. Dabei wird jeder Fall für sich betrachtet. Mit diesem Vorgehen wird sichergestellt, dass jede notwendige Dokumentation vorhanden ist aber unnötige Dokumentation nicht erstellt wird.

**Customer collaboration** over contract negotiation

Softwareprodukte werden stets für einen Kunden entwickelt. Das agile Manifest stellt die Zusammenarbeit mit dem Kunden ausdrücklich in den Mittelpunkt. In großen Projekten gibt es vertragliche Vereinbarungen. Damit diese Vereinbarungen nicht das tägliche Arbeiten bestimmen, akzeptiert Agile Softwareentwicklung, dass nicht alle Anforderungen im Vorfeld bekannt sind. Während der Entwicklung werden diese mit dem Kunden gemeinsam erarbeitet. Diese Vorgehensweise setzt voraus, dass der Kunde die Vorteile der agilen Entwicklung nutzen möchte und ein Grundverständnis für die daraus folgenden Prozessen beim Kunden vorhanden sind. Eine offene Zusammenarbeit mit dem Kunden ist die logische Konsequenz wobei Kunde und Auftragnehmer immer das Produkt im Fokus haben. Werden in der kooperativen Zusammenarbeit festgestellt, dass wesentliche Anforderungen in den Vertragsverhandlungen übersehen worden sind, werden diese gemeinsam nachgearbeitet. Durch diese Vorgehensweise wird sichergestellt, dass Softwarequalität und letztlich für Kundenzufriedenheit eine größere Gewichtung haben als zuvor geschlossene Vereinbarungen.

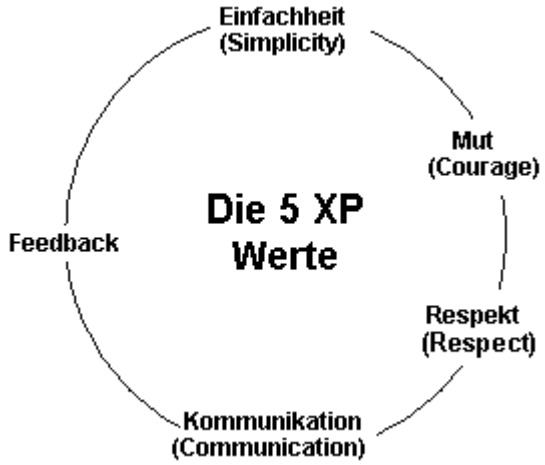
**Responding to change** over following a plan

Wenn der Mensch als maßgeblicher Faktor für funktionierende Software und die Kommunikation mit dem Kunden im Mittelpunkt steht, können diese Werte nur umgesetzt werden, wenn alle Projektbeteiligten offen für Veränderung sind. Dies gilt insbesondere dann, wenn akzeptiert wird, dass Anforderungen erst während des Projektverlaufes erkannt und umgesetzt werden. Durch diese offene Sichtweise ist es unabdingbar, Prozesse ständig anzupassen und alle gesetzten Pläne regelmäßig auf ihre Gültigkeit zu überprüfen. Agile Vorgehensmodelle stellen dabei eine Vielzahl von Mechanismen zur Planung und Bewertung von Softwareprojekten bereit. Tatsächlich ist agiles Vorgehen im hohen Maße von Planung bestimmt. Die dafür notwendigen Mechanismen werden ständig an die aktuellen Gegebenheiten angepasst.

## 2.2 *eXtreme Programming* – XP

Einer der Mitbegründer des Agilen Manifest Kent Beck, schlug im Jahre 2000 ein Vorgehensmodell mit dem Namen *eXtreme Programming (XP)* vor. Kent Beck arbeitete viele Jahre als Entwickler und Berater. In XP werden von Kent die erfolgreichsten Praktiken zu Methoden zusammengefasst. Auch in XP gibt es grundlegende Werte. Diese sind dem Agilen Manifest sehr ähnlich.

**Die Werte des eXtreme Programming**




[Abbildung 1: Werte eXtreme Programming]

Zusätzlich weist XP explizit darauf hin, dass diese Werte entsprechend den unternehmensweiten Werten angepasst werden müssen. Auch der Umgang mit Kollegen und Kunden soll hierbei überprüft werden. Wenn nötig müssen diese fünf Werte angepasst oder erweitert werden.

XP geht grundsätzlich davon aus, dass zu Beginn eines Projektes Kunde und Team noch nicht alle Anforderungen formulieren können und noch nicht über alle Informationen verfügen. Deshalb kommt dem Wert Kommunikation eine besondere Bedeutung zu. *eXtreme Programming* schlägt eine fortlaufende Iteration und den Einsatz mehrerer Einzelmethoden vor, um sich Schritt für Schritt dem optimalem Endergebnis zu nähern.

### 2.2.1 Rollen in XP

XP kennt nur einige festgeschriebene Rollen. Zwingend gibt es einzig die Rollen Entwickler und Kunde. Dabei wird die Rolle Entwickler keinem Spezialgebiet zugeordnet. Jede Person in einem Entwicklungsteam wird als Entwickler bezeichnet. Dadurch soll kein Druck gegenüber einer Person aufgebaut werden. Außerdem trägt dies dem Grundgedanken Rechnung, dass das ganze Entwicklungsteam grundsätzlich für alle Themenbereiche verantwortlich ist.

Der Kunde hat als Rolle einen wichtigen Stellenwert innerhalb von XP. Er ist dafür verantwortlich, dass Entwicklungsteam mit allen Informationen zu versorgen. Dabei muss der Kunde während des Projektverlaufs stets für Nachfragen zur Verfügung stehen. Damit ist der Kunde nicht nur Auftraggeber im Sinne eines Rechtsgeschäftes sondern auch ein wichtiger Teil des Entwicklungsprozesses selbst.

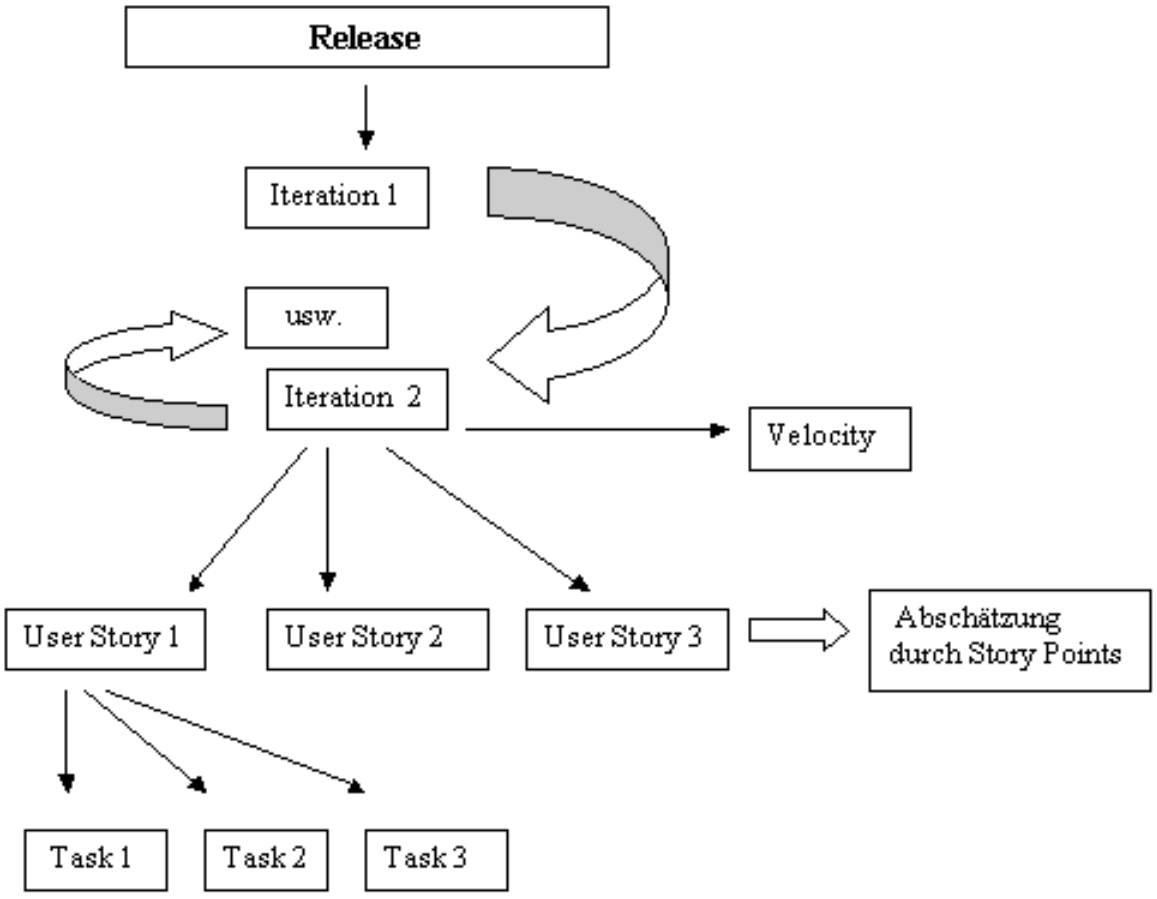
Zusätzlich zu den Rollen Kunde und Entwickler können innerhalb von XP noch folgende Rollen besetzt werden:

- Projektmanager
- Produktbesitzer
- Benutzer
- XP-Coach

Das Verständnis der Rolle Projektmanager entspricht auch bei XP dem üblichen Aufgabenumfang. Der Produktbesitzer ist meist eine Person innerhalb des Auftragnehmers. Diese Rolle soll als Schnittstelle zwischen dem Entwicklungsteam und dem Kunden dienen. Dabei sollen die Prioritäten des Kunden erkannt werden und das Entwicklungsteam zielführend unterstützt werden. Als Benutzer kann eine Person von dem Auftraggeber berufen werden. Diese Rolle soll schon frühzeitig die Sicht des Benutzers in der Entwicklung mit einfließen lassen. Insbesondere innerhalb der Anforderungsanalyse kann diese Rolle eine wichtige Position einnehmen. Treten Situationen auf, in denen das Entwicklungsteam selbst Unterstützung bei der Umsetzung des Vorgehensmodell *eXtreme Programming* benötigt, kann eine Rolle mit einem XP-Coach besetzt werden. Diese soll die Werte und Mechanismen an das Team vermitteln. Vor allem wenn Teammitglieder mit dem Vorgehensmodell nicht vertraut sind kann diese Roll von großem Vorteil sein.

### 2.2.2 Der XP-Prozess

Das Modell XP kennt zwei wesentliche Abschnitte: Das Release und die Iteration. Dabei stellt ein Release eine oder mehrere Funktionen bereit, die in sich geschlossen sind und eine neue Version der Software rechtfertigen. Um ein Release zu erreichen, werden meist mehrere Iterationen durchlaufen. Jede Iteration für sich dauert eine Woche. Innerhalb einer Iteration werden Arbeitspakte durch Stories beschrieben. Die Aufgaben, die sich mit einer Story verbinden, werden anhand von Tasks definiert. Eine Iteration wird so geplant, dass in dieser alle definierten Stories abgeschlossen werden. Um den Aufwand abzuschätzen, werden Stories mit Story-Points geschätzt.




[Abbildung 2: Phasen eines XP-Projektes]

Damit die Planungen von Releases, Iteration, Stories und Tasks gelingen kann werden folgende Schritte eingehalten:

- Der Kunde beschreibt grob dem Umfang des nächsten Release.
- Das Entwicklungsteam holt alle notwendigen Informationen beim Kunden ein, um eine grobe Schätzung über das Release vorzunehmen.
- Aufgrund der Schätzung des Teams beschreibt der Kunde detaillierte Anforderungen. Diese schreibt der Kunde in Stories nieder. Anschließend priorisiert der Kunde die Stories und legt so fest, welche in der kommenden Iteration bearbeitet werden sollen.
- Das Entwicklungsteam bearbeitet die Stories nach der Priorisierung des Kunden.
- Das Entwicklungsteam präsentiert die Ergebnisse und erhält Feedback vom Kunden.
- Die Planung der nächsten Iteration berücksichtigt die Probleme und Erkenntnisse, neue oder zu überarbeitende Anforderungen werden dabei berücksichtigt.

## 2.3 Scrum

Der Begriff Scrum kommt aus der Sportart Rugby und bedeutet soviel wie Gedränge. Dabei soll ausgedrückt werden, dass das Ergebnis in einem gemeinsamen Ringen von verschiedenen Akteuren, in dessen Zentrum das Ziel steht, erreicht wird. Scrum ist kein Vorgehensmodell explizit für die Softwareentwicklung. Tatsächlich stellt Scrum einen Managementrahmen dar, in dem vor allem zu erwartende Aufwände eingeschätzt werden können und mögliche Release-Zeitpunkte geplant werden können. Somit kann Scrum für unterschiedliche Aufgaben, die in einem Team bewältigt werden sollen, eingesetzt werden. In dieser Arbeit wird Scrum aus der Sicht der Softwareentwicklung betrachtet.

### 2.3.1 Rollen im Scrum

Scrum hat eine sehr klare Vorstellung von den notwendigen Rollen. Dabei können einzelne Rollen nicht weggelassen oder anders als vorgesehen interpretiert werden. Auch in diesem Vorgehensmodell wird in einem Team gearbeitet. Dieses besteht zwingend aus drei Rollen: Product-Owner, Scrum-Master und Team-Member.

**Product-Owner**

Der Product-Owner stellt den Vertreter des Kunden direkt im Team da. Diese Rolle beantwortet bei der täglichen Arbeit die fachlichen Fragen. Darüber hinaus priorisiert der Product-Owner die

Features und formuliert diese in Stories. Diese Stories stellen die Grundlage der Arbeit im Team dar. In Stories sind Funktionen und Anforderungen definiert. Nachdem eine Funktion implementiert wurde, nimmt der Product-Owner diese inhaltlich und formal ab. Somit stellt diese Rolle eine fachliche Trennung zwischen dem Entwicklungsteam und dem Kunden selbst dar.

**Scrum-Master**

Scrum geht davon aus, dass ein Team nur erfolgreich arbeiten kann, wenn es von äußeren Störungen nicht betroffen ist. Auch müssen für einen Erfolg zwingend gewisse grundsätzliche Prozesse eingehalten werden. Scrum geht davon aus, dass dies nicht vom Team selbst geleistet werden kann. Aus diesem Grund ist die Rolle Scrum-Master fest vorgeschrieben. Diese Rolle wacht darüber, dass alle Prozesse (ob verbindlich vorgeschrieben oder vom Team selbst festgelegt) eingehalten werden. Darüber hinaus, bearbeitet der Scrum-Master auftretende Störungen (Impediments), unabhängig, ob diese aus dem Team selbst oder außerhalb des Teams entstehen.

