

# **Vergleich der drei agilen Softwareentwicklungsprozesse Crystal, Scrum und Kanban**

STUDIENARBEIT

für die Prüfung zum

Bachelor of Engineering/Bachelor of Science

des Studiengangs Informatik  
Studienrichtung Angewandte Informatik

an der

Dualen Hochschule Baden-Württemberg Karlsruhe

von

Vorname Nachname

Abgabedatum

Matrikelnummer    Matrikelnummer

Kurs    Kursbezeichnung

Ausbildungsfirma    Firmenname,    Stadt

Betreuer [der Ausbildungsfirma]    Titel Vorname Nachname

[Gutachter der Studienakademie    Titel Vorname Nachname]

**Erklärung**

gemäß § 5 (3) der „Studien- und Prüfungsordnung DHBW Technik“ vom 22. September 2011.

Ich habe die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

---

Ort, Datum

Unterschrift

## **Zusammenfassung**

---

## **Abstract**

---

## Inhaltsverzeichnis

---

Zusammenfassung .....	3
Abstract .....	3
Inhaltsverzeichnis .....	4
Abbildungsverzeichnis.....	5
Tabellenverzeichnis.....	5
Abkürzungsverzeichnis.....	5
1. Einleitung .....	6
2. Agile Softwareentwicklung .....	8
2.1 Allgemein.....	8
2.1.1 Einführung.....	8
2.1.2 Das agile Manifest .....	10
2.1.3 Agile Prinzipien/Methoden .....	11
2.2 Crystal Clear .....	13
2.3 Scrum .....	16
2.4 Kanban .....	20
4. Vergleichskriterien für die Methoden.....	26
5. Wahl der Tools .....	33
6. Wahl des Projektes .....	44
7. Vorbereitung der Durchführung.....	49
8. Durchführung des Projektes.....	52
8.1 Crystal Clear .....	52
8.2 Scrum .....	52
8.3 Kanban .....	52
9. Ergebnisse .....	53
9.1 Analyse/Ursachenforschung.....	53
9.1.1 Crystal Clear .....	53
9.1.2 Scrum.....	53
9.1.3 Kanban.....	53
9.2 Auswertung der Tools.....	53
9.3 Gegenüberstellung der agilen Prozesse.....	53
10. Fazit .....	54

## **Abbildungverzeichnis**

---

Es konnten keine Einträge für ein Abbildungsverzeichnis gefunden werden.

## **Tabellenverzeichnis**

---

Es konnten keine Einträge für ein Abbildungsverzeichnis gefunden werden.

## **Abkürzungsverzeichnis**

---

## 1. Einleitung

In der Softwareentwicklung setzt man seit einigen Jahren immer mehr auf agile Entwicklungsprozesse. Diese versprechen durch vermehrten Kontakt mit dem Kunden und hohe Anpassungsfähigkeit an Produktänderungen ein schnelleres lieferbares Ergebnis. Der große Nutzen gegenüber linearen Prozessen wie Wasserfall und RUP lässt sich nicht mehr bestreiten. Agil ist jedoch ein sehr allgemeiner Begriff und so kann man nicht generell von dem agilen Prozess sprechen. Angefangen mit dem Extrembeispiel „Extreme Programming“ über die sehr häufig genutzte Variante Scrum, wurden durch diese agile Bewegung bis heute viele Prozesse entwickelt, die sich oft nur in wenigen Eigenschaften unterscheiden.

Gerade durch die Prozessvielfalt fällt es oft schwer eine Entscheidung zu treffen, welcher Prozess am besten zum Projekt passt. Aus diesem Grund ist es Ziel dieser Arbeit verschiedene agile Prozesse zu betrachten, auszuwerten und zu vergleichen. Es soll herausgefunden werden, in wie weit sich die einzelnen Prozesse tatsächlich unterscheiden und in welchen Fällen man einen bestimmten Prozess vorziehen sollte.

Hierzu wurden drei agile Prozesse ausgewählt:

- Scrum
- Crystal Clear
- Kanban

Als eine sehr verbreitete und beliebte Variante darf Scrum in diesem Vergleich nicht fehlen, vor allem weil es auch oft an den Hochschulen als „die“ agile Methode vorgestellt wird. Außerdem besteht unter den Teammitgliedern bereits Erfahrung in der Durchführung von Scrum, da es in einigen Unternehmen angewandt wird.

Daneben wurde ein eher unbekannter von Alistair Cockburn entworfener Prozess namens Crystal Clear gewählt, da er sich speziell für kleine Teams, wie es bei dieser Arbeit der Fall ist, eignet und eine abgeschwächte Form von XP sein soll (Cockburn, 2005). Darüber hinaus hat man sich für den Entwicklungsprozess Kanban entschieden, der eigentlich aus der Produktion stammt und speziell für sein Kanban-Board bekannt ist. Daher wäre es interessant den eigentlichen Prozess hinter dem Namen kennen zu lernen.

Die Arbeit wurde nach einem bestimmten Schema gegliedert, welches das zeitliche Vorgehen während der Durchführung widerspiegelt. Bevor auf den tatsächlichen Vergleich eingegangen wird, werden im ersten Teil die agile Softwareentwicklung und die drei genannten Prozesse vorgestellt, um einen ersten Einblick in das Thema zu vermitteln. In den darauf folgenden Kapiteln werden einige Vorüberlegungen beschrieben. Dazu gehört einerseits die Festlegung der Vergleichskriterien, welche bereits eine erste Gegenüberstellung auf Basis der Recherchen liefern soll.

Andererseits wird im Anschluss darauf ein Beispielprojekt definiert, auf welches die Prozesse angewandt werden sollen, um dadurch selbst erhobene Vergleichsdaten führen zu können. Ein weiterer Teil dieser Arbeit ist es die Verwendbarkeit von bestimmten Projektmanagementtools in Bezug auf die drei Prozesse zu testen. Dazu werden aus einem Pool von Tools nach einer ausführlichen Bewertung zwei ausgewählt und während des Beispielprojekts angewandt. Die Durchführung des Beispielprojekts mit drei Prozessen erweist sich jedoch als durchaus schwierig, aus diesem Grund wird hierzu eine Strategie entworfen und vorgestellt.

Das darauf folgende Kapitel beschreibt die Durchführung der Prozesse und soll speziell die Vorgehensweise und die selbst gewonnenen Erfahrungen dokumentieren. Es beinhaltet wie die Prozesse umgesetzt wurden und zeigt einige Ergebnisse aus Meetings, Reflexionen und Dokumentation. Hierbei soll der Fokus nicht auf dem zu entwickelnden Produkt sondern auf dem eigentlichen Vorgehen liegen. Danach folgt die eigentliche Analyse der Prozesse. Dabei wird für jeden Prozess auf die Frage eingegangen, was funktioniert oder nicht funktioniert hat und was wohl die Ursachen dafür waren. Außerdem werden die angewandten Tools ausgewertet und anschließend eine Gegenüberstellung der drei agilen Prozesse auf Basis der gewonnenen Erfahrungen aufgestellt. Hierbei werden die Punkte aus dem Vorabvergleich vom Anfang aufgegriffen und mit den selbst erhobenen Informationen verglichen.

Zum Schluss dieser Arbeit sollen auch mögliche Kombinationen der Entwicklungsprozesse abgewägt und Empfehlungen für bestimmte Projektgegebenheiten gegeben werden.

## **2. Agile Softwareentwicklung**

### **2.1 Allgemein**

#### **2.1.1 Einführung**

Agil stammt vom Lateinischen Wort agilis und bedeutet soviel wie „von großer Beweglichkeit zeugend; regsam und wendig“ (1). Mit Softwareentwicklung ist die „Verbesserung vorhandener oder Erarbeitung neuer Software“ (1) gemeint.

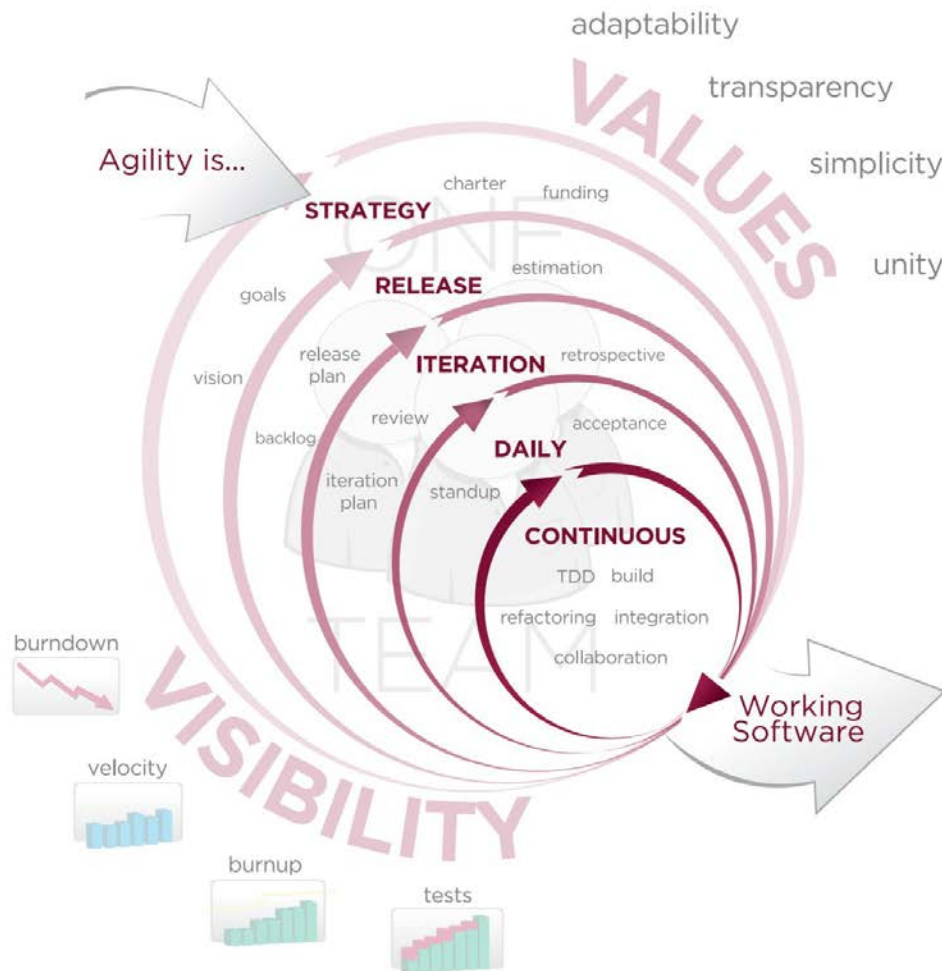
In Kombination wird der Begriff der agilen Softwareentwicklung meist als Gegensatz zur traditionellen Softwareentwicklung verwendet und bezieht sich im Allgemeinen auf den gesamten Entwicklungs- und Managementprozess. Hierbei wird oft hauptsächlich ein Vergleich mit dem relativ starren Wasserfallmodell vorgenommen, welches 1970 im Artikel „Managing the Development of Large Software Systems“(2) von Dr. Winston Royce erstmalig formell beschrieben wurde. Dabei erklärt Royce bereits, dass lineares Arbeiten für Softwareentwicklung ungeeignet ist. Stattdessen empfiehlt er einen iterativen Prozess, der heute in verschiedensten Ausführungen in allen Beispielen für Agile Softwareentwicklung zu finden ist.

Das Abheben von alten, starren Modellen ist jedoch nicht das Hauptziel der Agilen Entwicklung. Prinzipiell soll der gesamte Prozess flexibler gestaltet werden, um die Probleme aus klassischen Modellen zu verringern oder ganz zu vermeiden. Zu diesen Problemen zählen beispielsweise die Überdokumentation, die fehlende Reaktionsfähigkeit gegenüber sich ändernden Anforderungen und Ressourcen, gesetzlichen Rahmenbedingungen oder spät erkennbaren Risiken. Weitere Probleme sind große Fehler bei Zeitschätzungen am Anfang des Projekts sowie mangelnde Kommunikation und Wissens- und Erfahrungsaustausch zwischen festgelegten Rollen innerhalb des Projekts.

Abbildung 1 gibt einen Überblick über die verschiedenen Werte wie Transparenz oder Anpassungsfähigkeit, aber auch unterschiedliche Methoden und Hilfsmittel der Agilen Softwareentwicklung. Das Hauptziel der schnelleren Lieferung von „Working Software“ ist ebenso klar erkennbar wie die kontinuierliche Evaluation des Prozesses und der Ergebnisse.



# AGILE DEVELOPMENT



## ACCELERATE DELIVERY

Abbildung 1

### 2.1.2 Das agile Manifest

Im Februar 2001 haben 17 Softwareentwickler die Richtlinien der agilen Softwareentwicklung im agilen Manifest formuliert, welches bis heute Gültigkeit besitzt. Dadurch sollte eine kleine Revolution in der Softwareentwicklung angestoßen werden.

**„Individuen und Interaktionen** mehr als Prozesse und Werkzeuge

**Funktionierende Software** mehr als umfassende Dokumentation

**Zusammenarbeit mit dem Kunden** mehr als Vertragsverhandlung

**Reagieren auf Veränderung** mehr als das Befolgen eines Plans“<sup>1</sup>

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

Individuen und Interaktionen stehen für das selbstbestimmt Arbeiten an einem Projekt und die persönliche Kommunikation, welche oft falsche Verständigung minimiert.

Funktionierende Software hat beim agilen Prozess eine deutlich höhere Priorität als eine umfassende Dokumentation, denn das stellt den Kunden vorrangig zufrieden. Die enge Zusammenarbeit mit dem Kunden ermöglicht eine schnelle Rückfrage bei Unklarheiten und hilft Fehler frühzeitig zu erkennen und zu vermeiden.

