DHBW
Duale Hochschule
Baden-Württemberg
Karlsruhe

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

Die folgende Studienarbeit entstand während der Theoriephasen des 5. und 6. Semesters im Rahmen des dualen Studiums. Sie befasst sich mit den drei ausgewählten agilen Softwareentwicklungsprozessen Crystal Clear, Scrum und Kanban, welche anhand eines selbst gewählten Beispielprojekts verglichen werden sollen.

Zuerst werden die drei agilen Prozesse vorgestellt und in folge dessen die Vergleichskriterien und das durchzuführende Projekt festgelegt. Den *Hauptteil (am Ende prüfen, ob das stimmt)* bilden die Dokumentation der Durchführung und die aus der praktischen Anwendung resultierenden Erfahrungen und Eindrücke. Zusätzlich wird der Einsatz von zwei selbstgewählten Projektmanagementtools in Bezug auf die Unterstützung der drei Softwareentwicklungsprozesse getestet und evaluiert.

Das Ergebnis dieser Arbeit bietet eine Gegenüberstellung der drei Softwareentwicklungsprozesse als Hilfe zur Wahl des passenden Prozesses bei einem gegebenen Projekt und Abwägung von verschiedenen Projektmanagementtools.

Abstract

Inhaltsverzeichnis

Zusammenfassung 3
Abstract 3
Inhaltsverzeichnis 4
Abbildungverzeichnis 6
Tabellenverzeichnis 6
Abkürzungsverzeichnis 6
1. Einleitung 7
2. Agile Softwareentwicklung 9
2.1 Allgemein 9
2.2 Crystal Clear 14
2.3 Scrum 17
2.4 Kanban 21
3. Vergleichskriterien für die Prozesse 27
3.1 Begriffserklärung 27
3.2 Vergleichsmatrix 29
4. Vergleich der Tools 34
4.1 Vorauswahl 35
4.2 Bewertungskriterien 37
4.3 Allgemeine Kriterien 38
4.4 Bewertungskriterien für Scrum 40
4.5 Bewertungskriterien für Kanban 41
4.6 Bewertungskriterien für Crystal Clear 43
4.7 Nice to have – Kriterien 44
4.8 Ergebnis und Auswahl 47
5. Wahl des Projektes 48
6. Vorbereitung der Durchführung 53
7. Durchführung der Projekte 56
7.1 Überblick 56
7.2 Gemeinsame Aktivitäten 59
7.3 Crystal Clear 62
7.4 Scrum 70
7.5 Kanban 76
8. Ergebnisse 85
8.1 Gegenüberstellung der Prozesse 85
8.2 Auswertung der Tools 95

9. Fazit 100

Abbildungverzeichnis

Es konnten keine Einträge für ein Abbildungsverzeichnis gefunden werden.

Tabellenverzeichnis

Es konnten keine Einträge für ein Abbildungsverzeichnis gefunden werden.

Abkürzungsverzeichnis

RUP	Rational Unified Process

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.

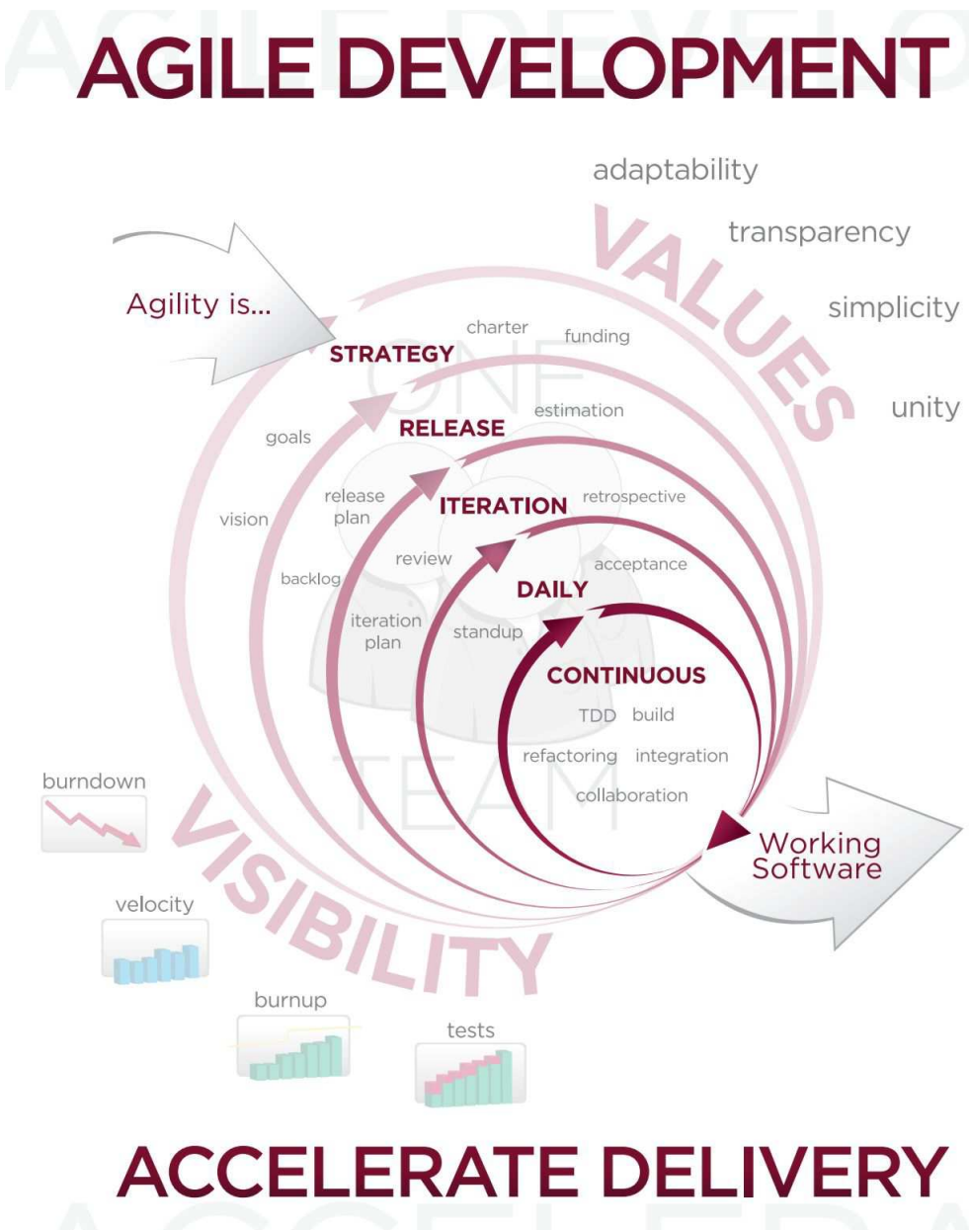

AGILE DEVELOPMENT
adaptability
transparency
simplicity
unity
VALUES
Agility is...
STRATEGY
charter
funding
RELEASE
estimation
goals
vision
release plan
ITERATION
retrospective
review
acceptance
backlog
DAILY
iteration plan
standup
CONTINUOUS
TDD
build
refactoring
integration
collaboration
Working Software
VISIBILITY
burndown
velocity
burnup
tests
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“1

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

[1] 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 Detailierungsgrad 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).

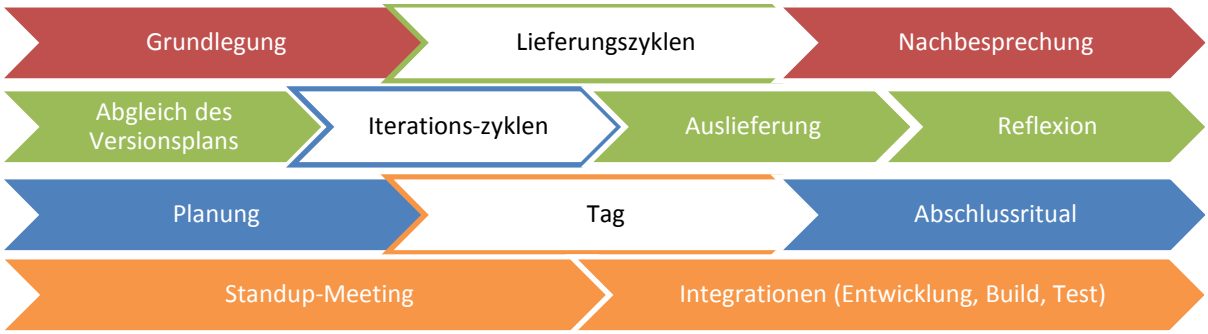
Eigenschaften

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 aus 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 Vison 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: unabhänig 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

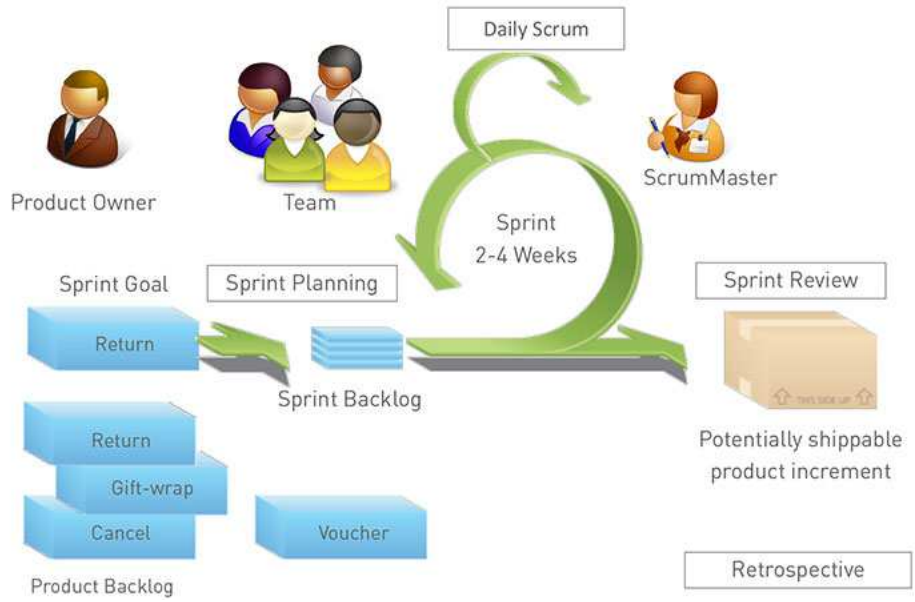
Während eines Sprints werden alle Tasks mit geschätzter Zeit und fertige Tasks mit benötigter Zeit versehen. Diese Zeiten werden dann im Burndown Chart grafisch dargestellt, dies dient dazu, dass das Team den Vorschritt des Sprints verfolgen kann und auch sieht ob die Zeitplanung eingehalten werden kann.

Der Sprint Review

Am Ende eines Sprints steht der Sprint Review, daran nimmt das Team, der Scrum Maser 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 Print 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 comunity 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 Optimierungsschrit 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). 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 sid erlaubt.

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

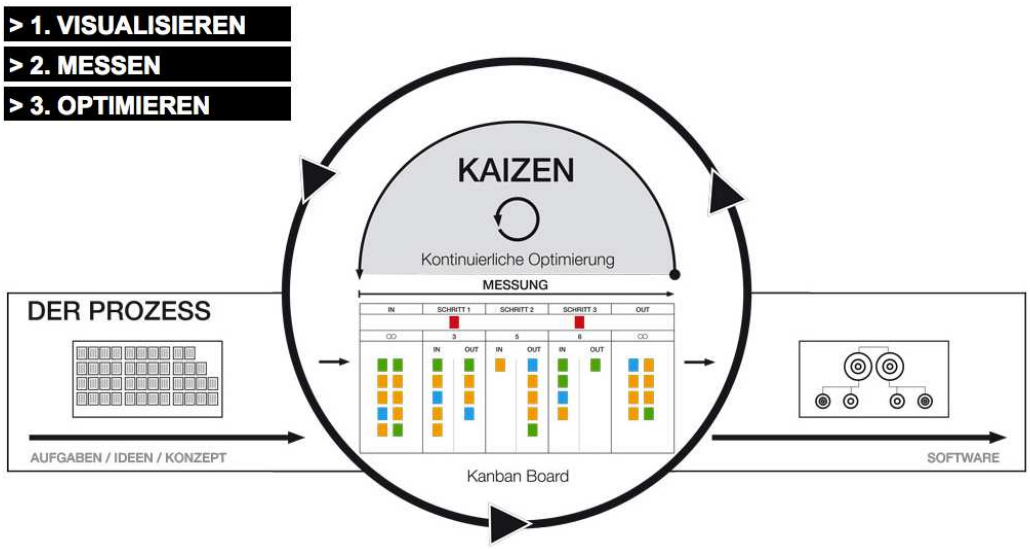


Abbildung 1: http://kanban-plakat.de/

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 gwöhnlich 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

derselbe: 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 durchlaugen 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 Paractise 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 weiße diese erfolgt.

Commitment/ Definition of Done: Ist die Festlegung wann eine User Story als erledigt makiert 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
Iterationen	- Iterationszyklen innerhalb einer Lieferung	- keine Pflicht - variable Länge oder gleichmäßige Iterationen	- gleichmäßige Länge - in der Regel 1-4 Wochen
Teamgröße	- 2 bis 6	- keine Begrenzung	- 7 bis 10
Rollen	- Chefdesigner - Endanwender - Auftraggeber - Programmierer/ Designer	- keine Vorgaben	- Scrum Master - Product Owner - Scrum Team
Techniken	- Verschiedenste agile Techniken	- WIP-Limit - Service-Level Agreement-Klassen (SLAs)	- Verschiedenste agile Techniken
Lieferung	- mindestens 2 Lieferungszyklen - nach einer Iteration	- Release - am Ende einer Iteration oder festes Datum	- nach jedem Sprint → gezeigt im Review
Änderung der Anforderungen	- vor jeder Iteration möglich	- immer möglich	- vor jedem Sprint möglich
Änderung der Arbeitsweise	- nach jeder Lieferung (Reflexion der Arbeitsweise)	- erwünscht (kontinuierliche Verbesserung) - immer möglich	- nach jeder Retrospektive

Kriterien	Crystal	Kanban	Scrum
Meetings/ Kommunikation	- osmotische Kommunikation - Reflexionsmeeting nach Iteration - weitere Meetings empfohlen	- keine Pflicht: - Daily Standup - After Meeting - Queue Replenishment Meeting - Release Planning Meeting - Triage - Issue Log Review - Retrospektive	- Sprint Planning Meeting - Daily Scrum - Review - Retrospektive
Dokumentation	- empfohlen Beispiele: - Projektplan - Versionsplan - Risikoliste	- keine Pflicht - Kanban Board - elektronisches Trackingsystem	- Scrumboard - elektronisches Trackingsystem
Definition of Done/ Commitment	- nicht festgesetzt	- nicht Pflicht	- Product Owner und Team gemeinsam
Größe der Tasks	- nicht vorgeschrieben - generell möglichst klein	- möglichst gleich groß	- möglichst klein - max. 1 Tag/Task
Aufwands-schätzungen	- Blitzplanung - Anpassung vor jeder Iteration	- nicht Pflicht	- Sprint Planning Meeting - Abschätzung mit Story Points

Kriterien	Crystal	Kanban	Scum
Priorisierung	- in Blitzplanung durch Auftraggeber - Anpassungen während des Projektes	- Queue Replenishment Meeting: Nicht-Entwickler - während Umsetzung: Entwickler oder FIFO - Ausnahme: SLAs	- Product Owner - Anpassungen durch Team bei Abhängigkeiten
Empirie	- nicht gegeben	- Cumulative Flow - Lead und Cycle Time (Durchsatz)	- Velocity (Story Points)

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 Länger 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. Chrystal 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 einen 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 einsetzten 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 lauf 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 Parctice. 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 Stroies 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 größer der Tasks ist bei allen Prozessen sehr ähnlich. Dabei sollte darauf geachtet werden das die einzelnen Tasks eine geringe Größe haben. Jedoch sollte als Spezialisierung bei Kanban darauf geachtet werden das die einzelnen Tasks ungefähr die gleiche Größe haben. Bei Scrum hingegen gilt es als Sinnvoll darauf zu achten, das ein einzelner Task in der Größe nur so groß ist das 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 Storiepoints innerhalb eines Sprints abarbeitbar sind.

4. Vergleich der Tools

Bei der Durchführung eines Projektes ist es stets 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 prozess-spezifische Anforderungen gestellt wurden.

Im Rahmen dieser Arbeit werden hierfür zwei Tools ausgewählt, mit deren Hilfe verschiedene Projektmanagement-Aspekte während der Projektdurchführung erleichtert werden sollen. Im Anschluss an die Durchführung wird eruiert, welche Vor- und Nachteile den Teammitgliedern aufgefallen sind und entschieden, welches Tool sich besser für jedes Prozessmodell eignet.

Zur Auswahl der beiden Tools wurden verschiedene Bewertungskriterien herangezogen, die in diesem Kapitel näher beleuchtet werden. Zunächst traf das Team eine Vorauswahl von fünf Tools. Danach entschied es sich sowohl für allgemeine als auch prozess-spezifische Kriterien für Scrum, Kanban und Crystal Clear, die das Tool erfüllen sollte. Als fünfter Bewertungsschwerpunkt wurden Wünsche der Teammitglieder in einer Nice-To-Have-Tabelle zusammengefasst. Die fünf Tools in der Endauswahl wurden mit verschiedenen Mitteln darauf geprüft, inwieweit sie diese Kriterien erfüllen. Die dabei vergebenen Punkte wurden aus den vier erstgenannten Bereichen ohne Nice-To-Have-Features aufaddiert und die beiden Tools mit der höchsten Gesamtpunktzahl ausgewählt. Die Wünsche der Teammitglieder dienten lediglich als Tie-Breaker, sollten Tools in ihrer Punktzahl sehr nah beieinander liegen und sich eines durch besonders interessante zusätzliche Features hervortun.