**Team-Member**

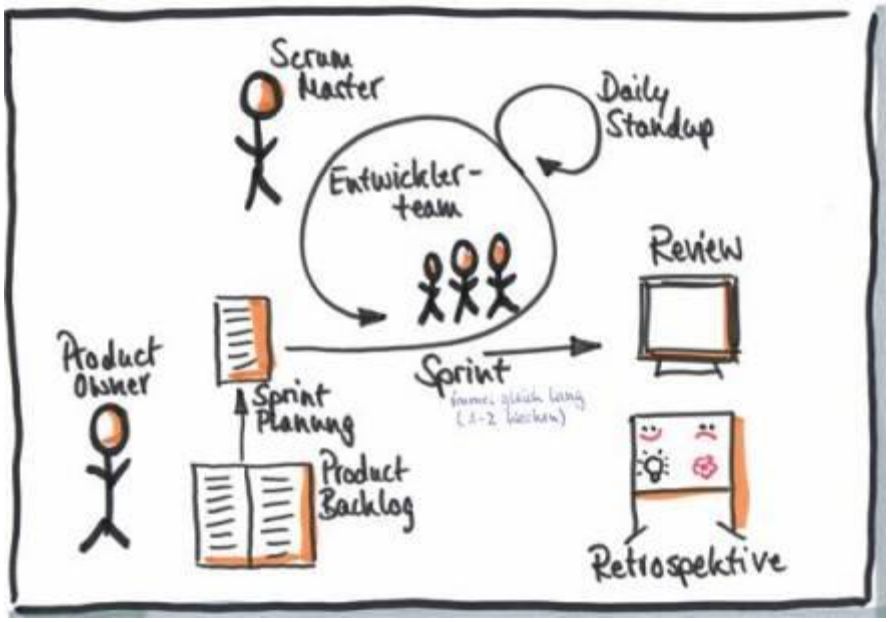
Für die Personen, die in einem Team die Software entwickeln kennt Scrum einzig die Rolle des Team-Member. Dabei spielt es keine Rolle ob ein Member ein besonderer Spezialist ist. Es werden also Entwickler, Designer, Tester und andere alle als Team-Member in der gleichen Rolle gesehen. Vielmehr geht Scrum davon aus, dass alle Team-Member interdisziplinär arbeiten und alle Aufgaben, die im Entwicklungsprozess anfallen, gleichermaßen bewältigen können. Das Team handelt dabei eigenverantwortlich und selbstbestimmt. Gegenüber dem Product-Owners handelt es selbstständig die Anforderungen aus und einigt sich auf den Umfang einzelner Stories (Commitment).

### 2.3.2 Der Scrum-Prozess

Im Zentrum des Scrum-Prozess steht der Sprint. Dieser ist ein festgelegter Zeitraum, in dem mehrere Stories bearbeitet werden. Innerhalb eines Sprints werden Stories als zentrales Artefakt bearbeitet. Dabei bildet jede Story für sich eine abgeschlossene Einheit. Jeder Sprint ist mit entsprechenden Sprintzielen verbunden. Diese können, aber müssen nicht den Kern einzelner Stories widerspiegeln. Die zu bearbeitenden Stories und die Sprintziele werden zu

Beginn des Sprints vom Team in Abstimmung mit dem Product-Owner festgelegt. Innerhalb des Sprints arbeitet das Team selbstständig und ohne Störungen von außen. Zum Ende des Sprints werden die Ergebnisse des Teams präsentiert und auf das Erreichen ihrer Sprintziele überprüft.

Der Sprint und die Stories werden mit folgenden Mechanismen abgearbeitet:




[Abbildung 3: Scrum Prozess]

**Product-Backlog**

Alle Anforderungen werden vom Product-Owner in einem Product-Backlog als Stories gesammelt. Naturgemäß werden diese Stories vom Product-Owner selbst unter Abstimmung mit dem Kunden erstellt. Aber auch das Team kann Anforderungen formulieren, die sich als Stories im Product-Backlog wiederfinden. Alle Stories im Product-Backlog werden regelmäßig vom gesamten Team gesichtet. Dabei werden Schätzungen über den zu erwartenden Aufwand anhand von Story-Points abgegeben und geklärt, ob alle notwendigen Informationen zur Bearbeitung einer Story vorhanden sind. Ist dies der Fall wird die betreffende Story mit dem Status „Definition of Ready“ vermerkt. Damit ist diese Story bereit, in einem der darauf folgenden Sprints bearbeitet zu werden.

### Sprint-Planing und Sprint-Backlog

Zu Beginn eines Sprints wird ein Sprit-Planing durchgeführt. Das Planing ist in zwei Phasen untergliedert. Zunächst stellt der Product-Owner alle Stories vor, die in dem kommenden Sprint bearbeitet werden sollen. Alle diese Stories haben den Status „Definition of Ready“. In diesem ersten Teil des Planings werden vorwiegend fachliche Fragen zu allen Themen diskutiert. Im Anschluss findet die zweite Phase des Planings statt. In diesem Teil besprechen die Team-Member die technischen Details der Stories. Dabei werden einzelne Aktivitäten für jede Story mit Tasks definiert und mit Stunden als Aufwand geschätzt. Anhand dieser Schätzung können die Team-Member ermitteln, welche Stories in dem aktuellen Sprint bearbeitet werden. Zum Ende dieser Phase des Planings stellen die Team-Member ihre Planung dem Product-Owner vor. Nicht alle vorgestellten Stories müssen zwingend in nächsten Sprint eingeplant werden. Die Team-Member selbst entscheiden eigenverantwortlich welchen Umfang vom Team im kommenden Sprint bewältigt werden kann. Zum Ende des Planings verständigen sich die Team-Member und der Product-Owner in einem Commitment über den finalen Umfang an Stories für den kommenden Sprint. Ist dies geschehen, werden alle Stories des nächsten Sprint in ein Sprint-Backlog überführt. Dieses Sprint-Backlog, in dem der aktuelle Stand aller Stories gepflegt wird, ist zentrales Dokumentationswerkzeug für den kommenden Sprint.

### Daily-Standup

Im Verlauf des Sprints wird täglich zur gleichen Uhrzeit ein Daily-Standup abgehalten. Diese Meetings werden aus Gründen des kurzen und prägnanten Austauschs von wesentlichen Inhalten in einem Standup durchgeführt. Hierbei werden von jedem Teilnehmer folgende drei Fragen beantwortet:

- Was habe ich seit dem letzten Daily-Standup für Aufgaben bewältigt?
- Welche Aufgaben möchte ich bis zum nächsten Standup bewältigen?
- Was könnte mich bei der Bewältigung meiner Aufgaben behindern?

### Review

Zum Ende eines Sprints stellt das Team seine Ergebnisse dem Product-Owner in einem Review vor. Dabei werden alle Stories explizit vorgestellt. Innerhalb des Reviews werden die Stories als erledigt vom Product-Owner abgenommen. Hier können innerhalb des Reviews neue Anforderungen formuliert, bzw. nicht fertig gestellte Stories diskutiert werden.

**Retrospektive**

Nach Abschluss eines Sprints wird von Team-Member, Product-Owner und Scrum-Master eine Retrospektive durchgeführt. Dabei wird der letzte Sprint kritisch betrachtet. Sowohl positive als auch negative Ereignisse werden diskutiert. Ziel der Retrospektive ist es Prozesse stetig zu verbessern. Diese Verbesserungen können im Team selbst durchgeführt werden. In diesem Fall wird ein Action-Item als Arbeitspaket definiert und einem Teilnehmer im Team zugewiesen. Dieser kümmert sich um die Erledigung dieses Arbeitspaketes. Ist es notwendig Veränderungen außerhalb des Teams herbei zu führen, wird der Scrum-Master mit der Durchführung beauftragt. Teil jeder Retrospektive ist die Überprüfung der Action Items aus zurückliegenden Retrospektiven und eine Bewertung des zu erwartenden Erfolges.

## 2.4 Kanban

Ursprünglich entstand Kanban innerhalb des Toyota-Produktionssystems. Das Wort Kanban bedeutet im Japanischen sinngemäß Signalkarte. Ziel war es, die Lagerbestände zu reduzieren und einen gleichmäßigen Fluss in der Fertigung zu gewährleisten. Grundidee ist dabei die Anzahl der aktuellen Aufgaben (Work in Progress WIP) zu minimieren und dadurch schnellere Durchlaufzeiten zu erzielen. Zusätzlich sollen dadurch mögliche Probleme schneller erkannt und behoben werden.

Üblicher Weise wird Kanban mit fundamentalen Prozessen und grundlegenden Praktiken beschrieben. In „Kanban from the Inside“ [3] wird im ersten Teil statt dessen ein System mit neun Werten vorgeschlagen. Diese Werte werden dann jeweils mit den Prozessen und Praktiken verbunden.

### 2.4.1 Die Kanban Werte

**Transparancy**

Damit Prozesse optimal bewertet werden können, müssen diese sichtbar sein. Deswegen nimmt der Wert Transparenz in Kanban eine zentrale Rolle ein. Diese Transparenz betrifft nicht nur Workitems. Auch Prozesse selbst sollen für jeden sofort einsichtig sein. Darüber hinaus muss klar sein, wer in einem Team welche Aufgaben übernimmt.

**Balance**

Oftmals können unterschiedliche Aspekte nicht konfliktfrei verfolgt werden. Nicht selten kann ein Aspekt nur erfolgreich erzielt werden, wenn ein anderer niedriger priorisiert wird. Es ist also notwendig ständig eine Balance zwischen verschiedenen Aspekten zu finden. Kanban versucht diese Herausforderung mit der Limitierung von Work-Items in einen bestimmten Zustand auf dem Board zu begegnen.

**Collaboration**

Kanban glaubt an eine echte Zusammenarbeit. Diese muss sowohl innerhalb eines Teams als auch über Teamgrenzen hinaus erfolgen. Dieser Wert ist von der Einsicht geprägt, dass ein wertvolles Produkt nur durch die Einbeziehung verschiedener Meinungen erstellt werden kann.

**Customer focus**

Bei der täglichen Arbeit sollten immer der Kunde und seine Bedürfnisse im Vordergrund stehen. Dieser Wert kann sich in unterschiedlichen Handlungen widerspiegeln. Zum Beispiel kann der Wert bedeuten, dass jede Story einen konkreten Nutzen für den Anwender beinhalten muss. Auf keinen Fall sollten Arbeiten durchgeführt werden, die einzig der Infrastruktur dienen jedoch nie zu einer Verbesserung des Produktes führen.

**Flow**

Jeder Team-Member kann den aktuellen Stand aller Work-Items auf dem Kanban-Board einsehen. Dabei lebt Kanban von der Idee, dass sich die Work-Items ständig durch die einzelnen Stationen auf dem Board bewegen. Wenn einzelne Items längere Zeit in einer Station verbleiben liegt offensichtlich ein Problem vor. Dieses sollte schnellstens gelöst werden. Optimal ist der Arbeitsprozess wenn Work-Items in einem „natürlichem Fluss“ über das Board wandern.

**Leadership**

In dem Wert Leadership sieht Kanban die Vereinigung von verschiedenen Grundwerten. Collaboration, Customer Focus, Flow, Transparancy and Balance. Diese Werte sollten im Leadership zusammengeführt werden und erfolgreich für das Team eingesetzt werden. Somit versteht Kanban Leadership selbst nicht als gesonderten Wert, sondern als eine Kombination der genannten Werte. Trotzdem wird der Wert Leadership separat genannt.

**Understanding**

Dieser Wert ist stark mit dem fundamentalem Kanban Prozess „Start with what you do now“ verbunden. Mit diesem Wert ist nicht gemeint, dass man zwingend immer ein 100%iges Verständnis aller Zusammenhänge der aktuellen Aufgabe haben muss. Unter „Understanding“ versteht Kanban eher das Verständnis der aktuellen Arbeit und die Auswirkungen auf Werte wie Customer Focus.

**Agreement**

Ähnlich vielen anderen agilen Vorgehensmodellen glaubt auch Kanban daran, dass eine Aufgabe im Team oft besser gelöst werden kann als wenn einzelne Spezialisten jeweils Teilaufgaben lösen. Damit dies funktionieren kann, ist es nötig, dass sich die Team-Member auf gemeinsame Punkte einigen. Kanban geht hierbei von einem Agreement bezogen auf eine evolutionäre Strategie aus. Alle Member sollten sich also darauf verständigen gemeinsam eine Kultur des ständigen Wandels zu verfolgen.

**Respect**

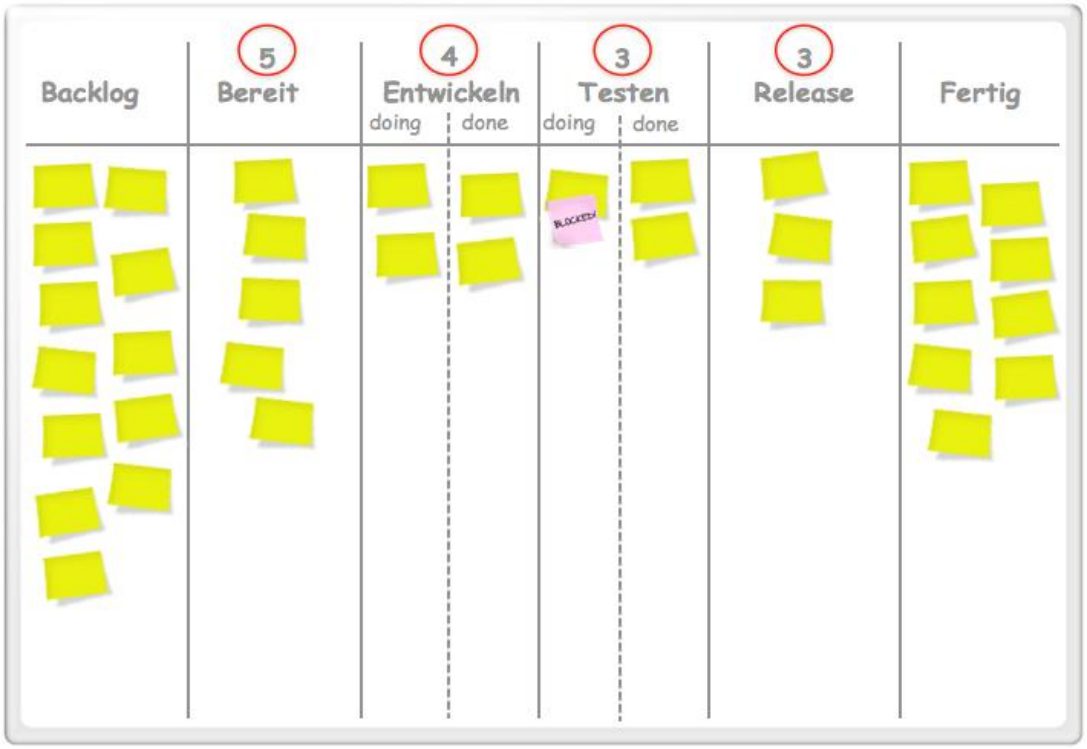
Wenn Arbeiten im Team im Mittelpunkt steht, muss natürlich Respekt ein zentraler Wert sein. Allerdings geht Kanban hier noch einen Schritt weiter und interpretiert noch mehr in den Wert Respekt. Wenn ein Team-Member eine Idee verfolgt und nach Personen für die Umsetzung sucht, soll nicht zentral nach Rollen gesucht werden („Don`t start with Roles“ [3]). Kanban denkt bei der Umsetzung von Ideen maximal pragmatisch („Be Like Water“ [3]). Respekt ist also in Kanban ein Wert, der auch den Gedanken trägt, dass jeder Member unabhängig seiner Rolle für die Verwirklichung einer Idee Beiträge leisten kann. Eine entsprechende Position ist nicht nötig um Entscheidungen von Tragweite mit zu beeinflussen.

### 2.4.2 Praktiken in Kanban

**Visualisierung des Arbeitsflusses**

Die verschiedenen Aktivitäten zu einer Anforderung (Tasks, Features oder User Stories) werden anhand eines Boards visualisiert. In der Regel verwendet man dazu ein großes Whiteboard, auf dem die unterschiedlichen Stationen als Spalten dargestellt sind. Die Tickets wandern von links nach rechts durch das Kanban Board. Für jeden ist der aktuelle Status der einzelnen Work-Items ablesbar. Folgende Informationen sind auf einen Blick vorhanden:

- Welche Arbeit ist blockiert?
- Wer arbeitet an welchem Work-Item?
- Welcher Arbeitsaufwand ist noch zu erwarten?




[Abbildung 4: Kanban Board]

**Limitierung des Work-in-Progress (WIP)**

Damit ein gleichmäßiger Fluss der einzelnen Items gewährleistet wird, werden die Anzahl der Items, die sich in einem Abschnitt des Boards befinden dürfen, begrenzt. Wenn zum Beispiel derzeit zwei Tickets in dem Status „Programmieren“ sind, und das Limit für diesen Abschnitt zwei beträgt, darf kein drittes Ticket in diesen Abschnitt eintreten. Durch dieses Vorgehen werden Items nicht einfach in einen nächsten Abschnitt weiter geschoben. Vielmehr holt sich jeder aus der Station zuvor ein Ticket.