Das Reagieren auf Veränderungen bezieht sich unter anderem auf eine Anpassung der Anforderungen während der Entwicklung und auf das Anpassen der Arbeitsweise.

Zusätzlich zum agilen Manifest wurden 12 Prinzipien zur agilen Softwareentwicklung aufgestellt. „Sie erklären detaillierter die Werte und Prinzipien der "Agilisten".

---

<sup>1</sup> Quelle: <http://www.heise.de/developer/meldung/10-Jahre-Agiles-Manifest-zur-Geburt-agiler-Softwareentwicklung-1188299.html>

### 2.1.3 Agile Prinzipien/Methoden

Neben den abstrakten Werten kann man die Prinzipien eher als die „gelebten“ Werte bezeichnen (Lundak, 2009). Während die Werte sehr stark im Unternehmen verankert sein müssen, werden die Prinzipien am besten von der Basis erarbeitet, um somit für diejenigen, die sie ausführen, eine höhere Akzeptanz zu schaffen.

In dem agilen Manifest haben sich die Autoren auf insgesamt zwölf Prinzipien geeinigt. Die höchste Priorität haben die regelmäßigen Lieferungen von hochwertiger Software an den Kunden, wobei die Lieferungszyklen so kurz wie möglich sein sollten. Dabei wird der ausgelieferte funktionierende Code als Fortschrittsmaßstab verwendet (Cockburn, 2003). Agil bedeutet außerdem, dass mögliche Änderungen an den Anforderungen positiv aufgenommen werden, um so durch die Einbindung von neuen Technologien und Geschäftsprozessen ein Wettbewerbsvorteil zu erlangen. Nur ein hochqualitatives und ständig überprüftes Design lässt sich problemlos an Änderungen anpassen. Nicht nur die Anforderungen können sich ändern, sondern auch die Arbeitsgewohnheit des Teams: mit Hilfe von regelmäßigen Reflexionen soll die Arbeitsweise Schritt für Schritt geprüft und verbessert werden, bis das Team am effektivsten arbeitet. Darüber hinaus wird eine enge Zusammenarbeit zwischen Entwicklern und Anwendern gefordert, um schnelles Feedback über die Funktionsweise der Software zu erhalten. Am effektivsten ist ein Projektteam, wenn es sich aus motivierten Mitarbeitern zusammensetzt, die sich selbst organisieren können und von außen gefördert werden. Das Unternehmen kann sich so auf gute Arbeit verlassen und muss nicht in das Geschehen durch festgelegte Methoden eingreifen. Im agilen Manifest wird auch gerade die Kommunikation als treibende Kraft für effektive Zusammenarbeit vorgestellt. Ein weiteres essenzielles Prinzip ist die Einfachheit – „die Kunst, die Menge nicht getaner Arbeit zu maximieren“ (Beck, et al., 2001). Das bedeutet, dass zum Beispiel durch weniger Arbeit auf Grund von einfachen und klaren Prinzipien trotzdem höhere Leistung erzielt werden kann.

Neben den vom agilen Manifest definierten Prinzipien empfiehlt es sich jedoch auch eigene zu erarbeiten, damit sich die Mitarbeiter besser damit identifizieren können (Lundak, 2009).

Zur Unterstützung und Umsetzung der Prinzipien wurden einige Praktiken und Techniken entworfen, sogenannte „Best Practices“. Zu den bekanntesten gehören Pair Programming, Reflexion, Refactoring, Test-Driven Development, kontinuierliche Code-Integration und kontinuierliche Tests. Um agil zu sein müssen nicht alle diese Praktiken ausgeführt werden, jedoch helfen sie dabei die agilen Werte im Projekt zu leben.

Aus diesem Grund gibt es viele verschiedene agile Prozesse, die auf unterschiedliche Weise versuchen die Prinzipien in den Projekten durchzuführen. Es gibt also nicht nur einen agilen Prozess, sondern viele verwandte Modelle, die sich von Projekt zu Projekt unterscheiden können.

Im Folgenden soll explizit auf die ausgewählten Prozesse Crystal Clear, Scrum und Kanban eingegangen und die wichtigsten Merkmale dargestellt werden.

## 2.2 Crystal Clear

### Die Crystal Familie

Jedes Projekt ist unterschiedlich und benötigt andere Methoden, um erfolgreich abgeschlossen zu werden. Aus diesem Grund hat Alistair Cockburn, einer der Urheber des agilen Manifests, eine Methodikfamilie entworfen namens Crystal. Sie enthält viele verschiedene Methodiken für unterschiedliche Projektarten, doch alle diese Methodiken haben einen „gemeinsamen genetischen Code“ (Cockburn, 2005). Mit Hilfe des Codes können Unternehmen ein neues Familienmitglied erzeugen, welches an die Bedürfnisse ihrer Projekte angepasst ist.

Die einzelnen Methodiken der Familie werden durch Teamgröße und Kritikalität charakterisiert, welche mit Hilfe von Farbe und Härtegrad angegeben werden. Desto dunkler die Farbe ist, umso größer ist das Projektteam. So wird Crystal Clear zum Beispiel für Teams mit ein bis sechs Mitgliedern ausgeführt.

Gemeinsam verfolgen die Crystal Methodiken alle dieselben Ziele: der positive Projektausgang soll sichergestellt werden, eine effiziente Entwicklung wird angestrebt und das Team soll sich mit den Konventionen wohlfühlen (Cockburn, 2005). Darüber hinaus legte Alistair Cockburn für die Crystal Familie fest, dass der Detaillierungsgrad der Dokumentation von den Projektgegebenheiten abhängt und nicht für jedes Projekt gleich sein muss. Als Ausgleich legt Crystal aber sehr viel Wert auf kurze und ergiebige Kommunikationspfade und regelmäßige Abstimmungen der Arbeitsgewohnheiten, um die Zusammenarbeit flexibel verbessern zu können.

Gute und effiziente Kommunikation ist eines der wichtigsten Prinzipien, aus diesem Grund gibt es speziell für kleine Teams ein auf osmotische (enge) Kommunikation spezialisiertes Familienmitglied namens Crystal Clear. Alistair Cockburn stellt bei der Definition der Methodik klar, dass Crystal Clear „nicht vollständig festgeschrieben“ ist, da sich auch alle Projekte unterscheiden (Cockburn, 2005). Die Methodik soll während eines Projektes Schritt für Schritt an das Projekt und das Team angepasst werden. Deshalb möchte er das Team nicht durch festgeschriebene Techniken und Methoden einengen, sondern versucht eher Empfehlungen zu geben. Alistair Cockburn schreibt, dass Crystal Clear ein „einfacher und toleranter Regelsatz sein soll, der das Projekt in sicheres Fahrwasser bringt“ (Cockburn, 2005).

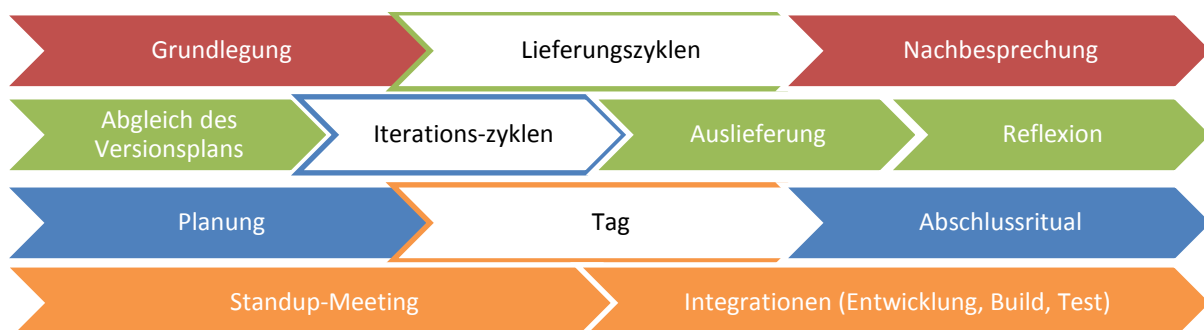
## Eigenschaften

Bei den Befragungen der besten Teams haben sich sieben Eigenschaften herausgestellt, die ein Projekt zu einem erfolgreichen Abschluss bringen. Crystal Clear erfordert die ersten drei Eigenschaften. Ein Crystal Clear Team kann jedoch noch mehr der empfohlenen Eigenschaften einhalten, „um weiter in den sicheren Bereich zu gelangen“ (Cockburn, 2005).

Zu den drei essenziellen Eigenschaften zählen regelmäßige Lieferungen, verdichtete (osmotische) Kommunikation und reflektierte Verbesserungen. Vor allem die osmotische Kommunikation wird bei Crystal Clear in den Vordergrund gestellt, weil die kleinen Teams auf engem Raum arbeiten können und somit eine viel effektivere Kommunikation stattfindet. Durch den großen Grad an Nähe können viele Informationen auch im Hintergrund aufgenommen werden und auch das Feedback erhöht sich, da der Kommunikation keine Hindernisse im Weg stehen, wie zum Beispiel über den Flur laufen zum nächsten Büro, um dann festzustellen, dass die gefragte Person nicht am Arbeitsplatz sitzt. Ebenso wie die Kommunikation im Team, muss jedoch auch die Kommunikation zum Kunden oder Endanwender durch regelmäßige Lieferungen gehalten werden. Dadurch wird sichergestellt, dass beide Parteien Feedback erhalten und das gewünschte Produkt entwickelt wird. Um die Effizienz zu steigern, müssen auch regelmäßig die Arbeitsweisen reflektiert und angepasst werden. Da Crystal Clear keine bestimmten Strategien und Techniken vorschreibt, sondern viele optional zur Verfügung stellt, liegt es an dem Team, die für sie geeignete Arbeitsweise durch Experimentieren herauszufinden.

## Prozesse

„Crystal Clear verwendet geschachtelte zyklische Prozesse“ (Cockburn, 2005) und jedes Projekt besteht aus den folgenden Zyklen:



Damit Crystal Clear eingehalten wird, müssen mindestens zwei Lieferungszyklen mit tatsächlichen Lieferungen an den Kunden ausgeführt werden.

### Die Rollen

Ein Crystal Clear Projekt definiert vier obligatorische Rollen und weitere vier zusätzliche Rollen (Cockburn, 2005).

Zum einen gibt es den *Auftraggeber*, der für die finanziellen Mittel des Projektes zuständig ist. Er weist dem Team die Richtung, indem er die Arbeitseinheiten vor jeder Iteration priorisiert und dabei Änderungen in den Geschäftsprozessen einfließen lässt.

Zum anderen benötigt das Team einen *erfahrenen Anwender*, der sich mit den Vorgängen und dem eingesetzten System auskennt und für regelmäßige Rücksprachen zur Verfügung stehen sollte. Er ist derjenige, der die ausgelieferte Software ausprobiert und den Entwicklern Feedback gibt.

Darüber hinaus wird ein *Chefdesigner* definiert, der vom fähigsten Designer ausgeführt wird und technischer Leiter des Projektes ist. Er übernimmt Aufgaben wie Projektmanagement und Förderung der Teammitglieder und fungiert als Bindeglied zwischen Auftraggeber und Projektteam.

Bei Crystal Clear wird die Rolle des *Designers* und des *Programmierers* kombiniert, da die Programmierung immer ein Design voraussetzt und die beiden Rollen somit nicht getrennt ausgeführt werden können.

Zusätzliche Rollen wie *Koordinator*, *Fachexperte*, *Tester* oder *Autoren* können wahlweise hinzugefügt werden, um die anderen Rollen zu entlasten.

### Die Arbeitsergebnisse

Auch bei den Arbeitsergebnissen gilt, dass weder alle erforderlich noch alle optional sind (Cockburn, 2005). Aus diesem Grund sind durch äquivalente Ersetzungen, Variationen und Anpassungen der Arbeitsergebnisse möglich. Gerade bei Crystal Clear kann die Anzahl und Notwendigkeit von Zwischenergebnissen stark reduziert werden, da diese durch interpersonelle Kommunikation, Anmerkungen auf den Whiteboards oder Lieferungen teilweise ersetzt werden. Das schließt die Dokumentation aber nicht vollkommen aus. Alle Projektrelevanten Arbeitsergebnisse müssen in irgendeiner Form dokumentiert werden. Hierbei ist der Formalismus eher nebensächlich, da auch Ergebnisse in Form von Whiteboard-Aufschrieben und Flipcharts angemessen sind. Jedes Dokument wird einer oder mehreren Rollen zugeordnet, damit allen bewusst ist, wer für welches Ergebnis verantwortlich ist.

### 2.3 Scrum

Der Begriff Scrum hat seinen Ursprung beim Rugby, wo dieser Begriff für „Gedränge“ steht.

Beim Scrum als agiles Vorgehensmodell geht es eher um das selbst Organisieren sowie um Eigenverantwortung.

„Scrum ist ein Management-Rahmenwerk zur Entwicklung komplexer Produkte“, welches besonders häufig in der Softwareentwicklung eingesetzt wird. „Das wichtigste an, Scrum ist, das man offen bleibt gegenüber neuen Einsichten oder Ideen, um Prioritäten und auch das Produkt anzupassen.“

Gerade in der Softwareentwicklung stehen zum Anfang der Planung und Umsetzung noch viele Unklarheiten im Raum, welche eine Änderung des Produktes und der Planung auslösen. Zudem ergeben sich während der Umsetzung meist komplexere Funktionen oder Probleme, welche in der Planung nicht berücksichtigt wurden.