4.1 Vorauswahl

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.

Es wurden verschiedene Online-Quellen hinzugezogen, um eine Vorauswahl und später auch die endgültige Wahl zu treffen. Die Hauptquelle zur Vorauswahl war die Webseite FindTheBest[2], 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.

[2] 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.

Tabelle 1: Vorauswahl der Tools

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

4.2 Bewertungskriterien

Im Folgenden werden die im Vorfeld erstellten Auswahlkriterien aus den fünf Bereichen tabellarisch zusammengefasst und kurz erläutert. Für die Grundlage der Bewertung orientierte sich das Team an der Idee des Qualitätshauses, 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). Die eher geringe Spannweite bei der Merkmalsbewertung kommt daher, dass nicht alle Tools ausgiebig getestet werden können, da beispielsweise keine freien Demos existieren und es somit schwierig wäre, eine noch differenziertere Einschätzung lediglich an Hand von Fotos, Grafiken oder Beurteilungen in Textform durchzuführen. Da dies zu sehr ähnlichen Punktzahlen der Tools führen kann, wurde der fünfte Bereich mit Nice-To-Haves als Entscheidungshilfe aufgenommen.

Tabelle 2: Bewertungsskala und ihre Bedeutung

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

4.3 Allgemeine Kriterien

Tabelle 3 gibt einen Überblick über allgemeine, nicht prozess-spezifische 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 in die Vorauswahl aufgenommen wurde, dass diese nicht erfüllt.

Tabelle 3: allgemeine Kriterien

	Prio	**Genius inside**	**Jira**	**Pivotal Tracker**	**Vision Project**	**Yodiz**
Erfahrung	3	0	2	1	0	0
Allgemeine Funktionen						
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

Die Erfahrung der Teammitglieder mit dem Tool wurde ebenfalls in die Bewertung 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 wichtig, aber 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.

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.

4.4 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 Scumboard 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.

Tabelle 4: Scrum-Kriterien

	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

4.5 Bewertungskriterien für Kanban

Obwohl Kanban allgemein nur sehr wenige Merkmale besitzt, die zwangsweise zum Prozess gehören, entschied sich das Team dafür, auch einige der freiwilligen Techniken als Kriterien aufzunehmen (siehe Tabelle 5).

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.

Tabelle 5: Kanban-Kriterien

	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

4.6 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.

Tabelle 6: Crystal-Kriterien

	Prio	**Genius inside**	**Jira**	**Pivotal Tracker**	**Vision Project**	**Yodiz**
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

4.7 Nice to have – Kriterien

Da bereits beim Aufstellen und Ausfüllen der prozess-spezifischen Kriterien bewusst war, 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.

Tabelle 7: Nice-To-Have-Kriterien

	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

Kalender → z.B. Termine	2	2	2	2	2	2
Time Tracking	2	2	2	2	2	2
GitHub-Integration o.Ä.	3	0	2	0	0	2
Summe		35	48	36	48	54

4.8 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, die später es in der Projektdurchführung so synchron wie möglich gehalten werden sollen.

Tabelle 8: Gesamtauswertung

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

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. Somit kann ein Teammitglied eine Aufgabe, die es eventuell bereits in einem der anderen Projekte bearbeitet hat, viel schneller lösen. 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.

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 (Scrum)	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 (Crystal)	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 Mindestpunktanzahl 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 (Kanban)	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.

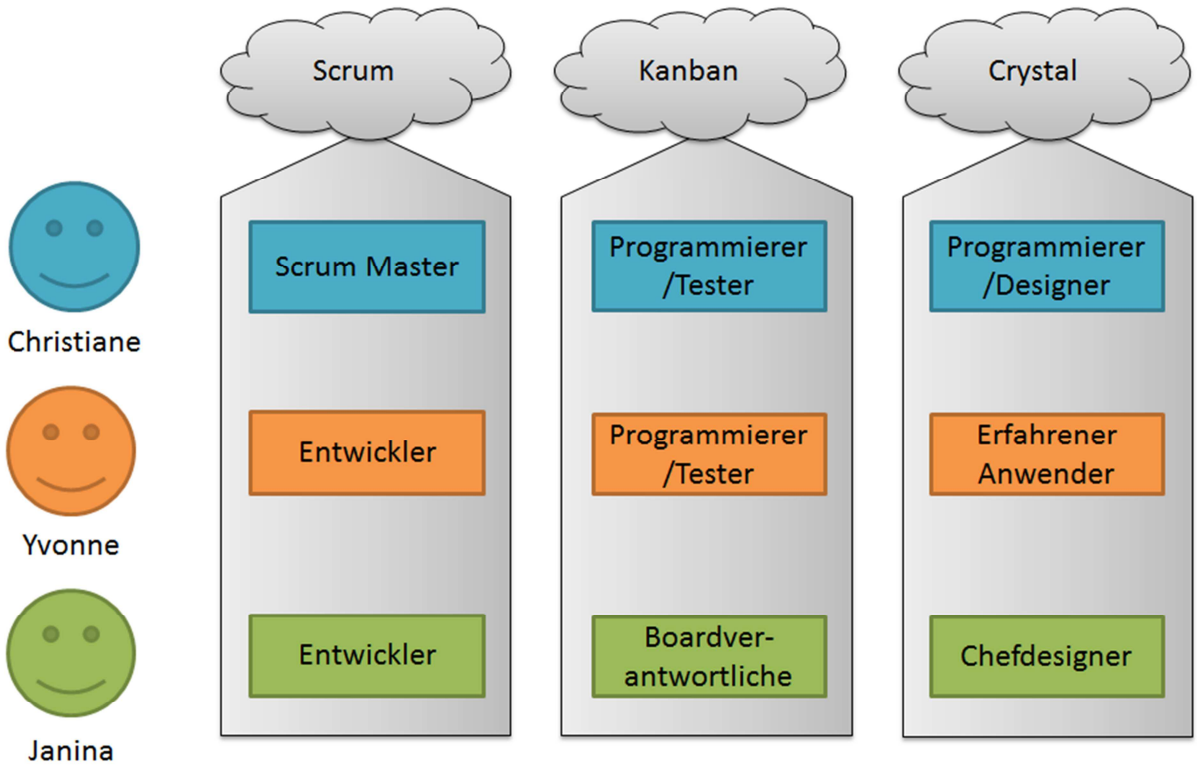
Ein Spieler kann ein Level so lange durchspielen, bis er das nächste Level erreicht hat. Außerdem soll eine Rangliste implementiert werden, welche die Spielergebnisse aller bisheriger Spieler vergleicht und dem Spieler so Feedback über seine Leistung gibt. Generell kann sich in einer Session immer nur 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 **Fehler! Verweisquelle konnte nicht gefunden werden.** 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 der Projekte

7.1 Überblick

Nach der allgemeinen Vorbereitung der Durchführung, soll im Folgenden speziell auf die Realisierung und Umsetzung der drei ausgewählten agilen Prozesse eingegangen werden.

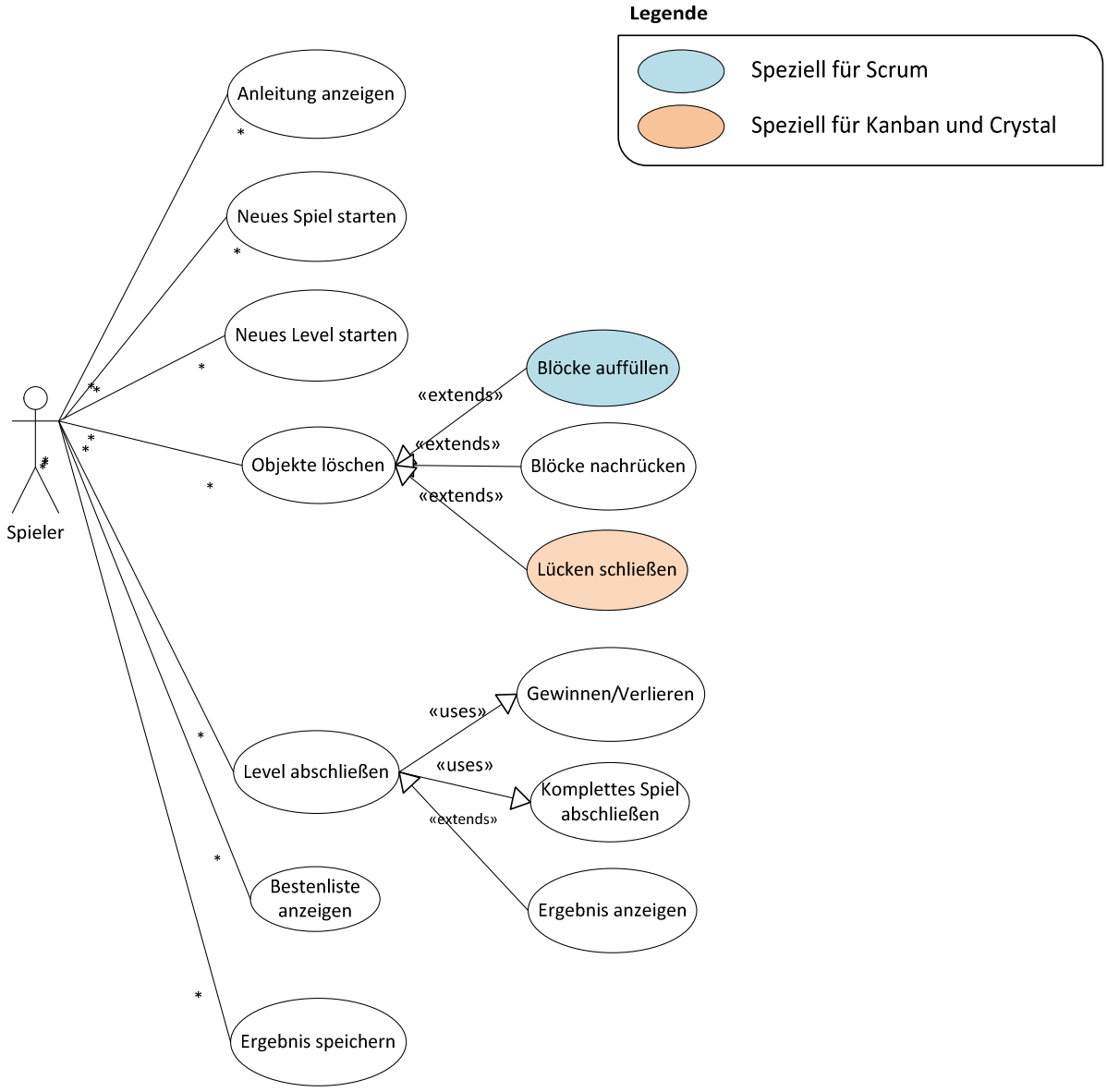
Der Einstieg erfolgt über eine kurze Vorstellung der Use Cases und ein sogenanntes Prozess-Diary, welches einen Überblick über den Projektverlauf der einzelnen Prozesse geben soll.

Bevor es zur eigentlichen Detailbetrachtung der einzelnen Prozesse geht, werden zuerst einige Aspekte betrachtet, die die Prozesse in ihrer Durchführung gemeinsam hatten. Danach wird in der Detailbetrachtung der einzelnen Prozessdurchführungen vor allem auf die Frage eingegangen, wie der Prozess umgesetzt wurde und welche Maßnahmen getroffen wurden, um prozesskonform zu sein. Darüber hinaus werden auch Ausschnitte aus den jeweiligen Arbeitsergebnissen gezeigt, da diese eine große Rolle in der Umsetzung spielen.

Use Cases

Vor der separaten Durchführung der drei Projekte wurden die Anforderungen anhand von Use Cases festgelegt. Normalerweise werden diese vom Auftragsgeber als Lastenheft eingereicht. Da die Projekte speziell für die Analyse der Prozesse definiert wurden und somit kein konkreter Kunde existiert, wurden die Anforderungen vom Projektteam erarbeitet.

In der weiteren Durchführung der Projekte werden die im Folgenden aufgeführten Use Cases (siehe Abbildung) als vom Auftraggeber gegeben betrachtet.



Prozess-Diary

Das Prozess-Diary soll eine Übersicht des Ablaufs der drei Projekte liefern. Dabei werden schon einige Unterschiede gezeigt. auf die jedoch erst später eingegangen wird. Die enthaltenen User Stories stammen aus den zuvor definierten Anforderungen.

Zeit		Crystal Clear	Scrum	Kanban
Woche 1	10.2 – 17.2	(1) Entwicklungsumg. (2) Boards konfigurieren	(1) Entwicklungsumg. (2) Boards konfigurieren (4) Projektplanung	(1) Entwicklungsumg. (2) Boards konfigurieren (4) Projektplanung
			Review 1	**Release 1**
Woche 2	18.2 – 24.2	(3) Grundlagenrecherche (4) Projektplanung (5) Design (6) Neues Spiel starten (7) Anleitung	(3) Grundlagenrecherche (5) Design	(3) Grundlagenrecherche (5) Design (6) Neues Spiel starten (11) Spielfeld
		Lieferung 1	**Review 2**	
Woche 3	25.2 – 3.3	(8) Spiel abbrechen (9) Endergebnis	(6) Neues Spiel starten (11) Spielfeld	
			Review 3	
Woche 4	4.3 – 10.3	(10) Bestenliste (11) Spielfeld (12) Statusanzeige	(13) Objekte löschen	**Release 2**
		Lieferung 2	**Review 4**	(8) Spiel abbrechen (23) Spielzüge prüfen
Woche 5	11.3 – 17.3	(13) Objekte löschen	(22) Board auffüllen (12) Statusanzeige	
			Review 5	
Woche 6	18.3 – 24.3	(14) Lücken schließen (15) Level abschließen (16) Zwischenergebnis (17) Neues Level starten	(8) Spiel abbrechen (18) Zeit anzeigen (15) Level abschließen	(13) Objekte löschen (14) Lücken schließen
				Release 3
		Lieferung 3	**Review 6**	
Woche 7	25.3 – 31.3	(18) Zeit anzeigen (19) Spiel abschließen (20) Ergebnis speichern (21) Improvements	(17) Neues Level starten (19) Spiel abschließen (16) Zwischenergebnis (9) Endergebnis	(15) Level abschließen (17) Neues Level starten
			Review 7	
Woche 8	1.4 – 7.4	(21) Improvements	(7) Anleitung (10) Bestenliste (20) Ergebnis speichern	(9) Endergebnis (12) Statusanzeige (19) Spiel abschließen (10) Bestenliste (20) Ergebnis speichern (7) Anleitung
		Lieferung 4	**Review 8**	**Release 4**

7.2 Gemeinsame Aktivitäten

Obwohl sich die drei angewendeten Prozesse in vielen Punkten unterscheiden, gab es auch einige Gemeinsamkeiten, welche die Durchführung betreffen. Dazu gehören die Durchführung eines Planungsmeetings am Anfang einer Iteration, ein tägliches Standup-Meeting sowie die grundlegende, initiale Umsetzung der Boards. Des Weiteren wurde zur Sicherung der Codequalität und Testabdeckung regelmäßig in allen Projekten SonarQube ausgeführt. Die eben genannten Punkte werden in diesem Kapitel allgemein für alle drei Prozesse beschrieben.

Planungsmeeting

Jede Iteration der drei Projekte, unabhängig von ihrer Länge, begann mit einem Planungsmeeting zur Festlegung der durchzuführenden User Stories. In Scrum wurde dieses Planning in drei Schritten durchgeführt. Zuerst wurden nach der vom Product Owner vorgegebenen Priorisierung im Product Backlog die obersten User Stories mit allen notwendigen Tasks versehen. Im späteren Verlauf des Projekts wurde dies zum Teil während des Sprints erledigt, damit das Planning nicht zu lange andauert, und hier nur noch geprüft, ob alle Tasks vorhanden sind. Danach fand die Abschätzung der User Stories mit Storypoints statt und das Team entschied, wie viele der User Stories bearbeitet werden könnten. Hierbei wurden auch eventuelle Ausfälle von Teammitgliedern durch Urlaub in die Planung. Im Verlauf des Projekts verbesserten sich diese Schätzungen auf Grund der steigenden Erfahrung.

Bei Kanban verlief die Planung fast analog zu Scrum, allerdings wurde zum Ende des Projekts die Menge der in die Iteration genommenen User Stories durch das WIP-Limit bestimmt. Dieses berechnete sich jedoch ebenfalls aus der Erfahrung der vorherigen Itertionen.

Auch das Planning für Crystal verlief ähnlich dem der anderen beiden Prozesse. Der Hauptunterschied bestand darin, dass die User Stories bereits aus der Blitzplanung vorabgeschätzt und einer Lieferung zugeordnet waren sowie ihre Tasks erhielten. Somit wurden im Planungsmeeting lediglich die User Stories der entsprechenden Lieferung daraufhin überprüft, ob die Tasks tatsächlich vollständig und die Schätzung noch zutreffend waren. Ansonsten wurden fehlende Tasks ergänzt, die User Stories neu abgeschätzt und anschließend entsprechend der Priorität und der zur Verfügung stehenden Zeit einer der 2 Iterationen der Lieferung zugeordnet.