**Steuerung und Messung des Arbeitsflusses**

Eines der zentralen Ziele in Kanban ist es Durchlaufzeiten zu verringern. Deswegen wird der Durchfluss von Work-Items ständig beobachtet. Diese Praktik kann auch etwas ausführlicher

beschrieben werden. „Manage flow, seeking smothness, timeliness, and good economic outcomes, anticipating customer needs“ [3]. Durch diese Formulierung wird die Verbindung zu den Werten customer focus und transparancy aus gedrückt. Ganz besonders viel Gewicht bekommt die Kombination der Praktiken „Limit Work-in-Progress“ und „Manage flow“. Durch die Visualisierung am Kanban-Board kann ständig die Einhaltung dieser Praktiken überprüft werden.

**Prozesse explizit machen**

Kanban glaubt daran, dass Regeln nur dann existieren sollten, wenn diese einen Mehrwert erbringen. Auf den ersten Blick scheint dies eine Selbstverständlichkeit. Oft werden Regeln aus guten Gründen eingeführt. Kaum jemand legt Regeln fest, wenn er sich davon keinen Vorteil verspricht. Allerdings werden Regeln nach der Einführung oft nicht hinterfragt. Nicht selten werden Regeln für Fälle angewendet, für die diese gar nicht gedacht waren. Deswegen fordert Kanban in dieser Praktik, dass Regeln explizit für einen Fall oder eine Gruppe von Fällen beschränkt werden. Diese Praktik wird oft missverstanden. Kanban propagiert nicht, dass weniger Regeln erstrebenswert sind. Wenn man Regeln explizit festlegt entstehen sogar zwangsläufig mehr Regeln. Wichtig ist jedoch, dass Regeln kritisch hinterfragt werden. Wenn eine Regel einen konkreten Mehrwert erbringt, wird diese befolgt.

**Verbesserungen durch bewährte Modelle**

Damit Erfolge bewertet werden können, sollen Prozesse vorwiegend iterative durchlaufen werden. Nur wenn am Ende eines Abschnittes eine Rückmeldung über ein Ergebnis eingeholt wird, kann man tatsächlich Prozesse stetig verbessern. Unser Kanban-Board könnte noch mit einer Spalte Review ergänzt werden. In diesem Schritt würde man das Ergebnis eines Work-Items vorstellen und gemeinsam diskutieren. Oft ergibt ein Review weitere Aufgaben. Dadurch entsteht eine iterative Arbeitsweise.

# 3 Softwarequalität und ASDM

## 3.1 Softwarequalität

Der Begriff der Softwarequalität kann grundsätzlich nur im Zusammenhang zu ihren Anforderungen verstanden werden (vergleiche [4]). Entspricht eine Software den zuvor festgelegten Anforderungen, kann die Qualität gemessen werden. In dieser Arbeit soll überprüft werden, ob bestimmte Vorgehensmodelle eher dazu dienen, Software mit hoher Qualität zu erzeugen als andere. Daher kann hier Softwarequalität als abstraktes Artefakt betrachtet werden. Eine konkrete Überprüfung von Qualität ist an dieser Stelle nicht möglich.

Die ISO-Norm 9126 nennt die Qualitätsmerkmale Funktionalität, Zuverlässigkeit, Benutzbarkeit, Effizienz und Änderbarkeit. Die Summe dieser Kriterien kann als Softwarequalität verstanden werden und zur Überprüfung der genannten Vorgehensmodelle herangezogen werden.

### 3.1.1 Funktionalität

Funktionalität beschreibt die Fähigkeit einer Software. Durch Ein-/Ausgabeverhalten lässt sich diese Fähigkeit nachweisen. Diese Fähigkeiten werden in funktionalen Anforderungen definiert. Auch wenn Anforderungen in agilen Prozessen nicht vollständig vor der Implementierung definiert werden, sind funktionale Anforderungen in der Regel frühzeitig bekannt. Die Implementierung von einzelnen Funktionen stellt somit kein Qualitätsrisiko dar.

Durch die Iterative Vorgehensweise von agilen Prozessen werden die einzelnen Funktionen in vielen kleinen Teilschritten implementiert. Aus diesem Grund besteht die Gefahr, bereits vorhandene Funktionen durch Seiteneffekte nachträglich mit Defekten zu versehen. Ein Vorgehensmodell, das dieses Risiko besonders minimiert, würde das Qualitätsmerkmal Funktionalität besonders gewährleisten.

### 3.1.2 Zuverlässigkeit

Wenn eine Software über einen festgelegten Zeitraum ein bestimmtes Leistungsniveau einhalten kann, ist dies ein Maß für Zuverlässigkeit. Ein weiteres Qualitätsmerkmal in diesem

Zusammenhang ist eine bestimmte Fehlertoleranz einer Software. Diese ist gegeben, wenn das Leistungsniveau trotz eines Defekt oder falschem Schnittstellenaufruf aufrecht gehalten werden kann.

Damit unterschiedliche Anwendungsszenarien untersucht werden können, kann ein System mit Konfigurationsdateien versehen werden. Ein Entwicklungsprozess der Konfigurationsmanagement im Besonderen berücksichtigt, würde Zuverlässigkeit als Qualitätskriterium unterstützen.

### 3.1.3 Benutzbarkeit

Unter Benutzbarkeit versteht man den Grad der Verständlichkeit von Benutzerfunktionen. Dieses Qualitätskriterium ist einzig bei interaktiven Systemen von Belangen. Da IT-Systeme zunehmend vernetzt arbeiten, wird in dieser Betrachtung davon ausgegangen, dass alle Systeme auch in der Benutzbarkeit gemessen werden müssen. In dieser Arbeit werden klassisch isolierte klassische Embedded-Systems (z.B. Waschmaschinen etc.) nicht berücksichtigt. Diese Systeme haben keine übliche softwaregestützte Benutzerschnittstelle und können daher nicht auf Benutzbarkeit überprüft werden.

Alle agilen Vorgehensmodelle zeichnen aus, dass der Benutzer besonders im Zentrum steht. Aus diesem Grunde sollten sich in untersuchten Modellen entsprechende Prozesse oder andere Mechanismen finden lassen, die Benutzbarkeit im Besonderen gewährleistet.

### 3.1.4 Effizienz

Das Qualitätskriterium Effizienz beschreibt den Einsatz von Betriebsmitteln. Dabei werden oft Kommunikationswege in Zeit- oder Datenmenge pro Zeit gemessen. Auch die Ausnutzung von Speicher wird hier berücksichtigt. Dieses Kriterium ist im hohen Maße von den zur Verfügung stehenden Betriebsmitteln abhängig. Zusätzlich müssen in verteilten Systemen das Verhalten von Nachbarsystemen betrachtet werden.

Diese Faktoren stehen in keinem Zusammenhang mit einem gewählten Vorgehensmodell. Aus diesem Grund wird das Kriterium Effizienz nicht in die Betrachtung einbezogen, da es auf die Auswertung keine Auswirkung hätte.

### 3.1.5 Änderbarkeit

Sehr oft werden Softwarelösungen über einen längeren Zeitraum eingesetzt. Dabei ändern sich Nachbarsysteme und auch Anforderungen an das System. Diese betreffen jedoch nicht alle Aufgabenfelder. Wie schon unter dem Punkt Benutzbarkeit beschrieben, werden hier bewusst Embedded-Systems aus der Betrachtung ausgeklammert. Für diese Systeme spielt das Kriterium Änderbarkeit faktisch keine Rolle.

In agilen Vorgehensmodellen wird kaum überprüft, ob Lösungen über einen längeren Zeitraum besonders geeignet sind, allerdings führt die iterative Vorgehensweise dazu, dass Software generell in einem Umfeld entwickelt wird, welches sich ständig ändert. Diese Tatsache kann als Qualitätskriterium verstanden werden. Die genannten Vorgehensmodelle werden unter diesem Gesichtspunkt betrachtet werden.

## 3.2 Agile Software Development Model (ASDM)

Die genannten Vorgehensmodelle werden im Allgemeinen textuell und damit unscharf beschrieben. Beim direkten Vergleich ist oft nicht klar, ob mit unterschiedlichen Begriffen das Gleiche gemeint ist oder ob es Abweichungen gibt. Im XP ist eine Rolle Produktbesitzer vorgesehen. Diese ist offensichtlich mit der Rolle Produkt-Owner im Scrum identisch. Ob der Prozess Iteration aus dem XP identisch mit dem Sprint aus Scrum ist, kann zunächst nicht eindeutig definiert werden.

Um einen direkten Vergleich der Vorgehensmodelle durchzuführen ist eine formale Beschreibung notwendig. Diese Beschreibung hat André Janus in seiner Dissertation vorgenommen [7] und als Agile Software Development (ASDM) zusammengefasst. Das ASDM unterscheidet dabei eine Menge von Basiskonzepten und eine Menge von Beziehungen der Basiskonzepte unter einander.

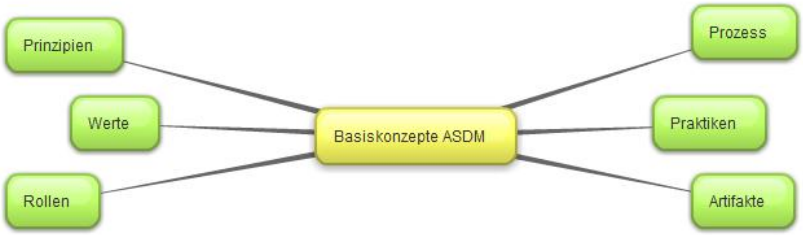
$$ASDM = (M_{ASDM}, R_{ASDM})$$

[Abbildung 5: ASDM]

Das Agile Software Development Model (ASDM) [6] gruppiert die Begriffe und Konzepte, um eine einheitliche Charakterisierung der agilen Vorgehensmodelle zu erreichen.

### 3.2.1 Basiskonzepte im ASDM

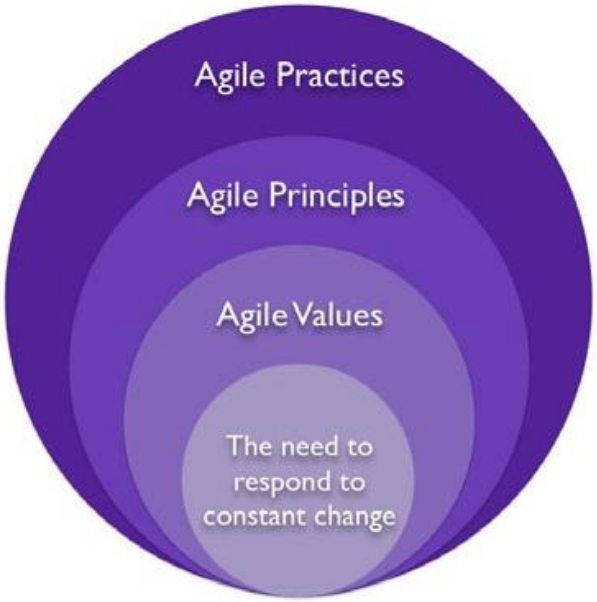
Das ASDM untergliedert die Vorgehensmodelle in unterschiedliche Basiskonzepte. Ausgehend vom Agilen Manifest werden Werte als grundlegendes Basiskonzept verstanden. In der Summe umfasst das ASDM die Basiskonzepte Werte, Prinzipien, Prozesse, Praktiken, Artefakte und Rollen. Diese Basiskonzepte sind gleichberechtigt.




[Abbildung 6: Basis-Konzepte des ASDM]

**Werte, Prinzipien und Praktiken**

In der Beschreibung von agilen Vorgehensmodellen werden die Begriffe Werte, Prinzipien und Praktiken eingesetzt. Hier zeigt sich die Notwendigkeit einer klaren und abgesprochenen Terminologie, damit die einzelnen Bereiche voneinander abgegrenzt sind.

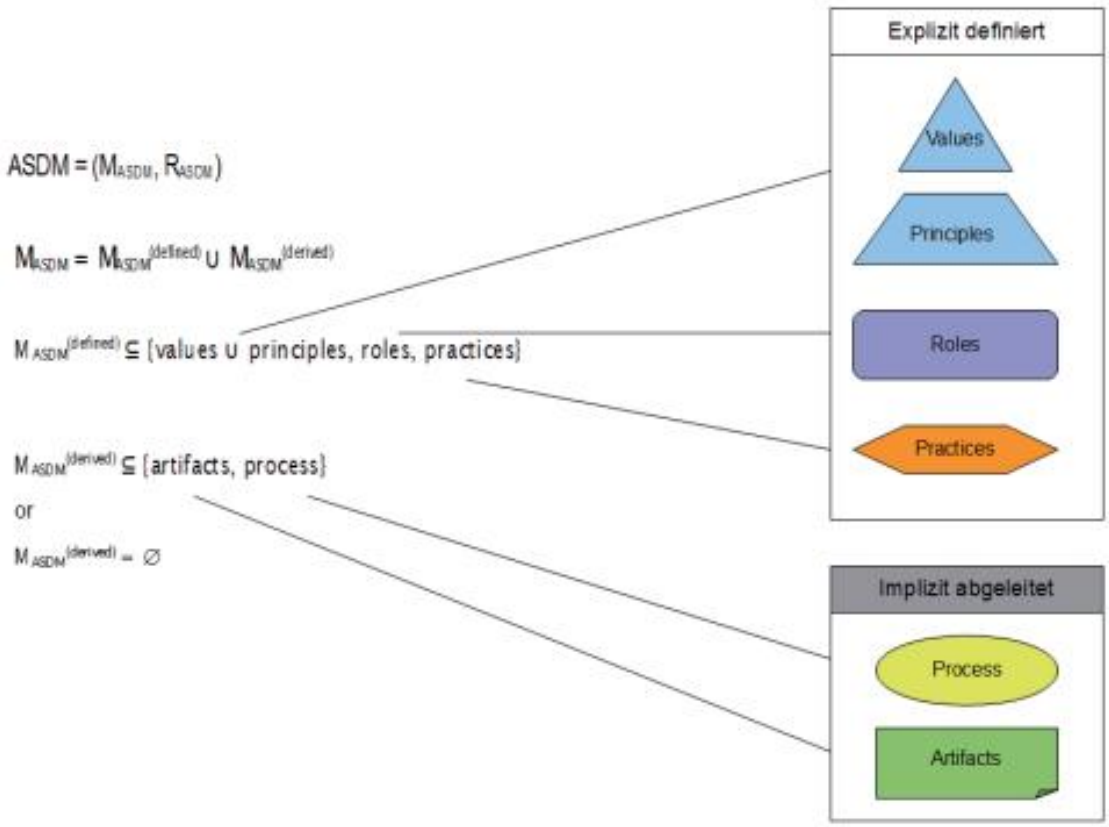



[Abbildung 7: Werte, Prinzipien und Praktiken]

Wie schon im Agilen Manifest beschrieben, stellen die Werte den Kern eines agilen Vorgehensmodells dar. Auf diesen Werten können Prinzipien aufbauen, die wiederum die Grundlage für Praktiken darstellen. Somit sind Praktiken die sichtbare und operative Umsetzung von Werten.

**Explizite und Implizite Basiskonzepte**

Nicht alle Basiskonzepte können eindeutig aus der textuellen Beschreibung zugeordnet werden. Deswegen unterscheidet das ASDM explizit definierte und implizite Basiskonzepte. In Abbildung 8 wird gezeigt, wie Werte, Prinzipien, Rollen und Praktiken implizit in einem Vorgehensmodell beschrieben werden. Prozesse und Artefakte wurden in dem Vorgehensmodell nicht beschrieben. Allerdings gibt es Beschreibungen, die diese Basismodelle herleiten können. Aus diesem Grund sind hier die Basiskonzepte Prozesse und Artefakte implizit beschrieben.