Bei den agilen Softwareentwicklungsprozessen muss im Gegensatz zu den klassischen Ansätzen kein Neustart des Projekts oder der Funktion erfolgen, sondern die Anforderungen können vor oder nach einem Sprint geändert werden.

#### Die Vision

Wie bei jedem Projekt steht auch bei Scrum die Vision am Anfang. In der Vision müssen das angestrebte Ergebnis des Projekts, der Grund für die Durchführung, der Nutzen des Projekts, der Einsatzbereich, die Branche, das Budget und der Zeitplan geklärt werden. Die Projektvision ist nicht technisch geprägt. Die Vision wird vom Product Owner in Zusammenarbeit mit dem Kunden erstellt um dem Scrum Master und dem Team ein fest definiertes Ziel zur Verfügung zu stellen

#### Das Release Planning Meeting

Während des Release Planning Meetings wird eine Liste von Anforderungen/ Funktionen erstellt, welche das Projekt enthalten soll. Anschließend wird jeder Anforderung eine Priorität zugeordnet. „Hauptfunktionen, welche die höchste Priorität besitzen und welche meist schon zu Anfang am klarsten formuliert sind, sollten auch am frühesten abgearbeitet werden“. Weiter muss im Release Planning Meeting die Definition von „done“ festgelegt werden, dabei wird geregelt, welche Anforderungen erfüllt sein müssen um den Eintrag im Produkt Backlog als „Erledigt“ zu markieren.



### **Der Product Backlog**

Die Liste von Anforderungen und Funktionen aus der Vision User Story, mit deren Prioritäten, die im Release Planning Meeting festgelegt wurden, wird Produkt Backlog genannt. User Stories sind vage Anforderungen aus Benutzersicht welche einen sichtbaren Mehrwert für die Kunden darstellt. Dabei sollten User Stories diese Eigenschaften aufweisen:

- Independent: unabhängig voneinander
- Negotiable: verhandelbar
- Valuable: Wert für den Kunden
- Estimatable: schätzbar
- Small: klein
- Testable: testbar

### **Der Produkt Owner**

Der Produkt Owner ist das Bindeglied zwischen dem Auftraggeber und der Softwarefirma, er ist unter anderem für die Priorisierung der Funktionen zuständig. „Zudem ist es die Aufgabe des Produkt Owners für Fragen jederzeit zur Verfügung zu stehen und die Interessen der Steakholder zu priorisieren und die dem Entwicklern näher zu bringen“.

Dazu muss der Produkt Owner Kenntnisse darüber besitzen, was:

- „der Kunde benötigt und worin der höchste Wert liegt“
- „der Markt und die Konkurrenz anbieten“
- „die Entwickler an Informationen benötigen“

### **Das Sprint Planning Meeting**

In Ersten Sprint Planning Meeting wird die Softwarearchitektur im Groben festgelegt um Abhängigkeiten bei der Planung zu berücksichtigen. Während des Sprint Planning Meetings werden die User Stories in so genannte Tasks unterteilt, welche in einem Sprint abgearbeitet werden. Dabei muss abgeschätzt werden in welcher Zeit das Team die einzelnen Tasks abarbeiten kann. „Kurze Zyklen mit fester Zeitdauer führen innerhalb kürzester Zeit zur besseren Einschätzung, was in dieser Zeit möglich ist.“ Dadurch ist es einfacher dem Kunden einen ungefähren Zeitplan zu vermitteln.

### **Der Sprint Backlog**

die einzelnen Tasks die innerhalb eines Sprints abgearbeitet werden sollen, werden in den Sprint Backlog eingetragen. Jede Aufgabe innerhalb eines Tasks sollte an einem Tag abzuarbeiten sein. Der Sprint Backlog dient dem Team sowie dem Scrum Master zur Übersicht wer welche Aufgabe gerade bearbeitet oder was evtl. kritisch wird mit der Umsetzung in diesem Sprint

Während eines Sprints sind die Einträge im Sprint Backlog fixiert und es dürfen keine fundamentalen Änderungen an ihnen vorgenommen werden.

### **Der Sprint**

Während eines Sprints, „welcher in der Regel variabel ist, jedoch in der Regel 1-4 Wochen dauert“, organisiert sich das Entwicklerteam selbst. Dabei weißt sich das Team selbst Aufgaben zu und klärt die jeweiligen Verantwortlichkeiten. Während eines Sprints arbeitet das Team komplett eigenverantwortlich. „Am Ende eines Sprints steht eine funktionierende Funktion, welche der Kunde ausführlich Testet“.

### **Das Daily Scrum Meeting**

An jedem Tag des Sprints steht ein Daily Scrum Meeting an, welches einen Zeitrahmen von 15 Minuten nicht überschreiten und jeden Tag zum selben Zeitpunkt stattfinden sollte. „Daily Scrum Meetings finden am besten im Stehen statt, das jeder beteiligte aktiv bleibt“. Während eines Daily Scrums sollte jeder beteiligte am Sprint folgende drei Fragen beantworten um dem Team den momentanen Entwicklungsstand zu erläutern:

- „Was habe ich seit dem Letzten Meeting erreicht?“
- „Was werde ich bis zum nächsten Meeting erreichen?“
- „Was blockiert mich?“

### **Der Scrum Master**

Der Scrum Master achtet auf die Einhaltung des Daily Scrums sowie auf dessen Struktur und regelt den Ablauf. Zudem überwacht er die Einhaltung des Sprint Ziels. Zusätzlich ist der Scrum Master dafür verantwortlich den Product Owner vom Entwicklungsteam während eines Sprints fernzuhalten.

### Das Sprint Burndown Chart

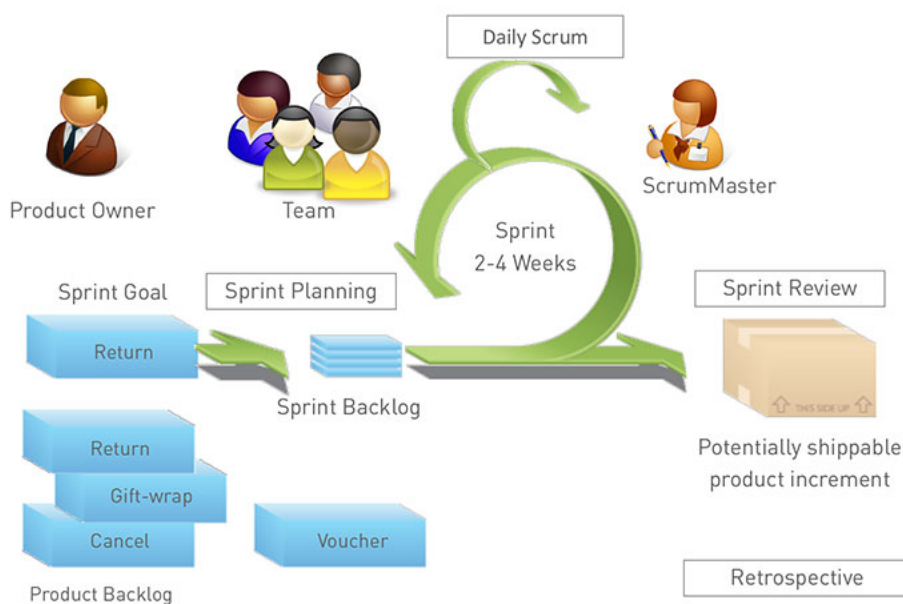
Während eines Sprints werden alle Tasks mit geschätzter Zeit und fertige Tasks mit benötigter Zeit versehen. Diese Zeiten werden dann im Burndown Chart grafisch dargestellt, dies dient dazu, dass das Team den Vorschritt des Sprints verfolgen kann und auch sieht ob die Zeitplanung eingehalten werden kann.

### Der Sprint Review

Am Ende eines Sprints steht der Sprint Review, daran nimmt das Team, der Scrum Master und der Produkt Owner, sowie eventuell der Kunde, zukünftige Anwender und der Geschäftsführer, teil. In diesem werden die umgesetzten User Stories des Sprints präsentiert. Dabei werden schnell Missverständnisse oder evtl. Fehlfunktionen aufgedeckt, welche dann im nächsten Sprint korrigiert werden müssen. Dabei sollte die Präsentation eine Live Demo sein. „Aus ökonomischer Sicht beinhaltet jede Änderung eine Verbesserung und daher sind diese Änderungen gewünscht.“

### Das Retrospektiv Meeting

Es erfolgt im Anschluss des Sprint Reviews, dabei wird der Sprint im Hinblick auf Probleme und Verbesserungen durchleuchtet. Das Meeting findet nur mit dem Team und dem Scrum Master statt, da es dabei um interne Strukturen geht, die nicht direkt mit dem Produkt zu tun haben müssen. Dabei werden Fragen wie: was war schlecht, oder was war gut gestellt. Hier geht es um Verbesserungen welche die Produktivität des Teams steigern, auch im Hinblick auf weitere Sprints und Projekte. Dabei wird eine Liste mit konkreten Verbesserungsmaßnahmen erstellt.



## 2.4 Kanban

„Kan-ban“ ist eine Zusammensetzung der beiden japanischen Worte für Signal und Karte. Diese Signalkarten wurden bei Toyota im Zuge der Umstrukturierung ihres Produktionsprozesses in den 40er Jahren eingesetzt. Dort dienten sie zur Aufforderung an ein später im Produktionsverlauf arbeitendes Team, etwas zu produzieren. Ohne diese Aufforderung fertigte keine Station etwas an, so dass die Menge der überschüssigen Teile gering blieb. Solch ein System wird auch Pull-System genannt. Grund für diese Art von Produktion war der Wunsch nach einer kontinuierlichen Produktionsmenge bei Toyota. So wurde Kanban die Basis für den Prozess der kontinuierlichen Verbesserung, japanisch Kaizen, den Toyota anstrebte. Im Bereich der Agilen Software-Entwicklung wurde das Pull-System kombiniert mit der Theory of Constraints und dem Lean Manufacturing und formte ein neues Verständnis des Begriffs Kanban. Die Theory of Constraints wurde maßgeblich von Eli Goldratt entwickelt und besagt grob, dass die langsamste Stelle, der sogenannte Engpass oder Bottleneck der Produktion, bestimmt, wie groß der Durchsatz des Systems ist. Deshalb muss man diese Engpässe erkennen und wenn möglich eventuell beheben. Dies geschieht iterativ, d.h. man erkennt erst einen Bottleneck, behebt diesen und eröffnet somit gleichzeitig den Weg für einen neuen an anderer Stelle. Während demnach die Theory of Constraints ihren Schwerpunkt auf Flaschenhälse legt, konzentriert sich Lean Manufacturing und das darauf aufbauende Lean Development auf die Verbesserung des Flusses oder Flows. Diese Eigenschaften versuchte David J. Anderson, der inoffizielle Begründer von Kanban in der Software-Entwicklung, zusammenzuführen. Im Folgenden wird hauptsächlich seine Vorstellung dieses agilen Software-Entwicklungsprozesses erläutert, die sich in seinem Buch „Successful Evolutionary Change for your Technology Business“ nachlesen lässt.

### **Eigenschaften von Kanban**

„Kanban [...] is used to refer to the methodology of evolutionary, incremental process improvement [...] and has continued to evolve in the wider Lean software development community in the years since.“ (e-book, Kapitel 1 Ende ?)

Diese Definition fasst bereits die wichtigsten Eigenschaften von Kanban in einem Satz zusammen. Im Gegensatz zu vielen anderen agilen Prozessen löst Kanban nicht die vorher existierende Methode in einem Schlag ab. Stattdessen soll der bereits existierende und den Mitarbeitern bekannte Prozess in kleinen Schritten verbessert werden. Die Idee dahinter ist, dass Menschen Gewohnheiten haben und mögen. Ihnen widerstrebt Veränderung. Deshalb ist es besser, diese Anpassung schrittweise durchzuführen, um auf so wenig Widerstand zu stoßen.

Abgesehen von der Idee der kontinuierlichen Verbesserung steht im Mittelpunkt von Kanban ein weiteres Konzept: der Flow oder Fluss. Hierbei ist gemeint, dass Tickets möglichst gleichmäßig durch das System wandern sollen, sie also so wenig wie möglich still stehen und warten müssen. Dies setzt voraus, dass Tasks sich jeweils in ihrer Größe nicht allzu sehr unterscheiden. Da dies nicht immer möglich ist, können Aufgaben Service-Level-Agreements und Item-Types zugeordnet werden. Diese können zur besseren Visualisierung mit verschiedenen Farben gekennzeichnet oder in Swim Lanes, abgegrenzten Zeilen am Board, zusammengefasst werden. So ist der Fluss auch bei verschieden großen Tasks möglich. Das Ziel von Kanban ist es nun, alle weiteren Behinderungen des Flows zu erforschen und möglichst zu beseitigen. Dafür werden die verschiedensten Metriken, Diagramme und Statistiken verwendet, so z.B. Burn-Charts, das Cumulative Flow Diagram oder der Durchsatz.

Insgesamt ist Kanban jedoch sehr frei und anpassbar in seinen Techniken und Prinzipien. Dies ist auch gar nicht anders möglich, da man auf dem vorhandenen Prozess aufbaut, und dieser von Team zu Team unterschiedlich ist. So gut wie alle im Folgenden vorgestellten Methoden haben sich zwar in vielen Teams bewährt, aber außer der Beschränkung des WIP gibt es keine Pflichtvorgaben. Sie sind nur Best Practices. Selbst das Kanban-Board ist nur ein Vorschlag zur Visualisierung.

