Die approbierte Originalversion dieser Diplom-/Masterarbeit ist an der Hauptbibliothek der Technischen Universität Wien aufgestellt (http://www.ub.tuwien.ac.at).

The approved original version of this diploma or master thesis is available at the main library of the Vienna University of Technology (http://www.ub.tuwien.ac.at/englweb/).

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Lean and Agile Software Development: Planung und Realisierung einer Verbindung von Kanban und Scrum

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Wirtschaftsinformatik

eingereicht von

Raoul Vallon
0525496

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung
Betreuer: Thomas Grechenig

Wien, 31.5.2011

(Unterschrift Verfasser) (Unterschrift Betreuer)

Lean and Agile Software Development: Planung und Realisierung einer Verbindung von Kanban und Scrum

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Wirtschaftsinformatik

eingereicht von

Raoul Vallon

0525496

ausgeführt am

Institut für Rechnergestützte Automation

Forschungsgruppe Industrial Software

der Fakultät für Informatik der Technischen Universität Wien

Betreuung:

Betreuer: Thomas Grechenig

Wien, 31.5.2011

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benützt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Wien, am 31.5.2011

Raoul Vallon

"It is difficult to understand the universe if you only study one planet."
Miyamoto Musashi
(Samurai, 17th Century)

Danksagung

Zuallererst möchte ich meiner Familie danken, deren Unterstützung mir dieses Studium der Wirtschaftsinformatik erst ermöglicht hat. Ich möchte mich bei allen Freunden bedanken, neuen wie alten, mit denen ich die Herausforderungen des Studiums so gut gemeinsam meistern konnte.

Ich möchte mich weiters bei Michael Müller-Wernhart für die wissenschaftliche aber auch persönliche Betreuung bedanken, der mir bei Fragen jeglicher Art zur Seite stand. Weiters möchte ich Thomas Grechenig für die Betreuung des Themas und die Schärfung des Konzeptplans danken.

Zu guter Letzt gilt den Mitarbeitern des österreichischen IT-Unternehmens großer Dank, in dem die Vorschläge meines Konzepts umgesetzt wurden. Die zahlreichen Interviews und Diskussionen über den gesamten Verlauf der Diplomarbeit hin sind ein zentraler Bestandteil der Arbeit. Ich möchte allen Personen im Unternehmen danken, die direkt oder indirekt von der Umsetzung meiner Vorschläge betroffen waren, und damit geholfen haben die Diplomarbeit voranzutreiben. Ich bedanke mich für die kompetente, freundliche und unkomplizierte Zusammenarbeit.

Kurzfassung

Lean Development, auch bekannt als „Just in Time“ Produktion, hat bereits seit einigen Jahren erfolgreich Einzug in die Software Entwicklung gehalten und gewinnt auch in Österreich zunehmend an Popularität. Der dynamische Prozess verwirklicht ein „Pull“-System, das jedoch auch zahlreiche Gemeinsamkeiten zu agilen Prozessen aufweist. Ziel dieser Arbeit ist die Erarbeitung eines Konzepts, das Lean Development anhand des Vertreters „Kanban“ und den agilen Softwareprozess „Scrum“ verbindet, sodass die Stärken der beiden Prozesse vereint werden.

Der erste Teil der Arbeit befasst sich mit der Erstellung eines Konzepts, in dem sowohl Kanban- als auch Scrum-Elemente enthalten sind. Den zweiten Teil stellt die praktische Erprobung des Konzepts dar. Als Fallbeispiel wurden Teile des Konzepts für eine Umsetzung im Rahmen einer Praxisphase ausgewählt. Diese Prüfung der Praxistauglichkeit bietet weiteres Feedback für eine abschließende Verbesserung des Konzepts.

Die Ergebnisse aus der praktischen Erprobung zeigen, dass die Kombination von Kanban und Scrum einen sinnvollen Ansatz für ein effizientes Projektmanagement darstellt. Beispielsweise kommt es in Software Projekten aus verschiedensten Gründen zu Änderungen der Priorisierungen. Mit dem dynamischen Kanban Workflow wird den Projektmitarbeitern ein Werkzeug gegeben, das aufgrund seiner Flexibilität die Möglichkeit bietet, dennoch eine effiziente Umsetzung des Projekts zu gewährleisten.

Auf der anderen Seite konnten die in der Praxis erprobten Scrum-Elemente des Konzepts, wie Retrospektive oder Review, Raum schaffen, auch im dynamischen Kanban-Prozess regelmäßig über den Prozess zu reflektieren und den Fortschritt zu überprüfen. Für den Einsatz dieser beiden Scrum-Elemente wurde ein im Rahmen des Konzepts entwickelter „Sprint ohne Commitment“ eingeführt, der sich als ideales Mittel herausstellte, um einen Zeitpunkt für Retrospektive und Review zu bestimmen und gleichzeitig jedoch vom Scrum-üblichen Commitment abzusehen, mit dem ein Kanban Workflow nicht durchführbar wäre.