Tägliches Standup-Meeting (Daily)

Tägliche Meetings wurden sowohl für Scrum als auch Kanban und Crystal zur Synchronisierung der Teammitglieder benötigt. Sie fanden in der Woche, wenn zeitlich möglich, vor der ersten Vorlesung am Morgen und zumindest an einem der Wochenendtage ebenfalls vormittags statt. Während ihres Verlaufs wurde rekapituliert, wer welche Tasks seit dem letzten Daily bearbeitet hatte und was am selben Tag erledigt werden konnte. Außerdem bestand die Möglichkeit, Probleme oder Fragen anzusprechen und um Hilfe zu bitten. Schließlich wurden die elektronischen Boards in beiden Tools aktualisiert und synchronisiert. Mit ihrer Hilfe war die Durchführung der Meetings sogar möglich, ohne dass alle Teammitglieder am gleichen Ort waren. Hierfür telefonierten sie mittels Skype.

Alle drei Dailys wurden immer direkt hintereinander durchgeführt, wobei die Reihenfolge nicht festgelegt war. Dadurch sah oder hörte jedes Teammitglied auch ein Meeting für ein Projekt, bei dem es kein Entwickler war. Für Scrum und Kanban waren sie stattdessen der Scrum Master bzw. der Boardverantwortliche. Beide Rollen erlaubten oder forderten sogar eine mindestens semi-aktive Rolle im Standup. Sie mussten auf das Einhalten der jeweiligen prozess-spezifischen Daily-Regeln achten und auf Verstöße oder Board-Unterschiede hinweisen. Bei Crystal Clear hingegen genügten die beiden Entwickler.

Board

Anhand der verwendeten Boards konnten die Aufgaben an die entsprechenden Entwickler verteilt und gegebenenfalls neu zugewiesen werden. Darüber hinaus waren die Boards ein hilfreiches Mittel, um den Fortschritt der User Stories in der entsprechenden Iteration zu sehen. So konnten schnell Maßnahmen ergriffen werden, wenn es den Anschein hatte, dass die Aufgaben nicht bis zum Ende der Iteration erledigt werden konnten.

Vor Beginn der Projektarbeit einigten sich sie Teams auf einen bestimmten Ablauf, dem alle Tasks bei ihrer Abarbeitung folgen sollten (Abbildung XYZ). Dieser wurde durch die Erweiterung von JIRAs Standard-Workflow so gut es ging auf die dortigen Boards und deren Spalten abgebildet und anschließend mit den durch Yodiz bereitgestellten Möglichkeiten in das jeweils entsprechende Yodiz-Board übertragen. Neu ist dabei nur eine Review-Spalte hinzugekommen, die jedes Ticket durchlaufen muss, bevor es geschlossen werden darf. Dabei muss ein anderer Entwickler sowohl die Funktonalität auf seinem System als auch den geschriebenen Code überprüfen.

Die Synchronisierung der Boards war eine der wichtigsten, aber auch schwierigsten Aufgaben während der Projektdurchführung, mit der alle drei Teams zu kämpfen hatten. Während der Ablauf und Boardaufbau für Scrum und Crystal im gesamten Projektverlauf gleich blieb, veränderte es sich bei Kanban mehrfach. Dies wird im Kapitel zur Durchführung von Kanban näher ausgeführt.

Abbildung XYZ

SonarQube

Für die regelmäßige Kontrolle der Testabdeckung sowie die Überprüfung der Codequalität an Hand einiger Metriken und Java Coding Guidelines wurde SonarQube ab der 3. Woche mindestens einmal wöchentlich am Iterationsende ausgeführt. Dieses Codeanalyse-Werkzeug stellt in einer Webseite überblicksartig verschiede Kennzahlen wie LOC (Lines Of Code) oder die Testabdeckung in Prozent dar (Screenshot XYZ) und half den Teams Problem- oder Hotspot-Klassen zu erkennen und den Code entsprechend der Hinweise zu refaktorieren. Diese Verbesserungen wurden im Sinne der drei Prozesse und der agilen Software-Entwicklung im Allgemeinen zeitnah, also meist bereits in der folgenden Iteration, durchgeführt.

Screenshot XYZ SonarQube-Dashboard von Crystal Clear (Datum)

7.3 Crystal Clear

Dieser Abschnitt soll speziell die Durchführung von dem Prozess Crystal Clear dokumentieren. Zuerst wird ein genereller Überblick über den festgelegten Ablauf des gesamten Projekts, der Lieferungen und der Iterationen vermittelt. Aufbauend auf diesem Projektablauf wird dann im Einzelnen auf spezielle Schritte, angewandte Techniken und daraus resultierenden Arbeitsergebnisse eingegangen. Darüber hinaus wird aufgezeigt, welche Maßnahmen getroffen wurden, um Crystal Clear konform zu bleiben.

Als Hilfe zur Umsetzung von Crystal Clear wurde vor allem das Buch “Crystal Clear” von Alistair Cockburn zu Rate gezogen, da er den Prozess entworfen hat und in seinem Buch viele Tipps zur Durchführung gibt. Darüber hinaus beschreibt er einige Techniken, die bei der Durchführung von Crystal Clear helfen können. Da Crystal generell sehr flexibel ist und keine konkreten Techniken vorgeschrieben werden, kann man die Durchführung des Prozesses sehr individuell gestalten.

Im Gegensatz zu den flexiblen und optionalen Techniken, folgt der Ablauf eines Crystal Clear Projekts jedoch immer einem bestimmten Muster, welches in Tabelle 9 grob dargestellt wird. Zu beachten ist hierbei, dass die dargestellte Lieferung, ebenso wie die Iteration, als Zyklus verstanden werden muss, welcher mehrfach durchlaufen wird.

Tabelle 9: Prozessablauf

Grundlegung			• Projektplan • Methodik oder Konventionen • Domänenmodell
Lieferung	**Abgleich des Versionsplans**		• Projektplan aktualisieren
	Iteration	**Planung**	• Prioritäten festlegen • Einschätzung der Dauer
		Tägliche Aktivitäten	• Standup-Meeting • Entwicklung, Integration, Build, Test
		Abschlussritual	
	Auslieferung		
	Abschluss mit Reflexion		• Reflexionsworkshop
Nachbesprechung			

Die Grundlegung

Das Projekt startete mit einer Grundlegung, bei der die flexiblen Parameter des Prozesses, wie Länge der Lieferungs- und Iterationszyklen oder anzuwendende Techniken, festzulegen sind.

Da die Anforderungen bereits außerhalb des Projekts global für alle drei Prozesse definiert wurden, konnte man diese direkt als gegeben annehmen und mit der Planung des Projektes starten. Dazu wurde die von Alistair Cockburn empfohlene Blitzplanung als Anhaltspunkt verwendet, an der alle verfügbaren Teammitglieder (Chefdesigner, Programmierer und Anwender) teilnahmen.

Die Teammitglieder haben dabei zuerst gemeinsam alle Use Cases auf Post-its festgehalten und die dazugehörigen Tasks gesammelt. Durch die Zusammenarbeit und Kommunikation konnten die Mitglieder sich so gut ergänzen und kamen schnell zu einem vollständigen Ergebnis. Danach mussten die Use Cases noch in eine Abhängigkeitsreihenfolge gebracht werden. Dazu wurden sie mit ihren zugehörigen Tasks an die Wand geheftet und je nach dem, ob sie parallel oder nacheinander abgearbeitet werden müssen, nebeneinander oder untereinander platziert. Außerdem wurde bei der Festlegung der Reihenfolge eine Priorisierung durchgeführt.

Nach dem die Reihenfolge geklärt war, wurde die Zeit für jeden Task geschätzt und danach auf den zugehörigen Use Case aufaddiert. Dabei wurde vor allem darauf geachtet, dass auch genügend Zeit für Tests geschätzt wird. Anhand der geschätzten Zeiten und der bereits festgelegten Lieferungslängen von 4 Wochen wurden die einzelnen Use Cases dann jeweils einer Lieferung zugeordnet. Das Ergebnis dieser Blitzplanung (siehe Abbildung 1) wurde anschließend in die Tools übertragen.

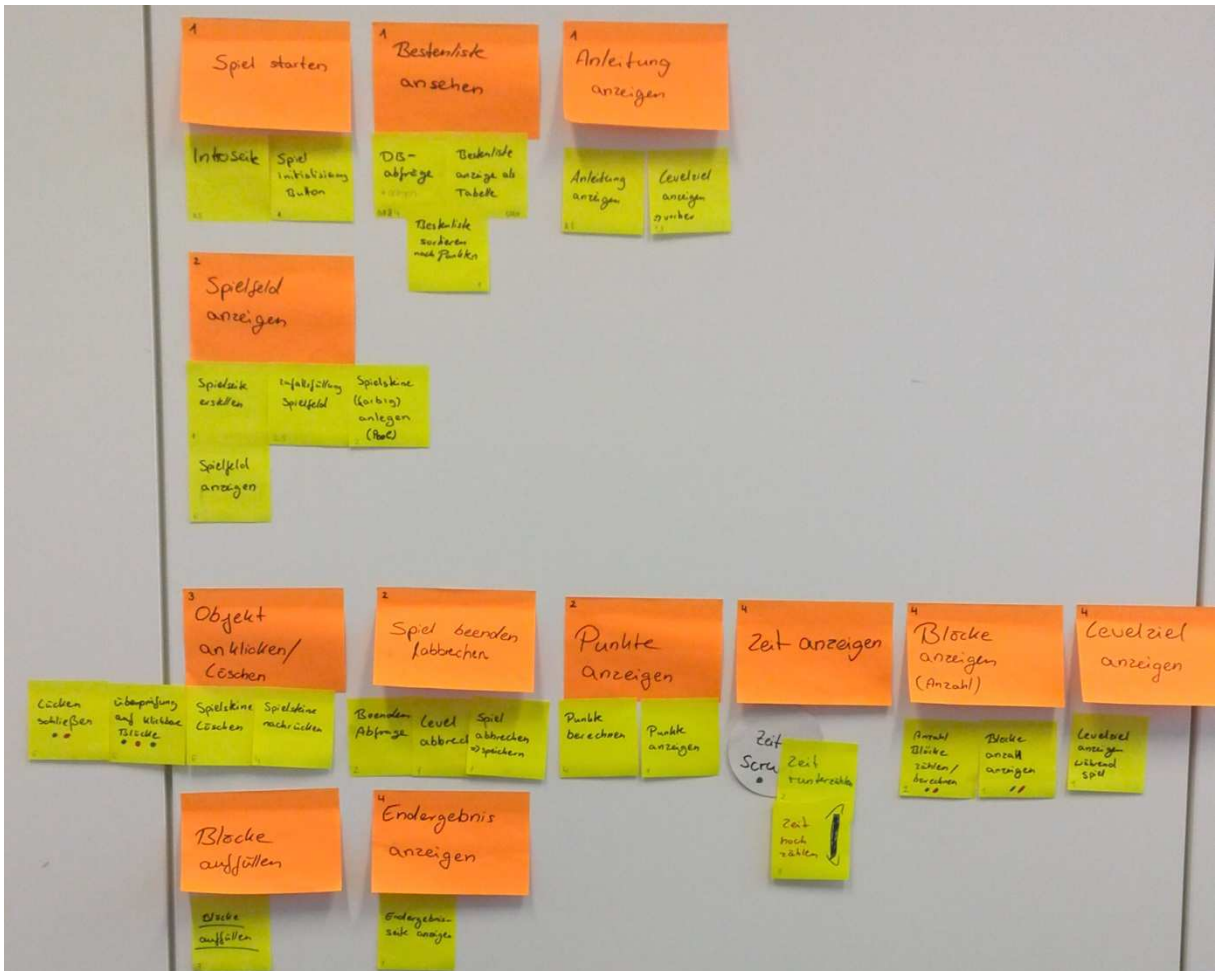


Abbildung 2: Ausschnitt aus der Blitzplanung

Nach der Projektplanung hat der Chefdesigner gemeinsam mit dem Programmierer die Konventionen festgelegt. Dabei hat man sich zum Beispiel für die Einhaltung des Modell-View-Controller Prinzips entschieden und hat bestimmte Richtlinien für die Namensgebung von Variablen, Methoden und Klassen festgelegt. Außerdem wurden Regeln zum Coding (Einzug, Reihenfolge von bestimmten Methoden) sowie für Kommentare und Zugriffsrechte zwischen den Klassen definiert. Neben den Konventionen wurden zudem auch ein paar Techniken wie Pair-Programming und Refactoring als Einstieg festgelegt.

Abschließend wurde darüber hinaus noch das Design der verschiedenen Seiten und die Klassenarchitektur entworfen.

Die Lieferungen

Nach der Grundlegung des Projekts begann die eigentliche Umsetzung. Das gesamte Projekt wurde in vier Lieferungen mit je zwei Wochen eingeteilt. Zu Beginn einer Lieferung wurde der Projektplan, der während der Blitzplanung entstanden ist, als Vorgabe benutzt, was nach den zwei Wochen erledigt sein muss. In diesem Schritt wurden außerdem Anpassungen am Projektplan gemacht, da durch die gewonnene Erfahrung die Aufgaben besser eingeschätzt werden konnten. Dabei kam es auch zu Umpriorisierungen der Use Cases in Absprache mit dem Auftraggeber.

Zum Schluss jeder Lieferung wurde dem Anwender ein Prototyp präsentiert. Der Anwender hatte so die Chance die bisher entwickelten Funktionen auszuprobieren und gegebenenfalls abzunehmen oder Verbesserungswünsche zu äußern, welche umgehend als sogenannte Bugs in die Tools aufgenommen wurden.

Die Iterationen

Während einer Lieferung wurden zwei wöchentliche Iterationen durchgeführt, welche jeweils separat wie zuvor in dem Abschnitt “allgemeinen Aktivitäten” beschriebenen Planungsmeeting geplant wurden.

Jede Iteration verlief nach einem gewissen Muster. Nach der Planung wurden jeden Tag im Schnitt bis zu 2 Stunden für die Bearbeitung der Use Cases verwendet. Außerdem wurde versucht jeden Tag ein sogenanntes Standup-Meeting durchzuführen, um schnell Schwierigkeiten und Probleme zu erkennen und sich gegebenenfalls helfen zu können. Gearbeitet wurde oftmals in den Räumlichkeiten der DHBW, um gemeinsam zum Beispiel durch Pair-Programming an Aufgaben zu arbeiten oder sich gegenseitig zu unterstützen. Dadurch wurde im Sinne der osmotischen Kommunikation die Zusammenarbeit gefördert und die Informationsaufnahme erleichtert. Die meiste Bearbeitungszeit wurde jedoch von zu Hause verrichtet, wodurch die Kommunikation auf Chats oder kurze Telefonate eingeschränkt war.

Darüber hinaus konnten Use Cases aus späteren Iterationen eingereiht werden, wenn das Team schneller als geplant fertig war. Dabei hatten die Use Cases der laufenden Lieferung die höchste Priorität. Danach erst konnten die Use Cases aus anderen Lieferungen in Absprache mit dem Auftraggeber ausgewählt werden. Aus diesem Grund wurde zum Beispiel der Use Case „Neues Spiel starten“ nicht wie geplant in der vierten Lieferung sondern in der zweiten Lieferung realisiert.

Nach Abschluss einer Iteration wurden Use Cases, die nicht vollständig fertiggestellt wurden, mit in die nächste Iteration übernommen und erhielten dort die höchste Priorität. Generell wurde versucht diese Übernahme der Aufgaben in anschließende Iterationen zu vermeiden. War dies jedoch nicht möglich sollte immerhin der Task in der gleichen Lieferung abgeschlossen werden. Hauptsächlich am Anfang des Projektes als die Einschätzungen noch nicht optimal waren, kam es zu solchen Fällen. Ein Beispiel dafür ist die Anzeige der Bestenliste, welche in Iteration 2 begonnen und in Iteration 4 beendet wurde. Dabei kam es ausnahmsweise auch zu einem Wechsel der Lieferung. Erklären lässt sich dies, durch die damalig mangelnde Erfahrung mit Hibernate und JSF, wodurch die Aufgabe erst nach der intensiven Einarbeitung in die Themen bearbeitet werden konnte.

Die Reflexionen

Das von Alistair Cockburn definierte Abschlussritual nach einer Iteration wurde weitestgehend weggelassen, da dies nur Sinn macht, wenn das Team täglich bis zu acht Stunden an dem Projekt arbeitet und die Iteration so gedanklich abgeschlossenen werden muss. Außerdem ist der Zeitaufwand bei einer einwöchigen Iteration zu groß und nicht rentabel. Dennoch wurde auf die von Crystal Clear groß geschriebene Reflexionen nicht vollständig verzichtet: Nach jeder Lieferung wurde wie vorgesehen ein Reflexionsworkshop durchgeführt. Dabei wurde die abgeschlossene Lieferung noch einmal reflektiert und überlegt, was gut war, was Probleme bereitet hat und was man in der nächsten Lieferung anders machen könnte. Dabei ging es vor allem um die Verbesserung der Zusammenarbeit, der Kommunikation und der Arbeitsgewohnheiten. Die folgende Tabelle zeigt die Ergebnisse der ersten drei Reflexionen.