Ebenso freiwillig sind Iterationen in Kanban; sie werden jedoch fast von allen Quellen trotzdem empfohlen. Allerdings ist es möglich, verschiedene Taktfrequenzen (englisch „cadences“) bei Planung, Release oder Priorisierung zu setzen. So kann ein Meeting zur Priorisierung jede Woche stattfinden, aber eine Lieferung nur aller vier Wochen.

Kanban selbst schreibt weiterhin auch keine Rollen vor, so dass diese aus dem vorher existierenden Prozess beibehalten werden können, und auch die Priorisierung des Backlogs ist freiwillig. Dies ist aus Kundensicht aber eher störend, da der Kunde selten seine Aufgaben nach dem First-In-First-Out-Prinzip abgeordnet haben will.

Auf Grund dieser doch sehr starken Modifizierbarkeit von Kanban, wird die Methode noch immer kontrovers diskutiert. Allerdings ist weniger umstritten, dass Teams, die diese Methode intensiv nutzen, messbar verbesserte Produktivität, Qualität, Kunden- und auch Mitarbeiterzufriedenheit sowie kürzere Lieferzeiten vorweisen können.

### **Kanban einführen**

Die bereits erwähnte kleinschrittige Einführung von Kanban unterliegt wie alles andere ebenso keiner festen Regel oder Reihenfolge. Trotzdem haben sich einige Schritte bewährt.

Der erste und einfachste Schritt ist die Visualisierung des Workflows. Dies wird für gewöhnlich mit Hilfe eines Boards realisiert wie es auch viele andere agile Softwareprozesse verwenden. Jeder Schritt im Workflow erhält mindestens eine Spalte am Board. Allerdings hat ein Kanban-Board auch einige Besonderheiten. Hierzu zählen die Begrenzung des Work-In-Progress aus Schritt 2 und die mögliche Anordnung der Aufgaben in Swim Lanes. Das Ziel dieser Visualisierung ist einerseits das Bewusstmachen und Verinnerlichen des Arbeitsprozesses für alle Mitarbeiter. Andererseits werden so bereits relativ zeitig die bereits angesprochenen Bottlenecks sichtbar. Jede Spalte innerhalb eines Kanban-Boards kann als Work-Queue verstanden werden. Das bedeutet, alle Tickets bewegen sich nach und nach durch die Spalten bis zum Ende des Boards. Wenn sich in einer Queue nun immer mehr Tickets ansammeln, ist die Station unmittelbar danach der Engpass, denn sie kann die Tickets nicht schnell genug abarbeiten und nachziehen.

Schritt 2 kann bereits schwieriger zu implementieren sein. Mit der Begrenzung des Work in Progress wird paralleles Arbeiten verringert. Am Board wird dies durch Begrenzungen der Spalten ausgedrückt. Jede Spalte erhält eine maximale Anzahl an Tickets. Diese Zahl steht für gewöhnlich über der Spalte am Board und darf nur in wenigen Ausnahmefällen überschritten werden. Damit findet weniger Multi-Tasking statt, was wiederum bedeutet, man beendet eine Aufgabe bevor man eine neue beginnt. Damit bewegen sich die Tickets automatisch schneller zum Ende des Boards statt in einigen Spalten lange zu verweilen, d.h. der Durchsatz wird

verbessert. Daraus wiederum resultieren schnellere Releases an den Kunden, der eher Feedback zum gelieferten Produkt geben kann. Gleichzeitig ist ein Entwickler gezwungen bei Problemen nicht einfach eine andere Aufgabe zu suchen, sondern stattdessen sofort an der Lösung des Problems zu arbeiten, damit sie den Workflow nicht zu lange blockiert. Damit ist Kooperation und Hilfe untereinander gefragt.

Schritt 3 ist der Optimierungsschritt und betrifft die Kontrolle des Flusses, indem verschiedene Größen gemessen werden. Dazu zählen der Durchsatz, Warteschlangenlängen oder der Flow selbst. Ziel dabei ist es, die Planung zu erleichtern und sich immer mehr an eine möglichst korrekte Zielvorgabe für den Kunden anzunähern.

In Schritt 4 sollten die selbstaufgestellten Regeln des Teams schriftlich festgehalten werden. Dazu können Dinge wie eine Definition of Done gehören oder auch wie die Wahl des nächsten Tickets vonstatten gehen soll. Damit wird Unsicherheit über den Prozessablauf bei den Teammitgliedern vorgebeugt.

Schließlich sollen im letzten Schritt Modelle verwendet werden, „um Chancen für kollaborative Verbesserungen zu erkennen“ (Quelle Zitat: [http://de.wikipedia.org/wiki/Kanban\\_%28Softwareentwicklung%29](http://de.wikipedia.org/wiki/Kanban_%28Softwareentwicklung%29)). Diese Modelle können aus den verschiedensten Bereichen übernommen werden, so z.B. aus der Theorie of Constraints, und sind ebenfalls wieder frei wählbar. Auch Eigenentwicklungen oder Modifikationen sind erlaubt.

Die eben vorgestellte Einführung wird oft auch kürzer in nur drei Schritten zusammengefasst (Abbildung 1): Visualisieren, Messen und Optimieren.

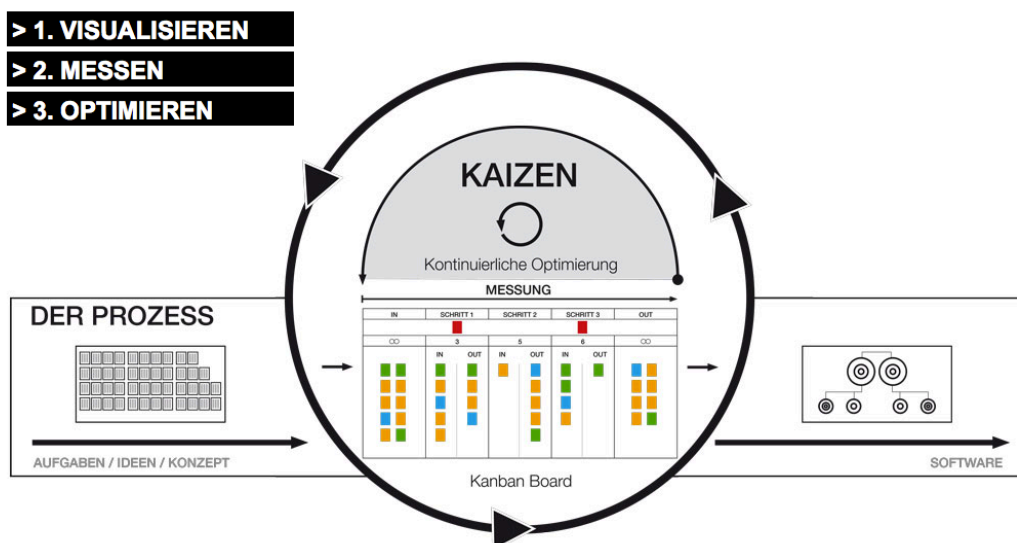


Abbildung 1: <http://kanban-plakat.de/>

### **Kommunikation und Koordination**

Je nach Umsetzung gibt es verschiedene Meetings, die den Arbeitsprozess mit Kanban vereinfachen. Dazu gehört zu allererst das Daily, ein tägliches Standup-Meeting, bei dem der Projektfortschritt am Board betrachtet wird. Dabei soll nach Anderson jedoch im Gegensatz zu anderen agilen Prozessen nicht erläutert werden, wer was tut und tun wird. Dies sollte aus dem Board schnell ersichtlich sein. Stattdessen wird lediglich überprüft, ob Tickets existieren, die nicht weiter kommen, oder ob Bottlenecks auffallen, die behoben werden müssten und können. Als mögliche Idee schlägt er vor, Tickets, die sich einen Tag nicht bewegt haben, zu markieren, beispielsweise mit einem Punkt für jeden Tag. Damit fallen sie im nächsten Daily sofort wieder auf.

Alle Themen, die für das normale Daily zu detailliert oder nicht ganz passend sind, können im After Meeting gleich im Anschluss besprochen werden. Hier finden sich eher kleine Gruppen von zwei bis drei Personen zusammen, die ein gemeinsames Problem lösen möchten oder andere Informationen austauschen, die nicht für alle am Daily Beteiligten wichtig sind.

Die Queue Replenishment Meetings dienen zur Priorisierung der Input-Queue, also der Spalte am Board, in der die abzuarbeitenden Tickets zu finden sind. Die Priorisierung wird vom Product Owner und eventuellen anderen Stakeholdern durchgeführt. Das Entwicklungsteam ist für gewöhnlich nicht involviert. Es ist möglich diese Meetings regelmäßig einzutakten oder eher spontan (on-demand) abzuhalten, wenn das Team die Priorisierung von allein zufriedenstellend erledigt. Die Dauer zwischen den regelmäßigen Meetings beeinflusst die benötigte Größe der Queue und sollte eher kurz sein, z.B. eine Woche.

Ähnliches gilt für Release Planning Meetings. Auch sie können regelmäßig stattfinden, was regelmäßige Releases zur Folge hat. Damit ist keine weitere Koordination der Teilnehmer notwendig. Werden sie unregelmäßig durchgeführt, muss der Kunde mit unregelmäßigen Releases einverstanden sein. Sie erfordern sehr viel mehr Planung, um allen, die möchten, die Teilnahme zu ermöglichen.

Während der Triage bzw. Backlog-Triage wird für jedes Item im Backlog geprüft, ob es gelöscht werden kann oder noch relevant ist. Für gewöhnlich nehmen auch hier eher Stakeholder teil, die nicht an der technischen Umsetzung beteiligt sind. Eine Alternative zur Triage ist die Möglichkeit, regelmäßig alle Items im Backlog zu löschen, die ein bestimmtes Alter überschritten haben. Sinn von beiden Varianten ist



der selbe: den Backlog nicht zu sehr anwachsen zu lassen, so dass das Queue Replenishing Meeting zur Priorisierung schneller vonstatten gehen kann.

Schließlich hat auch Kanban ein Meeting, in dem es um die Verbesserung des Prozesses selbst geht - den Operations Review. Die Durchführung kann prinzipiell unregelmäßig sein, sollte jedoch möglichst nicht nur das Entwicklungsteam selbst enthalten, sondern auch Teilnehmer aus allen Bereichen, mit denen das Team zusammenarbeitet. Somit wird sichergestellt, dass auch an den Schnittstellen des Teams nach außen Probleme auffallen und verbessert werden können.

Abgesehen von all diesen freiwilligen Meetings, sollte das Team hauptsächlich mit Hilfe des Kanban-Boards oder einem Tool zum elektronischen Tracking kommunizieren. Bei ersterem können Probleme in Form von blockierenden oder blockierten Tickets sowie Bottlenecks im Ablauf schnell erkannt und gemeinsam behoben werden. Das Gegenstück eines Bottlenecks, eine Station, die nicht ausgelastet ist, bietet ebenfalls Raum für Verbesserung, eventuell indem das WIP-Limit erhöht wird oder indem die Zahl der Teammitglieder für diese Station im Workflow reduziert wird. Das elektronische Tracking meint für gewöhnlich einen Issue-Tracker, der zusätzlich zur Auflistung aller Tickets mit Details meist ein virtuelles Board enthält. Best Practice bei Kanban ist das Verwenden von beidem zusammen.

### **Fazit**

Kanban unterscheidet sich von anderen agilen Prozessen, da es einen bestehenden Prozess verbessert, statt ihn zu ersetzen. Es bringt ebenso eine Verbesserung der Selbstorganisationsfähigkeiten der beteiligten Personen und gute Visualisierungsmöglichkeiten des Workflows und damit mehr Transparenz und Verständnis. Schließlich verlangt Kanban mehr Kooperation und Fokussierung, um die Vorhersagbarkeit von Terminen für den Kunden zu verbessern. Dies führt zu höherer Kundenzufriedenheit und Mitarbeitermotivation.

### 3. Vergleichskriterien für die Prozesse

Anhand der nachfolgenden Vergleichskriterien werden die drei agilen Prozesse, Crystal, Kanban und Verglichen. Die Auswertung der einzelnen Prozesse erfolgt auf Grundlage der Recherchen der einzelnen Prozesse. Die Vergleichskriterien wurden anhand einzelner Besonderheiten der einzelnen Prozesse aufgestellt. Die Vergleichskriterien dienen auch nach Abschluss des Projekts zum Vergleich der agilen Prozesse anhand unserer gewonnenen Erfahrungen. Die Gegenüberstellung anhand der Matrix ist besonders hilfreich um Unterschiede oder auch Gemeinsamkeiten hervorzuheben.

#### 3.1 Begriffserklärung

**Iterationen:** Iterationen sind Abschnitte die immer wieder nach einem gleichen Muster durchlaufen werden. In den einzelnen Iterationen werden Teile der Software innerhalb eines festgelegten Zeitraums entwickelt

**Teamgröße:** die Teamgröße beschreibt die Optimale beziehungsweise mindestens benötigte Teamgröße in einem Projekt mit der jeweiligen Entwicklungsmethode.

**Rollen:** Unter Rollen versteht man die Zuweisung unterschiedlicher Aufgabenbereiche, welche in einem Projekt vorhanden sein sollen.

**Techniken:** die Techniken beschreiben die Vorgehensweise während der Entwicklung, welche Techniken zum Best Practice gehören und welche möglich sind.

**Lieferung:** Die Lieferung beschreibt die Termine an denen Jeweils ein fertiger Softwareteil dem Kunden übergeben werden kann. Dies kann entweder nach bestimmten Iterationen oder immer zu festen Zeiten erfolgen.