Ein weiteres Ergebnis der vorliegenden Arbeit stellt die Erweiterung des Scrum Task Boards zu einem Kanban-Board dar: Durch die realitätsnahe Visualisierung des Workflows konnten Flaschenhälse identifiziert werden und natürliche Wartezeiten eines Prozesses in Form von Queues modelliert werden.

Keywords: *Lean, Kanban, Agil, Scrum, Just in Time, Software Prozess, Software Entwicklung*

Abstract

Lean Development, also known as „Just in Time“ Production, has found its way into Software Engineering in recent years and is also beginning to grow in popularity in Austria. The dynamic software process is implementing a “Pull” system that also shows similarities to Agile Processes. The aim of this master thesis is to develop a concept that connects Lean Development by example of “Kanban” with the Agile Software Process “Scrum” in order to combine the strengths of both processes.

The first part of the thesis deals with the development of a concept, which contains characteristics of both Kanban and Scrum. The second part covers a practical test of the concept. Parts of the concept have been selected for this case study. This practical examination of the concept promises additional feedback to be used for further improvement of the concept.

The results of the practical test show that the combination of Kanban and Scrum presents a sound approach to efficient project management. Priorities in software projects, for instance, are subject to change due to a variety of unforeseen reasons. By employing the dynamic Kanban workflow IT professionals are given a tool, which offers the flexibility to cope with changes in priorities and thus ensure the efficient execution of the project.

Scrum-own properties of the concept, on the other hand, like Retrospective and Review, managed to introduce reflecting and progress-checking procedures to the dynamic environment of Kanban. As a timeframe for these procedures a “Sprint without commitment” has been developed as part of the concept, which proved to be a very suitable means to set a time for Retrospective and Review while not submitting to Scrum’s Sprint Commitment, which couldn’t be handled by a Kanban workflow.

Another result of the thesis at hand is the extension of the Scrum Task Board to a Kanban Board: By means of the workflow’s close-to-reality visualization, bottle necks could be identified and natural waiting times that every process has could be modeled as queues.

Keywords: *Lean, Kanban, Agile, Scrum, Just in Time, Software Development, Software Process*

Inhaltsverzeichnis

Abbildungsverzeichnis IV

Tabellenverzeichnis V

1 Einleitung 1
1.1 Problemstellung 1
1.2 Motivation und Zielsetzung 1
1.3 Methodisches Vorgehen 2
1.4 Aufbau der Arbeit 3
1.5 Praxisphase 4

2 Theoretischer Hintergrund 5
2.1 Linear: Wasserfallmodell 5
2.1.1 Ursprung 5
2.1.2 Prozessüberblick 5
2.1.3 Rollen, Meetings und Artefakte 7
2.2 Iterativ: Rational Unified Process (RUP) 7
2.2.1 Ursprung 7
2.2.2 Prozessüberblick 7
2.2.3 Rollen, Meetings und Artefakte 10
2.3 Agil: Scrum 10
2.3.1 Ursprung 10
2.3.2 Prozessüberblick 11
2.3.3 Rollen 12
2.3.4 Meetings 14
2.3.5 Artefakte 16
2.4 Lean: Kanban 18
2.4.1 Ursprung 18
2.4.2 Prozessüberblick 19
2.4.3 Rollen, Meetings und Artefakte 20
2.5 Analyse Scrum und Kanban 21
2.5.1 Gemeinsamkeiten 21
2.5.2 Unterschiede 23
2.5.3 Zusammenfassung 27
2.6 State of the Art 28
2.6.1 „SPI-LEAM“-Methode 29
2.6.2 To Pull or Not to Pull 30
2.6.3 Issues and Advantages of Using Agile and Incremental Practices 31
2.6.4 Weitere 32

3 Praktische Erkenntnisse ... 33
3.1 Projektverlauf ... 33
3.1.1 Lead Time ... 34
3.1.2 Work in Progress ... 36
3.2 Scrum ... 38
3.2.1 Ablauf ... 38
3.2.2 Vorzüge ... 39
3.2.3 Probleme ... 42
3.3 Kanban ... 44
3.3.1 Ablauf ... 44
3.3.2 Vorzüge ... 45
3.3.3 Probleme ... 48
3.4 Generelle Erkenntnisse ... 53