Tabelle 10: Reflexionsergebnisse

Lieferung 1 (24.02)	**Keep** - dailies **Problems** - mehr Kommunikation - lokale Datenbank	**Try** - Side-by-Side/ Pair Programming - Gemeinsame Iterationsplanung - Review - Testen - Datenbank gleich pflegen - Häufigeres pushen
Lieferung 2 (10.03)	**Keep** - dailies - Pair-Programming - Tests (JUnit) **Problems** - Tests (JSFUnit) - Probleme zwar erkannt ABER nicht behoben	**Try** - Anwender zwischen durch fragen - Pair-Programming ausbauen - Tickets für Probleme mit hoher Priorität
Lieferung 3 (24.03)	**Keep** - dailies - Reviews - Tests (JUnit) **Problems** - Zu wenig Pair Programming → Zu wenig Zeit → Tasks zu klein/ leicht	**Try** - Git integrieren → Commits an Tickets hängen → leichterer Review

Wie man aus den Reflexionsergebnissen gut sehen kann, wurde am Anfang noch sehr viel ausprobiert und im Laufe des Projektes dann entweder gefestigt, wie zum Beispiel Testen und Reviews oder verworfen wie Side-by-Side Programming, da dazu keine passenden Aufgaben gefunden wurden. Wie von Alistair Cockburn empfohlen wurde mit einer geringen Anzahl von festgeschriebenen Methoden und Techniken begonnen. Dadurch konnte die Methodik im Laufe des Projekts in kleinen Schritten erweitert werden und die zu Begin definierte Basis blieb das komplette Projekt bestehen. Trotz der ständigen Kommunikation während der Iterationen konnten einige Probleme tatsächlich erst in den Reflexionen erkannt werden. Was in den Reflexionen auch deutlich wird, ist der ständige Versuch das Pair-Programming umzusetzen. Leider hat dafür oft die Zeit gefehlt und die passenden Aufgaben.

Ein weiterer Punkt der in der Reflexion der zweiten Lieferung zum Ausdruck kommt, ist der ständige Kontakt zum Anwender. Trotz der einfachen Kontaktaufnahme wurde die Kommunikation mit dem Anwender oft nicht wahrgenommen, wodurch Fehlimplementierungen erst bei der tatsächlichen Lieferung auffielen. Im Laufe des Projektes hat sich dieser Aspekt jedoch verbessert.

Die Nachbesprechung

Nach Abschluss der letzten Lieferung hat sich das Projektteam noch einmal zusammengefunden, um eine letzte Reflexion durchzuführen. Da es keine folgende Lieferung mehr gab, für die man sich Verbesserungsvorschläge überlegen konnte, hat man sich auf die Analyse der positiven und negativen Aspekte der Lieferung beschränkt.

Darüber hinaus wurde jedoch das gesamte Projekt mit Hilfe der erarbeiteten Reflexionsergebnissen rückblickend beurteilt und Verbesserungsvorschläge gesammelt, die im Hinblick auf weitere Projekte mit Crystal Clear umgesetzt werden könnten.

Die dabei gewonnen Erkenntnisse werden konkret in der Analyse des Prozesses im nächsten Teil dieser Arbeit aufgegriffen und erläutert.

Umsetzung der Crystal Clear Eigenschaften

Im Folgenden wird noch einmal konkret aufgegriffen, in wie weit die von Crystal Clear geforderten Eigenschaften umgesetzt wurden.

Häufige Lieferungen	Dem Anwender wurden im zwei Wochen Rhythmus Prototypen geliefert.
Osmotische Kommunikation	Das Projektteam hat versucht viele Teile der Arbeit im selben Raum zu verrichten, jedoch musste die meiste Zeit von zu Hause gearbeitet werden, wodurch die osmotische Kommunikation nur schwierig umzusetzen war.
Reflektierte Verbesserung	Nach jeder Lieferung wurde ein Reflexionsworkshop durchgeführt, bei dem überlegt wurde, was an der Arbeitsgewohnheit beibehalten werden sollte, wo Probleme bestanden und was man in der nächsten Lieferung ausprobieren könnte.
Einfacher Kontakt zum Anwender	Generell wurde der Kontakt zum Anwender sehr einfach gestaltet. Die Entwickler hatten die Möglichkeit den Anwender vor Ort persönlich anzusprechen oder telefonisch zu kontaktieren.

7.4 Scrum

In diesem Kapitel werden die acht Artefakte von Scrum: Release Planning Meeting, Product Backlog, Sprint Planning Meeting, Sprint Backlog, Sprint, Daily, Sprint Review und Retrospektiv, welche in der Durchführung umgesetzt wurden erläutert. Zusätzlich werden auch Probleme, so wie Besonderheiten welche auftraten und umgesetzt wurden geschildert.

Release Planning Meeting

Am Anfang des Release Planning Meetings waren die globalen Anforderungen, welche für alle Projekte gleich sind, bereits gegeben, so mussten lediglich die speziellen Anforderungen für Scrum definiert werden. Dazu zählen:

- Die Countdown Funktion
- Auffüllen mit neuen Blöcken
- Erreichen einer minimal Punktzahl

Weiterhin wird die Sprintlänge auf die Dauer von einer Woche festgelet. Aufgrund der Anforderungen wurde anschließend ein Architekturmodell erstellt um Abhängigkeiten zu klären und sicherzustellen, dass keine grundlegenden Funktionen zu dieser Zeit vergessen wurden. Die Architektur im Planning beschreibt grob einen Kreislauf welcher jedoch auch Nebenläufigkeiten beinhaltet.

Architektur

So kommt man im Kreislauf von der Startseite zum ersten Level, vom Beendeten Level zur Ergebnisseite und von dieser entweder ins nächste Level oder zum speichern. Nach dem Speichern gelangt man zur Rangliste und von dort aus wieder zur Introseite. Zudem ist es ebenso möglich von der Introseite zur Rangliste zu springen. Weiterhin zählen hierbei zu den Nebenläufigkeiten das Abbrechen des Levels und das Auslassen des Speicherns welches beide eine Weiterleitung zur Introseite nach sich ziehen.

Aufgrund der vorgestellten Architektur und den Anforderungen wurde zusammen mit dem Product Owner eine Priorisierung der User Sories vorgenommen. Eine Priorisierung auf Basis einer bereits bestehenden Architektur hat den Vorteil, dem Product Owner bereits Abhänigkeiten verständlich aufzeigen zu können und diese in der Priorisierung auch zu berückksichtigen.

Im Release Planning Meeting soll zudem eine erste Definition of Done festgehalten werden. In dieser wurde beschlossen das eine User Story als erledigt „Done“ gilt,

sobald alle Tasks innerhalb der jeweiligen User Story abgearbeitet sind und jeder Task von dem zweiten Teammitglied gereview wurde

Product Backlog

Aufgrund der Priorisierung im Release Planning Meeting wurden die User Stories absteigend ihrer Priorität in den Produkt Backlog übernommen und in die beiden eTracking Programme Jira und Yodiz eingetragen. So wurde in diesem Projekt, die Einrichtung der Entwicklungsumgebung, die Konfiguration der Bords, die Grundlagenrecherche sowie das Design, besonders hoch Priorisiert. Dabei am niedrigsten priorisiert wurde das Anzeigen der Anleitung, die Anzeige der Bestenliste sowie das Speichern der Ergebnisse.

Sprint Planning Meetingstory

Die Abschätzung der User Story in Scrum erfolgt in Story points. Story Points sind keine feste Zeiteinheit sondern sie beziehen sich auf die kleinste User Story. Ein Story Point ist in Scrum also so groß wie die kleinste User Story. In diesem Projekt ist die kleinste User Story das Anlegen der Startseite.

Zum Abschätzen wurden für jede User Story welche in den Sprint gezogen werden sollte eine unabhänige Schätzung der Story Points gemacht werden. In dem jedes Teammitglied verdeckt eine Anzahl von Stoy Points welche den Fibonacci Zahlen entsprechen, auf eine Karte schrieb und diese gleichzeitig mit dem anderen Teammitglied vorzeigte.

Wenn beide Teammitglieder die gleiche Anzahl an Story Points geschätzt haben wird diese Anzahl in die User Story übernommen. Wenn jedoch eine unterschiedliche Anzahl geschätzt wurde, was deutlich häufiger auftrat erklärten beide die Gründe weshalb sie die Story Points so abgeschätzt hatten. Zu Angang des Projktes schrieb dann jedes Teammitglied erneut eine Anzahl an Story Points, welche dann erneut verglichen wurde. Wenn immer noch keine Einigung erzielt werden konnte wurde die größere der beiden Zahlen in die User Story übernommen. Ab circa der hälfte des Projektes wurde bei einer fehlenden Übereinstimmung der Story Points, wieder eine Erklärung jedes Mitglieds abgegeben, jedoch hat das Mitglied welche die höhere Zahl geschätzt hat angegeben ob sie die geschätzten Stroy Points nach unten korrigiert oder ob sie bei der Anzahl bleibt. Wenn das andere Teammitglied die Anzahl über die Story Points jetzt teilt wird diese Anzahl in die User Story übernommen, andernfalls die höchste der beiden Zahlen. Für die Plattform Yodiz war

es zudem nötig gegen die Konventionen von Scrum auch die Tasks innerhalb der User Storys abzuschätzen. Dies musste hierbei in Minuten erfolgen. Innerhalb des Planing Meetings ist es auch möglich das der Product Owner, welcher in diesem Projekt aus Personalmange auch der Scrum Master war, eine Umpriorisierung vornimmt. Innerhalb des Projektes kam es zweimal zu einer Umpriorisierung. So wurde einmal die User Story Punkte anzeigen und Spiel abbrechen vertauscht. Zudem wurde die Anleitung, welche am Anfang eine hohe Priorisierung hatte vom Product Owner deutlich herabpriorisiert und wurde so in die letzte Iteration verschoben.

Sprint Backlog

Im Sprint Backlog werden die User Storys, welche im Sprint Planing abgeschätzt wurden und im Sprint abgearbeitet werden festgehalten. Der Sprint Backlog stellt sicher das jedes Teammitglied weiß was in diesem Sprint für Aufgaben erledigt werden müssen und welche Aufgaben bereits erledigt sind. Zudem sieht man wer an welchen Aufgaben gerade arbeitet. Dies war in der Studienarbeit besonder wichtig, da an verschiedenen Standorten entwickelt wurde, um eine Doppelarbeit zu vermeiden. Zudem sind die User Stories die im Sprint Backlog festgehalten sind auch tatsächlich fest und können während einem Sprint nicht getauscht werden. Im Projekt gab es während eines Sprints keine Probleme mit den festgelegten User Stories.

Weiterhin muss erwähnt werden das die User Stories ihre Priorität wie auch im Product Backlog behalten und die Aufgaben so abgearbeitet werden müssen. Dies hat eher zu Problemen geführt denn wenn ein Teammitglied sich die wöchentliche Arbeitszeit eher am Ende der Woche eingeplant hat, aber verantwortlich war für einen der oberen User Stories hat dieses Mitglied die Arbeit des anderen Mitglieds behindert/ verhindert.

Sprints

Die Länge der Sprints wurde wie in der Einleitung schon erwähnt auf eine Woche festgelegt. Dies Stellt einen Kontrast zu den anderen beiden Prozessen da, da mach jeder Woche ein Release zu machen war. Der Sprint beginnt jeweils Montags und endet jeweils Montags. Die tägliche Arbeitszeit wurde auf eine Stunde am Tag festgehalten, wobei auch zwei oder drei Arbeitstage in einem Arbeitstag zusammengefasst wurden um anderen Studentischen oder privaten Verpflichtungen gerecht werden zu können.

Die Zeitpunkte der Sprintlänge erwieß sich jedoch im Laufe der Entwicklung als ungünstig, denn an den Wochenenden fiel die Kommunikation der Teammitglieder schwerer als erwartet. So war es auch nur unter der Woche möglich Pairprogramming umzusetzten. Das Pairprogramming wurde jedoch trotz der „Terminprobleme“ so häufig wie möglich umgesetzt. Es wurde darauf geachtet mindestens einmal in der Woche Pairprogramming umgesetzt wurde. Zu Anfangs wurden die Tests etwas vernachlässigt, was auch daran lag das es Probleme bei der Aufsetzung der Testumgebung gab. Hieraus resultierte das die JSF Komponenten nicht getestet wurden.

Dailies

Die Dailies wurden jeden Tag meist vor Unibeginn abgehalten. In den Dailies wurde besprochen was jeder am Vortag an Aufgaben des Sprints erledigt hat, was ihn dabei behindert hat und was er an diesem Tag erledigen will. Da am Anfang des Projekts nicht immer alle dieser drei Fragen beantwortet wurden und es auch für den Scrummaster schwierig war darauf zu achten wer welche Fragen nicht beantwortet hatte, da wir die Dailies zum teil an verteilten Orten über Skype durchgeführt hatten. Entschloss sich das Team eine Liste einzuführen in der die Fragen welche beantwortet wurden abgehakt wurden. Dies wurde so lange gemacht bis sich das Team an die Fragen gewöhnt hatte und die Liste nicht mehr benötigt wurde.

Sprint Review

Nach einem Sprint steht der Review, wobei dem Product Owner die vollständig erledigten User Stories vorgestellt wurden, welche er dann testen konnte. Da in diesem Projekt wie schon erwähnt kein unabhäniger Product Owner zur Verfügung stand übernahm dies auch wieder der Scrummaster. Da wurden die User Stories auf dem Laptop des Scrummasters getestet. Innerhalb der ersten zwei Sprints hatte der Product Owner keine Beanstandungen und nahm die fertigen User Stories ab. Ab dem dritten Review kam immer wieder Kritik an einzelnen Use Cases auf, da entweder einzelne Sachen geändert oder hinzugefügt werden sollten. Wenn die genannten Änderungen nicht in der Beschreibung der Use Cases oder als Tasks aufgeführt waren beschloss das Team mit dem Product Owner dies als Change Request in den Product Backlog aufzunehmen. Fall jedoch ein Task nicht nicht oder vergessen wurde umzusetzen galt diese User Story als nicht vollständig und musste somit im nächsten Sprint wieder mit aufgenommen werden.

Retrospektive

Bei der Retrospektive welche nach jedem Sprint stattfindet werden gute oder schlechte Ereignisse während des letzen Sprint tematisiert. Innerhalb des Projektes wurde dies bis auf eine Ausnahme so umgesetzt. Zu dieser einen Ausnahme kam es da ein Teammitglied zu diesem Zeitpunkt erkrankt war. Während jeder Retrospektive wurde ein Zeitstrahl an das White Board gemalt welcher den Vergangenen Sprint abbildete. Jedes Teammitglied bekam dann Post Its worauf es dann jeweils genau eine positive oder negative Empfindung des Sprints geschrieben wurde. Positive Empfindungen waren nicht rein auf den Sprint bezogen sondern konnten auch ein besonderes Lob des anderen Teammitglieds darstellen. Die Methode das „versteckten“ Schreibens, wie bei der Schätzung der Story Points, hatte zum Vorteil das jeder unabhängig vom anderen seine Meinungen und Empfindungen äußern konnte ohne sofort Kritik zu befürchten. Im Anschluss hefteten die Teammitglieder nacheinander die jeweiligen Post Its an die entsprechende Stelle des Zeitstrahls.

Die negativen Punkte welche am Zeitstrahl eingetragen wurden, wurden analysiert und deren Verbesserungsvorschläge in der Retrospektiv Liste Festgehalten

Retrospektiv Liste

Retro-Nr.	Zeitraum	Änderungen
1	17.02-24.02	- Velocity maximal 10 Stroy Points - Beantwortung der 3 W-Fragen im Daily - Größe pro Task maximal 3 Stunden
2	24.02-03.03	- Abschätzung User Stories vor Sprint nochmal - Taskgröße von 3 beachten - Daily Überwachung mit Liste - Tests als Tasks im Board - Einführung Unit Tests - Neue Dafinition of Done
3	03.03-10.03	- Taskgröße für Pairprogramming maximal 5 Stunden - Yodiz: Task mit Comment und Zuweisung bei Review - Tasks innerhalb User Story schätzen → besseren Schätzen User Story
4	10.03-17.03	- 1 Daily am Wochenende - Rückfragen Review spätestens am nächsten Tag - Velocity erhöhen maximal 15

5	17.03-31.03	- Dailies ohne Liste - Velocity in 1. Woche zu niedrig
6	31.03-07.04	- Reviews von Bugs besser Koordinieren

Review des gesamten Projektes

Im Anschluss an die letzte Rerospektive wurde eine Beurteilung des gesamten Projektes in gleicher weise wie die Reviews durchgeführt. Dabei wurde festgehalten was bei zukünftigen Projekten mit Scrum beibehalten oder verbessert werden sollte.

Positiv	- die Zeiteinteilung - das Pairprogramming - die Dailies
Verbesserungen	- früher Tests schreiben - Reviews mehr Beachtung schenken - schnellere Reaktion auf Kundenänderungen -aktuell halten des Boards - mehr Wissenstransfair - mehr Pairprogramming

7.5 Kanban

Im folgenden Kapitel werden verschiedene Aspekte der Durchführung des Kanban-Projekts näher beleuchtet. Da dieser agile Prozess dazu genutzt wird, einen bestehenden Prozess zu verbessern und in der Wahl seiner Mittel große Freiräume lässt, wurden die vom Team gewählten Methoden zur Prozessoptimierung teilweise zur Laufzeit des Projekts ausgewählt. Dazu zählen das Kanban-Board inklusive des Work-In-Progress-Limits, die Wahl der Sprint-Länge und der damit verbundenen Taktfrequenz der Planning- und Release-Meetings, die Verwendung verschiedener Diagramme wie beispielsweise des Cumulative Flow Diagramms zur Flusskontrolle sowie die Verbeserung des Prozessablaufs mittels Vorschlägen aus dem Operations Review oder aus Dailys.