**Änderung der Anforderungen:** In den Verschiedenen agilen Prozessen ist es an verschiedenen Stellen möglich die Anforderungen des Kunden zu ändern, dies kann durch unvorhergesehene Einflüsse neue Erkenntnisse oder durch wirtschaftliche Vorkommnisse unter anderem nötig sein

**Änderung der Arbeitsweise:** Eine Änderung der Arbeitsweise ist je nach Vorgehensmodell an verschiedenen Stellen möglich. Solche Änderungen sind nötig wenn festgestellt wird, dass eine bestimmte Arbeitsweise für dieses Projekt nachteilig ist.

**Meetings/Kommunikation:** Die Kommunikation ist in den einzelnen Agilen Prozessen sehr unterschiedlich geregelt, ebenso die Zeiten wann eine Kommunikation untereinander stattfindet und auf welche Weise.

**Dokumentation:** Unter Dokumentation wird unter anderem die Dokumentation des Fortschritts eines Projektes, sowie vorherige Planungsdokumente verstanden und in welcher Weise diese erfolgt.

**Commitment/ Definition of Done:** Ist die Festlegung wann eine User Story als erledigt markiert werden kann.

**Größe der Tasks:** Die Größe der Tasks richtet sich danach in welcher Zeitspanne ein einzelner Task abgearbeitet werden kann.

**Aufwandsschätzungen:** Für die Aufwandsschätzungen gibt es verschiedene Möglichkeiten. Zu Teil können diese einen Realen Ursprung wie die Zeit haben oder Punkte.

**Priorisieren:** Bei der Priorisierung geht es darum wer die einzelnen User Stories in einem Projekt priorisiert.

**Empirie:** Empirische Erhebung von Informationen zur Erstellung von Statistiken über den Entwicklungsprozess (z.B. Velocity)

### 3.2 Vergleichsmatrix

Die Gegenüberstellung anhand der Matrix ist besonders hilfreich um Unterschiede oder auch Gemeinsamkeiten hervorzuheben. Die einzelnen Prozesse werden in ihrer Ursprünglichen, nicht modifizierten Art gegenüber gestellt. Denn gerade in der Praxis werden die Prozesse nicht immer in Ihrer Ursprünglichen Art genutzt, sondern den Bedürfnissen des Teams angepasst.

Kriterien	Crystal	Kanban	Scrum
<b>Iterationen</b>	- Iterationszyklen innerhalb einer Lieferung	- keine Pflicht - variable Länge oder gleichmäßige Iterationen	- gleichmäßige Länge - in der Regel 1-4 Wochen
<b>Teamgröße</b>	- 2 bis 6	- keine Begrenzung	- 7 bis 10
<b>Rollen</b>	- Chefdesigner - Endanwender - Auftraggeber - Programmierer/ Designer	- keine Vorgaben	- Scrum Master - Product Owner - Scrum Team
<b>Techniken</b>	- Verschiedenste agile Techniken	- WIP-Limit - Service-Level Agreement-Klassen (SLAs)	- Verschiedenste agile Techniken
<b>Lieferung</b>	- mindestens 2 Lieferungszyklen - nach einer Iteration	- Release - am Ende einer Iteration oder festes Datum	- nach jedem Sprint → gezeigt im Review
<b>Änderung der Anforderungen</b>	- vor jeder Iteration möglich	- immer möglich	- vor jedem Sprint möglich
<b>Änderung der Arbeitsweise</b>	- nach jeder Lieferung (Reflexion der Arbeitsweise)	- erwünscht (kontinuierliche Verbesserung) - immer möglich	- nach jeder Retrospektive

Kriterien	Crystal	Kanban	Scrum
<b>Meetings/ Kommunikation</b>	<ul style="list-style-type: none"> <li>- osmotische Kommunikation</li> <li>- Reflexionsmeeting nach Iteration</li> <li>- weitere Meetings empfohlen</li> </ul>	<ul style="list-style-type: none"> <li>- keine Pflicht:</li> <li>- Daily Standup</li> <li>- After Meeting</li> <li>- Queue Replenishment Meeting</li> <li>- Release Planning Meeting</li> <li>- Triage</li> <li>- Issue Log Review</li> <li>- Retrospektive</li> </ul>	<ul style="list-style-type: none"> <li>- Sprint Planning Meeting</li> <li>- Daily Scrum</li> <li>- Review</li> <li>- Retrospektive</li> </ul>
<b>Dokumentation</b>	<ul style="list-style-type: none"> <li>- empfohlen</li> <li>Beispiele:</li> <li>- Projektplan</li> <li>- Versionsplan</li> <li>- Risikoliste</li> </ul>	<ul style="list-style-type: none"> <li>- keine Pflicht</li> <li>- Kanban Board</li> <li>- elektronisches Trackingsystem</li> </ul>	<ul style="list-style-type: none"> <li>- Scrumboard</li> <li>- elektronisches Trackingsystem</li> </ul>
<b>Definition of Done/ Commitment</b>	<ul style="list-style-type: none"> <li>- nicht festgesetzt</li> </ul>	<ul style="list-style-type: none"> <li>- nicht Pflicht</li> </ul>	<ul style="list-style-type: none"> <li>- Product Owner und Team gemeinsam</li> </ul>
<b>Größe der Tasks</b>	<ul style="list-style-type: none"> <li>- nicht vorgeschrieben</li> <li>- generell möglichst klein</li> </ul>	<ul style="list-style-type: none"> <li>- möglichst gleich groß</li> </ul>	<ul style="list-style-type: none"> <li>- möglichst klein</li> <li>- max. 1 Tag/Task</li> </ul>
<b>Aufwands- schätzungen</b>	<ul style="list-style-type: none"> <li>- Blitzplanung</li> <li>- Anpassung vor jeder Iteration</li> </ul>	<ul style="list-style-type: none"> <li>- nicht Pflicht</li> </ul>	<ul style="list-style-type: none"> <li>- Sprint Planning Meeting</li> <li>- Abschätzung mit Story Points</li> </ul>

Kriterien	Crystal	Kanban	Scrum
<b>Priorisierung</b>	<ul style="list-style-type: none"> <li>- in Blitzplanung durch Auftraggeber</li> <li>- Anpassungen während des Projektes</li> </ul>	<ul style="list-style-type: none"> <li>- Queue Replenishment Meeting: Nicht-Entwickler</li> <li>- während Umsetzung: Entwickler oder FIFO</li> <li>- Ausnahme: SLAs</li> </ul>	<ul style="list-style-type: none"> <li>- Product Owner</li> <li>- Anpassungen durch Team bei Abhängigkeiten</li> </ul>
<b>Empirie</b>	<ul style="list-style-type: none"> <li>- nicht gegeben</li> </ul>	<ul style="list-style-type: none"> <li>- Cumulative Flow</li> <li>- Lead und Cycle Time (Durchsatz)</li> </ul>	<ul style="list-style-type: none"> <li>- Velocity (Story Points)</li> </ul>

In der direkten Gegenüberstellung fallen zuerst die Gemeinsamkeiten bei Iterationen auf, denn jeder Prozess hat, beziehungsweise kann Iterationen haben. Lediglich in der Art der Länge unterscheiden sie sich. So haben Crystal und Scrum die Vorgabe, dass es mehrere Iterationszyklen innerhalb der Entwicklung geben muss. Bei Scrum sind die Bestimmungen noch enger gefasst, denn hierbei ist die optimale Länge einer Iteration vorgegeben. Im Gegensatz dazu ist Kanban hierbei sehr offen, denn hier ist nicht vorgeschrieben ob es eine oder mehrere Iterationen gibt.

Ebenso bei der Teamgröße erweist Kanban die höchste Flexibilität, denn hier gibt es keine Begrenzung der Teamgröße. Anders sieht es hier bei Crystal und Scrum aus. Crystal ist eher für kleinere Entwicklungsgruppen konzipiert, wohingegen Scrum für ein etwas größeres Team ausgelegt ist.

Ebenso offen wie bei der Teamgröße ist Kanban auch bei der Rollenverteilung. Hier können sowohl die Rollen von Scrum wie auch von Crystal übernommen werden, oder ganz klassisch mit einem Projektleiter. Die Rollen bei Crystal und Scrum hingegen sind fest definiert.

Die Techniken sind auch wie die Rollen wieder sehr unterschiedlich, da bei Scrum und Crystal alle möglichen agilen Techniken wie Pairprogramming, Side by side Programming und Side by side Testing zum Einsatz kommen können. In Kanban hingegen stehen Techniken wie WIP oder Service-Level-Agreement-Klassen im Vordergrund. Bei WIP (Work in Progress) wird festgelegt, welche Anzahl von Tasks sich in einem Entwicklungsschritt befinden dürfen. Das Festlegen und einsetzen von Service-Level-Agreement-Klassen (SLAs) beschreibt den Einsatz von verschiedenen Ticketarten. Darunter fallen unter anderem Bugs und Change Requests.

In allen Prozessen erfolgt die Lieferung nach einer Iteration. Jedoch können bei Kanban die Lieferungen auch außerhalb von Iterationen stattfinden, da es ja wie oben bereits erwähnt wurde nicht zwingend notwendig ist mehrere Iterationen zu durchlaufen.


Sowohl bei der Änderung der Anforderungen so wie der Arbeitsweise sind wieder Scrum und Crystal gleich. Die Änderungen der Anforderungen erfolgen bei beiden direkt von der Iteration. Bei Crystal hingegen sind Änderungen jeder Zeit möglich. Auch dies hat wieder den Hintergrund das es nicht zwingend Iterationen gibt.

Für die Änderung der Arbeitsweise gilt das gleiche, hier sind ebenfalls Crystal und Scrum gleich. Bei beiden kann die Arbeitsweisen nach einer Iteration geändert werden.

Um eine ungeschickte und eventuell fehleranfällige Arbeitsweise in der nächsten Iteration zu verbessern.

Bei Kanban hingegen ist eine kontinuierliche Verbesserung der Arbeitsweise während der Iterationen gerade zu erwünscht. Denn jede Verbesserung der Arbeitsweise zieht auf Kanban eine Verbesserung des Produktes nach sich.

Die Kommunikation und die Meetings sind bei allen Prozessen sehr unterschiedlich. So ist bei Crystal die osmotische Kommunikation, bei der alle Teammitglieder im gleichen Raum sitzen sollen, das best Practice. Zudem kommen Reflexionsmeetings nach den Iterationen. Weitere Meetings sind keine Pflicht können aber zum Beispiel durch Dailys ergänzt werden. Bei Kanban lautet das Motto wie vorher schon „alles kann aber nichts muss“. So zählen zu den Empfohlenen Meetings zum Beispiel ein Daily Standup, ein After Meeting, das Queue Replenishment Meeting, ein Release Planning Meeting, ein Triage und ein Issue Log Review. Beim Triage wird nach alten Tasks im Backlog gesucht und diese gelöscht. Bei Scrum hingegen sind die Meetings fest vorgeschrieben, dazu zählen Sprint Planning Meeting, Daily Scrum, Review und Retrospektive

Zur Dokumentation haben alle drei Prozesse ein Board an dem die User  und die dazugehörigen Tasks für das ganze Team sichtbar sind. Zudem werden Technische Hilfsmittel wie ein elektronisches Trackingsystem immer häufiger verwendet. Bei Crystal können zum Board und dem Trackingsystem noch weitere Dokumentationen hinzu. Hier gilt der Grundsatz je mehr Kommunikation je weniger Dokumentation. Zu den Beispielen für die Dokumentation zählen: der Projektplan,

der Versionsplan, die Risikoliste sowie eine Architekturbeschreibung. Die Definition of Done oder auch Commitment besagt ab wann eine User Story vom Kunden als erledigt abgenommen wird. Dabei ist dies nur bei Scrum genau festgehalten und wird vom Product Owner festgelegt. Bei den anderen zwei Prozessen ist eine feste Definition of Done nicht festgeschrieben

Die Größe der Tasks ist bei allen Prozessen sehr ähnlich. Dabei sollte darauf geachtet werden dass die einzelnen Tasks eine geringe Größe haben. Jedoch sollte als Spezialisierung bei Kanban darauf geachtet werden dass die einzelnen Tasks ungefähr die gleiche Größe haben. Bei Scrum hingegen gilt es als Sinnvoll darauf zu achten, dass ein einzelner Task in der Größe nur so groß ist dass er innerhalb eines Tages abarbeitbar ist.

Bei der Aufwandschätzung unterscheiden sich die drei Prozesse sehr. Crystal führt zu Anfang des Projektes eine Blitzplanung durch in dem die jeweiligen Zeiten für die einzelnen Tasks geschätzt wird. Vor jeder Iteration können die Zeiten bezüglich gewonnener Erfahrungswerte angepasst werden. Bei Kanban ist eine Zeitabschätzung nicht vorgeschrieben. Natürlich kann auch hier eine Schätzung zu jeder Zeit erfolgen. Bei Scrum wird im Sprint Planning Meeting die Zeiten für die einzelnen Tasks die im Sprint abgearbeitet werden sollen vom gesamten Team geschätzt.

Bei der Priorisierung der Tasks sind sich Crystal und Scrum wieder sehr ähnlich hier werden die Tasks vom Auftraggeber beziehungsweise vom Product Owner durchgeführt. Jedoch können die Entwickler während der Entwicklung Tasks oder ganze User Stories umpriorisieren, falls dazu die Notwendigkeit besteht. Dies kann der Fall sein wenn User Stories oder Tasks von anderen abhängig sind.

Kanban (fehlt noch)

Die Erhebung von Statistiken so genannte Empiere gibt es nur in Kanban und in Scrum. Bei Kanban gibt es unter anderem den cycle time, dabei wird die Zeit der Arbeit am Ticket beginnen bis zu dessen Ende ins Verhältnis gesetzt. Bei Scrum gibt es das Velocity, dies besagt wie viele Storypoints innerhalb eines Sprints abarbeitbar sind.