4 Konzept zur Verbindung von Scrum und Kanban ... 57
4.1 Herleitung ... 57
4.2 Ergebnis praktische Basis ... 58
4.2.1 Vor- und Nachteile Scrum ... 58
4.2.2 Vor- und Nachteile Kanban ... 59
4.2.3 Vor- und Nachteile von Scrum und Kanban konsolidiert ... 61
4.2.4 Verbesserungsvorschläge aus Interviews ... 64
4.3 Problemlösungen ... 65
4.4 Detaillierte Erläuterung des Konzepts ... 67
4.4.1 Wechsel von Scrum auf Kanban ... 68
4.4.2 Das „richtige“ WiP Limit finden ... 69
4.4.3 Feedback Loop verbessern ... 70
4.4.4 Retrospektive einführen ... 71
4.4.5 Analyse mehr integrieren ... 72
4.4.6 Aufwandsschätzung in Kanban ... 72
4.4.7 Burndown-Chart und Cumulative Flow Diagram ... 74
4.4.8 Board-orientiertes Standup Meeting ... 75
4.4.9 „Team Impediments“-Liste für Product Owner ... 76
4.4.10 Kanban-Sprint einführen ... 77
4.4.11 Limit für „To Do“-Spalte stets gering halten ... 78
4.4.12 Umgang mit externen Blockierungen ... 78
4.4.13 Concurrency in Kanban ... 79
4.4.14 Backlog in Kanban richtig benutzen ... 80
4.4.15 Kanban Release Management ... 81
4.4.16 Priorisieren in (und mit) Kanban ... 81
4.4.17 Bug Handling ... 83
4.4.18 Einführung von Queues ... 83
4.4.19 Filter für „Closed“-Spalte ... 84

4.4.20 Unterteilung des Kanban-Boards ... 85
4.4.21 Wartung von Tasks im Ticket Management System ... 87
4.4.22 Nicht einbezogene Lösungsvorschläge ... 87
4.5 Probleme und Konzeptlösungen gegenübergestellt ... 87

5 Praktische Erprobung des Konzepts ... 91
5.1 Vorgehensweise ... 91
5.2 Feedback der ITU ... 91
5.2.1 Initiales Feedback ... 91
5.2.2 Umzusetzende Verbesserungsvorschläge ... 95
5.2.3 Praktische Erfahrungen ... 100

6 Diskussion der Ergebnisse ... 106
6.1 Verbesserung des Konzepts ... 106
6.1.1 Umgesetzte Vorschläge ... 107
6.1.2 Nicht umgesetzte Vorschläge ... 113
6.2 Abschließende Beantwortung der Fragestellung ... 115

7 Conclusio ... 117

Literaturverzeichnis ... 120

Anhang ... i
Interview-Leitfäden ... i
„Kick Off“ mit Geschäftsführer und Scrum Master ... i
Interview mit Scrum Master ... i
Interview mit Geschäftsführer ... ii
Interview mit Entwickler ... iii
Interview mit Leitung Qualitätssicherung ... iii
Interview mit Mitarbeiter Qualitätssicherung ... iv
Interview mit Analytiker ... iv
Interview mit Product Owner ... v

Abbildungsverzeichnis

Abbildung 1: Einfaches Wasserfallmodell [BrBe11] 6
Abbildung 2: Erweitertes Wasserfallmodell [BrBe11] 6
Abbildung 3: Iteration vs. Inkrement [BeRa03] 7
Abbildung 4: Überblick Rational Unified Process [IBM10] 8
Abbildung 5: Scrum Überblick [Pich08] 11
Abbildung 6: Sprint Burndown Chart [Pich08] 17
Abbildung 7: End-of-Sprint Chart [Pich08] 18
Abbildung 8: Lead Time im Kanban-Board [KnSk10] 20
Abbildung 9: Indirekte Kontrollfaktoren [KnSk10] 22
Abbildung 10: Prozessregeln [KnSk10] 24
Abbildung 11: Sprint Velocity [KnSk10] 26
Abbildung 12: Inventories und Dimensionen [PeWo10] 29
Abbildung 13: Vorteile Inkrementelle Agile Entwicklung [PeWo09] 31
Abbildung 14: Nachteile Inkrementelle Agile Entwicklung [PeWo09] 31
Abbildung 15: Lead Time der Tickets in „Projekt A“ 34
Abbildung 16: Verteilung des Work in Progress in „Projekt A“ 37
Abbildung 17: Einzelne Darstellung der Status in „Projekt A“ 38
Abbildung 18: Agile Feedback Loop [KnSk10] 70
Abbildung 19: Gleichzeitige Aktivitäten innerhalb einer Spalte im Kanban-Board [Ande10] 80
Abbildung 20: Neuer Workflow in der ITU als UML Zustandsdiagramm 112
Abbildung 21: Kanban Workflow 118