[Abbildung 8: explizite und implizite Basis-Konzepte des ASDM]

Nicht alle Basiskonzepte dürfen im ASDM durch eine implizierte Beschreibung hergeleitet werden. Die Basiskonzepte Rollen und Prinzipien müssen in einer expliziten Beschreibung vorliegen, damit das Vorgehensmodell im Rahmen des ASDM beschrieben werden kann.

### 3.2.2 Beziehungen im ASDM

Zusätzlich zu den Basiskonzepten beschreibt das ASDM Beziehungen. Dabei werden Ausprägungen und Beziehungen zwischen identifizierten Basiskomponenten beschrieben. Diese Beziehungen erfüllen zwei Funktionen:

Nicht explizit beschriebene Basiskonzepte werden durch Beziehungen implizit beschrieben. Das Beispiel in Abbildung 9 zeigt wie durch die Beziehung zwischen einer Praktik und einer Rolle ein Prozess implizit beschrieben wird.

$$r^{(process)} \in R_{ASDM} : practice \times role \rightarrow process$$

[Abbildung 9: implizite Beschreibung des Basiskonzeptes Prozess]

Abstrakte Werte werden aufgezeigt, indem zum Beispiel konkrete Praktiken und deren Wechselwirkung beschrieben werden. Der Nachweis aus Abbildung 10 zeigt den abstrakten Wert Prinzip.

$$r^{(principle)} \in R_{ASDM} : practice_1 \times practice_2 \rightarrow principle$$

[Abbildung 10: ein abstraktes Prinzip wird durch die Beziehung zweier Praktiken beschrieben]

### 3.2.3 Verwendung des ASDM in dieser Arbeit

In dieser Arbeit sollen die genannten Vorgehensmodelle mit dem ASDM untersucht werden. Dabei wird zunächst jedes Vorgehensmodell auf die genannten Basiskonzepte untersucht. Anschließend wird anhand einer Beziehung überprüft, ob die Basiskonzepte auch in der praktischen Arbeit tatsächlich angewendet werden. Wichtig ist dabei, dass sich die Werte in einer Beziehung zwischen Praktik und Prinzip nachweisen lassen.

**Wert -> Praktik x Prinzip**

Wie in Kapitel 3.2.1 beschrieben, stellen Werte den Kern eines agilen Vorgehensmodells dar. Auf diesen Werten bauen Prinzipien und Praktiken auf, wobei die Praktiken schlussendlich die sichtbare Handlung der Werte in einem agilen Vorgehensmodell sind.

Auf dieser Überlegung basierend, können die genannten Basiskonzepte (Wert, Praktik und Prinzip) zu einer Beziehung verbunden werden. Anhand dieser Beziehungen lassen sich die Werte, und deren Umsetzung in Praktiken überprüfen.

Nur wenn sich für alle Werte schlüssige Beziehungen von Praktiken und Prinzipien finden lassen, ist sicher gestellt, das die Werte in der täglichen Arbeit auch tatsächlich umgesetzt werden.

**Beispiel:**

Im dem Vorgehensmodell Scrum gibt es den Wert Mut. Für sich alleine genommen ist das zunächst ein abstrakter Begriff und es bleibt unklar, was dieser Wert für das Vorgehensmodell und die beteiligten Personen bedeutet.

Als Prinzip nennt Scrum unter anderen auch „Änderungen willkommen heißen“. Ein Prinzip ist schon konkreter als ein Wert. Aber auch bei einem Prinzip bleibt letztlich unklar, was dies für die tägliche Arbeit bedeutet.

Wenn man zu dem Mut als Wert und Änderungen willkommen heißen als Prinzip die Praktik Retrospektive hinzunimmt, kann eine Beziehung zwischen den drei Basiskonzepten gebildet werden.

***Mut** -> Retrospektive x Änderungen willkommen heißen*

Durch diese Beziehung werden alle drei genannten Basiskonzepte in einen Kontext gesetzt. Dies erlaubt das „Wesen“ der einzelnen Konzepte einzuordnen und später Vergleiche einzelner Beziehungen durchzuführen.

Im letzten Schritt wird mit Beziehungen untersucht, ob die in Kapitel 3.1 genannten Qualitätskriterien nachgewiesen werden können. Dabei wird für jedes Qualitätskriterium eine Beziehung aus Rolle, Praktik und Artefakt gebildet.

**Qualitätskriterium -> Rolle x Praktik x Artefakt**

Die Praktik wurde in die Beziehung mit aufgenommen, da diese die direkte Verbindung zu den Werten in einem Vorgehensmodell darstellt. Durch den zuvor genannten Nachweis (Wert -> Praktik x Prinzip) ist nachgewiesen, dass die betreffende Praktik in dieser Beziehung zwingend zur Erfüllen des Qualitätskriteriums dient. Dabei können unterschiedliche Praktiken für den Nachweis des Wertes und des Qualitätskriteriums in die Beziehungen aufgenommen werden.

Damit eine Praktik ein Qualitätskriterium erfüllen kann, muss es eine Person innerhalb eines Entwicklungsteam geben, die diese Praktik durchführt. Da die Rollen explizit beschrieben sein müssen, ist ein direkter formaler Nachweis erbracht.

Am Ende der durchgeführten Praktik durch einen Team-Mitglied muss das Qualitätskriterium anhand eines konkreten Artefaktes nachgewiesen werden. Nur dann kann überprüft werden ob das Qualitätskriterium tatsächlich erfüllt wurde.

**Beispiel:**

In Kapitel 3.1.3 wurde das Qualitätskriterium Benutzbarkeit erläutert. In einem Vorgehensmodell soll sicher gestellt werden, dass dieses Qualitätskriterium auch tatsächlich berücksichtigt wird.

Der Product-Owner ist naturgemäß eine Rolle die ein sehr großes Interesse an der Benutzbarkeit der implementierten Funktionen hat. Er alleine kann isoliert dieses Qualitätskriterium nicht erfüllen.

Innerhalb der Praktik Sprint-Planing stellt der Product-Owner seine Stories, und damit seine Funktionen, dem Entwicklungsteam vor. Die zu implementierende Funktion wird im Rahmen dieser Praktik ausführlich besprochen. Durch diese Praktik hat der Product-Owner die Möglichkeit, die Benutzbarkeit sicher zu stellen.

Jede Funktion für sich isoliert, stellt die Benutzbarkeit noch nicht sicher. Oft sind einzelne Funktionen nur in Kombination mit anderen Funktionen überhaupt erst sinnvoll. Damit die Benutzbarkeit sicher gestellt ist, pflegt der Product-Owner einen Releaseplan. In diesem ist der zeitliche Ablauf der einzelnen Veröffentlichungen von Funktionalitäten aufgeführt.

***Benutzbarkeit*** -> *Produkt-Ownner x Sprint-Planing x Releaseplan*

Durch die Beziehung der drei Basiskonzepte kann in diesem Fall die Benutzbarkeit nachgewiesen werden. Bei der Untersuchung kann die Anzahl der gefundenen Beziehungen als ein Maß der Softwarequalität eingesetzt werden.

# 4 XP, Scrum und Kanban im ASDM

## 4.1 XP im ASDM

Das Vorgehensmodell Extreme Programming (XP) ist eines der ältesten agilen Vorgehensmodelle dessen Begründer Kent Beck ist. In seinem Buch *Extreme Programming Explained* [8] beschreibt er dieses Vorgehensmodell.

### 4.1.1 Werte

Kent beschreibt fünf zentrale Werte. Diese können als explizit definiert in das ASDM übernommen werden [9].

**Kommunikation** (engl. Cummunication)

Im XP wird Kommunikation sowohl für die Definition von Problemen als auch zu deren Lösung beschrieben. Zusätzlich wird mangelnde Kommunikation als Urheber von Problemen beschrieben.

**Einfachheit** (engl. Simplicity)

Mit diesem Wert ist immer die schlichteste Lösung gemeint, die trotzdem die Anforderungen erfüllt gemeint. Einfach darf nicht mit trivial verwechselt werden. Aus diesem Grund wird dieser Wert als größte intellektuelle Herausforderung im XP beschrieben.

Im Focus sollte eine aktuelle Lösung stehen, die nicht von Dauer sein muss und sich den Anforderungen und Ergebnissen stellt und somit flexibel ist. Die Anpassung ist die Folge und wird in Kauf genommen.

**Rückkopplung** (engl. Feedback)

Im XP wird Softwareentwicklung als ein permanenter Prozess verstanden. Wie in allen agilen Vorgehensmodellen wird auch im XP zur Kenntnis genommen, dass es fast nie möglich ist, schon zu Beginn alle relevanten Anforderungen zu kennen. Um diesem Umstand Rechnung zu tragen, wird Feedback als wesentlicher und unabdingbarer Wert verstanden. Dabei wird Feedback als Teil der Kommunikation verstanden, der von allen Rollen im Team gelebt werden muss.

**Mut** (engl. Courage)

Mut wird als Wert zur direkten Bekämpfung von Angst in XP gesehen. Dabei wird Angst als menschlicher Faktor und somit als Bestandteil der Softwareentwicklung akzeptiert. Mit Mut ist ein bestimmter Anspruch an jeden im Team verbunden. Ist ein Problem bekannt, dann sollte jeder mutig dieses kommunizieren und an einer Lösung arbeiten. Dabei werden Risiken bewusst in Kauf genommen.

**Respekt** (engl. Respect)

XP glaubt nicht daran, dass die vier zuvor genannten Werte für sich alleine Bestand haben können. Wenn alle Personen in einem Team auf einander Rücksicht nehmen und sorgfältig mit dem Projekt umgehen, ist ein Erfolg erreichbar. Dieser Umgang wird unter dem Wert Respekt zusammengefasst.

### 4.1.2 Prinzipien

Die beschriebenen Werte stellen eher abstrakte Leitlinien dar. In der Praxis stellt sich eher die Frage <Was bedeutet dieser Wert für meine Arbeit?> Der Mehrwert hängt von dem Kontext ab in dem Werte betrachtet werden. In der Folge werden die von Kent vorgeschlagenen Prinzipien beschrieben. Diese lassen einen stärkeren Bezug auf die tägliche Arbeit zu und erläutern wie die Werte zu interpretieren sind.

Wie die Werte sind auch die Prinzipien explizit beschrieben und können in das ASDM übernommen werden.

**Menschlichkeit** (engl. Humanity)

Softwareentwicklung wird von Menschen betrieben. Diese einfache Erkenntnis wird in Entwicklungsprozessen nicht ausreichend berücksichtigt. Dafür werden meist ökonomische Gründe genannt. Diese Unterlassung wird in XP als kurzsichtig angesehen. Entwickler benötigen ein menschliches Umfeld, um kreativ zu sein und damit einen echten Mehrwert zu schaffen.

Dabei identifiziert XP folgende Schwerpunkte [8]:

- Grundlegende Sicherheit (frei von Hunger, physischer Bedrohung usw.)
- Befähigung (die Fähigkeit und Gelegenheit das Team zu unterstützen)
- Zugehörigkeit (Identifikation mit einer Gruppe und einem gemeinsamen Ziel)

- Wachstum (Gelegenheit zur Verbesserung der Fähigkeiten)
- Vertrautheit (Gegenseitiges Verständnis im Team)

**Wirtschaftlichkeit** (engl. Economics)

XP ist sich der Tatsache bewusst, dass tägliche Arbeit finanziert werden muss. Softwareentwicklung darf nicht nur den „technischen Erfolg“ im Blick haben. Alle Handlungen müssen auf ihr ökonomisches Risiko und auf den wirtschaftlichen Mehrwert untersucht werden. Dabei identifiziert XP zwei Aspekte von ökonomischen Ausprägungen: Zum Einen den Zeitverlauf bis eine Investition sich im Umsatz niederschlägt. Zum Anderen formuliert XP die Erkenntnis, dass Softwareentwicklung besonders profitabel ist, wenn ein Team oder die Technologie die Möglichkeit hat, in der Zukunft auch neue oder andere Anforderungen zu erfüllen.

**Gegenseitiger Vorteil** (engl. Mitual Benefit)

Menschen in einer Gruppe verfolgen am erfolgreichsten ein Ziel, wenn nicht jeder nur auf seinen Vorteil bedacht ist. Viel erfolgreicher ist es, wenn jeder auch den Vorteil des anderen im Blick hat. In diesem Zusammenhang spricht man von einer Win-Win-Situation.

**Selbstähnlichkeit** (engl. Self-Similarity)

Sehr oft sind neue Probleme nur eine Abwandlung von schon bekannten Aufgaben. Dadurch lassen sich meist vorhandene Lösungen auf das neue Problem adaptieren. In diesem Zusammenhang werden in der Softwareentwicklung häufig sogenannte Frameworks eingesetzt, die eine Klasse von Problemen lösen.

**Verbesserung** (engl. Improvement)

Dieses Prinzip lebt von der Erkenntnis, dass gefundene Lösungen und Strategien oft nur eine begrenzte Zeit optimal sind. Deswegen versucht XP ständig gefundene Lösungen und auch Prozesse zu optimieren.

**Mannigfaltigkeit** (engl. Diversity)

Softwareentwicklung wird als kreativer Prozess akzeptiert. Ein einheitliches Profil für eine Rolle in einem Entwicklungsteam gibt es nicht. Vielmehr ist es wünschenswert, dass die unterschiedlichen Personen in einem Team unterschiedliche Schwerpunkte mit einbringen.

Durch die verschiedenen Sichtweisen auf ein Problem kann eine möglichst optimale Lösung gefunden werden.

**Reflexion** (engl. Reflection)

Nur wenn ein Entwicklungsteam bereit ist, aus den vergangenen Erfahrungen zu lernen, ist sichergestellt, dass auch zukünftige Probleme und Herausforderungen gemeistert werden können. Aus diesem Grund verlangt XP von allen Personen in einem Team eine selbstkritische Haltung. Diese Haltung gilt in erster Linie für Personen, wobei die Prozesse der Überprüfung ebenfalls standhalten müssen.

**Fluss** (engl. Flow)

Die Arbeit in XP soll ohne Unterbrechungen einen stetigen Fluss zulassen. Das Entwicklungsteam soll dabei ungestört und ohne Einwirkung von außen seine Aufgaben eigenständig lösen.

**Gelegenheit** (engl. Opportunity)

Häufig werden Probleme als etwas Unangenehmes wahrgenommen. XP begreift Aufgaben und Fragestellungen als etwas Positives und nicht als Problem und formuliert diese aus diesem Grund als Gelegenheit. Denn nur wenn Probleme als Gelegenheit wahrgenommen werden, ist es möglich aus diesen zu lernen und sich stetig zu verbessern.

**Redundanz** (engl. Redundancy)

In der Informatik wird meist Redundanz eher vermieden. Technische Lösungen sollen so oft wie möglich wieder verwendet werden. Allerdings gibt es auch Systeme und Prozesse, die auch noch funktionieren sollen wenn Teile ausfallen. Dies gilt für technische Systeme aber auch für ein Entwicklungsteam. In einem Team soll vermieden werden, dass Aufgaben nur von speziell ausgebildeten Personen erledigt werden können. Ein Mindestmaß an Abhängigkeit von einzelnen soll so erzielt werden.

**Fehlschlag** (engl. Failure)

XP akzeptiert Fehlschläge als Teil des Lebens und der Softwareentwicklung. Diese Misserfolge zu akzeptieren heißt eine Auseinandersetzung mit Problemen und damit Lerneffekte zu erzielen. Darüber hinaus führt die überlegte Beschäftigung mit Misserfolgen unweigerlich zu neuen Lösungsansätzen.