## 4. Wahl der Tools

### Vergleich der Tools

Bei der Durchführung eines Projektes ist es immer hilfreich, mindestens für einige der Managementaufgaben ein oder mehrere Hilfsmittel zu verwenden. In diesem Fall werden drei Projekte mit verschiedenen Prozessmodellen parallel bearbeitet, während sich die Teammitglieder zusätzlich den Großteil der Zeit nicht im gleichen Raum oder Gebäude aufhalten. Deshalb entschied sich das Team für die Nutzung eines elektronischen Tools, an welches sehr verschiedene, allgemeine als auch technik-spezifische Anforderungen gestellt wurden. Im Rahmen dieser Arbeit sollen hierfür zwei Tools ausgewählt werden, mit deren Hilfe verschiedene Projektmanagement-Aspekte erleichtert werden sollen. Im Anschluss an die Durchführung wird eruiert, welche Vor- und Nachteile den Teammitgliedern aufgefallen sind und welches Tool sich besser für jedes Prozessmodell eignet. Da alle drei durchgeführten Prozesse im Bereich der agilen Software-Entwicklung anzusiedeln sind, sollten die gewählten Hilfsmittel entweder spezifisch für agiles Projektmanagement entwickelt worden sein oder zumindest erkennbar mit den daher rührenden Anforderungen umgehen können. Das bedeutet insbesondere, dass Iterationen in irgendeiner Form abbildbar sein müssen. Außerdem arbeiten an den Projekten Entwickler mit verschiedenen Betriebssystemen, in diesem Fall OS X und Windows. Das bedeutet, das Tool muss entweder online und im Browser nutzbar sein oder für die beiden genannten Betriebssysteme zur Verfügung stehen.

Da die Projektmitarbeiter über eine begrenzte Erfahrung mit Projektmanagement-Tools und Issue-Trackern verfügen, wurden verschiedene Online-Quellen hinzugezogen, um eine Vorauswahl und dann auch die endgültige Wahl zu treffen. Die Hauptquelle zur Vorauswahl war die Webseite FindTheBest<sup>2</sup>, welche Informationen über verschiedenste Projektmanagementsoftware sowie Reviews und Bewertungen von Nutzern zur Verfügung stellt. Außerdem ist es möglich, schnell und übersichtlich einen Vergleich zwischen mehreren Produkten anzuzeigen. Des Weiteren wurden Videos, Demos und Rezensionen auf den Herstellerseiten zu Rate gezogen, diese jedoch immer mit dem Wissen, dass der Hersteller selbst weniger objektiv sein kann als eine unabhängige Quelle.

<sup>2</sup> <http://project-management-software.findthebest.com/>

Schließlich wurden die in Tabelle 1 dargestellten fünf Tools in die nähere Auswahl aufgenommen. Dabei kamen JIRA und Pivotal Tracker allein wegen ihrer Bekanntheit in die Vergleichsrunde. Die drei anderen Produkte zeigten auf FindTheBest die besten Bewertungen für Issue-Tracker, die sowohl mit agilem Projektmanagement und Kanban umgehen können, als auch für kleine Projekte handhabbar sind.

<b>JIRA mit Greenhopper</b>	allen Teammitgliedern mehr oder weniger gut bekannt
<b>Pivotal Tracker</b>	einem Teil des Teams bekannt
<b>Genius Inside</b>	Beste Smart Rating – User Rating Kombination einer Software für kleine Projekte (Quelle: FindTheBest)
<b>Vision Project</b>	Nach Genius Inside beste Software, die Kanban und Issue-Tracking können soll (Quelle: FindTheBest)
<b>Yodiz</b>	Nach Vision Project beste Software, die Kanban und Issue-Tracking können soll (Quelle: FindTheBest) ansprechende Oberfläche im Video (Quelle: FindTheBest)

Tabelle 1: Software

### Bewertungskriterien

Im Folgenden werden die im Vorfeld erstellten Auswahlkriterien tabellarisch zusammengefasst und kurz erläutert. Dabei wurden allgemeine von prozess-spezifischen Kriterien getrennt und zusätzlich Wünsche der Teammitglieder unter „nice to have“ gelistet. Die letztgenannten Kriterien waren hauptsächlich als Tie-Breaker gedacht, falls alle anderen nicht ausreichen, um sich klar für zwei Tools zu entscheiden.

Für die Grundlage der Bewertung orientierte sich das Team an einer Idee aus dem Qualitätshaus, einer Matrix aus dem Quality Function Deployment. Die Wichtigkeit eines Merkmals wird mit einer Priorität zwischen 1 und 5 bewertet, wobei 5 für „sehr wichtig“ und 1 für „unwichtig“ steht. Die Entscheidung, ob ein Tool das gewünschte Merkmal vorweist, wird wiederum mit 0, 1 oder 2 bewertet (siehe Tabelle 2).

0	definitiv nicht vorhanden
1	könnte ungefähr so vorhanden sein, wie benötigt oder eine andere Funktionalität kann vermutlich dafür „missbraucht“ werden oder es wurden keine Angaben gefunden, könnte also vorhanden sein
2	so vorhanden wie benötigt

Tabelle 2: Bewertungsskala und ihre Bedeutung

# Ihr Stand: Man einen einleitenden Text.

## Allgemeine Kriterien

Tabelle 3 gibt einen Überblick über allgemeine Kriterien bei der Tool-Auswahl. Zu diesen zählen theoretisch auch Preis und Plattform bzw. Betriebssystem als rein organisatorische Kriterien. Der Preis spielte eine untergeordnete Rolle und wurde deshalb nicht in die Bewertungsmatrix aufgenommen. Die Ausführbarkeit auf Computern mit verschiedenen Betriebssystemen hingegen war wie bereits zu Anfang erwähnt eine zwingende Anforderung, so dass kein Tool aufgenommen wurde, dass diese nicht erfüllt.

Schließlich wurde auch die Erfahrung der Teammitglieder mit dem Tool einbezogen, da die Vertrautheit mit der Software hilfreich bei der Projektdurchführung sein sollte.

Allerdings kann bei einem bereits bekannten Produkt die Schwierigkeit der Einarbeitung nicht mehr bewertet werden. Nach Beratung im Team wurde entschieden, dass Vertrautheit mit dem Tool wichtiger, aber trotzdem keine Pflicht sein soll.

Neben den organisatorischen Merkmalen gibt es auch allgemeine Funktionen, die in den gewählten Software-Entwicklungsprozessen gleich oder zumindest ähnlich vorkommen und abbildbar sein müssen. Dazu zählen das Aufstellen von User Stories, Priorisierung von Tickets, eine Art Backlog und ein anpassbares Board zur Visualisierung der Tickets. Dies sind Mindestanforderungen an die gewählte Software aus der agilen Entwicklung. Das anpassbare Board ist besonders für Kanban hilfreich und wichtig, da es den Workflow abbildet, den Kanban zu optimieren versucht. Das bedeutet auch, dass das Board angepasst werden sollte, so z.B. die Anzahl der Spalten oder deren Namen.

Aufgrund der geographischen Trennung der Teammitglieder ist neben dem elektronischen Board auch eine Möglichkeit zur Kommunikation wichtig. Diese kann über einfache Kommentare an Tickets, Emails, Persönliche Nachrichten oder Chats erfolgen. Mindestens eine dieser Varianten sollte durch das Produkt gegeben sein. Eine 2 erhielten in diesem Fall Produkte, die mehr als eine Art der Kommunikation erlauben.

die Kriterien sollten aus dem heraus fallen, was man analysieren will

↑  
besser strukturieren

Schließlich sollte das Tool nicht nur als Ersatz für ein nicht vorhandenes physisches Board genutzt werden sondern unbedingt auch ein Issue-Tracker sein. Das bedeutet insbesondere, dass User Stories und Tasks, die nur überblicksartig auf dem Board sichtbar sind, mit genaueren Details außerhalb des Boards gehandhabt werden können, oder auch ganz auf das Board verzichtet werden kann, falls beispielsweise ein physisches Board existiert. Außerdem soll eine Zuordnung an Mitarbeiter und eine Art Klassifikation der Tickets möglich sein, z.B. in Backlog-Items oder Bugs.

	Prio	Genius inside	Jira	Pivotal Tracker	Vision Project	Yodiz
Erfahrung	3	0	2	1	0	0
<b>Allgemeine Funktionen</b>						
User Stories	4	2	2	2	2	2
Priorisierung	4	2	2	2	2	2
Backlog (Produktbacklog)	5	2	2	1	2	2
anpassbares Board	4	0	2	0	1	1
Möglichkeiten zur Kommunikation	4	2	1	1	2	2
Issue tracking	4	2	2	2	2	2
Summe		42	52	36	46	46

Tabelle 3: allgemeine Kriterien

### Bewertungskriterien für Scrum

Das Scrum-Team hat für seinen Prozess die in Tabelle 4 dargestellten Kriterien aufgestellt. Dabei wurde insbesondere die Durchführung von Sprints, möglichst auch mit dieser Benennung, gewünscht. Für Scrum bedeutet dies, dass man eine Iterationen in eine Timebox stecken, d.h. mit einem Start- und Enddatum versehen kann. Ein automatisches Öffnen und Schließen des Sprints wäre schön, ist aber kein Muss-Kriterium und wurde deshalb nicht in bei der Bewertung berücksichtigt.

Um Scrum entsprechend umzusetzen, muss das Tool außerdem unbedingt eine Art Sprintbacklog erlauben, d.h. es muss möglich sein, aus allen vorhandenen Tickets des Projekts eine bestimmte Anzahl zu einem Sprint zuzuordnen.

Da das Team an verschiedenen Orten entwickeln sollte, war ein virtuelles Scrumboard ebenfalls sehr wichtig. Hierbei gab es in der initialen Entwicklung der Kriterien keine Anforderung an die Konfigurierbarkeit. Auch diese war eher ein Kann-Kriterium.

Das Vorhandensein von User Stories und Tasks bzw. die Möglichkeit mindestens zwei verschiedene Arten von Tickets anzulegen, wurde ebenfalls als wichtig für Scrum eingeordnet. Außerdem sollten zu einem Ticket in irgendeiner Weise Unteraufgaben zugeordnet werden können. Da das Team nicht immer bei der Entwicklung zusammensitzt, sollten Tickets in kleinere Aufgaben unterteilbar sein, so dass nicht jedes Teammitglied an einer eigenen User Story arbeiten muss, um anderen nicht in die Quere zu kommen. Dafür würden diese User Stories sehr lange „in progress“ sein, eine unerwünschte Eigenschaft bei Scrum.

Zur Auswertung und möglichen Verbesserung des Scrum-Prozesses ist u.a. nötig einige Kennzahlen zu berechnen. Das Team entschied, dass zumindest ein Burndown Chart automatisch mit Hilfe des Tools erzeugbar sein sollte, um hierbei behilflich zu sein und den Fortschritt auf andere Art als nur im Board zu visualisieren. Dies wurde besonders durch den Scrum Master gewünscht, erhielt allerdings nur eine geringere Priorität, da notfalls auch eine Abbildung per Hand beispielsweise in Excel möglich ist.

	Prio	Genius inside	Jira	Pivotal Tracker	Vision Project	Yodiz
Sprints	5	2	2	1	1	2
Sprint Backlog	5	2	2	2	0	2
Scrumboard	4	2	2	1	1	2
User Stories und Tasks	4	1	2	1	1	2
Abhängigkeiten zwischen User Stories / Untertasks	4	1	2	0	2	1
Burndown Chart	2	2	1	2	2	2
Summe		40	46	27	25	44

Tabelle 4: Scrum-Kriterien

**Bewertungskriterien für Kanban**

Obwohl Kanban allgemein nur sehr wenige Merkmale besitzt, die zwangsweise dazugehören, entschied sich das Team dafür, auch einige der freiwilligen Techniken zu als Kriterien in Tabelle 5 aufzunehmen.

Das Kanban-Board ist eines dieser Kriterien, wurde jedoch unbedingt vom Team zur schnellen Übersicht aus Mangel an einem physischen Board gewünscht. Obwohl ein Kanban-Board im Prinzip einem Scrum-Board entsprechen kann, gab es auch hier nur 2 Punkte, wenn das Board veränderbar war, so dass ein geänderter Workflow übertragen werden könnte. Zusätzlich dazu sollte es möglich sein, die Anzahl der Tickets pro Spalte zu begrenzen. 2 Punkte gab es jedoch nur, wenn eine Verletzung der Begrenzung möglich und gut sichtbar zu erkennen war. Bei allen Produkten außer JIRA musste man sich hierbei auf Rezensionen verlassen, da in den Demos hierfür möglicherweise weiterführende Berechtigungen nötig gewesen wären. Da sich das Team zu Beginn des Projekts noch nicht festgelegt hatte, wurde dem Vorhandensein von Swim Lanes innerhalb des Boards dagegen eher eine niedrigere Priorität verliehen.

Für die Darstellung der verschiedenen SLAs oder Tickettypen sowie eine Unterscheidung zwischen Ticket und Task sollte irgendeine Art der Einstellung möglich sein. Am besten wären farbliche Unterschiede wie es für Kanban zur schnellen Unterscheidung am Board vorgeschlagen wird. Da dies notfalls mit Swim Lanes in etwa ersetzt werden kann, wurde lediglich die Priorität 3 gewählt.