Visualisierung: Kanban-Board und Work-in-Progress-Limit (WIP-Limit)

Das Kanban-Team nutzte wie auch die anderen Teams ein elektronisches Board zur Synchronisation und Koordination der Teams. In Yodiz entsprach dieses den beiden anderen Prozessen, aber in JIRA wurde ein Kanban-Board statt des Scrum-Boards verwendet. Dieses hat prinzipiell keine Iterationen sondern zeigt alle Tickets sofort in der “To Do”-Spalte an. Die im Folgenden beschriebenen Änderungen wurden ausschließlich in JIRA durchgeführt, da das Board bei Yodiz nicht flexibel genug für Kanban ist und WIP-Limits, Swimlanes oder zusätzliche Spalten nicht unterstützt.

Um eine der wichtigsten Eigenschaften von Kanban, das WIP-Limit, umzusetzen, entschied das Team für den zweiten Release-Zyklus zuerst, sowohl das “In Progress” Limit als auch das “Review” Limit auf zwei zu setzen. Somit sollten die Entwickler gezwungen werden, stets nur an einem Task zu arbeiten, statt an mehreren und gleichzeitig den Review nicht aus den Augen zu verlieren.

Diese Lösung stellte sich relativ schnell als unpraktisch heraus, da die Entwickler im Team besonders am Wochenende und abends in der Woche nicht gleichzeitig arbeiteten. Dadurch kam es oft zu Überschreitungen in der Review-Spalte, die JIRA mit einem leuchtend roten Hintergrund anzeigt. Ein Bottleneck war gefunden. Deshalb wurde das WIP-Limit dort auf vier angehoben. Selbst wenn ein Entwickler nun am Abend allein entwickelte, konnte er für alle Tasks des Anderen den Review durchführen und danach vier eigene in die Spalte schieben. Dies genügte in den meisten Fällen (Ausnahme: Abbildung BOARD 5)

Um die oben erwähnte unübersichtliche “To Do”-Spalte aufzuräumen, wurde für den

dritten Release-Zyklus eine weitere Spalte in JIRA eingeführt, die “Backlog-Spalte” (Abbildung BOARD 5). Bis dahin konnte in JIRA nicht gesehen werden, welche User Stories noch in der laufenden Iteration zu bearbeiten waren. Dafür wurde Yodiz genutzt. Mit Einführung der neuen Spalte wurden nun in jedem Planning die entsprechenden User Stories in “To Do” geschoben. Ein WIP-Limit gab es hier jedoch zu Beginn nicht, da es dem Team nicht gelungen war, die Tasks ansatzweise gleich groß zu halten. Bei sehr unterschiedlicher Größe ist es aber schlecht möglich ein WIP-Limit für diese Spalte und damit effektiv für die Planung einzuführen, da die Taskanzahl zu sehr schwankt.

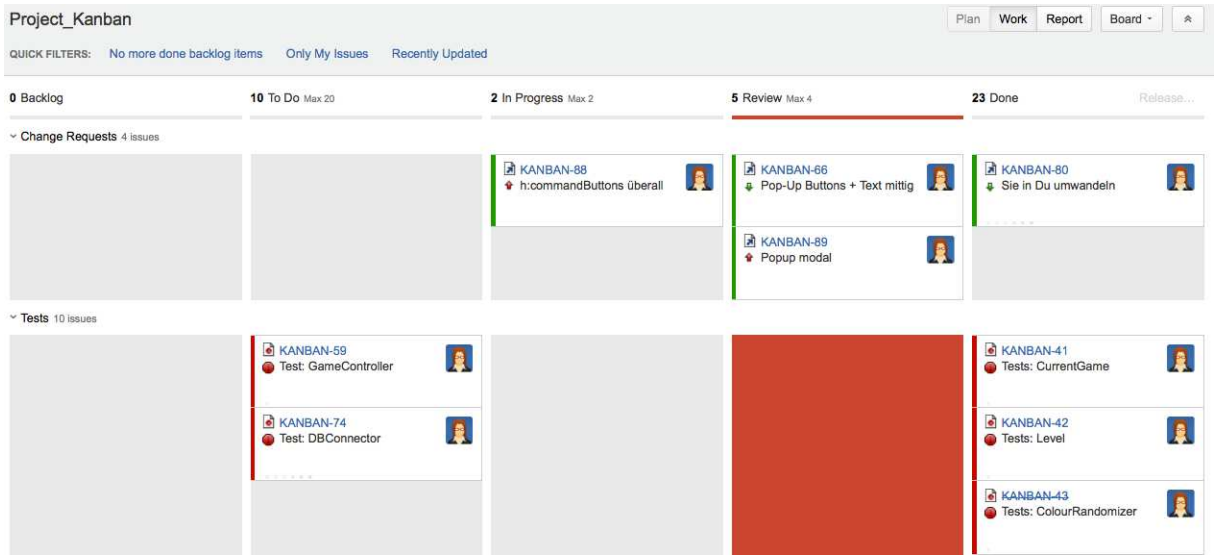


Abbildung 3: Board 5

Deshalb wurden innerhalb des dritten Release-Zyklus Swimlanes eingeführt. Da das Einteilen von Tasks in Servive Level Agreements (SLAs) in beiden Boards nicht möglich ist, wurde zumindest in JIRA mittels Swimlanes diese Einteilung simuliert. SLAs dienen dazu, mit verschieden großen Tasks umzugehen. Im Fall dieses Projekts entschied sich das Team für insgesamt drei Swimlanes (Abbildung SWIMLANES). “Change Requests” sind Tickets, die vom Kunden beim Release erstellt wurden oder kleine Bugs, die das Team selbst entdeckte. Sie wurden alle auf höchstens 30 Minuten Aufwand geschätzt. Tests enthielten Tickets für Tests, alle mit ein bis zwei Stunden abgeschätzt. User Stories enthielten alle User Stories mit den dazugehörigen Tasks und lagen meist zwischen 30 Minuten und zwei Stunden. Zusätzlich dazu gab es Tasks, die mittels Pair-Programming durchgeführt werden sollten und meist bei drei bis vier Stunden anzusiedeln waren. Es wurde kurz diskutiert auch diese in eine eigene Swimlane auszulagern, aber der Zusammenhang

zur User Story wäre damit schlechter sichtbar gewesen. Deswegen wurde die Idee einer eigenen Swimlane wieder verworfen. Stattdessen durften im Planning höchstens 2 Tasks dieser Art in “To Do” gezogen werden.

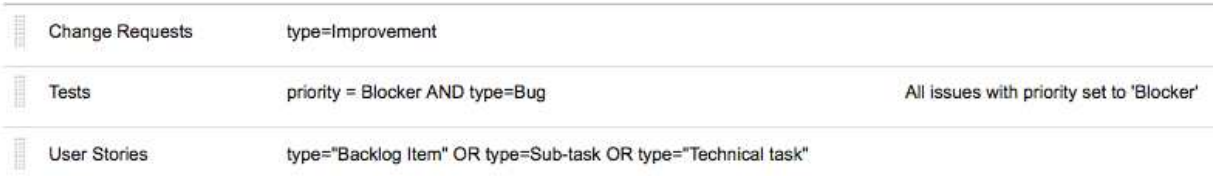


Abbildung 4: Swimlanes

Im Operations Review zum dritten Release-Zyklus stellte sich heraus, dass die ursprüngliche Anordnung der Swimlanes bei JIRA ungünstig war. Sowohl Tests als auch Change Requests wurden unterhalb der User Stories nicht beachtet, da sie nur durch Scrollen der Webseite sichtbar wurden. Deshalb änderte das Team die Reihenfolge der Swimlanes, so wie in Abbildung SWIMLANES sichtbar. Somit mussten Entwickler nun jedes mal scrollen, um zu den User Stories zu gelangen, und sahen automatisch die beiden oberen Zeilen zuerst.

Nachdem die Planung beim dritten Release mittels Swimlanes gut funktioniert hatte, wurde für die Iterationen im vierten Release-Zyklus außerdem ein WIP-Limit für die “To Do”-Spalte eingeführt. Es durften höchstens 5 Tickets bei Tests, 5 bei Change Requests und 10 bei User Stories eingeplant werden - höchstens 2 von ihnen Pair-Programming Tickets. Diese Einzellimits konnten bei JIRA nicht umgesetzt werden. Es ist lediglich möglich, ein Gesamtlimit pro Spalte zu setzen. Die eingetragenen 20 Tickets mussten dann beim Planning entsprechend auf die Spalten aufgeteilt werden. Insgesamt vereinfachte und beschleunigte dieses Limit das Planungsmeeting sehr und stellt den letzten Schritt in der Evolution des Boards während dieses Projekts dar.

Iteration, Iterationslänge und Taktfrequenz von Planung und Releases

Prinzipiell ist es bei Kanban möglich, die Länge der Iterationen und Release-Zeiträume verschieden zu wählen und zu variieren. Da diese Möglichkeit in Scrum und Crystal nicht vorhanden ist, entschied sich das Team, dies auf jeden Fall in der Umsetzung auszuprobieren.

Begonnen wurde das Projekt jedoch mit dem Prozessablauf identisch zum Scum-Prozess. Das bedeutet, die erste Iteration hatte eine Länge von einer Woche, beendet mit einem Release. Es gab ein Planungsmeeting zu Beginn der Iteration und

ein Meeting zur Auswertung, den Operations Review, am Ende. Allerdings bestand diese Iteration lediglich aus technischen und organisatorischen Vorbereitungen für das Projekt wie beispielsweise die Einrichtung der Entwicklungsumgebung, so dass es kein wirklichen für den Kunden sicht- oder nutzbares Ergebnis als Release gab.

Nach der ersten Woche wurde mit dem Kunden zusammen entschieden, welche User Stories der erste “richtige” Release des Projekts enthalten sollte. Danach schätzte das Team diese User Stories ab und setzte den Release-Termin. Es wurde das Scrum-Vorgehen beibehalten, dass ein Release das Ende einer Iteration signalisierte. Somit verlängerte sich die zweite Iteration von einer Woche auf zweieinhalb Wochen. Bereits bei der Planung zeigte sich das Team jedoch unsicher, einen so langen Zeitraum korrekt einschätzen zu können. Nach etwa ein bis eineinhalb Wochen kristallisierte sich schließlich eine gewisse Unzufriedenheit im Team heraus, was den Überblick über den momentanen Stand der Entwicklung betraf. Niemand wusste, ob der Prototyp am Ende der Iteration tatsächlich fertig werden würde. Das Board war durch die Länge der Iteration voller als vorher und es fiel den Entwicklern schwer, einzuschätzen ob sie noch in der Zeit lagen bzw. schneller oder langsamer waren. Auch stellte sich in der abschließenden Besprechung nach der Iteration heraus, dass beide Entwickler zu Beginn der Arbeit das Gefühl hatten, nicht vorwärts zu kommen, da die Menge der offenen Tasks sehr lange sehr groß blieb, und das Board allgemein auf Grund der Größe unübersichtlich geworden war. Insgesamt wurde die Iteration als eindeutig zu lang für das Vorwissen und die Vorerfahrung der Entwickler im Bereich Planung und Abschätzung empfunden.

Deshalb wurde für den dritten Release entschieden, die Sprintlänge vom Release zu entkoppeln. Wie für den zweiten Release entschied der Kunde mit den Entwicklern gemeinsam den Umfang der zu bearbeitenden User Stories. Danach wurden diese ebenfalls grob abgeschätzt und ein erster Release-Termin angesetzt. Gleichzeitig entschied sich das Team wieder für eine einwöchige Iterationsdauer. Das bedeutete in diesem Fall, dass der dritte Release mitten innerhalb einer Iteration lag. Da der Termin jedoch nur auf einer groben Abschätzung beruhte, wurde mit dem Kunden vereinbart, dass bei Verzögerungen die restliche Iteration als Puffer zur Verfügung stand. Dies empfanden Team und Kunde als gute Lösung, falls die Schätzung nicht korrekt war oder ungeplante Verzögerungen auftraten. Die Planungsmeetings wurden weiterhin zu Beginn der Iterationen durchgeführt - nun also wieder

wöchentlich, während der Operations Review weiterhin nur am Ende des Release-Zeitraums stattfand. Während die Entwickler sehr viel besser mit der Planung einer wochenlangen Iteration zurecht kamen, fiel es ihnen schwerer mit einem Release mitten in der Planung umzugehen. Einerseits musste alles Fehlende für den Release vollendet werden, andererseits sollten bereits User Stories für den nächsten Release eingeplant werden. Das Meeting zur Planung des nächsten Releases fand aber erst nach dem Planning der Iteration statt. Diese Reihenfolge funktionierte nach Meinung aller Teammitglieder überhaupt nicht.

Auf Grund der Verwirrung entschied das Team sich zu einer letzten Änderung innerhalb des Projekts. Da die Grobabschätzung für Release 2 und 3 überraschend gut funktioniert hatte, wurde entschieden, dass diese für Release 4 beibehalten werden könnte. Auch die einwöchigen Iterationen mit einem anfänglichen Planungsmeeting blieben erhalten. Allerdings wurde nun der tatsächliche Release an das Ende einer Iteration gelegt. Wenn das Ergebnis der Grobabschätzung eine Zahl zwischen zwei und drei Wochen war, so wurde der Release auf das Ende der dritten Iteration gelegt. Somit gab es bereits automatisch Puffer. Falls das Team tatsächlich eher fertig wurde, konnte es die übriggebliebene Zeit für projektunabhängige Aufgaben nutzen oder den Kunden für eine Pre-Release-Vorführung einladen. Das hierdurch gewonnene Feedback konnte somit noch am Ende der Iteration vor dem eigentlichen Release angenommen und mögliche Kritikpunkte umgesetzt werden.

Insgesamt war das Team zufriedener und fühlte sich sicherer mit einer kürzeren Iteration, am besten einer Woche. Das Planning am Anfang dieses Zeitraums ergab mit der Zeit gute Resultate. Den Release über mehrere Iterationen laufen zu lassen war ebenfalls kein Problem. Der Versuch, den Release mitten in die Iteration zu legen, wurde eher negativ gesehen, vermutlich weil dies im Gegensatz zu anderen Prozessen wie Scrum ungewohnt war. Das Abschlussmeeting nur zum Release durchzuführen statt am Ende jeder Iteration führte dazu, dass viele Ideen bereits wieder vergessen wurden. Obwohl alle Teammitglieder bei Kanban auch während der Iteration im Daily jederzeit Vorschläge für Abänderungen des Prozessses machen dürfen, geschah dies eher selten. Stattdessen wurden Ideen im Operations Review vorgestellt und in der nächsten Iteration umgesetzt. Auch diese Art der Daily-Nutzung war dem Team vorher unbekannt, was möglicherweise ein Grund dafür ist, dass sie kaum genutzt wurde.

Flusskontrolle

Zur Flusskontrolle bei Kanban können verschiedene Diagramme herangezogen werden, u.a. das Cumulative Flow-Diagram sowie Cycle Time und Lead Time.

Das Cumulative Flow Diagram ist eine Weiterentwicklung der Burnup-Charts und hilfreich bei der Darstellung der verschiedenen Stationen im Arbeitsprozess. Jede Station oder Spalte am Board erhält eine Farbe. Die Menge der Tasks wird auf der y-Achse, die Zeit auf der x-Achse abgebildet (Abbildung CUMULATIVE FLOW).

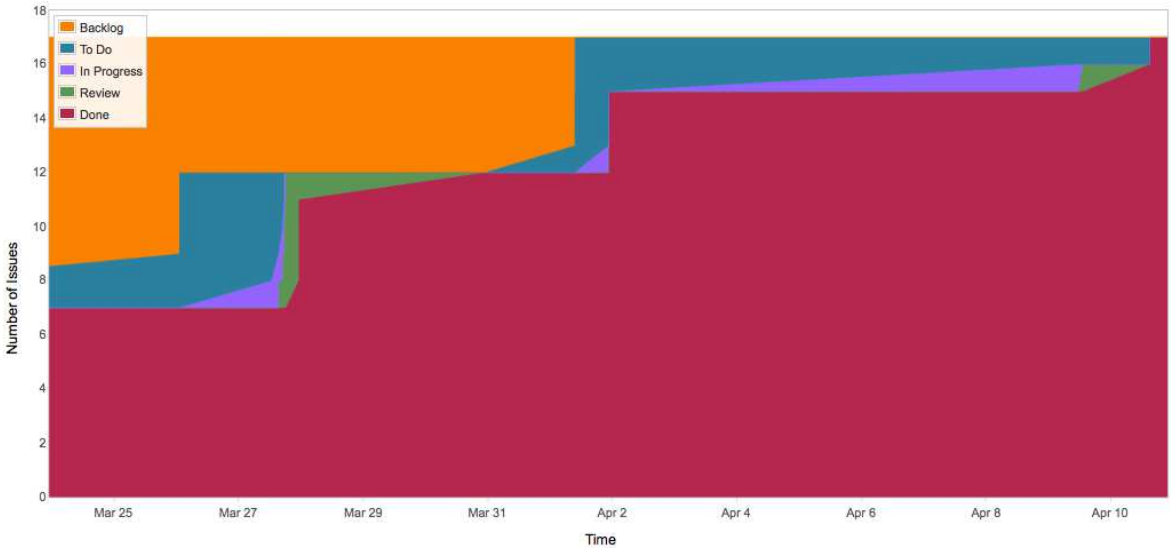


Abbildung 5: Cumulative Flow (Auschnitt Release 4)

Die rote Fläche zeigt an, wie viele Tickets bereits abgearbeitet (“Done”) sind. Ist die Linie oberhalb der Fläche waagerecht wie beispielsweise zwischen 24. und 28. März, wurden in dieser Zeit keine User Stories abgeschlossen. Dies kann bedeuten, dass zumindest bis 26. März niemand gearbeitet hat oder dass wie in diesem Fall ein Entwickler vergessen hatte, seine Tickets im Board weiter zu hängen, so dass sie erst ab 26. März als “In Progress” zählten.