**Qualität** (engl. Quality)

Jeder möchte mit seiner Arbeit zufrieden sein und eine gute Qualität als Ergebnis erzielen. Oft muss man allerdings in der Umsetzung einer Lösungsstrategie Qualität gegen den zeitlichen Aufwand aufwiegen. XP entscheidet sich grundsätzlich bei dieser Frage immer für die Qualität. Dabei steht nicht der ästhetische Anspruch im Vordergrund. Vielmehr erkennt XP, dass qualitativ hochwerte Lösungen von heute eine Effektivitätssteigerung und damit Kostenersparnis für die Zukunft bringen.

**Babyschritte** (engl. Baby Steps)

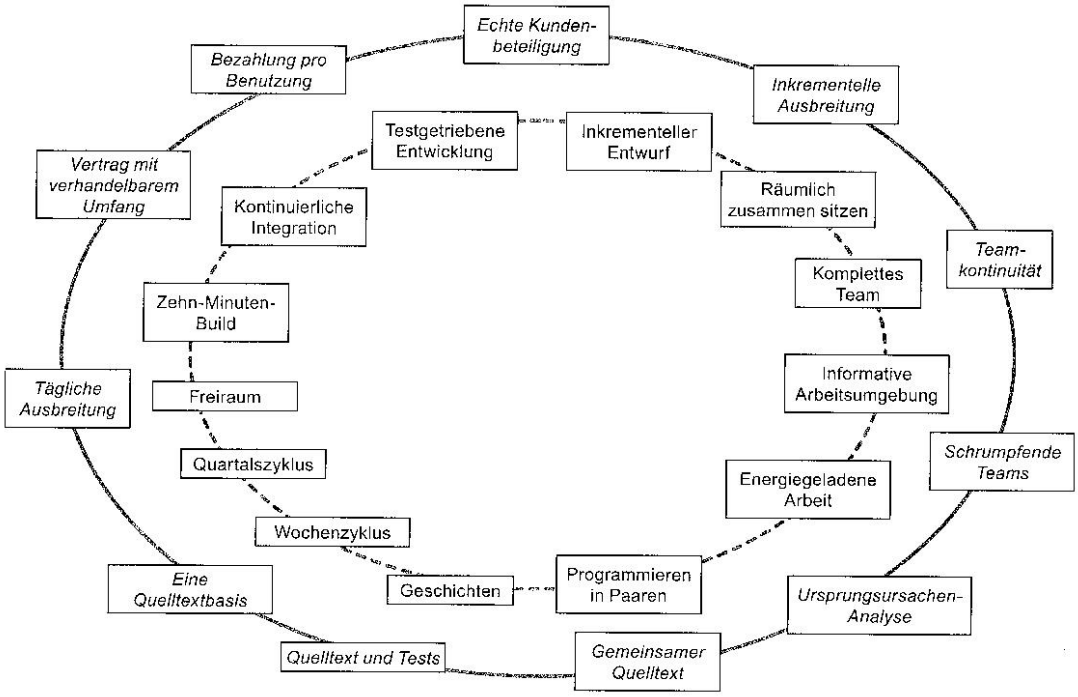
Aus der Erfahrung der Wasserfallmodelle werden in XP große Schritte vermieden. Dabei wird das Risiko minimiert und auch alle zuvor genannten Prinzipien wie Reflexion und andere kommen stärker zum Tragen. Bei zu großen, wenig zeitnahen Zyklen ist der Gewinn, aus Fehlern oder Erkenntnissen Schlüsse zu ziehen, gering.

**Akzeptierte Verantwortlichkeit** (engl. Accepted Responsibility)

XP glaubt an die Eigenverantwortung von einzelnen Personen innerhalb eines Entwicklungsteam. Allerdings geht XP noch einen Schritt weiter. Verantwortung, die einzig von Vorgesetzten an Personen weiter gegeben wird, kann nicht wahrgenommen werden. Die betreffenden Personen müssen diese Verantwortung ehrlich akzeptieren. Diese Akzeptanz betrifft in gleichem Maße Entwicklerteam und Kundenseite.

### 4.1.3 Praktiken

XP unterscheidet Primär- und Folgepraktiken. Dabei sind die Primärpraktiken für sich bereits vollständig. Darüber hinaus empfiehlt XP zunächst nur die Primärpraktiken einzusetzen. Erst wenn ein Team diese vollständig beherrscht, sollten die Folgepraktiken eingesetzt werden.




[Abbildung 11: Übersicht Primär- und Folgepraktiken von XP]

Da die Folgepraktiken einzig eine weitere Ausprägung der Primärpraktiken sind, werden in der Betrachtung im Rahmen des ASDM, nur die Primärpraktiken berücksichtigt. Damit ähnliche Ansetze nicht doppelt bewertet werden und damit keine irreführende Ergebnisse erzielt werden, werden die Folgepraktiken in dieser Betrachtung nicht berücksichtig.

Die 13 Primärpraktiken von XP sind:

**Räumlich zusammen sitzen** (engl. Sit Together)

Eine gute Kommunikation in einem Team ist unabdingbar für eine zielgerichtete Zusammenarbeit und um das Teamwork zu unterstützen. Daraus ergibt sich die Notwendigkeit, eine räumliche Nähe aller Team-Member herzustellen, um die geforderte Aufgabe optimal zu lösen.

**Komplettes Team** (engl. Whole Team)

Ein Team soll für sich selbstständig arbeiten und Entscheidungen treffen. Dies ist nur möglich, wenn alle notwendigen Qualifikationen im Team vorhanden sind. Die dafür notwendigen Rollen können unterschiedlich und der Aufgabenstellung entsprechend sein. Entscheidend ist, dass das Team keine Ressourcen von außerhalb benötigt und somit autark arbeiten kann.

**Informative Arbeitsumgebung** (engl. Informative Workspace)

Wie oben beschrieben, begreift XP Softwareentwicklung als einen kreativen Prozess. Dies bedeutet, dass der Arbeitsalltag von ständigen Veränderungen geprägt ist. Damit das gesamte Team in diesem Prozess den Überblick behält, ist es nötig den aktuellen Stand direkt in der Arbeitsumgebung abzulesen. Alles was dazu nötig ist, soll vom Team genutzt werden. Üblich sind in diesem Zusammenhang Flipchart-Zettel und Magnetboards. Aber auch einfache Ausdrucke können diesen Zweck erfüllen.

**Energiegeladene Arbeit** (engl. Energized Work)

Ein kreativer Prozess kann innovative Lösungen zu Tage bringen, wenn die Beteiligten ein echtes Interesse an dem Ergebnis haben. Deswegen erwartet XP eine entsprechende Arbeitseinstellung von allen Personen. Diese Primärpraktik ist nur schwer zu definieren. Trotzdem ist sie für den Erfolg entscheidend.

**Programmieren als Paar** (engl. Pair Programming)

Da XP von unterschiedlichen Charakteren bei der Teambesetzung ausgeht, können sich die verschiedenen Personen optimal ergänzen. Zusätzlich neigt eine einzige Person oft dazu, Fehler zu übersehen, da man sich auf einen Aspekt konzentriert. Um diesen Faktoren Rechnung zu tragen, schlägt XP das Programmieren zu zweit vor. Dieses ist einer der bekanntesten und wichtigsten Praktiken in XP überhaupt.

**Geschichten** (engl. Stories)

Die Anforderungen durch den Kunden werden in XP nicht in Grob- und Feinkonzepte formuliert sondern eine nicht formale Beschreibung in Stories ist ausreichend. Diese Stories werden von der Größe so bemessen, dass diese in einer Iteration abgearbeitet werden können. Zusätzlich muss jede Story für sich einen Mehrwert erbringen und getrennt releast werden können.

**Wochenzyklus** (engl. Weekle Cycle)

Der natürliche Rhythmus ist eine Arbeitswoche. Dieser Zyklus ist klein genug um ihn gut zu überschauen. Zudem können unerwartete Entwicklungen bis zu einem gewissen Rahmen abgefangen werden. Aus diesen Gründen wird der Wochenzyklus in XP als Iteration festgelegt. In diesem Zeitraum soll eine Story vollständig abgeschlossen sein.

**Quartalszyklus** (engl. Quarterly Cycle)

Der ausgedehnte Zyklus in XP, ist der Quartalszyklus. Dieser Zeitraum legt ein Release fest. Alle Stories werden zusammengefasst und als ein auslieferbares Artefakt releast.

**Freiraum** (engl. Slack)

Damit Softwareentwicklung ein kreativer Prozess bleibt, müssen Entwickler einen gewissen Teil ihrer Arbeitszeit frei einsetzen können. Die Zeit soll bewusst nicht genutzt werde, um aktuelle Stories oder Probleme zu lösen. Vielmehr sollen Entwickler die Zeit nutzen um andere Lösungsstrategien kennen zu lernen oder sich auch mit ganz anderen Klassen von Problemen beschäftigen.

**Zehn-Minuten-Build** (engl. Ten-Minute-Build)

In der täglichen Arbeit werden oft nur Teile eines Artefakts tatsächlich kompiliert und die dazu gehörigen Tests ausgeführt. Der Grund liegt hier in der Zeitersparnis. Dies führt allerdings oft dazu, dass Fehler auf Integrationsebene erst sehr spät gefunden werden und ein Release gefährdet ist. Um diesem Umstand zu begegnen, fordert XP einen kompletten Build innerhalb von 10 Minuten. In großen Projekten kann es schwer sein, diese Anforderung zu erfüllen. Allerdings führt diese Anforderung zu einer intelligenten Buildstrategie mit mehrstufiger Testausführung um den Buildprozess zu beschleunigen.

**Kontinuierliche Integration** (engl. Continuos Integration)

Alle Entwickler erarbeiten zusammen einen gemeinsamen Stand des Quellcodes. Jede lokale Änderung wird spätestens nach Abschluss einer Story in diesem gemeinsamen Stand integriert. Somit liegt ständig eine lauffähige Version der Gesamtsoftware als Artefakt vor.

**Testgetriebene Entwicklung** (engl. Test-First Programming)

Diese Primärpraktik wurde später zu einem eigenständigen Vorgehensmodell namens Test-Driven-Development weiter entwickelt. Im Kern steht der Gedanke, zunächst für eine Funktionalität einen Test zu schreiben und erst im Anschluss den eigentlichen produktiv Code zu entwickeln. Dadurch soll ein besseres Verständnis der Funktionalität und damit eine höhere Qualität erzielt werden.

**Inkrementeller Entwurf** (engl. Incremental Design)

Wie andere agile Methoden, geht XP davon aus, nicht zu Beginn den kompletten Überblick über alle Teilaspekte einer Implementierung zu kennen. Aus diesem Grund ist es nicht möglich einen Entwurf im Gesamten anzufertigen. Vielmehr wird der Entwurf mit jeder neuen Story überarbeitet und erweitert.

### 4.1.4 Prozesse

In Kapitel 2.2.2 wurden die Prozesse in XP beschrieben. Diese Beschreibung ist nicht explizit aus XP zu entnehmen. Die genannten Prozesse ergeben sich implizit aus der Beschreibung der Werte, Prinzipien und Praktiken.

Es können folgende Prozesse im Rahmen des ASDM identifiziert werden:

- Iteration
- Release
- Pair-Programming
- Continuous-Integration
- User-Stories und Tasks

### 4.1.5 Rollen

Die in Kapitel 2.2.1 beschriebenen Rollen untergliedern sich in Basis-Rollen und Hilfs-Rollen. Diese Rollen sind explizit beschrieben und können in das ASDM übernommen werden:

Basis-Rollen:

- Kunde
- Entwickler

Hilfs-Rollen:

- Produktmanager
- Produktbesitzer
- Benutzer
- XP-Coach

### 4.1.6 Artefakte

In XP werden Artefakte nicht explizit beschrieben. Aus der Beschreibung der Werte, Prinzipien und Prozesse können die folgenden Artefakte implizite für das ASDM abgeleitet werden.

- Release-Artefakt
- Lokaler-Source-Code
- CI-Infrastruktur
- Test-Infrastruktur
- Live-Infrastruktur
- User-Stories und Tasks

### 4.1.7 Beziehungen

Beziehungen sind in XP nicht explizit beschrieben. Diese werden implizit durch die bisher genannten Basiskonzepte beschrieben. Bei der Beschreibung wird zunächst überprüft, ob im XP die Werte tatsächlich umgesetzt werden.

Die Überprüfung erfolgt nach folgenden Beziehungen:

***Wert -> Praktik x Prinzip***

**Folgende Werte können durch Beziehungen im XP nachgewiesen werden:**

***Kommunikation*** -> *Räumlich zusammen sitzen* x *Reflexion*

***Einfachheit*** -> *Inkrementeller Entwurf* x *Babyschritte*

***Rückkopplung*** -> *Informative Arbeitsumgebung* x *Fluss*

***Mut*** -> *Energiegeladene Arbeit* x *Fehlschlag*

***Respekt*** -> *Freiraum* x *Menschlichkeit*

Im nächsten Schritt soll überprüft werden, ob die in Kapitel 3.1 beschriebenen Qualitätskriterien mit den beschriebenen Praktiken erfüllt werden. Zusätzlich zu den Praktiken werden hier die Rollen miteinbezogen.

Die Überprüfung erfolgt nach folgenden Beziehungen:

***Qualitätskriterien -> Rolle x Praktik x Artefakt***

**Folgende Qualitätskriterien können durch Beziehungen im XP nachgewiesen werden:**

***Funktionalität*** *-> Entwickler x Kontinuierliche Integration x CI-Infrastruktur*

Häufig können Funktionen nur im Zusammenspiel mit anderen Funktionalitäten bewertet werden. Deswegen ist die qualitative Beurteilung von Funktionalität isoliert betrachtet in diesen Fällen unmöglich.

Mit der Praktik der kontinuierlichen Integration wird jede neue Funktion sofort in die vorhandene Software eingebunden. Damit dies praktisch umgesetzt werden, kann muss eine CI-Infrastruktur bereitgestellt werden. Dieses Artefakt und die damit verbundene Praktik wird von der Rolle Entwickler gelebt.

***Funktionalität*** *-> Kunde x Testgetriebene Entwicklung x Release-Artefakt*

Nach Abschluss der Implementierung ist der Kunde derjenige, der in seiner Rolle eine neue Funktion nutzen soll. In XP gibt es durch die Praktik der testgetriebenen Entwicklung die Möglichkeit den Kunden aktiv schon zur Entwicklungszeit mit einzubeziehen. Dabei erstellt das Entwicklungsteam zunächst Tests und spricht diese mit dem Kunden ab. Nachdem die Funktionalitäten implementiert wurden werden die Tests auf dem Release-Artefakt ausgeführt. Durch diese Beziehung wird Funktionalität vor allem aus Kundensicht sichergestellt.

***Funktionalität*** *-> Produktbesitzer x 10-Minuten-Build x User-Stories und Tasks*

Funktionalität textuell anhand von User-Stories zu beschreiben ist für den Produktbesitzer oft eine große Herausforderung. In der Regel kann eine User-Story immer nur einen Teil der tatsächlichen Wirklichkeit beschreiben. Vergessene Details gehören fast immer zum Alltag.

Durch die Praktik des 10-Minuten-Build hat der Produktbesitzer unmittelbar nach der Implementierung die Möglichkeit, die konkrete Funktionalität zu überprüfen. Dadurch können kleine Korrekturen noch eingebunden werden und im Gesamtergebnis wird das Qualitätskriterium Funktionalität unterstützt.

***Zuverlässigkeit*** *-> Entwickler x Kontinuierliche Integration x CI-Infrastruktur*

Durch die Praktik Kontinuierliche Integration in eine CI-Infrastruktur kann frühzeitig überprüft werden wie ein Gesamtsystem und neue Funktionen auf mögliche Fehler reagieren. In diesem Zusammenhang werden von den Entwicklern meist verschiedene Konfigurationsdateien als Tests eingebunden. Damit lassen sich permanent Fehlerzustände erzeugen.