Das Ziel der kontinuierlichen Verbesserung des Arbeitsablaufs kann nur dann erreicht werden, wenn regelmäßige Übersichten über Durchsatz, Flow und Geschwindigkeit der Arbeit erstellt und analysiert werden. Hierfür wären die zwei am häufigsten bei Kanban verwendeten Diagramm-Typen wünschenswert: Cumulative Flow Diagram und Burnup bzw. Burndown-Chart. Auch hier kann man die niedrige Priorität der Burncharts mit der Möglichkeit eines manuellen Ersatzes erklären.

Neben diesen beiden sind besonders Lead Time und Cycle Time hilfreiche Richtwerte und sollten deshalb möglichst leicht ablesbar sein.

Als eines der wenigsten agilen Prozessmodelle kann Kanban ganz ohne Iterationen durchgeführt werden. Auch wenn dies nicht das Ziel des Teams war, sollte die niedrige Priorisierung diesen Fakt hervorheben: Iterationsfähigkeit ist kein Muss für das Produkt. Da sie jedoch ebenso für Scrum und Crystal Clear nutzbar sein sollten, erhielten alle Produkte 2 Punkte.

	Prio	Genius inside	Jira	Pivotal Tracker	Vision Project	Yodiz
Kanban Board	5	1	2	1	1	1
WIP- Begrenzung	4	1	2	1	1	1
Swim Lanes	2	1	2	0	1	1
Tickettypen / SLAs	3	1	2	2	1	1
Burn Up Chart	2	0	0	0	2	2
Cumulative Flow Diagram	4	0	2	0	2	1
Durchsatz: Lead and Cycle Time	4	1	2	0	2	1
Iterationen	1	2	2	2	2	2
Summe		20	46	17	32	28

Tabelle 5: Kanban-Kriterien

### Bewertungskriterien für Crystal Clear

Die Bewertungskriterien für Crystal Clear finden sich kurz zusammengefasst in Tabelle 6. Ähnlich der Liste von Scrum ist die wichtigste Anforderung eine Möglichkeit zur Organisation einer Iteration. In diesem Zusammenhang ebenfalls wünschenswert wäre die Planung und Umsetzung von Releases. Damit können häufige Lieferungen an den Endanwender abgebildet werden.

Des Weiteren sollte ein Rollensystem implementiert werden. Für die Vergabe einer 1 genügte es, wenn man Nutzern verschiedene vorgegebene Rollen mit verschiedenen Zugriffs- und Ausführungsrechten zuordnen kann. Für eine 2 sollte es die Software dem Anwender erlauben, seine eigenen Rollen zu kreieren, zu benennen sowie deren Berechtigungen zu setzen. Die Einschätzung dieses Kriteriums war am schwierigsten, da in einer Online-Demo selten genug Rechte vorhanden sind, um eine solche Funktionalität zu testen. Ebenso wenig wurde in Vorführvideos darauf eingegangen. Somit erhielt lediglich JIRA 2 Punkte, da hier aus der Erfahrung heraus bekannt war, dass neue Rollen im Administratorbereich angelegt werden können. In Bezug auf die Reflektion und im Sinne der kontinuierlichen Verbesserung des Arbeitsprozesses sollte das gewählte Tool außerdem im Nachhinein noch einmal

Einblick in alte Iterationen geben können, die Teil der letzten Lieferung waren. So kann auf diese in der Reflektion noch einmal eingegangen werden.

Schließlich sollte ähnlich wie bei Scrum zumindest eine Art Burnchart im System integriert sein – entweder ein Burnup- oder ein Burndown-Chart. Diese sind besonders wichtig für den Überblick über den momentanen Fortschritt im Sprint. Somit sollte schnell erkennbar sein, falls das Team nicht mehr in der Zeit liegt und möglicherweise der Release-Termin nicht eingehalten werden kann.

	Prio	<b>Genius inside</b>	<b>Jira</b>	<b>Pivotal Tracker</b>	<b>Vision Project</b>	<b>Yodiz</b>
Iterationen	5	2	2	2	2	2
Releases	3	2	2	2	2	2
Rollen	4	1	2	1	1	1
Burncharts (Burnup oder Burndown)	4	2	2	2	2	2
Analyse der letzten Lieferung	4	2	2	1	2	2
Summe		39	36	32	40	40

Tabelle 7: Crystal-Kriterien



**Nice to have – Kriterien**

Da bereits beim Aufstellen und Ausfüllen der prozess-spezifischen Kriterien auffiel, dass auf Grund des begrenzten Wissens über die Produkte die Punkte sehr nahe beieinander liegen würde, wurde zusätzlich Tabelle 7 mit Wünschen der Teammitglieder gefüllt. Diese sollte herangezogen werden, falls es zu einem Unentschieden zwischen Systemen kam oder die Punkte sehr dicht beieinander lagen.

Die Frage nach der allgemein benutzerfreundlichen und intuitiven Oberfläche wurde zusammen mit der Übersichtlichkeit am höchsten priorisiert. Hier erhielten die Tools ihre Bewertung nach einer kurzen Nutzung der Demo, wenn möglich, und mit Hilfe von Screenshots und Videos. Insbesondere Drag-and-Drop-Möglichkeiten wurden mit 2 Punkten bewertet.

Das Vorhandensein eines Dashboards als Einstiegs- und Übersichtsseite über das Projekt wurde ebenso mit „sehr wichtig“ eingestuft, allerdings wurde hier lediglich das Vorhandensein einer solchen Seite bereits mit 2 Punkten belohnt.

Zusätzlich zu den bisher betrachteten Features, die sich hauptsächlich um die Darstellung auf einem Board und verschiedene Analysemethoden drehten, wurden bei den Nice-To-Have-Kriterien auch allgemeinere Projektmanagement-Kriterien aufgenommen. Dazu gehören sowohl ein Dokumentenmanagement als auch ein Wiki zu Dokumentationszwecken. Letzteres kann durch das Schreiben von Kommentaren an Tickets oder in die Tickets selber leichter ausgeglichen werden, wenn auch weit weniger übersichtlich. Ersteres wäre zum Beispiel hilfreich für verschiedene Planungsnotizen und allgemeine Dokumente oder Screenshots, die an Tickets angefügt werden sollen, zu denen sie gehören. Bei JIRA wird das Wiki von Confluence, einem separaten aber leicht mit JIRA kombinierbaren Produkt übernommen. Des Weiteren wurde die Möglichkeit des Time Tracking gewünscht, so dass kein zusätzliches Produkt verwendet werden müsste, um dies zu übernehmen. Außerdem wäre ein Kalender praktisch, um insbesondere die verschiedenen Meetings leicht ersichtlich zur Hand zu haben und eventuell sogar mit Hilfe des Tools zu organisieren. Dazu würden dann auch das Einladen der beteiligten Personen und das Festlegen der Räumlichkeit gehören. Confluence regelt dies ebenfalls anstatt JIRA.

Schließlich wäre es praktisch aus Programmierer-Sicht, wenn die Software die Integration von Versionsmanagement wie beispielsweise github, bitbucket oder Ähnlichem unterstützt. Damit könnten Tickets mit entsprechenden Commits verknüpft werden, was das Abarbeiten dieser Tickets oder auch das Bugfinden erleichtern könnte.

	Prio	Genius inside	Jira	Pivotal Tracker	Vision Project	Yodiz
benutzerfreundliche/intuitive Oberfläche	5	1	2	2	1	2
Übersichtlichkeit	5	2	2	1	2	2
Dashboard	5	2	2	2	2	2
Dokumenten- management	3	2	0	1	1	2
Wiki	2	0	2	0	2	0
<i>Kalender → z.B. Termine</i>	2	2	2	2	2	2
Time Tracking	2	2	2	2	2	2
<i>GitHub-Integration o.Ä.</i>	3	0	2	0	0	2
<i>Summe</i>		35	48	36	48	54

Tabelle 7: Nice-To-Have-Kriterien

### Ergebnis und Auswahl

Nach der Auflistung aller Kriterien und der Berechnung der Gesamtpunktzahl mit und ohne Nice-To-Have waren zwei Tools mit über 200 Punkten vorn (siehe Tabelle 8).

Der Sieger wurde JIRA. Dies liegt sehr wahrscheinlich daran, dass es neben Pivotal Tracker das einzige dem Team bekannte Produkt war und eine genauere Einschätzung der Funktionalität aus der Erfahrung heraus ermöglichte. Dadurch erhielt JIRA mehr Zweien als seine Konkurrenten.

Auf Platz 2 befindet sich Yodiz, ein Tool, welches seit dem ersten Blick auf seine eher bunte Oberfläche als sehr verschieden von JIRA auffiel. Laut der einzelnen Zwischenpunktzahlen sollte JIRA sich um einiges besser für Kanban eignen, während beide in allen anderen Kategorien außer der allgemeinen sehr nahe beieinander liegen. Da das Team Yodiz nur von der Demo her kennt, sollte hier ein wenig Zeit zur Einarbeitung gegeben werden.

Insgesamt wurden am Ende zwei doch sehr unterschiedlich aussehende Tools gewählt.

	<b>Genius inside</b>	<b>Jira</b>	<b>Pivotal Tracker</b>	<b>Vision Project</b>	<b>Yodiz</b>
allgemein	42	52	36	46	46
Scrum	40	46	27	25	44
Kanban	20	46	17	32	28
Crystal	39	36	32	40	40
<b>Gesamt 1</b>	<b>141</b>	<b>180</b>	<b>112</b>	<b>143</b>	<b>158</b>
nice to have	35	48	36	40	50
<b>Gesamt</b>	<b>176</b>	<b>228</b>	<b>148</b>	<b>183</b>	<b>208</b>

Tabelle 8: Gesamtauswertung

## 5. Wahl des Projektes

Um den Vergleich der drei ausgewählten agilen Softwareentwicklungsprozesse auf eigene praktische Erfahrungen stützen zu können, sollen sie an realen Projekten durchgeführt werden. Die Prozesse können so anhand der Durchführung analysiert und verglichen werden.

Nun stellt sich die Frage, wie man drei Prozesse so durchführt, dass ein optimales Vergleichsergebnis erarbeitet werden kann. Die optimalste Lösung wäre es für jeden Softwareentwicklungsprozess ein eigenes Projektteam zu definieren und jedem Team die gleiche Aufgabenstellung zu geben. Wenn man davon ausgeht, dass die Qualifikationen der Teams ausgeglichen sind, besteht somit für den Vergleich die ideale Basis: gleiches Projektziel, vergleichbares Team, identischer Zeitraum und generell gleiche Gegebenheiten.

Im Rahmen dieser Studienarbeit kann der Ansatz jedoch nicht realisiert werden, da das Projektteam nur aus drei Personen besteht, die gerade die Mindestanzahl für ein Projekt darstellt. Aus diesem Grund werden im Folgenden einige Lösungsansätze erörtert, die sich mit der Problemstellung befassen, wie drei Prozesse zeitgleich von einem einzigen dreiköpfigen Team durchgeführt werden können.

### Lösungsansätze

Eine Möglichkeit wäre jeweils ein Projekt für jeden Prozess durchzuführen. Dadurch kann die Technologie unabhängig von den anderen Projekten definiert werden.

Außerdem beeinflussen sich so die Projekte nicht, da sie unterschiedliche Aufgabenstellungen haben und die Teammitglieder ihre Erfahrungen aus den anderen Projekten nur bedingt benutzen können. Die Projekte sind aus diesem Grund aber auch schwerer vergleichbar, denn die Projektgegebenheiten unterscheiden sich, wodurch keine einheitliche Vergleichsbasis existiert. Darüber hinaus ist das Management von drei separaten Projekten sehr aufwendig.

Um die Nachteile des ersten Ansatzes zu verringern wurde eine zweite Lösung entworfen, bei der ein Projekt in drei Module unterteilt wird und jedes dieser Module mit einem anderen Prozess bearbeitet wird. Dadurch wird das Management etwas leichter aber problematisch wird hierbei die Projektfindung, da man durch die Trennbarkeit in drei Module sehr eingeschränkt ist. Selbst bei diesem Ansatz ist keine gute Vergleichbarkeit gegeben, da die Module sich auch trotz des gemeinsamen Ziels in den Aufgabengebieten unterscheiden.

Da sich die Wahl von drei separaten Projekten oder Modulen nicht als vergleichbar erweist, ist es eher ratsam sich auf die Wahl von einem Projekt zu beschränken. Ein denkbarer Ansatz wäre die drei Prozesse gleichzeitig auf ein Projekt anzuwenden. Aber bereits hier stellt sich die Frage, wie dies möglich sein soll. Ein Projektteam, das drei Prozesse gleichzeitig auf genau dasselbe Projekt durchführen sollen, wäre mit Sicherheit überfordert, da es schwierig ist die Prozesse zu trennen. Darüber hinaus würden sich die Prozesse gegenseitig beeinflussen oder sogar behindern. Ein Vergleich wäre somit unmöglich.

Aus diesem Grund hat man sich entschieden zwar nur ein Projekt zu definieren, dieses jedoch separat für jeden Prozess durchzuführen. Dadurch werden für die definierten Projektanforderungen drei getrennte Lösungen entwickelt, die sich sehr gut als weiteres Mittel für den Vergleich eignen. Ein Nachteil ist, dass die Projekte sich gegenseitig beeinflussen, da durch die Einteilung der gleichen Teammitglieder in jedem Projekt ein Wissenstransfer entsteht. Dieser Faktor kann aber weitgehend ignoriert werden, denn der dadurch simulierte Erfahrungsaustausch könnte in normalen Umständen auch durch Berater stattfinden. Mit dieser Variante ist somit die beste Vergleichsbasis geboten.