Die lila Fläche über der Waagerechten zwischen 4. und 9. April zeigt wiederum an, dass in diesem Zeitraum Tickets aus “In Progress” nicht zum Review und auch nicht zu “Done” geschoben wurden. In diesem Fall gab es CSS-Probleme, die alle User Stories betrafen - also zumindest einen kurzfristigen Bottleneck.

Der Abstand zwischen den vertikalen Linien der Zustände zeigt an, wie lange ein Ticket, das an einem bestimmten Tag begonnen wurde, durchschnittlich bis zu seinem Abschluss bzw. allgemein zum Statuswechsel benötigt, also die durchschnittliche Cycle Time. JIRA ermöglicht es sogar zu sehen, um welches spezifische Ticket es sich handelt.

Das beinahe Verschwinden der “Review”-Spalte im obigen Diagramm resultiert aus der Splittung der User Stories in relativ kleine Tasks und der Einhaltung des WIP. Sofern sie innerhalb eines Tages abgearbeitet wurden, und durch den Review gingen, wechselte das Ticket innerhalb des Tages ohne Probleme von “To Do” in “Done”. Analoges gilt für die “In Progress”-Spalte. Beide sollten in diesem Diagramm kaum sichtbar sein.

Die Lead und Cycle Times von Tickets zeigt an wie lange sie innerhalb des Projekts am Leben waren bevor sie beendet wurden (Lead Time) bzw. wie lange die Bearbeitung gedauert hat (Cycle Time). Abbildung LEAD CYCLE TIME EXPLANATION zeigt den Unterschied zwischen beiden.

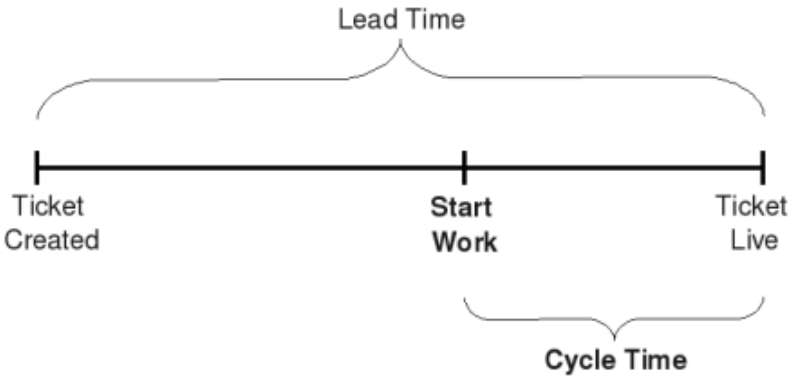


Abbildung 6: Lead Cycle Time Explanation

(http://stefanroock.files.wordpress.com/2010/03/leadtimes3.png?w=450&h=213)

JIRA bietet auch diese beiden Werte in Form eines Diagramms an (Abbildung: LEAD UND CYCLE TIME (selber Zeitraum)). Auch hier ist es möglich, die Cycle Time einzelner Tickets zu bestimmen, um z.B. festzustellen, in welchen Spalten Tickets besonders viel Zeit verbringen. Dies wäre wiederum ein Anzeichen für einen möglichen Bottleneck.

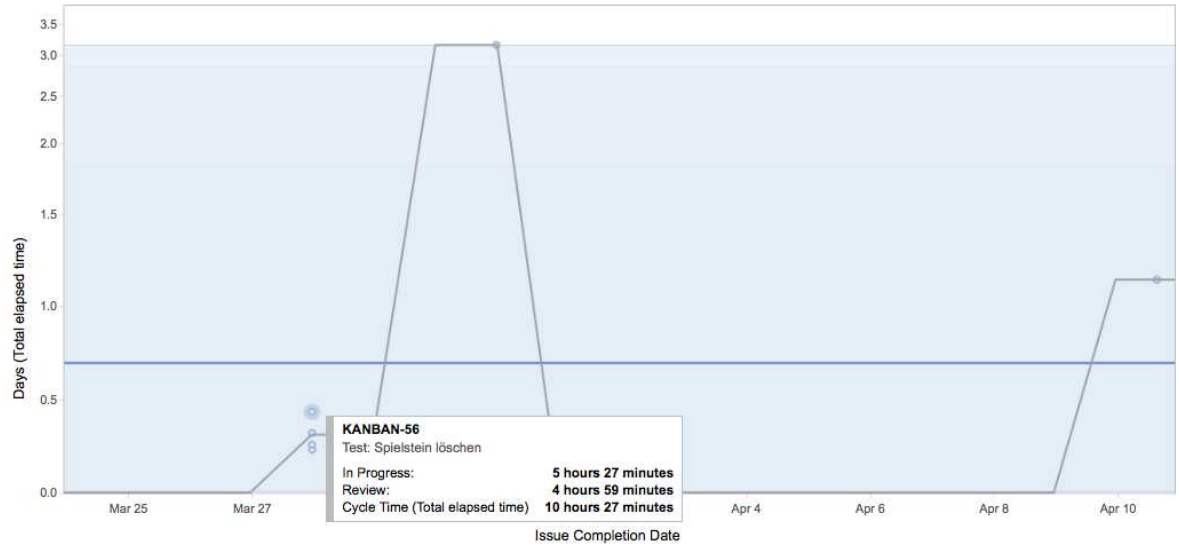


Abbildung 7: Lead und Cycle Time (selber Zeitraum)

Ablauf verbessern

Bei Kanban ist es während der gesamten Iteration möglich, den Arbeitsablauf zu verändern. Insbesondere das Daily ist ein guter Ort, um Verbesserungsvorschläge an das Team heranzutragen und wenn möglich sofort umzusetzen. Die Einführung der “To Do”-Spalte ist eine dieser im Daily vorgeschlagenen Änderungen.

Zusätzlich dazu gibt es einen Operations Review, der am Ende jedes Releases stattfindet und zum Überdenken der Abläufe gedacht ist. Für dieses Projekt fanden drei dieser Meetings statt, da nach Release 1 noch keine Veränderung nötig war.

Operations Review	Probleme	Ideen zur Veränderung
06.03.2014	2,5 Wochen Iteration → schwer zu planen (lang)	kürzere Iterationen (1 Woche) Release und Iterationen → verschiedene Cadences flexibles Release-Datum
	WIP-Limits → ungewohnt, schwer einzuhalten	beibehalten → Ziel: nicht parallel mehrere Tickets anfangen ohne sie abzuschließen
	Umgehen mit verschieden großen Tasks (Vorarbeit für WIP-Limit für To-Do-Spalte)	Swimlanes einführen: User Stories Change Requests Tests
22.03.2014	Change Requests und Tests	Reihenfolge der Swimlanes ändern

	übersehen	
		WIP-Limit für To-Do-Spalte

Der letzte Operations Review zum Abschluss des Projekts am 09.04.2014 stellte klar heraus, dass der Prozess noch an einigen Stellen verbesserungswürdig war. So wurde insbesondere die Swimlane für Tests in Frage gestellt, nachdem ein Entwickler einen Test für eine Funktionalität schreiben wollte, die noch nicht implementiert war. Grund hierfür war der nicht mehr sichtbare Zusammenhang zu den User Stories. Auch die mangelnde Kommunikation mit dem Kunden wurde als negativ empfunden, da dieser lediglich zum Release vor Ort war. Hier meinten die Entwickler, dass eine Art Projektleiter oder zumindest ein Verantwortlicher für Kundenkommunikation hilfreich wäre, der das Team ab und zu daran erinnert, den Kunden direkt anzusprechen. Außerdem könnte dieser Verantwortliche das Team im Daily daran erinnern, aktiv an der Gestaltung des Arbeitsablaufs mitzuwirken. Es fiel den Programmierern schwer, sich neben der Software-Entwicklung zusätzlich darauf zu konzentrieren, wie man den Prozess verbessern könnte, wenn dies normalerweise ein Projektverantwortlicher tut.

Insgesamt ist ein gewisses Maß an Umdenken nötig, will man das volle Potential von Kanban nutzen. Dies war innerhalb des kleinen Teams und der kurzen Zeit nur schwer möglich.

8. Ergebnisse

8.1 Gegenüberstellung der Prozesse

Vor der Durchführung der drei Projekte haben die Teams eine Gegenüberstellung der agilen Prozesse durchgeführt. Die Ergebnisse dieses theoretischen Vergleichs wurden in Kapitel (X) vorgestellt. Während und nach der Durchführung wurden die vorher aufgestellten Vergleichskriterien in der Praxis umgesetzt und positiv oder negativ bewertet. Eine Zusammenfassung dieser Auswertung findet sich in diesem Kapitel.

Iterationen

Die Iterationslänge wurde bei Scrum und Crystal Clear gleichermaßen auf eine Woche festgelegt. Für beide Prozesse wurde die Länge als passend für die Planung und Übersicht über den Fortschritt des Teams bei der Entwicklung empfunden. Bei Kanban veränderte sich die Sprintlänge während des Projekts, so wie es die Methode erlaubt. Sie schwankte zwischen einer und zweieinhalb Wochen. Auch hier entschied das Team sich jedoch zum Ende des Projekts dafür zur einwöchigen Iteration zurückzukehren, da es den Entwicklern während der längeren Phasen schwer fiel, den Überblick über die Planung und den Entwicklungsstand zu behalten. Dies liegt jedoch insbesondere an der mangelnden Erfahrung der Teammitglieder beim Schätzen. Besonders schwierig wurde es, nachdem die Iterations- und Release-Zyklen nicht mehr synchron liefen. Auch dies erfordert einiges an Erfahrung der Teammitglieder in der Organisation von Projekten. Die große Flexibilität von Kanbans Iterationen kann demnach von einem unerfahrenen Team vermutlich nicht sofort genutzt werden.

Teamgröße und Rollen

Das eigentliche Entwickler-Team bestand bei allen Prozessen nur aus zwei Personen. Die dritte Person nahm jeweils eine andere Rolle im Prozess ein. Da es bei Kanban eigentlich keine explizit festgelegten Rollen gibt, gab es lediglich einen Kunden und einen Board-Verantwortlichen. Die geringe Größe des Entwicklungsteams machte die Synchronisation sehr einfach und verursachte kaum Overhead bei der Organisation. Allerdings entstehen vermutlich bei mehr Mitarbeitern auch mehr Verbesserungs- und Veränderungsvorschläge. Außerdem fehlte beiden Entwicklern jemand, der ab und zu die Leitung oder Organisation des Teams übernimmt. Vermutlich fällt es allgemein schwer, von einer Organisationsform mit Projektleiter oder Ähnlichem zu einer so freien Form der Selbstorganisation zu

wechseln.

Für ein Scrum-Team ist die Anzahl zu gering. Scrum Master und Product Owner / Kunde mussten in einer Person existieren. Dies ist denkbar schlecht, da der Scrum Master die Aufgabe hat, das Team vor Eingriffen des Kunden während des Sprints zu schützen – in diesem Fall also gegen sich selbst. Beide Rollen sollten unbedingt getrennt gehalten werden. Die Menge der Entwickler war dagegen kein Problem. Allerdings würden bei einem größeren Projekt auch mehr Entwickler nötig sein, um das Vorankommen des Projekts zu gewährleisten.

Ähnliches gilt für Crystal Clear. Auch hier war die Team-Größe von zwei Entwicklern für die Größe des Projekts angemessen und gut zu koordinieren. Die Rollen des Anwenders und Auftraggebers konnten sehr gut in einer Person vereinigt werden. Manchmal fehlte dem Team jedoch eine Art Koordinator – eine Rolle, die bei genügend Mitarbeitern dem Chefdesigner bei der Organisation hilft und ihn freier für die Entwicklung selbst gemacht hätte.

Techniken

Weder Scrum noch Kanban schreiben das Verwenden bestimmter Techniken vor. Deshalb wurde aus dem Wissensrepertoire der Entwickler Pair Programming gewählt und in beiden Projekten regelmäßig umgesetzt. Alle Entwicklerteams empfanden dies als sehr hilfreiches Mittel zur Wissensverteilung und –erweiterung und würden es jederzeit wieder anwenden, auch wenn damit zuerst die doppelte Zeit für eine Aufgabe aufgewendet wird. Dies wurde später durch den Zuwachs an Wissen und Können sowie das Durchsprechen von Konzeptionen und damit frühzeitiger Erkennung von Schwachstellen oder Fehlern aufgewogen. Allerdings sollte unbedingt darauf geachtet werden, wer tippt – nämlich der Entwickler, der weniger weiß.

Auch bei Crystal Clear werden keine Techniken vorgeschrieben. Es gibt jedoch einige Vorschläge, die in der Projektdurchführung umgesetzt wurden. Hierzu zählt ebenfalls Pair Programming, aber auch die Blitzplanung am Anfang des Projekts. Diese empfand das Team insgesamt als sehr hilfreich für die Projektplanung, da sowohl Anwender als auch Programmierer anwesend waren und verschiedene Blickwinkel auf das Endprodukt einbezogen werden konnten.

Insgesamt gesehen wurden in allen Prozessen die Entscheidung getroffen, mit wenigen Prozessen zu beginnen und diese beherrschen zu lernen statt vieles

gleichzeitig zu testen.

Lieferung

Die Datum der Lieferungen unterschied sich bei allen drei Prozessen. Bei Scrum erfolgten sie wie in der Theorie angegeben am Ende jedes Sprints. Im Review konnte der Kunde das vorhandene Produkt auf seinem eigenen Rechner testen. Da dies ein Mac war und die Entwicklung auf Windows-PCs mit unterschiedlichen Auflösungen durchgeführt wurde, kam es insbesondere beim Layout öfter zu Unzufriedenheit und Ablehnung der User Story. Dies konnte im Rahmen des Projekts nur dadurch behoben werden, dass der Kunde seinen Computer für Tests zur Verfügung stellte. Der Sprint war jedoch so kurz, dass es meist schwierig war, den Kunden selbst für ein vorzeitiges Testen zu gewinnen. Ein weiterer negativer Aspekt der nur einwöchigen Iterationsdauer war die geringe Menge an User Stories. Falls der Sprint nur ein oder zwei User Stories umfasste, führt das Ablehnen durch den Kunden zu einer sehr niedrigen Velocity von bis zu 0. Dies ist demotivierend für das Team und macht die Planung des nächsten Sprints schwierig, bei der man sich an der Velocity orientiert.

Diese Nachteile gab es bei Crystal Clear nicht, da ein Release jeweils zwei Iterationen umfasste, also zwei Wochen. Außerdem ist der Anwender Teil des Teams und kann jederzeit zum Testen oder einer Besprechung herangezogen werden. Dies umzusetzen fiel dem Team jedoch unerwartet schwer, da das direkte Ansprechen des Kunden oder Anwenders eine völlig neue Erfahrung darstellte. Außerdem präsentierte das Team vorwiegend den Release, statt den Anwender noch einmal testen zu lassen, es hat also die Chancen der Kundenkommunikation und Rückmeldung noch nicht ausgeschöpft. Abgesehen davon ist die Blitzplanung eine sehr große Hilfe zur Organisation der Releases. Hier wird bereits festgelegt, welche User Stories vermutlich in welchen Release fallen werden. So hat der Kunde bereits zu Anfang eine gewisse Vorstellung über die Fertigstellung aller Features, was zu mehr Sicherheit seinerseits und in gewissem Maße mehr Zufriedenheit führte. Es sollte lediglich darauf geachtet werden, diese Planung nach jedem Release neu zu prüfen. Dem Kunden darf nicht das Gefühl gegeben werden, er könne an der Reihenfolge nichts mehr ändern. Dieser Fehler in der Kommunikation wurde bei der Durchführung des Projekts gemacht.

Kanban schließlich nutzte sowohl synchrone als auch asynchrone Iterations- und Release-Zyklen. Der asynchrone Zyklus fiel beiden Entwicklern sehr schwer, da ein

Release mitten in der Iteration stattfand. Beiden war nicht klar, was sie während der restlichen Iteration tun sollten. Stattdessen wurden schließlich Releases nur ans Ende von Iterationen gelegt, aber die Anzahl der Iterationen war variabel. Dies stellte eine gute und sehr flexible Lösung dar.

Änderungen der Anforderungen und der Arbeitsweise

Der Zeitpunkt, zu dem der Kunde ändernde Anforderungen kommunizieren durfte, war bei allen drei Prozessen entsprechend der Definition von Agilität in der Software-Entwicklung nicht eingeschränkt. Allerdings traten sie in allen drei Prozessen nur während der Vorstellung der Lieferungen auf, da hier der Kunde direkt mit dem Produkt in Kontakt kam. Ansonsten konnten insbesondere neue Anforderungen jeweils frühestens in der nächstfolgenden Iteration berücksichtigt werden, was den Kunden jedoch kommuniziert worden war. Abgesehen davon konnten bei Crystal durch das Einbeziehendes des Kunden und Anwenders ins Team direkt während der Iteration Änderungen vorgenommen werden. Insbesondere das Ablehnen von User Stories am Ende der Iteration wurde so teilweise verhindert.