Gerade bei Weiterentwicklungen ist diese Beziehung besonders Interessant. Oft werden verbindliche Antwortzeiten eines Systems festgelegt. Bei der ersten Implementierung wird in der Regel die Einhaltung dieser Kennwerte beachtet. Werden später Weiterentwicklungen an einem System vorgenommen, können diese Kennwerte übersehen werden. Diese Schwäche wird durch die beschriebene Beziehung ausgeglichen und das Qualitätskriterium Zuverlässigkeit unterstützt.

Interessant ist, dass die gleiche Beziehung von Entwickler, Kontinuierliche Integration und CI-Infrastruktur sowohl das Qualitätskriterium Funktionalität als auch Zuverlässigkeit erfüllt. Dies ist werde Zufall noch ungünstig. Es zeigt im Gegenteil, wie sie agile Methoden insgesamt unterstützen können.

***Zuverlässigkeit*** *-> Entwickler x Pair-Programming x Lokale- Source-Code*

Das Qualitätskriterium Zuverlässigkeit soll verhindern, dass neue Funktionen schon vorhandene Funktionalitäten beeinträchtigen. Die Zusammenhänge, die zu solchen Beeinträchtigungen führen, sind in der Regel nicht trivial. Hinzu kommt, dass es von großem Vorteil ist, mögliche Schwachstellen der eigenen Implementierung frühzeitig zu erkennen.

Mit der Praktik des Pair-Programming ist sicher gestellt, dass jede Implementierung von zwei Entwicklern durchgeführt wird. Dadurch werden immer zwei Sichtweisen bei einem Lösungsansatz berücksichtigt. Mögliche negative Seiteneffekte werden so minimiert und die Zuverlässigkeit gesteigert.

***Benutzbarkeit*** *-> Produktbesitzer x Freiraum x Test-Infrastruktur*

Eine gut zu benutzende Softwarelösung ist keine einfache Definition. Jeder Produktbesitzer muss für seine Produkte eigene Kriterien aufstellen, die ihm angemessen erscheinen. Damit dies erfolgreich ist, muss der Produktbesitzer die Praktik Freiraum anwenden. In der täglichen Arbeit heißt das für den Produktbesitzer die Benutzbarkeit auf dem Artefakt der Test-Infrastruktur zu überprüfen. Die Beziehung der genannten Basiskonzepte stellt das Qualitätskriterium Benutzbarkeit sicher.

***Benutzbarkeit*** *-> Kunden x Komplettes Team x User-Stories und Tasks*

Die Einschätzung des Kunden von der Benutzbarkeit ist letztlich bei einer Softwarelösung entscheidend. Durch die Praktik Komplettes Team wird in XP sicher gestellt, dass alle im Team die Einschätzung des Kunden direkt erhalten. Diese werden in den Stories und Tasks als Artefakt festgehalten.

Auch wenn alle drei Basiskonzepte für sich stehen, kann in XP die aufgezeigte Beziehung erstellt werden. Durch diesen direkten Einfluss des Kunden auf die User-Stories wird die Benutzbarkeit gesteigert.

***Effizienz*** *-> Entwickler x Informative Arbeitsumgebung x Test-Infrastruktur*

Damit eine Software Lösung effizient arbeitet, müssen die Entwickler die betroffenen Nachbarsysteme möglichst gut kennen. Die Praktik Informative Arbeitsumgebung unterstützt die Entwickler, alle notwendigen Informationen für die aktuelle Lösung ein zu holen.

Bevor die Weiterentwicklung in den Livebetrieb geht, können die Entwickler in der eigenen Test-Infrastruktur überprüfen, ob die Änderungen negative Auswirklungen auf das Gesamtsystem haben.

***Änderbarkeit*** *-> Kunde x Wochenzyklus x Release-Artefakt*

Die Praktik Wochenzyklus führt zu einem iterativen Ansatz in der Softwareentwicklung. Mit jedem Wochenzyklus wird eine nächste Evolutionsstufe erreicht. Dieses Vorgehen bedeutet eine ständige Änderung. Wobei die Schritte immer in feste Einheiten zeitlich getrennt sind.

Der Kunde kann an das Entwicklungsteam permanent Änderungswünsche adressieren. Damit beeinflusst er die Inhalte der Wochenzyklen. Die erzielten Ergebnisse werden vom Entwicklungsteam an den Kunden mit dem Release-Artefakt ausgeliefert. Nach Auslieferung hat der Kunde die Möglichkeit die nächsten Änderungswünsche an das Entwicklungsteam zu geben.

## 4.2 Scrum im ASDM

Scrum ist eines der verbreitetsten agilen Vorgehensmodelle. Damit lassen sich sehr gut Businessprozesse abbilden. Vor allem bei Unternehmen, die agile Entwicklung zum ersten Mal einsetzen möchten, ist Scrum sehr beliebt. In der Folge werden die Basiskonzepte und Beziehungen für Scrum nach „*Scrum in der Praxis*“ [10] beschrieben.

### 4.2.1 Werte

Die folgenden Werte sind in Scrum explizit beschrieben, [10] und können so in das ASDM übernommen werden.

**Verpflichtung (engl. Commitment)**

Damit ein Team ein gemeinsames Ziel haben kann und dieses erfolgreich erreicht, müssen sich alle Teammitglieder auf gemeinsame Ziele und Werte verständigen. Diese Vereinbarung gilt als verbindlich für alle und nennt sich Commitment.

**Fokussierung (engl. Focus)**

Konzentriertes Arbeiten wird oft durch Einflüsse von außen verhindert. Damit ein Team effektiv und qualitativ arbeiten kann, müssen diese Ablenkungen vermieden werden. In diesem Zusammenhang hat der Scrum Master eine besondere Verantwortung. Seine Aufgabe ist es, alle störenden Außeneinflüsse vom Team fernzuhalten.

**Offenheit (engl. Openness)**

Ein Team kann nur aus Fehlern lernen, wenn alle Personen im Team offen mit Problemen und Fehlern umgehen. In diesem Zusammenhang ist der Wert Mut sehr wichtig. In einem agilen Umfeld ist es wichtig, dass alle Teilnehmer ohne Angst ihre Meinung vertreten können. Ein Team muss verstehen, dass unterschiedliche Meinungen eine Bereicherung sind und letztlich zu einem besseren Gesamtergebnis beitragen.

**Respekt (Respect)**

In der Regel werden Menschen immer Erfolg anstreben. Auch wenn Fehler oder Schwächen von einzelnen erkennbar sind, sollte ein erfolgreiches Team immer davon ausgehen, dass der Betreffende versucht das bestmögliche Ergebnis anzustreben. Aus diesen Gründen sollte jede Person mit dem entsprechenden Respekt behandelt werden. Nie darf Kritik persönlich werden. Ziel ist es, Fehler auf einer sachlichen Ebene zu diskutieren.

**Mut (Curage)**

Wenn die Arbeit in einem Team durch Offenheit und Respekt geprägt ist, werden die Personen in dem Team automatisch den Mut entwickeln neue Wege zu gehen. Der Scrum-Master sollte dabei immer beobachten, dass Mut nicht zu Übermut wird. Denn auch in bestimmten Situationen „Nein“ zu sagen gehört zum Mutigsein.

**Einfachheit (Simplicity)**

Oft neigt Softwareentwicklung dazu, komplexe Lösungen zu erarbeiten. Selbstverständlich können sehr komplexe Probleme nicht mit trivialen Lösungen beantwortet werden. Damit Software gut zu warten ist und möglichst wenig Fehler auftreten, sind immer einfache Lösungen anzustreben.

**Kommunikation**

Die Abstimmung zwischen den Beteiligten ist in allen agilen Umgebungen wichtig. Selten können Lösungen von einzelnen Personen eigenständig entwickelt werden. Ein kommunikativer Umgang ermöglicht es dem Team, erfolgreich seine Ziele zu erreichen.

**Feedback**

Aus dem Vergangenen lernen, bedeute für ein Team sich ständig zu verbessern. Wenn Probleme oder Fehler auftreten, kann man diese in der Zukunft nur vermeiden, wenn man gewillt ist aus diesen zu lernen. Wenn alle Mitglieder eines Teams Feedback geben und bereit sind selbst Feedback von anderen zu akzeptieren, können sich alle Mitglieder des Teams persönlich weiter entwickeln und vorhandene Prozesse erfolgreich reformieren.

### 4.2.2 Prinzipien

Scrum übernimmt die 12 Prinzipien des agilen Manifest in das eigene Vorgehensmodell. Diese Prinzipien [10] sind explizit beschrieben und können so in das ASDM übernommen werden.

**Den Kunden zufrieden stellen**

Scrum stellt den Kunden in den Mittelpunkt der Softwareentwicklung. Deswegen wird die Zufriedenheit als ein wichtiges Prinzip angesehen. Dies soll vor allem durch eine schnelle und kontinuierliche Auslieferung von Software ermöglicht werden.

**Änderungen willkommen heißen**

Scrum akzeptiert, dass sich Softwareentwicklung ständig anpassen muss, deswegen sollen Änderungen nicht als störend empfunden werden. Innerhalb der Scrum-Prozesse sind

Veränderungen durchaus erwünscht und sollten von allen Teammitgliedern begrüßt werden.

**Häufige Auslieferung**

Damit der Kunde kurzfristig Feedback zum aktuellen Stand geben kann, ist es nötig kontinuierlich Software auszuliefern. Dadurch sollen Fehlentwicklungen möglichst schnell aufgedeckt werden. Lösungen können auf diesem Wege frühzeitig gefunden und damit Kosten eingespart werden.

**Crossfunktionale Zusammenarbeit**

Für die Lösung sind oft unterschiedliche Fähigkeiten in einem Team gefragt. Wichtig ist dabei, dass die Personen effektiv zusammen arbeiten. Dabei muss jeder ausreichend Einblick in die

Arbeit einer „Nachbardisziplin“ haben damit eine echte crossfunktionale Zusammenarbeit möglich ist.

**Unterstützung leisten und Vertrauen schenken**

Wenn Teammitglieder aufgrund von fehlenden Informationen oder Ressourcen nicht arbeiten können, sollte jeder bereit sein, bei der Problemlösung behilflich zu sein. Auch wenn in Scrum der Scrum-Master eine besondere Stellung bei Beseitigung von organisatorischen Problemen hat, sollte jedes Teammitglied andere unterstützen und stets bereit sein, Vertrauen zu schenken.

**Direkte persönliche Kommunikation**

Damit alle tagesaktuellen Themen schnell besprochen werden können, macht es Sinn, eine Kommunikation auf kurzem Weg herzustellen. Grundsätzlich sollte immer eine betreffende Person direkt angesprochen werden. Mögliche Missverständnisse treten im persönlichen Kontakt eher selten auf. Auch können Nachfragen im persönlichen Gespräch sofort geklärt werden.

**Funktionierende Software**

Jeder Entwickler in einem Team muss die Möglichkeit haben, sich zu jeder Zeit den aktuellen Stand an lauffähiger Software als lokale Kopie zu holen. Nur so können unterschiedliche Themen parallel bearbeitet und ein Integrationsaufwand minimiert werden. Damit dies sichergestellt ist, muss es zu jeder Zeit eine funktionierende Software mit allen aktuellen Änderungen geben.

**Nachhaltige Geschwindigkeit**

Hohe Schwankungen und kurzzeitige Mehrarbeit wirken sich negativ auf die Effektivität eines Teams aus. Scrum begegnet diesem Umstand, indem es nachhaltige Geschwindigkeit fordert. Damit ist ein kontinuierlicher Durchsatz von Aufgaben über einen längeren Zeitraum gemeint.

**Streben nach technischer Exzellenz**

Routine in der täglichen Arbeit bedeutet, sich nicht zu verbessern. Scrum fordert eine permanente Verbesserung von Prozessen, insbesondere von jedem einzelnen Teammitglied. In der Softwareentwicklung spiegelt sich dies in diesem Prinzip wieder.

**Einfach ist besser**

Um der Komplexität in der Softwareentwicklung zu begegnen, erkennt Scrum, dass eine einfache Lösung gegenüber einer komplizierten grundsätzlich vorzuziehen ist. Dieses Prinzip ist von der Erkenntnis geprägt, dass dies nicht immer einzuhalten ist. Jeder Entwickler sollte im Zweifel immer eine einfache Lösung vorziehen.

**Selbstorganisiert agieren**

Ein Scrum-Team soll eigenverantwortlich seine Themen umsetzen. Nur wenn jeder einzelne eine echte Verantwortung für seine Arbeit übernimmt, kann ein Team erfolgreich arbeiten. Eine Selbstorganisation des Teams ist Voraussetzung für dauerhaften Erfolg. Die Übernahme von Verantwortung setzt im Team die Unabhängigkeit von Prozessen und Einflüssen voraus.

**Überprüfen und Anpassen**

Ein regelmäßiges Feedback innerhalb eines Teams führt zu Erkenntnissen, die sich in geänderten Prozessen oder Lösungen niederschlagen. Dieses Vorgehen wird auch als „Inspect and Adapt“ bezeichnet.

### 4.2.3 Praktiken

In Scrum werden Praktiken mit Events bezeichnet. Damit sind Praktiken nicht explizit beschrieben. Diese werden alle aus den Events abgeleitet. In Abschnitt 2.3.2 wurden im Rahmen einer allgemeinen Beschreibung des Scrum-Prozess schon einige wichtige Praktiken ausführlich beschrieben. Diese sind:

- Sprint-Planing
- Daily-Scrum
- Review
- Retrospektive

Ergänzend zu diesen wendet Scrum noch folgende Praktiken an:

**Backlog-Grooming**

Das Backlog-Grooming soll ein gemeinsames Verständnis über kommende Aufgabe liefern. Im Rahmen dieses Meetings werden kommende Themen besprochen und mögliche Probleme

erörtert. Diese Themen befinden sich bewusst noch nicht in der Bearbeitung. Durch dieses Vorgehen werden möglichst frühzeitig mögliche Hindernisse überwunden.

**Estimation**

Damit in Scrum Aufwände abgeschätzt werden können, werden unterschiedliche Schätzverfahren angewendet. Das bekannteste ist das Schätzen nach Story-Points. Dabei werden Stories nicht mit einem Zeitaufwand geschätzt. Es soll die Komplexität einer Aufgabe eingeschätzt werden. Oft sind Stories mit einem gewissen Anteil an Unsicherheit verbunden. Diese Unsicherheit kann in Story-Points besser als in einer Zeiteinheit abgebildet werden.

### 4.2.4 Prozesse

**Sprint**

Der zentrale Prozess in Scrum ist der Sprint. Dieser wurde in Abschnitt 2.3.2 ausführlich beschrieben. Diese Beschreibung wird in Scrum explizit vorgenommen und kann so in das ASDM übernommen werden.

**Releaseplanung**

Nicht jeder durchgeführter Sprint muss automatisch ein neues Release erzeugen. Auch können verschiedene Features in einem Release mittels eines Feature-Toggles zeitweilig deaktiviert werden. Der Product-Owner entscheidet in einem Sprintteam, wann welche Funktion tatsächlich live gestellt wird. Diese Aufstellung der einzelnen Funktionen wird als Releaseplanung bezeichnet. Dabei plant ein Team mit Hilfe der Velocity, wann welche Funktion fertig gestellt werden kann. Die Velocity ist eine Metrik, die in einem Team beschreibt, wie viele Story-Points ein Team durchschnittlich pro Sprint bewältigen kann. Mit Hilfe dieser Metrik lassen sich auch vergleichsweise langfristige Planungen aufstellen. Diese Technik wird vom Team für die Releaseplanung angewendet.

### 4.2.5 Rollen

Im Abschnitt 2.3.1 wurden bereits die Rollen im Scrum ausführlich beschrieben. Diese Beschreibung liegt in Scrum explizit vor und kann so in das ASDM übernommen werden. Scrum verfolgt den Ansatz der crossfunktionalen Rollen. Aus diesem Grund gibt es nur drei Rollen:

- Product-Owner

- Scrum-Master
- Team-Member

### 4.2.6 Artefakte

Die folgenden Artefakte sind bereits im Kapitel 4.1.6 im Rahmen der Beschreibung von XP ausführlich dokumentiert. Diese Artefakte werden in der gleichen Bedeutung auch in Scrum verwendet.