### **Vorgaben**

Bei der Wahl des Projektes müssen einige Vorgaben beachtet werden. Einerseits sollte es machbar sein, den Projektscope in 6-8 Wochen abzuarbeiten, um so den zeitlichen Rahmen der Studienarbeit nicht zu sprengen. Andererseits sollte das Projekt aber auch genügend Umfang bieten, um die Prozesse ausführlich anwenden und vergleichen zu können. Darüber hinaus sollen Projektanforderungen so gewählt sein, dass sie die Teammitglieder zwar fordern, aber nicht zu viele neue Kenntnisse verlangen. Denn aus Zeitmangel kann keine lange Einarbeitungsphase eingeplant werden.

*Gleiche Projekte einfacher beim 3. Teil?*

### **Technologie**

Aus diesem Grund hat man sich für eine Webanwendung entschieden, welche mit Hilfe von JSF (Java Server Faces) implementiert werden soll. Vorteil dieser Technologie sind die bereits bestehenden Kenntnisse der Teammitglieder in der Programmierung mit Java und HTML, welche die Hauptbestandteile von JSF darstellen. Als Alternative lässt sich auch die Verwendung von PHP nicht ausschließen, jedoch wäre die Einarbeitungsphase bei dieser Programmiersprache wesentlich aufwendiger.

Um JSF optimal nutzen zu können, soll die Entwicklungsumgebung Eclipse eingerichtet werden, welche mit weiteren Frameworks wie Hibernate erweitert wird, um das Datenbank Mapping zu realisieren. Für die Datenbank wird die OpenSource Lösung MySQL verwendet. Darüber hinaus kann das in Eclipse integrierte Maven Tool genutzt werden, welches einen automatischen Build ermöglicht. Eine weitere wichtige agile Praktik ist das Testen, das mit dem Unit-Test Framework JSFUnit umgesetzt werden kann. Aufgrund der Inkompatibilität von JSFUnit mit anderen Servern, muss für das Deployment der Webapplikation ein JBoss Server eingerichtet werden. Da das Produkt des Projektes nicht direkt einem Kunden ausgeliefert werden muss, genügt eine lokale Implementierung der Applikation.

### **Projektanforderungen**

Ziel des gewählten Projektes ist es, ein Spiel als Webapplikation umzusetzen. Bei dem Spiel handelt es sich um eine gitterartige Oberfläche auf der verschiedenfarbige Spielsteine verteilt sind. Der Spieler hat die Möglichkeit Spielsteine, die in einer Gruppierung von gleichartigen Steinen vorliegen, auszuwählen und zu löschen. Beim Löschen werden alle Spielsteine der Gruppierung gelöscht und der Spieler erhält proportional zur Anzahl der gelöschten Objekte Punkte. Danach werden die entstandenen freien Plätze durch Aufrücken der Objekte von oben nach unten wieder gefüllt.

An dieser Stelle wurden drei Varianten erarbeitet wie ein Level abgeschlossen werden kann, um so für einen geringen Unterschied zwischen den Projekten zu sorgen, welcher sich aber nicht auf den Vergleich auswirken sollte. Generell ist ein Spiel zu Ende, wenn keine gleichfarbigen Gruppierungen von Spielsteinen mehr vorliegen und der Spieler somit nicht in der Lage ist, einen weiteren Spielzug durchzuführen. Im weiteren Verlauf sprechen wir hierbei von dem generellen Abbruchkriterium.

**Variante 1: Zeit**

Auf der einen Seite ist es möglich das Spiel nach einer vordefinierten Zeit, welche pro Level gesetzt wird, zu beenden. Der User muss somit in einer bestimmten Zeit genügend Punkte sammeln, um die Mindestpunktzahl zu erreichen und im Level aufzusteigen. Hierbei wird das Spielfeld nach jedem Zug wieder mit neuen Spielsteinen aufgefüllt, so dass der Spieler immer einen Spielzug durchführen kann.

**Variante 2: Punkte**

Andererseits kann man das Spiel auch so implementieren, dass die durch das Löschen entstehenden Lücken bestehen bleiben und höchstens bei kompletten leeren Zeilen die Blöcke aufgerückt werden. Dazu wird wieder eine Mindestpunktzahl festgelegt, die der Spieler erreichen muss, um ein neues Level zu erreichen. Der Spieler hat danach jedoch die Möglichkeit weitere Spielzüge durchzuführen, um noch mehr Punkte sammeln zu können. Das Spiel ist also erst dann beendet, wenn das generelle Abbruchkriterium erreicht wird.

**Variante 3: Blöcke**

Eine weitere Variante wäre es eine Mindestblockanzahl zu definieren, welche zum erfolgreichen abschließen eines Levels führt. Der Spieler muss es also schaffen, so viele Gruppierungen wie möglich zu löschen, damit zum Schluss nur noch die definierte Anzahl an Blöcken vorhanden ist. Auch in diesem Fall ist das Spiel erst dann fertig, sobald das generelle Abbruchkriterium gilt, um dem Spieler die Möglichkeit zu geben noch mehr Punkte zu erreichen, da diese später in der Rangliste relevant sind.

Ein Spieler kann ein Level so lange durchspielen, bis er das nächste Level erreicht hat. Außerdem soll eine Rangliste implementiert werden, welche die Spielergebnisse aller bisheriger Spieler vergleicht und dem Spieler so Feedback über seine Leistung gibt. Generell kann sich in einer Session immer nur ein Spieler anmelden. Es muss also nur ein Einzelspielermodus erarbeitet werden. Die Benutzer können sich mit ihrem Namen und dem Ergebnis in der Rangliste eintragen, aber eine Registrierung muss nicht stattfinden. Die Anwendung beginnt somit für jeden Spieler bei einer neuen Session beim ersten Level und er muss sich von neuem hocharbeiten.

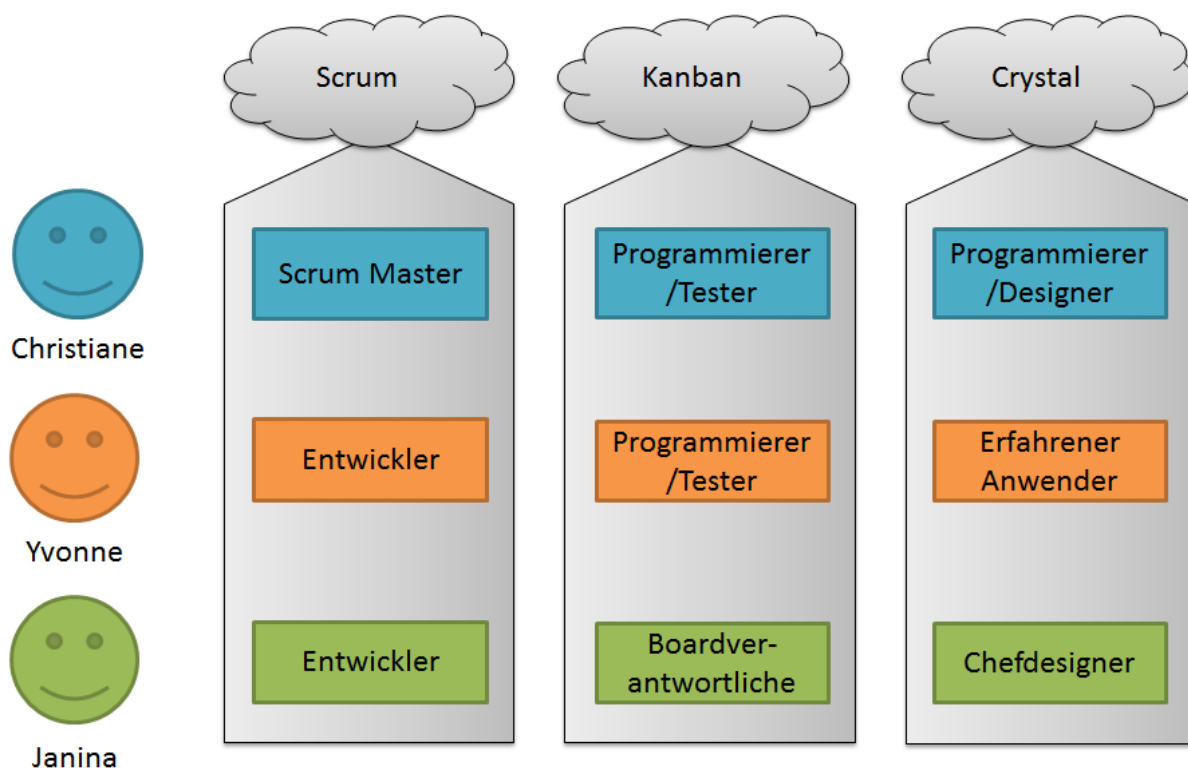


## 6. Vorbereitung der Durchführung

Bevor die Projekte mit den jeweiligen Prozessen durchgeführt werden können, müssen noch einige Vorbereitungen getroffen werden. Hier gilt es die Ressourcen optimal einzuteilen und die gemeinsamen Grundlegungen wie Organisation und Kommunikation zu definieren.

Für jedes Projekt müssen, wie bereits in Kapitel 2.1 erläutert, abhängig vom Prozess bestimmte Rollen besetzt werden. Zur Verfügung stehen drei personelle Ressourcen, deshalb wurde generell die Teamgröße auf drei Personen festgelegt. Daraus folgt, dass eine Ressource in jedem der drei Projekte eine Rolle einnehmen muss. Jedes Teammitglied dieser Studienarbeit hat sich auf einen der drei agilen Prozesse spezialisiert und erhält aus diesem Grund eine aktive Rolle in seinem Projekt. Um den Arbeitsaufwand und die Belastung jedoch so gering wie möglich zu halten, muss eine Person nur in zwei Projekten eine aktive Rolle einnehmen. Eine aktive Rolle beinhaltet die Tätigkeiten eines Programmierers, Designers, Testers oder Projektmanagers. In dem jeweils dritten Projekt wird daher eher eine passive Rolle vergeben, wie die des Scrum Masters, des Boardverantwortlichen oder des Anwenders.

Die folgende Abbildung zeigt die genaue Rolleneinteilung für jedes prozessspezifische Projekt.



Jedem Prozess fehlt nun noch eine wichtige Rolle. Bei Scrum muss die Rolle des Product Owner noch besetzt werden, welche quasi analog zum Auftraggeber bei Crystal Clear ist. Ebenso fehlt die Besetzung der Rolle des Kunden bei Kanban. Da diese Rollen hauptsächlich die Funktionen eines Sponsors ausführen, hat man sich dafür entschieden den Betreuer der Studienarbeit dafür einzusetzen, wodurch dieser auch einen aktiven Einblick in die Projekte erhält.

Neben der Rolleneinteilung wurden auch noch weitere organisatorische Festlegungen getroffen. Das Projekt wurde auf einen Zeitraum von 8 Wochen fixiert, was sich auf die Wahl der Iterationslängen bei Scrum und Crystal auswirkt. Generell wurde eine Iterationslänge von einer Woche festgelegt, wodurch sich das Projekt insgesamt in 8 Iterationen einteilen lässt. Bei Crystal wurden außerdem Lieferungszyklen definiert, welche jeweils zwei Iterationen umfassen. Es entstehen vier Lieferungen während des Projektes wodurch das von Crystal verlangte Minimum von zwei Lieferungen eingehalten wird. Für die Iterationsplanung wurde Montag festgelegt, damit die Iteration passend zum Wochenzyklus der Teammitglieder am Dienstag starten kann und am jeweils darauffolgenden Montag endet. Es wurde absichtlich Montag und nicht Sonntag als Iterationsende gewählt, da bei den geplanten Lieferungen und Reflektionen die Teammitglieder und gegebenenfalls der Betreuer sich zusammenfinden müssen. Kanban betreffen diese Vorgaben nicht, da es nicht direkt Iterationen vorschreibt und die Iterationslänge deshalb variabel gehalten wird.

Da die Projektteilnehmer parallel noch Vorlesungen besuchen müssen, stehen sie pro Tag nur jeweils 1-2 Stunden den Projekten zur Verfügung. Neben den bisher erwähnten Meetings wie Auslieferung oder Review und Reflektion bzw. Retrospektive sollen die agilen Prozesse außerdem durch tägliche Standup-Meetings unterstützt werden. Diese werden innerhalb der Iterationsplanung für die jeweilige Woche festgelegt, da einige Abhängigkeiten mit dem Vorlesungsplan beachtet werden müssen. Die Kommunikation findet dabei im besten Fall direkt in den Räumlichkeiten der DHBW statt oder als Alternative über das Video- und Sprachkonferenztool Skype statt.

Des Weiteren werden die beiden gewählten Tools für jedes Projekt eingerichtet. Dabei muss speziell darauf geachtet werden, dass sie identisch gepflegt sind und während der Projekte synchron verwendet werden, damit sie immer auf dem gleichen Stand sind. Gepflegt werden die Tools von der jeweiligen Prozessverantwortlichen. Das ist zwar eher unüblich, wenn diese als einfacher Entwickler in ihrem Projekt tätig ist, aber in dem Fall ist es durchaus sinnvoll, da sie sich am besten mit dem Prozess auskennt und weiß wie das Board zu pflegen ist. Weitere Prozessspezifische Vorkehrungen werden in dem nächsten Kapitel der eigentlichen Durchführung behandelt.

## **7. Durchführung des Projektes**

### **7.1 Crystal Clear**

### **7.2 Scrum**

### **7.3 Kanban**

## **8. Ergebnisse**

### **8.1 Analyse/Ursachenforschung**

#### **8.1.1 Crystal Clear**

#### **8.1.2 Scrum**

#### **8.1.3 Kanban**

### **8.2 Auswertung der Tools**

### **8.3 Gegenüberstellung der agilen Prozesse**

## 9. Fazit