Um die Arbeitsweise kontinuierlich zu verbessern, besitzen alle drei Prozesse Meetings zur ständigen Evaluierung. Bei Kanban insbesondere wurde der Operations Review am Ende eines Releases durchgeführt, was zum Teil zu sehr langen Zeiträumen ohne Veränderung führte. Obwohl es prinzipiell erlaubt und erwünscht ist, jederzeit Vorschläge zu machen, beispielweise zum Daily Standup Meeting, wurde dies so gut wie nie wahrgenommen. Auch hier liegt der Grund in der ungewohnt offenen Vorgehensweise. Die Entwickler waren es eher gewohnt, zu bestimmten Meetings Verbesserungsvorschläge zu machen. Außerdem betrafen einige dieser Ideen den Board-Aufbau, so z.B. das Einführen von Swimlanes, und diesen wollte man nicht mitten in der Iteration ändern.

Bei Crystal Clear und Scrum hingegen ist es tatsächlich nur vorgesehen, im entsprechenden Meeting Änderungen im Ablauf vorzuschlagen und zu beschließen. Der Reflexions-Workshop bei Crystal und die Retrospektive bei Scrum finden ebenfalls jeweils am Ende des Releases statt. Während die kurze einwöchige Phase bei Scrum als optimal empfunden wurde, waren die zwei Wochen bei Crystal bereits zu lang, um sich an Einzelheiten zu Beginn des Releases zu erinnern. Deshalb wäre es vielleicht besser, das Meeting nach jeder Iteration durchzuführen und damit einmal wöchentlich. Dieser kurze Rhythmus wurde von allen Teams begrüßt beziehungsweise für ihren Prozess vorgeschlagen. Auch eine Scrum-Projekt mit

vierwöchigen Sprints sollte nicht nur alle vier Wochen eine Retrospektive abhalten. Dies ist zu unflexibel und erst recht zu lang, um sich an der Anfang der Iteration zu erinnern.

Meetings und Kommunikation

Sowohl die Umsetzung der verschiedenen Meetings als auch die Teamkommunikation gestalteten sich in den Prozessen sehr unterschiedlich. Prinzipiell wurden jedoch alle Evaluierungs-Meetings dazu genutzt, den jeweiligen Prozess für dieses Projekt und dieses Team zu verbessern und zu optimieren. Alle Teams hatten hierbei das Problem, dass sie nicht notwendigerweise im gleichen Raum waren, wenn die Dailies stattfanden. Dies war im Prinzip kein Problem solange alle das gleiche Board sehen.

Bei Kanban insbesondere wurden die Daily Meetings dazu genutzt, bereits Umsetzungsdetails für den aktuellen Tag zu klären. Das Triage-Meeting, welches regelmäßig prüft, ob User Stories entfernt werden können, weil sie zu alt oder schon erledigt sind, wurde hingegen nicht durchgeführt. Das Team sollte schließlich alle User Stories umsetzen. Allerdings wäre es hilfreich gewesen, um zu sehen, dass einige der vorhandenen User Stories bereits bei der Umsetzung anderer User Stories vollendet und damit überflüssig waren. Ohne dieses Meeting stellte das Team dies immer erst im Planning fest, wenn es darum ging, Tasks zu schreiben.

Bei Crystal Clear liefen alle Meetings wie gewünscht ab, allerdings entpuppte sich die Umsetzung der osmotischen Kommunikation als schwierig, da das Team meist räumlich getrennt war. Gleichzeitig benötigt man mindestens drei Personen, damit sie überhaupt funktioniert. Bei größeren Teams könnte sich diese Art der Kommunikation jedoch auch als störend für einige Teammitglieder erweisen.

Scrum schließlich hielt sich strikt an die Vorschläge für den Scrum-Prozess. Da beide Entwickler die drei Fragen im Daily meist übergingen, wurden sie als Pflichtaussagen mit Tabelle aufgenommen, die der Scrum Master täglich abhakte. Dies erleichterte die Kommunikation stark, da niemand zu sehr abschweifte und Unwichtiges erzählte. Außerdem wurde so das Daily kurz gehalten. Das Planning wurde mit wachsender Erfahrung kürzer und die Abschätzung mit Story Points genauer, so dass sich ein positives Gefühl im Team entwickelte. Dies gilt prinzipiell jedoch auch für die beiden anderen Prozesse.

Dokumentation

Die Menge und Art der Dokumentation weicht in allen drei Prozessen voneinander ab. Den größten Aufwand hatte definitiv das Crystal-Team. Hier wurde am Anfang des Projekts die Grundsteinlegung erstellt, die bei der Organisation des weiteren Projekts jedoch sehr hilfreich war und den Chefdesigner als eine Art Ersatz-koordinator zur Seite stand. Ansonsten wurden lediglich ein Klassenmodell und das Design verschriftlicht. Diese sollten jedoch öfter auf Aktualität geprüft werden, sonst helfen sie wenig. Prinzipiell ist die Art der Dokumentation jedoch auch hier flexibel auswählbar; dies waren nur Vorschläge.

Dokumentation bei Scrum ist genauso freiwillig. Das Team dokumentierte lediglich die Retrospektiven und die daraus resultierende Definition of Done (DoD), die erklärt, wann ein Task geschlossen werden kann. Selbst das Burn-Down-Chart wurde bereits automatisiert vom Computer berechnet und nicht vom Scrum Master per Hand gezeichnet. Die DoD half besonders dabei, einen bestimmten Arbeitsablauf einzuhalten und führte zu mehr Abnehmen von Tickets durch den Kunden, und das Grundgerüst kann auch auf andere Projekte übertragen werden.

Das Kanban-Team schließlich dokumentierte gar nichts außerhalb des Kanban-Boards. Änderungen wurden sofort umgesetzt oder ans Board geschrieben. Der einzige Nachteil daraus entstand durch das Fehlen von Sprints in JIRA, so dass es schwierig war, auf vergangene Iterationen zurückzublicken und beispielsweise die früheren WIP-Limits oder Iterationslängen nur vage aus dem Gedächtnis bekannt waren. Dies würde das Team im nächsten Kanban-Projekt unbedingt schriftlich festhalten. Das Weglassen jeglicher Dokumentation wurde demnach als keine gute Arbeitsweise empfunden.

Task-Größe und Aufwandsschätzung

Crystal und Scrum haben keine genauen Vorgaben zur Größe der Tasks oder User Stories. Beide Teams entschieden sich jedoch nach den ersten zwei bis drei Iterationen für möglichst kleine Tasks, da diese leichter fertig zu stellen und abzuschätzen waren. Beides erleichterte ebenso die Planung. Die Abschätzung bei Crystal Clear erfolgte bereits für User Stories und Tasks in der Blitzplanung zu Beginn und wurde später während den Planungsphasen für die Iterationen überarbeitet. Dies war nötig, da sich wie bei den übrigen Prozessen das Schätzungsvermögen des Teams ständig verbesserte. Trotzdem erleichterte die Vorabschätzung aus der Blitzplanung diesen Prozess. Falls ein Entwickler nicht sicher bei der neuen Abschätzung war, konnte er sich notfalls daran orientieren.

Bei Scrum wurden vor jedem Sprint im Sprint Planning User Stories abgeschätzt, indem die beiden Entwickler gleichzeitig eine bestimmte Anzahl Finger zeigten. Dies wurde alternativ zum Planning Poker durchgeführt, da keine Karten vorhanden waren. Für kleine Teams ist das eine gute Alternative, für größere wären vermutlich Karten besser, da man schnell den Überblick verliert und Entwickler so ihre Fingerzahl noch ändern können. Das ist aber nicht gewünscht. Auf Grund der Eigenheit von Yodiz musste das Team für dieses Board die Tasks auch abschätzen. Dies stellte sich als weitaus einfacher und akkurater heraus als die User Stories und wäre zumindest in neuen Teams, die das Abschätzen erst noch lernen, vielleicht eine gute Alternative zum Anfang. Dann müssen nur die Task Story Points addiert werden um zur User-Story zu gelangen.

Für Kanban sollten die Tasks möglichst gleich groß sein, so dass sie gleichmäßig durch das Board „fließen“ können. Dies stellte sich als nicht durchsetzbar für das Team heraus, insbesondere bei Pair Programming Tasks, die automatisch die doppelte Zeit beanspruchen. Deshalb wurden Swimlanes genutzt, um ähnlich große Tasks wie Tests zu gruppieren. Allerdings ging dadurch der Zusammenhang zu den User Stories verloren – also keine gute Idee in diesem Fall. Die Change-Request – Swimlane allerdings funktionierte mit einer Dauer von 30 Minuten pro Ticket sehr gut. Alternativ sollte mit SLAs statt Swimlanes gearbeitet werden, was jedoch leider in beiden Tools nicht vorgesehen ist und deshalb nicht vom Team getestet wurde. Die Aufwandsschätzung wurde bei Kanban direkt über Schätzung der Tasks und nicht der User Stories durchgeführt. Sie verbesserte sich wie bei den anderen Teams von Iteration zu Iteration und erlaubte es am Ende auch das nächste Release-Datum und damit effektiv drei Iterationen ziemlich akkurat zu schätzen.

Priorisierung

Die Priorisierung ging bei allen Prozessen relativ gleich vonstatten. Der Kunde oder Anwender konnte prinzipiell am Anfang des Projekts eine Reihenfolge der User Stories festlegen. Diese konnte jederzeit geändert werden für alles, was nicht bereits in der laufenden Iteration bearbeitet wurde. Während bei Scrum die Abarbeitung des Boards von oben nach unten strikt eingehalten wurde, war dies bei Kanban nicht immer der Fall. Durch diesen Freiraum war es möglich, dass ein Teammitglied an einer User Story arbeitete, die es allein fertig stellen konnte, statt wie bei Scrum auf einen anderen Entwickler warten zu müssen, der behilflich ist. Dies war insbesondere bei flexiblen Arbeitszeiten sehr hilfreich, die sich nur zum Teil

überschneiden, machten das Daily aber besonders wichtig für die Absprache.

Empirie

Kanban ist eine Prozess, bei dem Empirie eine sehr große Rolle für den Verbesserungsprozess spielt. Allerdings stellte sich Yodiz als nicht hilfreich hierfür heraus, so dass lediglich JIRA verwendet werden konnte, um Bottlenecks im Fluss zu finden. Dies gelang beispielsweise als der Review der Tasks vom Team ignoriert wurde. In Burn-Up-Chart wäre insbesondere für die längeren Iterationen hilfreich um wünschenswert gewesen, um dem Team ein Gefühl für ihren Fortschritt zu geben.

Die beiden anderen Prozesse nutzten Empirie wenig bis gar nicht. Bei Scrum wurde das Burn-Down-Chart nicht zum Vergleich des Voranschreitens in der Iteration verwendet, obwohl es vorhanden war, da JIRA lediglich abgeschlossene User Stories anzeigt und deren Anzahl für den sehr kurzen Sprint und das sehr kleine Team entsprechend gering war. Bei größeren Teams und eventuell längeren Sprints hingegen ist das Diagramm ein sehr hilfreiches Tool. Zumindest die berechnete Velocity nutzte das Team aber für die Planung des nächsten Sprints. Crystal nutzte keinerlei empirische Methoden während des Projekts. Trotzdem funktionierten das Team und die Planung sehr gut. Möglicherweise halfen hier andere Maßnahmen wie beispielsweise die Blitzplanung oder auch die gute Absprache zwischen den Teammitgliedern.

Kundenkontakt

Der Kundenkontakt stellte sich bereits als erstaunlich schwierig heraus, wenn man bedenkt, dass in diesem Fall für alle drei Projekte die Kunden genauso oft verfügbar waren wie die Entwickler. Trotzdem gab es in allen Gruppen lediglich Kommunikation mit dem Kunden zum Release-Datum. Dies war besonders bei Kanban und Crystal während der längern Zyklen zu selten. Allerdings sollten insbesondere bei Crystal der Kunde und der Anwender ständig gefragt werden können – sie sind schließlich Teil des Teams. Aber auch bei den beiden anderen Prozessen ist die Kommunikation mit dem Kunden während der Iteration erlaubt und erwünscht. Darauf sollte unbedingt bei der Umsetzung aller drei Prozesse in neuen Teams geachtet werden, da es ein Umdenken zum „alten“ Vorgehen erfordert, bei dem ich als Entwickler vielleicht zusammen mit anderen Entwicklern entscheide, was der Kunde wahrscheinlich gemeint haben könnte.

Fazit/ Endergebnis/ Zusammenfassung

Insgesamt gesehen hatte jeder Prozess Vor- und Nachteile, die oft mit der Teamgröße und Vorerfahrung zusammenhingen. Aus den gemachten Erfahrungen und dem theoretischen Vorwissen über die Vorgehensweisen, entstand die folgende Einschätzung dafür, wann welcher Prozess gut geeignet ist.

Crystal Clear befindet sich durch die Blitzplanung und die Dokumentation mittels Grundlegung noch recht nahe am „nicht-agilen“ Projektmanagement und bietet somit vielleicht einen guten Einstieg in Agilität – sowohl für Kunden als auch für Entwickler. Die Dokumentation ist nicht plötzlich weggefallen und auch eine Art Planung des gesamten Projekts ist vorhanden und gibt eine gewisse Sicherheit für alle Beteiligten. Das Team sollte jedoch nicht zu klein gewählt werden und man benötigt für osmotische Kommunikation tatsächlich Mitarbeiter von Kundenseite, die mit den Entwicklern im selben Raum sitzen – und zwar langfristig. Dies ist wohl am ehesten möglich bei innerbetrieblichen Projekten.

Scrum hingegen eignet sich eher für einen Einstieg, bei dem Kunde und Entwickler damit zufrieden sind, dass nicht alles vollständig vorher geplant und dokumentiert ist. Auch hier ist eine gewisse Mindestgröße des Teams nötig, um die geforderten Rollen auszufüllen. Dafür muss der Kunde und Anwender nicht ständig mit den Entwicklern im selben Raum oder Gebäude sein, solange anderweitige Kommunikation z.B. via Telefon möglich ist. Der geregelte Ablauf in Scrum, d.h. die gleich bleibende Iterationslänge, die verschiedenen Meetings und Artefakte, ist gut geeignet für Einsteiger in agiles Projektmanagement. Sie helfen dabei, sich in die selbstorganisierte Arbeitsweise einzufinden und mit der recht großen Planungsfreiheit umzugehen.

Kanban kann sich ebenfalls als Einstieg in agiles Prozessmanagement eignen, allerdings nicht sofort mit all seinen Freiheiten und Möglichkeiten. Dinge wie asynchrone Iterationen und Releases erfordern sehr viel Erfahrung mit dem Ablauf und sind für Einsteiger eher ungeeignet. Das WIP-Limit und ein einfaches Kanban-Board hingegen, welches praktisch dem Scrum-Board gleicht, sind sehr gute Einstiegspunkte. Da Kanban prinzipiell dazu dient, den bestehenden Prozess stufenweise zu verändern, kann es jedoch auch in einem Team genutzt werden, das bereits andere agile Techniken wie Scrum oder Crystal einsetzt. Dann können auf Grund der Vorerfahrung auch gleich fortgeschrittene Techniken wie Swimlanes eingesetzt werden. Was insbesondere für den Einsatz von Kanban spricht, ist also

eine Teamstruktur, bei der der Prozess eher langsam geändert werden soll, statt ihn komplett zu ersetzen. Außerdem funktioniert Kanban sowohl bei kleinen als auch großen Teams sehr gut, insbesondere wenn verschiedene Mitglieder spezifische Aufgaben erfüllen, wie beispielsweise Konzeptersteller, Programmierer oder Tester.

8.2 Auswertung der Tools

Die Gegenüberstellung der Tools Jira und Yodiz erfolgt auf der Auswertung der Anwendung im Vergleich zu den Angaben der Hersteller. Dazu wurden die beiden Tools parallel für alle drei Prozesse eingesetzt. Die dabei endeckten Vor- und Nachteile werden zuerst anhand einer Matrix, später dann ausführlich erläutert.

Zuerst muss gesagt werden, dass beide Tools gut für den Einsatz von Scrum und Crystal geeignet sind. Die Vertreiber von Yodiz gaben an, dass ihre Software ein Kanban-Board unterstützt. Jedoch musste hier leider schnell festgestellt werden das dies nicht im Ansatz der Fall ist. Denn viele Kanban spezifische Eigenschaften, wie Swimlanes werden nicht bereitgestellt.