- Release-Artefakt
- Lokaler-Source Code
- CI-Infrastruktur
- Test-Infrastruktur
- Live-Infrastruktur
- User-Stories und Tasks

Zusätzlich gibt es in Scrum folgende explizit beschriebene Artefakte:

**Produkt-Backlog**

Alle kommenden Stories werden vom Product-Owner in einem Produkt-Backlog gepflegt. In diesen Stories sind zukünftig Funktionen beschrieben, die im Rahmen des Backlog-Groomings weiter detailliert besprochen werden.

**Sprint-Backlog**

Alle aktuellen Stories in einem Sprint sind in dem Sprint-Backlog beschrieben. Diese Stories müssen mit entsprechenden Tasks versehen sein. Auch müssen alle Stories einen Status „Definition of Ready“ haben und mit Story-Points geschätzt sein.

**Impediment-Backlog**

Im Rahmen des Retro-Meetings werden mögliche Hindernisse identifiziert. Diese werden in dem Impediment-Backlog gepflegt. Dieses Backlog wird vom Scrum-Master beaufsichtigt. Für jedes Hindernis wird eine konkrete Maßnahme beschlossen. In der Folgezeit wird im Rahmen der Retro überprüft, ob die Maßnahmen erfolgreich waren.

**Scrum-Board**

Im laufenden Sprint werden alle Stories auf einem Scrum-Board abgebildet. Alle Teammitglieder können zu jeder Zeit den aktuellen Stand jeder einzelnen Story sehen. Das Daily-Scrum-Meeting wird direkt vor diesem Scrum-Board abgehalten.

**Burndown-Chart**

Über den Verlauf eines Sprints wird vom Scrum-Master ein Brundown-Chart gepflegt. In diesem ist erkennbar, wie viele Stories bereits erfolgreich bearbeitet wurden. Es wird auch die Anzahl der gefunden und offenen Bugs aufgeführt. Mit diesem Hilfsmittel ist es möglich, während des Sprintverlaufs eine Voraussage über den Erfolg des Sprints zu treffen.

**Definition of Done**

Jedes Entwicklungsteam bestimmt für sich eine Anzahl von Kriterien, um den Status einer vollständig abgeschlossenen Story zu definieren. Damit legt das Team für sich verbindliche Qualitätsregeln für die eigene Arbeit fest.

**Releaseplan**

Als Ergebnis der Releaseplanung wird ein Releaseplan erstellt. In diesem Plan ist festgehalten, wann welche Funktion tatsächlich live ist und dem Kunden zur Verfügung steht. Siehe auch 4.2.4 Prozesse.

### 4.2.7 Beziehungen

Beziehungen sind in Scrum nicht explizit beschrieben. Diese werden implizit durch die bisher genannten Basiskonzepte beschrieben. Bei der Beschreibung wird zunächst überprüft, ob im Scrum die Werte tatsächlich umgesetzt werden.

Die Überprüfung erfolgt nach folgenden Beziehungen:

***Wert -> Praktik x Prinzip***

**Folgende Werte können durch Beziehungen im Scrum nachgewiesen werden:**

***Verpflichtung*** *-> Sprint-Planing x Den Kunden zufrieden stellen*

***Fokussierung*** *-> Backlog-Grooming x Nachhaltige Geschwindigkeit*

***Offenheit*** *-> Review x Direkte persönliche Kommunikation*

***Respekt*** *-> Estimation x Unterstützung leisten und Vertrauen schenken*

***Mut*** *-> Retrospektive x Änderungen willkommen heißen*

***Einfachheit*** *-> Backlog-Grooming x Einfach ist besser*

***Kommunikation*** *-> Daily-Scrum x Direkte persönliche Kommunikation*

***Feedback*** *-> Retrospektive x Überprüfen und Anpassen*

Im nächsten Schritt soll überprüft werden, ob die in Kapitel 3.1 beschriebenen Qualitätskriterien mit den beschriebenen Praktiken erfüllt werden.

Damit ein Qualitätskriterium erfüllt werden kann, muss es eine handelnde Person geben, die diese Praktik durchführt. Diese handelnde Person wendet in seiner Rolle die entsprechende Praktik an und als Ergebnis dieser Handlung entsteht ein Artefakt.

Die Überprüfung erfolgt nach folgenden Beziehungen:

***Qualitätskriterien -> Rolle x Praktik x Artefakt***

**Folgende Qualitätskriterien können durch Beziehungen im Scrum nachgewiesen werden:**

***Funktionalität*** *-> Produkt-Owner x Sprint-Planing x User-Stories und Tasks*

In Scrum erstellt und pflegt der Product-Owner die Stories. Diese wurden schon vor der Umsetzung im Rahmen des Backlog-Groomings mit dem Team besprochen. Zu Beginn der Umsetzung, stellt der Product-Owner jede Story in seinen finalen Zustand dem Team im Rahmen des Sprint-Planing vor. Jeder im Team kann in diesem Rahmen erneut Fragen stellen, um die einzelne Funktion tiefgreifend zu verstehen.

Im zweiten Teil des Sprint-Planing erstellt das Entwicklungsteam aus der einzelnen Story entsprechende Tasks, die zur Umsetzung notwendig sind. Diese Praktik erfolgt nicht isoliert vom

Product-Owner, oft tauchen in der detailierten Planung noch Fragen auf. Diese werden ständig mit dem Product-Owner besprochen.

***Funktionalität*** -> *Team-Member x Backlog-Grooming x Sprint-Backlog*

Schon weit vor der eigentlichen Implementierung werden im Scrum die Stories im Rahmen des Backlog-Groomings besprochen. Diese Meetings werden von allem Rollen im Team wahrgenommen. Dadurch wird sichergestellt, dass möglichst alle potentiellen Probleme rechtzeitig diskutiert werden können. Durch diese Praktik ist auch sichergestellt, dass der Product-Owner seine Stories in einer verständlichen Form verfasst. Die Funktionalitäten, die in den Stories beschrieben sind, werden durch diese Praktik ausreichend beleuchtet.

Nach Abschluss dieser Praktik sind diese Stories im Sprint-Backlog vorhanden. Somit stellt dieses das Artefakt und somit das Ergebnis dieser Praktik dar.

***Zuverlässigkeit*** -> *Team-Member x Review x CI-Infrastruktur*

Nach Implementierung einer Story wird ein Review dieser Story durchgeführt. Dabei stellen die Team-Member das Ergebnis vor und der Product-Owner kann überprüfen ob das Ergebnis seinen Vorstellungen entspricht.

Sehr oft werden durch Implementierungen von neuen Funktionalitäten schon vorhandene Funktionen beeinträchtigt. Scrum setzt zwingend eine CI-Infrastruktur voraus. Innerhalb dieser Infrastruktur werden mit der Implementierung von neuen Funktionen zeitgleich Tests für die vorhandenen Funktionen durchgeführt. Dadurch ist zu jeder Zeit sichergestellt, dass die Team-Member Funktionen in einem Review zeigen, die keine negativen Effekte auf bereits zuvor implementierte Funktionen haben. Durch diese Praktik von dem Team-Membern unter Nutzung des Artefakts CI-Infrastruktur ist das Qualitätskriterium Zuverlässigkeit sichergestellt.

***Benutzbarkeit*** *-> Produkt-Ownner x Sprint-Planing x Releaseplan*

Naturgemäß hat der Product-Ownder ein ganz besonderes Interesse an der Benutzbarkeit der implementierten Funktionen. Deswegen ist dies eine zentrale Rolle, wenn dieses Qualitätskriterium erfüllt werden soll.

Sehr oft sind einzelne Funktionen nur im Zusammenhang mit weiteren Funktionalitäten sinnvoll. Deswegen ist die Reihenfolge der Veröffentlichung von Funktionen entscheidend. Diesen zeitlichen Ablauf pflegt der Product-Owner in einem Releaseplan. Somit ist sichergestellt, dass jede Funktion zum richten Zeitpunkt dem Benutzer angeboten wird.

Alle Stories werden von Product-Owner im Rahmen des Sprint-Planing detailiert vorgestellt. Hier hat der Product-Owner die Möglichkeit, die aktuelle Funktion mit dem Wissen des Releaseplanes vorzustellen. Häufig werden zunächst Funktionen in Teilen umgesetzt. In dem Sprint-Planing kann der Product-Owner das Entwicklungsteam über die nächsten Schritte informieren und Hinweise geben, was bisher implementiert wurde und welche Funktionen als nächstes kommen werden.

***Effizienz*** *-> Team-Member x Sprint-Planing x User-Stories und Tasks*

Bei verteilten Anwendungen spielt die Effizienz eine immer entscheidendere Rolle. Einzelne Funktionen können für sich optimal implementiert sein und doch im Gesamtsystem nur mangelhaft funktionieren.

Damit dieses Qualitätskriterium erfüllt werden kann, ist es wichtig im Vorfeld keine technische Spezifikation vor zu geben. Aus diesem Grund werden in den Stories ausschließlich fachliche Anforderungen formuliert. Diese Stories werden im Rahmen des Sprint-Planing von allen Team-Membern zusammen in einzelne Tasks gegliedert. Dabei stellen die Tasks die technische Umsetzung der fachlichen Anforderung aus den Stories dar.

Durch das Erstellen der technischen Umsetzung anhand von Tasks erst kurz vor der Implementierung im Sprint-Planing von dem Team-Membern, ist ein Maximum an Effizienz bei der gewählten Lösung sicher gestellt.

***Änderbarkeit*** *-> Produkt-Owner x Retrospektive x Test-Infrastruktur*

Software wird häufig mit unterschiedlichem Funktionsumfang angeboten. Nicht jeder Benutzer benötigt alle Funktionen. Anbieter erreichen dadurch unterschiedliche Kundengruppen. Damit dies realisiert werden kann, ist ein hohes Maß an Änderbarkeit von Software nötig.

Innerhalb des Entwicklungsteams steht dem Product-Owner eine Test-Infrastruktur zur Verfügung. Nachdem die ersten Funktionen implementiert wurden, kann der Product-Owner diese lange vor dem Livegang testen.

In der Praktik Retrospektive werden die Erkenntnisse von dem Product-Owner an das Entwicklungsteam zurück gegeben. Dabei ist entscheidend, dass nicht einzelne Stories diskutiert werden, sondern der Gesamtumfang beleuchtet werden kann.

## 4.3 Kanban im ASDM

Gegenüber XP und Scrum ist Kanban ein vergleichsweise junges Vorgehensmodell. Im Zentrum steht die Idee, angefangene Arbeitspakete schnell zu beenden. Ursprünglich kommt diese Idee aus der Autoindustrie und sollte dort die Lagerkosten verringern. In der Softwareentwick

lung soll damit ein Übermaß an Kontextwechsel und kurze Releasezeiten erreicht werden.

### 4.3.1 Werte

Die Werte in Kanban wurden bereits in Kapitel 2.4.2 beschrieben. Diese sind „explizit“ beschrieben und können in das ASDM übernommen werden.

- Transparancy
- Balance
- Collaboration
- Customer focus
- Flow
- Leadership
- Understanding
- Agreement

- Respect

### 4.3.2 Prinzipien

Im Gegensatz zu XP und Scrum werden in Kanban keine Prinzipien beschrieben. Es werden bewusst „nur“ Werte und Praktiken genannt. Damit ist nicht gemeint, dass ein Entwicklungsteam keine Prinzipien haben soll. Diese werden nur von dem Vorgehensmodell nicht vorgegeben. Kanban beschreibt mehr *dass* etwas gemacht werden soll, weniger *wie* es zu machen ist. [11] Jedes Entwicklungsteam für sich kann sich Prinzipien aufstellen. Zwingend gefordert ist dies bei Kanban nicht.

Aus diesem Grund können keine Prinzipien im Rahmen des ASDM definiert werden.

### 4.3.3 Praktiken

In Kapitel 2.4.2 wurden ein Teil der Praktiken in Kanban bereits beschrieben. Diese Beschreibung ist „explizit“ und kann so in das ASDM übernommen werden.

- Visualisierung des Arbeitsflusses
- Limitierung des Work-in-Progress (WIP)
- Steuerung und Messung des Arbeitsflusses
- Prozesse explizit machen
- Verbesserungen durch bewährte Modelle

### 4.3.4 Prozesse

Stärker als in XP und Scrum stellt Kanban den Gedanken der ständigen Verbesserung in den Mittelpunkt. In Kanban wird dieser Prozess als Kaizen beschrieben, das eine stetige Wandlung zum Besseren meint. Dieses Vorgehen wird auch als „Evolutionäres Change Management“ bezeichnet.

Weitere Prozesse sind in Kanban nicht beschrieben und können auch nicht zwingend implizit abgeleitet werden.

### 4.3.5 Rollen

Bei der Rollenverteilung in einem Entwicklungsteam verfolgt Kanban die gleiche Strategie wie bei den Prinzipien. Es wird erklärt, welche Aufgaben zu bewältigen sind aber nicht welche Rollen dafür notwendig sind. Die Rollen könnten zwar anhand der Praktiken implizit abgeleitet werden. Das ASDM verlangt zwingend, dass Rollen explizit beschrieben sein müssen. [vgl. 7 Seite 112]

Aus diesem Grund können keine Rollen für das ASDM definiert werden.

### 4.3.6 Artefakte

In Kanban werden Artefakte nicht explizit beschrieben. Jedoch können aus den Beschreibungen folgende Artefakte implizit abgeleitet werden:

- Release-Artefakt
- CI-Infrastruktur
- Live-Infrastruktur
- User-Stories und Tasks
- Product-Backlog
- Kanban-Board
- Releaseplan

Die Beschreibung der Artefakte ist identisch mit dem in Kapitel 4.2.6.

### 4.3.7 Beziehungen

Für die Vorgehensmodelle XP und Scrum wurde durch Beziehungen nachgewiesen, dass die Werte tatsächlich umgesetzt werden (Wert -> Praktik x Prinzip). In Kanban gibt es keine Beschreibung des Basiskonzeptes Prinzip. Aus diesem Grund, können für die Werte keine Beziehungen aufgestellt werden und anhand des ASDM kein Nachweis über die Umsetzung der Werte erstellt werden.

Die Qualitätskriterien wurden in XP und Scrum durch Beziehungen von Rollen, Praktiken und Artefakte nachgewiesen (Qualitätskriterien -> Rolle x Praktik x Artefakt). In Kanban gibt es

keine explizite Beschreibung der Rollen, somit können anhand des ASDM keine Qualitätskriterien für Kanban nachgewiesen werden.

# 5 Vergleich agiler Vorgehensmodelle

## 5.1 Vergleich von XP, Scrum und Kanban

Im Kapitel 4 wurden die Vorgehensmodelle im Rahmen des ASDM untersucht. Soweit wie möglich wurden die Basiskonzepte identifiziert. Dabei konnten nicht für alle Vorgehensmodelle alle Basiskonzepte zugeordnet werden.