	Vorteile	Nachteile
Beides		• sortiert automatisch
Jira	• Board/ Workflow anpassbar • Swimlanes möglich (Kanban) • merkt Einstellungen • git commits--> Tickets verknüpfbar • beliebige Zeit logbar • viele Addons	• alle Tickets im Board (Kanban) • keine Iterationen (Kanban) • schlechte Trennung Obertasks und Subtasks • komplex • nicht live logbar • Addons kosten
Yodiz	• wenige Klicks • übersichtlich • leichtes Handling • gute Steuerung von Lieferung/Release • Zeit live loggen • Verlinken von Bugs + User Stories • Planning board • TO-DO-Liste	• Board nicht anpassbar • Iterationen → Muss • Tasks → Muss • keine kanban-Diagramme • keine Swimlanes • manuelles Loggen → nur 15 min Intervall • keine Verlinkung von Tasks • manche Tasks nicht logbar • von externem Server abhängig →Ausnahme Enterprise Edition

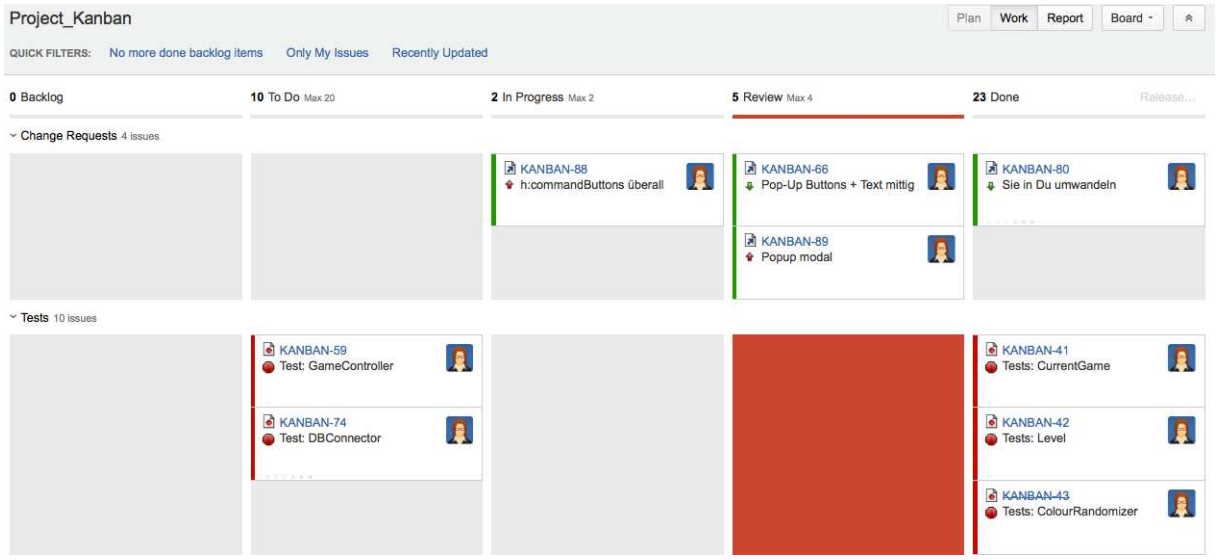
Einen Nachteil, den beide eTracking Systeme gemeinsam haben, ist dass die Ticketnummern in der Reihenfolge, in der sie angelegt werden, nummeriert und somit auch priorisiert werden. Dabei wird auch nicht berücksichtigt wo die einzelnen Tasks per Drag und Drop hingeschoben werden. Dies bedeutet, wenn ein Ticket per Drag und Drop weiter nach oben gezogen werden soll, weil es jetzt doch früher abgearbeitet werden soll übernimmt keines der beiden Programme diese Änderung. Hier sind in beiden Fällen die Änderungen der Priorisierung deutlich aufwendiger.

Bei Jira haben sich unter anderem die folgenden Vorteile herauskristallisiert. So ist

es bei Jira unter anderem möglich den Workflow anzupassen was auch eine Veränderung des Boards nach sich zieht. So kann hier Beispielsweise eine neue Spalte Review, wie in diesem Projekt genutzt, eingefügt werden. Dabei wird dann in den Einstellungen angegeben von welcher Spalte man in die Review Spalte gelangt und wo hin man von Review Spalte kommt. Weiterhin kann auch der ganze Lifecycle eines Tickets verändert werden. Das heißt hier können Änderungen gemacht werden von wo ein Ticket kommt und in in welche Spalte es geschoben werden darf. Zudem merkt sich Jira die Einstellung des letzten Tickets um das Anlegen neuer Tickets zu erleichtern. Weiterhin ist die Zusammenarbeit von Git und Jira sehr gut, denn hier kann man die Tickets mit den commits in Git durch einen Link verknüpfen. Weiterhin bietet Jira die Option die Arbeitszeit der Tickets Minuten genau zu loggen.

Weiterhin bietet es für Kanban die Möglichkeit Swimlanes sowie eine maximale Anzahl an Tasks welche gerade in einer bestimmten Spalte sein dürfen.

Nachteilig bei Jira ist im Bezug auf das Kanban Board das keine Iterationen angelegt werden können und somit alle Tasks immer sichtbar sind. Die einzelnen Tasks können lediglich durch ein Release Ticket aus dem Board entfernt werden. Weiterhin ist es schwierig die Anzeige von Obertasks und Untertasks zu trennen, wenn bereits wie in Kanban üblich, Swimlanes genutzt werden. Da die Trennung von Ober-und Untertasks üblicherweise durch Swimlanes erfolgt. In dem unten abgebildeten Screenshot sieht man sowohl die Swimlanes für Change Requests wie auch für Tests zudem die Maximale Anzahl an Work in Progress Tasks, welche in der Review Spalte überschritten wurden.

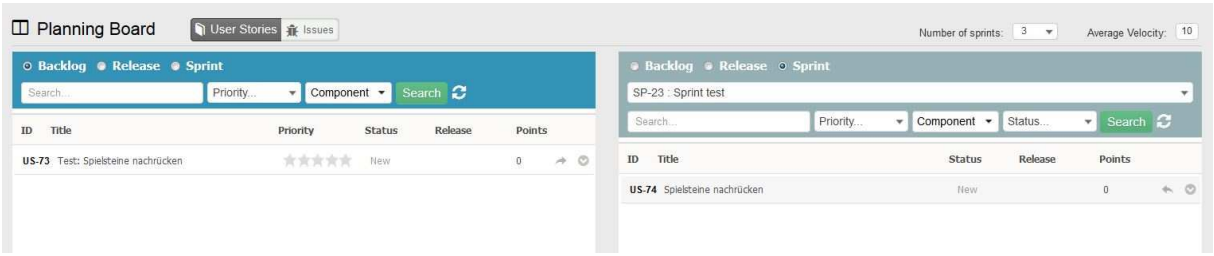


Ein weiterer Nachteil ist das Jira sehr komplex ist und für unerfahrene Benutzer sehr

viel Einarbeitungszeit benötigt, da besonders für die Einrichtung viele Klicks erforderlich sind. Im Gegensatz zu Yodiz bei dem die Zeit live loggbar ist, kann Jira dies ohne zusätzliches Add- On leider nicht. Nachteilig hierbei ist auch noch das sämtliche Add-Ons mit zusätzlichen Kosten verbunden sind.

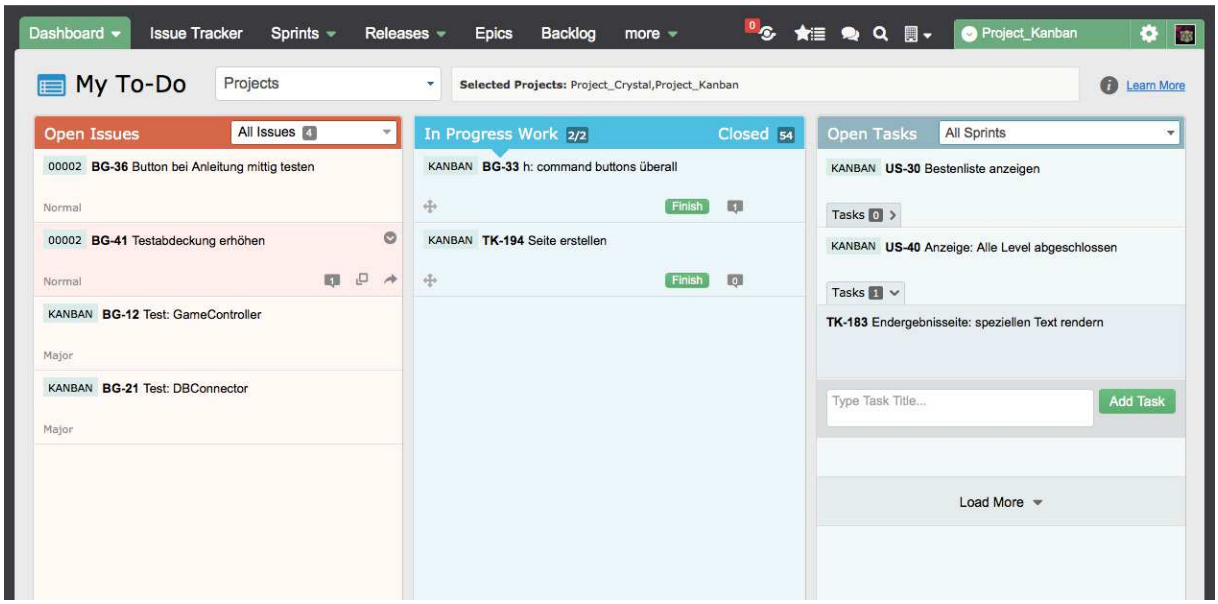
Yodiz hat gegenüber von Jira die Vorteile das die Einarbeitung innerhalb sehr kurzer Zeit möglich ist. Dies hat zur Folge das nur wenige Klicks benötigt werden um an das gewünschte Ziel zu kommen. Weiterhin ist es Yodiz sehr übersichtlich Aufgebaut was eine intuitive Nutzung und somit ein leichtes Handling begünstigt. Zudem Bietet Yodiz die Möglichkeit live auf einem Task die Zeit mit zu loggen. Dazu muss aber erwähnt werden das dies nur von einer Person und nicht von mehreren zur Selben Zeit möglich ist.

Zwei weitere gute Eigenschaften von Yodiz werden sowohl beim Planning Board wie auch bei der Planung von Releases/Lieferungen sichtbar. Hier ist es sehr einfach per Drag an Drop User Stories von Planning Board in die Iteration zu ziehen, oder von der Iteration in den Release.



Weiterhin hat man bei Yodiz die Möglichkeit einen Bug mit einer bestehenden User Story zu verbinden, um genau zu sehen welcher Bug mit welcher User Story in Verbindung steht. Hierbei nimmt Yodiz einen Teil der Arbeit ab, denn man muss zwar den Bug mit der User Story verbinden die umgekarte Verbindung von der User Story zum Bug wird jedoch automatisch von Yodiz hinzugefügt.

Ein weiterer Vorteil von Yodiz bietet das MY-TO-DO Board, dieses Bord ist besonders sinnvoll wenn man wie bei dieser Studienarbeit an mehreren Projekten gleichzeitig arbeitet. In diesem Board kann man sich über die Projektgrenzen hinaus anzeigen lassen was innerhalb dieser Iteration noch an Aufgaben, von dieser Person, abgearbeitet werden müssen.



Wie jedes Tool bietet Yodiz aber auch Nachteile, so lässt sich unter anderem das Board nicht Anpassen, was bedeutet das weder der Workflow noch das Board angepasst werden können. Dies ist ein Nachteil wenn zum Beispiel die Vereinbahrung getroffen wurde das jede User Story gereviewed werden soll sobald sie fertig ist. Da man hier keine Review Spalte einfügen kann ist es für die anderen Teammitglieder schwer ersichtlich ob die User Stroy oder der Task bereits fertig sind und gereviewed werden können oder ob die noch in Bearbeitung sind. Hierzu hat man zwar die Möglichkeit Comments an die einzelnen Tasks anzuhängen, jedoch werden die anderen Benutzer nicht aktiv darauf aufmerksam gemacht, das ein neuer, von ihnen noch ungelesener Comment existiert.

Weiterhin bietet Yodiz nicht nur die Möglichkeit automatisch die Zeit mitzuloggen sondern auch das manuelle Loggen der Zeit per Hand. Jedoch genau hier zeigt die Plattform zwei Schwächen, denn einerseits können beim manuellen Loggen nur vorbestimmte Zeiten im 15 Minuten Tackt geloggt werden. Zum anderen ist es bei dem letzten Task innerhalb einer Spalte an einigen PCs nicht möglich die Zeit zu loggen, da hier bedingt durch die Bildschirmgröße und Auflösung die Buttons zum Loggen nicht mehr angezeigt werden.

Ein weiterer Nachteil sind die fehlenden Verlinkungen von Tasks untereinander. So können die Tasks unter anderem nicht in eine Abhängigkeit gesetzt werden.

Ein weiterer Nachteil ist das es ausschließlich in der Enterprise Edition möglich ist das Programm mit den eigetragenen Daten selbst zu hosten. In allen anderen Fällen

ist man von dem externen Server der Firma abhängig.

Dies kann zur Folge haben das die Benutzer bei der Wartung des Webservice nicht auf ihr eTracking System zugreifen können. Dies kann zu extremen Problemen führen wenn die Teammitglieder nicht am gleichen Standort arbeiten und sich somit auch nicht persönlich absprechen können, wer welche Aufgaben erledigt.

Wie schon erwähnt erfüllt Yodiz auch nicht die Anforderungen welche an ein Kaban Board gestellt werden. So muss man zum Beispiel Iterationen zwingend anlegen, obwohl Kanban auch ohne Iterationen durchführbar ist. Weiterhin ist man gezwungen Tasks innerhalb einer User Story anzulegen. Weiterhin werden auch keine Swimlanes angeboten, welche typisch sind für ein Kanban Board.

Abschließend muss auch noch erwähnt werden das Yodiz sehr wohl viele Diagrammtypen zur Darstellung verschiedenster Gegebenheiten anbietet, jedoch auch hier keine speziellen Kanban Diagramme angeboten werden.

9. Fazit

Im Folgenden werden die Durchführung und die Ergebnisse dieser Arbeit noch einmal rückblickend kritisch betrachtet. Darüber hinaus wird auf die Frage eingegangen, wie die gewonnenen Ergebnisse zu verstehen und ob diese allgemeingültig sind.

Wie zuvor bei der Wahl der Projekte beschrieben, war es nicht so einfach die drei agilen Prozesse gleichzeitig durchzuführen. Die gewählte Methode, bei der im Grunde das gleiche Produkt in drei Variationen entwickelt wurde, hatte trotz der positiven Aspekte durchaus auch einige Nachteile.

Einerseits war es für die Teammitglieder eine große Herausforderung in zwei Projekten gleichzeitig zu arbeiten. Dadurch konnten sie sich einerseits nicht auf ein Projekt konzentrieren und andererseits war es oftmals schwierig die Projekte auseinander zu halten. Um dem entgegen zu wirken wurde auch speziell im Daily Meeting jedes Projekt separat besprochen und darauf geachtet, dass sich die Projekte nicht vermischen. Bei der Abarbeitung der einzelnen Tasks wiederum musste jedes Teammitglied selbst entscheiden, welches Projekt die höhere Priorität erhielt. Zum Teil wurden dadurch bestimmte Projekte bevorzugt bearbeitet oder die Teammitglieder waren zum Schluss einer Iteration oder Lieferung überfordert. Somit musste bereits bei der Planung beachtet werden, dass die Teammitglieder nicht Vollzeit zur Verfügung stehen.

Andererseits musste durch den zusätzlichen Vergleich von zwei Softwareentwicklungstools der doppelte Aufwand zur Pflege und Synchronisation der Tools betrieben werden. Hier kam es des öfteren zu Inkonsistenzen der zu pflegenden Boards, weil es für die Entwickler ungewohnt war, zwei Tools gleichzeitig zu benutzen und zu pflegen.

Außerdem war die mangelnde Zeit durch paralleles Studium und Bearbeitung von mehreren Projekten problematisch für die effektive Durchführung der Prozesse. Jede Iteration musste an den flexiblen Stundenplan der Teilnehmer angepasst werden und konnte somit nicht wie gedacht geregelt und gleichmäßig geplant werden. Der normalerweise sehr routinierte Tagesablauf bei agilen Prozessen, welcher meistens mit einem Standup-Meeting beginnt und aus einer Abfolge von Entwicklung, Tests, Builds und Integrationen besteht, konnte durch die Vorlesungen nur teilweise eingehalten werden. Darüber hinaus fehlte die Zeit, um gemeinsam im selben Raum

zu arbeiten um Techniken wie Pair-Programming und osmotische Kommunikation wirkungsvoll einsetzen zu können. Im Allgemeinen konnten die generellen Ansätze der verschiedenen Prozesse während der parallelen Durchführung jedoch gut umgesetzt werden und die einzelnen Vor- und Nachteile und Unterschiede wurden deutlich sichtbar.

Ebenso stellt sich die Frage, ob die Durchführung der Prozesse in dieser Arbeit vergleichbar mit dem produktiven Einsatz in Unternehmen ist. Zum Einen erscheint eine Arbeitsstunde pro Tag nicht sehr aussagekräftig, da im Normalfall acht Stunden täglich gearbeitet wird. Dennoch muss es auch im Unternehmen nicht immer der Fall sein, dass man ausschließlich an einem Projekt beteiligt ist. Außerdem müssen die Teammitglieder oftmals während der Projekte ihre Aufgaben der Linie bzw. des Alltagsgeschäfts erledigen, wodurch auch hier nicht die komplette Tageszeit zur Verfügung steht.

Die Ergebnisse dieser Arbeit sind demnach nicht als allgemeingültig zu deuten, da ein solcher Vergleich von vielen Faktoren wie Projektanforderungen, Teamzusammensetzung bezüglich Größe und Fähigkeiten sowie äußere Gegebenheiten, abhängt und dadurch auch teilweise mehr subjektiv ist, da die Erfahrungen der Teammitglieder einfließen. Dennoch soll der Vergleich als Hilfestellung dienen und eine Idee vermitteln, auf welche Aspekte man bei der Prozess- und Toolwahl achten sollte.