Es ergibt sich folgende Übersicht:

| | Werte | Prinzipien | Praktiken | Prozesse | Rollen | Artefakte |
|---|---|---|---|---|---|---|
| **XP** | X | X | X | X | X | X |
| **Scrum** | X | X | X | X | X | X |
| **Kanban** | X | | X | X | | X |

Folgende Basiskonzepte konnte identifiziert werden:

| Werte | | |
|---|---|---|
| **XP** | **Scrum** | **Kanban** |
| Kommunikation | Verpflichtung (engl. Commitment) | Transparancy |
| Einfachheit | Fokussierung (engl. Focus) | Balance |
| Rückkopplung | Offenheit (engl. Openness) | Collaboration |
| Mut | Respekt (engl. Respect) | Customer focus |
| Respekt | Mut (engl. Curage) | Flow |
| | Einfachheit (engl. Simplicity) | Leadership |
| | Kommunikation | Understanding |
| | Feedback | Agreement |
| | | Respect |

| Prinzipien | | |
|---|---|---|
| **XP** | **Scrum** | **Kanban** |
| Menschlichkeit | Den Kunden zufrieden stellen | |
| Wirtschaftlichkeit | Änderungen willkommen heißen | |
| Gegenseitiger Vorteil | Häufige Auslieferung | |
| Selbstähnlichkeit | Crossfunktionale Zusammenarbeit | |
| Verbesserung | Unterstützung leisten und Vertrauen schenken | |
| Mannigfaltigkeit | Direkte persönliche Kommunikation | |
| Reflexion | Funktionierende Software | |
| Fluss | Nachhaltige Geschwindigkeit | |
| Gelegenheit | Streben nach technischer Exzellenz | |
| Redundanz | Einfach ist besser | |
| Fehlschlag | Selbstorganisiert agieren | |
| Qualität | Überprüfen und Anpassen | |
| Babyschritte | | |
| Akzeptierte Verantwortlichkeit | | |

| Praktiken | | |
|---|---|---|
| **XP** | **Scrum** | **Kanban** |
| Räumlich zusammen sitzen | Sprint-Planing | Visualisierung des Arbeitsflusses |
| Komplettes Team | Daily-Scrum | Limitierung des Work-in-Progress (WIP) |
| Informative Arbeitsumgebung | Review | Steuerung und Messung des Arbeitsflusses |
| Energiegeladene Arbeit | Retrospektive | Prozesse explizit machen |
| Programmieren als Paar | Backlog Grooming | Verbesserungen durch bewährte Modelle |
| Geschichten | Estimation | |
| Wochenzyklus | | |
| Quartalszyklus | | |
| Freiraum | | |
| Zehn-Minuten-Build | | |
| Kontinuierliche Integration | | |
| Testgetriebene Entwicklung | | |
| Inkrementeller Entwurf | | |

| Prozesse | | |
|---|---|---|
| **XP** | **Scrum** | **Kanban** |
| Iteration | Sprint | Evolutionäres Change Management |
| Release | Releaseplanung | |
| Pair-Programming | | |
| Continuous-Integration | | |
| User-Stories und Tasks | | |

| Rollen | | |
|---|---|---|
| **XP** | **Scrum** | **Kanban** |
| Kunde | Product-Owner | |
| Entwickler | Scrum-Master | |
| Produktmanager | Team-Member | |
| Produktbesitzer | | |
| Benutzer | | |
| XP-Coach | | |

| Artefakte | | |
|---|---|---|
| **XP** | **Scrum** | **Kanban** |
| Release-Artefakt | Release-Artefakt | Release-Artefakt |
| Lokaler-Source-Code | Lokaler-Source-Code | CI-Infrastruktur |
| CI-Infrastruktur | CI-Infrastruktur | Live-Infrastruktur |
| Test-Infrastruktur | Test-Infrastruktur | User-Stories und Tasks |
| Live-Infrastruktur | Live-Infrastruktur | Product-Backlog |
| User-Stories und Tasks | User-Stories und Tasks | Kanban-Board |
| | Product-Backlog | Releaseplan |
| | Sprint-Backlog | |
| | Impediment-Backlog | |
| | Scrum-Board | |
| | Burndown-Chart | |
| | Definition of Done | |
| | Releaseplan | |

Aus den identifizierten Basiskonzepten konnten zum Teil die Werte der Vorgehensmodelle durch Beziehungen nachgewiesen werden.

Die Überprüfung erfolgt nach folgenden Beziehungen:

***Wert -> Praktik x Prinzip***

Folgende Werte konnten durch Beziehungen nachgewiesen werden:

| Nachweis der Werte durch Beziehungen | | |
|---|---|---|
| **XP** | **Scrum** | **Kanban** |
| Kommunikation -> Räumlich zusammen sitzen x Reflektion | Verpflichtung -> Sprint-Planing x Den Kunden zufrieden stellen | |
| Einfachheit -> Inkrementeller Entwurf x Babyschritte | Fokussierung -> Backlog-Grooming x Nachhaltige Geschwindigkeit | |
| Rückkopplung -> Informative Arbeitsumgebung x Fluss | Offenheit -> Review x Direkte persönliche Kommunikation | |
| Mut -> Energiegeladene Arbeit x Fehlschlag | Respekt -> Estimation x Unterstützung leisten und Vertrauen schenken | |
| Respekt -> Freiraum x Menschlichkeit | Mut -> Retrospektive x Änderungen willkommen heißen | |
| | Einfachheit -> Backlog-Grooming x Einfach ist besser | |
| | Kommunikation -> Daily-Scrum x Direkte persönliche Kommunikation | |
| | Feedback -> Retrospektive x Überprüfen und Anpassen | |

Aus den identifizierten Basiskonzepten konnten zum Teil die Qualitätskriterien aus Kapitel 3.1 anhand von Beziehungen nachgewiesen werden.

Die Überprüfung erfolgt nach folgenden Beziehungen:

***Qualitätskriterien -> Rolle x Praktik x Artefakt***

| Nachweis von Qualitätskriterien durch Beziehungen für XP | |
|---|---|
| **Funktionalität** | Entwickler x Kontinuierliche Integration x CI-Infrastruktur<br>Kunde x Testgetriebene Entwicklung x Release-Artefakt<br>Produktbesitzer x 10-Minuten-Build x User-Stories und Tasks |
| **Zuverlässigkei** | Entwickler x Kontinuierliche Integration x CI-Infrastruktur<br>Entwickler x Pair-Programming x Lokaler-Source-Code |
| **Benutzbarkeit** | Produktbesitzer x Freiraum x Test-Infrastruktur<br>Kunden x Komplettes Team x User-Stories und Tasks |
| **Effizienz** | Entwickler x Informative Arbeitsumgebung x Test-Infrastruktur |
| **Änderbarkei** | Kunde x Wochenzyklus x Release-Artefakt |

| Nachweis von Qualitätskriterien durch Beziehungen für Scrum | |
|---|---|
| **Funktionalität** | Produkt-Owner x Sprint-Planing x User-Story<br>Team-Member x Backlog-Grooming x Sprint-Backlog |
| **Zuverlässigkeit** | Team-Member x Review x CI-Infrastruktur |
| **Benutzbarkeit** | Produkt-Owner x Sprint-Planing x Releaseplan |
| **Effizienz** | Team-Member x Sprint-Planing x User-Stories und Tasks |
| **Änderbarkeit** | Produkt-Owner x Retrospektive x Test-Infrastruktur |

| Nachweis von Qualitätskriterien durch Beziehungen für Kanban | |
|---|---|
| **Funktionalität** | Kein Nachweis möglich |
| **Zuverlässigkei** | Kein Nachweis möglich |
| **Benutzbarkeit** | Kein Nachweis möglich |
| **Effizienz** | Kein Nachweis möglich |
| **Änderbarkei** | Kein Nachweis möglich |

## 5.2 Auswertung

Für Kanban konnte keine Nachweise der Qualitätskriterien erbracht werden. Deswegen wird hier Kanban nicht berücksichtigt.

Im direkten Vergleich von XP und Scrum konnten für XP mehr Nachweise für Beziehungen der Qualitätskriterien erbracht werden:

| | XP | Scrum |
|---|---|---|
| Funktionalität | 3 | 2 |
| Zuverlässigkeit | 2 | 1 |
| Benutzbarkeit | 2 | 1 |
| Effizienz | 1 | 1 |
| Änderbarkeit | 1 | 1 |
| **Summen** | **9** | **6** |

Aus dieser Betrachtung muss XP gegenüber Scrum bei der Berücksichtigung von Qualitätskriterien als vorteilhafter eingeschätzt werden.

**Gründe:**

Bei den untersuchten Beziehungen stehen die Praktiken im Mittelpunkt. Diese entscheiden ob eine Handlung ausgeführt wird oder nicht. In diesem Basiskonzept hat XP eindeutig einen Vorteil. XP stellt 13 und Scrum 6 Praktiken zur Verfügung. Aus diesem Grund können viel eher Handlungen abgeleitet werden, die die Qualitätskriterien belegen. Bei der Anzahl der Rollen bietet XP auch eine höhere Anzahl. Diese ist aber gegenüber den Praktiken nicht so entscheidend. Einzig bei der Anzahl der Artefakte hat Scrum eine größere Auswahl. Eine Vielzahl der Artefakte beschäftigen sich allerdings mit dem Thema Impediment und aktueller Fortschritt im Sprint. Die Anzahl der Artefakte, die sich mit der tatsächlichen Software beschäftigen, sind bei Scrum und XP nahezu identisch.

**Fazit:**

Führt man sich noch einmal den historischen Hintergrund vor Augen, erscheint es durchaus logisch, dass XP besser Beziehungen abbildet, die die Qualitätskriterien erfüllen. Der Kern von

XP ist das Pair-Programming, also die Idee, dass zwei Entwickler zusammen an einem Stück Software arbeiten. Dadurch sollten Fehler vermieden werden und möglichst unterschiedliche Aspekte in die entworfene Lösung eingearbeitet werden. Eine solche Arbeitsweise fördert grundsätzlich Softwarequalität.

Scrum wurde als Agiles Vorgehensmodell entwickelt, mit dem vor allem Business-Prozesse gut abgebildet werden. Dadurch sollte es gerade großen Firmen erleichtert werden, die Vorteile von agiler Entwicklung in die vorhandenen Prozesse zu integrieren. Nach diesen Anforderungen findet sich vor allem eine Vielzahl von Praktiken rund um die Rolle Scrum-Master. Viele dieser Praktiken sollen den Arbeitsfluss optimieren und den aktuellen Arbeitsstand für Entscheider aus der Business-Ebene transparent machen. Diese Punkte erleichtern womöglich die tägliche Arbeit, eine besondere Steigerung der Softwarequalität enthalten sie nicht.

# 6 Zusammenfassung

In dieser Arbeit konnte das ASDM erfolgreich zur formalen Beschreibung von agilen Vorgehensmethoden eingesetzt werden. Die Untergliederung in Basiskonzepte und Beziehungen hat sich für XP und Scrum als gutes Mittel erwiesen, um Handlungen nachzuweisen welche Softwarequalität fördern.

In der Praxis kann diese Arbeit verwendet werden, um klare Praktiken zu identifizieren, die nötig sind, um Softwarequalität im Entwicklungsprozess zu steigern. Dies kann als echter Mehrwert betrachtet werden.

Für Kanban hat sich das ASDM als unzureichend erwiesen. Alleine durch das Fehlen von festen Rollen konnte das ASDM nicht mehr eingesetzt werden. Dabei ist die Tatsache, dass ein Vorgehensmodell keine festen Rollen definiert, kein Beleg für mangelnde Softwarequalität. Kanban bietet mit dem Prozess des Evolutionären Change Management ein sehr wirkungsvolles Mittel, um permanent Verbesserungen zu realisieren. Bei einem richtigen Einsatz kann dies die Softwarequalität entscheidend verbessern.

Die große Stärke im ASDM liegt in den Beziehungen. Diese erlauben es sehr abstrakte Beschreibungen in einen Kontext zu setzten. Damit diese Beziehungen aufgestellt werden können, verlangt das ASDM die beschriebenen Basiskonzepte. Hier liegt die Schwachstelle des ASDM.

In der aktuellen Praxis ist es üblich in so genannten Crossfunktionalen-Teams zu arbeiten. Das bedeutet, dass alle Teammitglieder unterschiedliche Aufgaben wahrnehmen. Zwar haben einzelne Personen immer noch Schwerpunkte, aber die feste Rollenbeschreibung gehört zunehmend der Vergangenheit an. Für das Ergebnis und die tägliche Arbeit ist es letztlich auch nicht von Belangen, wer eine Praktik durchführt. Entscheidend ist, dass es feste Praktiken gibt und die Durchführung sichergestellt ist.

In der Konsequenz wäre eine Modernisierung des Basiskonzeptes Rolle denkbar. Diese könnte in Zukunft als Zuständigkeit verstanden werden. Damit wäre stärker eine zeitbezogene Verantwortung von einem Mitglied des Entwicklungsteams für eine Aufgabe gemeint. Diese Person könnte wechseln ohne das eine klare Zuständigkeit verloren ginge.

# 7 Anhang

## 7.1 Literatur

[1] **Agile Muster und Methoden** von Manfred Steyer, 2010 erschienen bei entwickler.press

[2] **The Agile Samurai** von Jonathan Rasmusson, 2010 erschienen bei Pragmatic Bookshelf

[3] **Kanban from the Inside** von Mike Burrows, 2014 erschienen bei Blue Hole Press

[4] **Basiswissen Softwaretest** von Andreas Spiller und Tilo Linz, 2005 erschienen bei dpunkt.verlag

[5] **Agile Testing** von Manfred Baumgartner, Marting Klonk, Helmut Pichler, Richard Seidl und Siegfried Tanczos, 2013 erschienen bei Carl Hanser Verlag München

[6] **Auf dem Weg zu einem allgemeinen Agilen Software Entwicklungs- und Vorgehensmodell (ASDM**) von A. Janus & R. Dumke, Merikon 2011

[7] **Dissertation von André Janus**, Konzepte für Agile Qualitätssicherung und –bewertung in Wartungs- und Weiterentwicklungs-Projekten, Otto-von-Guerick-Universität Magdeburg, 2012

[8] **Extreme Programming Explained**, von Kent Beck und Cynthia Andres, 2004 erschienen bei Addison-Wesley

[9] **Agile Softwareentwicklung**, von Wolf-Gideon Bleek und Henning Wolf, 2008 erschienen bei dpunkt.verlag

[10] **Scrum in der Praxis**, von Sven Röpstorff und Robert Wiechmann, 2012 erschienen bei dpunkt.verlag

[11] **Kanban in der IT**, von Klaus Leopold und Siegfried Kaltenecker, 2012 erschienen bei Carl Hanser Verlag

## 7.2 Abbildungsverzeichnis

### Abbildung 3: Scrum Prozess

http://www.agile-living.de/AgileMethods.aspx

### Abbildung 4: Kanban Board

http://www.it-agile.de/wissen/methoden/kanban/

### Abbildung 5: ASDM

Dissertation von André Janus, Konzepte für Agile Qualitätssicherung und –bewertung in Wartungs- und Weiterentwicklungs-Projekten, Otto-von-Guerick-Universität Magdeburg, 2012

### Abbildung 6: Basiskonzepte des ASDM-1

Selbst erstellte Mind Map nach Anregungen aus der Dissertation von André Janus

### Abbildung 7: Werte. Prinzipien und Praktiken

https://msdn.microsoft.com/en-us/library/hh273032(v=vs.88).aspx

### Abbildung 8: explizite und implizite Basiskonzepte des ASDM-2

Dissertation von André Janus, Konzepte für Agile Qualitätssicherung und –bewertung in Wartungs- und Weiterentwicklungs-Projekten, Otto-von-Guerick-Universität Magdeburg, 2012

### Abbildung 9: implizite Beschreibung des Basiskonzeptes Prozess

Dissertation von André Janus, Konzepte für Agile Qualitätssicherung und –bewertung in Wartungs- und Weiterentwicklungs-Projekten, Otto-von-Guerick-Universität Magdeburg, 2012

### Abbildung 10: abstraktes Prinzip wird durch die Beziehung zweier Praktiken beschrieben

Dissertation von André Janus, Konzepte für Agile Qualitätssicherung und –bewertung in Wartungs- und Weiterentwicklungs-Projekten, Otto-von-Guerick-Universität Magdeburg, 2012

### Abbildung 11: Übersicht Primär- und Folgepraktiken von XP

Aus Agile Softwareentwicklung, von Wolf-Gideon Bleek und Henning Wolf, 2008 erschienen bei dpunkt.verlag, Seite 144

# Versicherung über Selbstständigkeit

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

*Hamburg, den* _____