Tabellenverzeichnis

Tabelle 1: Prinzipien des Agile Manifesto in Scrum 12
Tabelle 2: Gegenüberstellung Lean Software Development und Toyota Produktionssystem 19
Tabelle 3: Unterschiede zwischen Scrum und Kanban 28
Tabelle 4: Projektverlauf „Projekt A“ 33
Tabelle 5: Lead Time je Status (Scrum) 35
Tabelle 6: Lead Time je Status (Kanban) 36
Tabelle 7: Vor- und Nachteile von Scrum (praktische Anwendung der ITU) 59
Tabelle 8: Vor- und Nachteile von Kanban (praktische Anwendung in der ITU) 60
Tabelle 9: Vor- und Nachteile von Scrum und Kanban konsolidiert 63
Tabelle 10: Verbesserungsvorschläge aus Interviews 64
Tabelle 11: Lösungsvorschläge für konsolidierte Probleme 66
Tabelle 12: Metriken für Kanban (vgl. [KnSk10]) 74
Tabelle 13: Aufteilung des Kanban-Boards 86
Tabelle 14: Gegenüberstellung Probleme und Konzeptlösung 90
Tabelle 15: Zusammenfassung des initialen Feedbacks der ITU 94
Tabelle 16: Umzusetzende Verbesserungsvorschläge des Konzepts 99
Tabelle 17: Zusammenfassung des Praxis-Feedbacks der ITU 104

1 Einleitung

1.1 Problemstellung

Gegenstand dieser Diplomarbeit ist die Verknüpfung von „Lean Development“[1] mit bekannten agilen Methoden in der Softwareentwicklung. Kanban und Scrum sind die betrachteten Vertreter von „Lean“ respektive „Agile Software Development“. Das Kernziel ist die Untersuchung sowohl der Gemeinsamkeiten als auch vorhandener Unterschiede. Neben einer theoretischen Analyse der beiden Prozesse steht jedoch vor allem die Erarbeitung eines Konzepts für den praktischen Einsatz von Kanban-Prinzipien in Verbindung mit Scrum im Vordergrund.

Die Erarbeitung eines generell praxisrelevanten Konzepts für agile Softwareentwicklung ist jedoch nicht Bestandteil dieser Arbeit, da dieses Thema bereits erschöpfend in der Literatur behandelt worden ist (vgl. [Cohn09], [HeBl10] oder [Pich08]). Vielmehr handelt es sich um den Entwurf einer konkreten möglichen Umsetzung beziehungsweise eine Integration von Kanban in vorhandene Scrum-Strukturen. Neben der Erarbeitung eines konkreten Konzepts wird dieses auch auf seine Relevanz und Umsetzbarkeit in einem österreichischen IT-Unternehmen[2] untersucht werden und bei Bedarf entsprechend angepasst werden.

Die Fragestellung stützt sich auf das auf Toyota zurückgehende „Lean Development“ (vgl. Neuauflage [TaOh09]). Die Grundidee war damals den „Work in Progress“ zu minimieren, das heißt die Anzahl paralleler Arbeiten bewusster zu kontrollieren und zu begrenzen, um schnellere Durchlaufzeiten zu erreichen, und im Zuge dessen Probleme und Engpässe sichtbar zu machen. Dieser Ansatz soll nun auch auf die Softwareentwicklung übertragen werden. Nicht zuletzt scheint der immer weiter verbreitete Einsatz von agilen Entwicklungsmethoden nun auch den Weg für „Lean Development“ geebnet zu haben.

1.2 Motivation und Zielsetzung

Die persönliche Motivation basiert auf der Idee, Kanban-Elemente erfolgreich in Scrum einzubinden und damit einen effektiveren Prozess zu schaffen, indem die Stärken beider Seiten vereint werden. Ein weiterer Aspekt liegt in der praktischen Erprobung des theoretisch erarbeiteten Konzepts und damit die Möglichkeit zu schaffen, dieses aufgrund der gewonnenen Erfahrung robuster zu gestalten.

---

[1] „lean“ auf Deutsch auch häufig mit „schlank“ übersetzt

[2] Das IT-Unternehmen wird auf gegenseitige Vereinbarung hin in der Diplomarbeit nicht genannt und alle gewonnenen Daten oder Erfahrungen gehen zu 100% anonymisiert in die Arbeit ein.

Ziel dieser Arbeit ist es daher, Lean Development in agile Softwareentwicklung so zu integrieren, dass die Stärken beider Prozesse genutzt werden können. Konkret lässt sich folgende Forschungsfrage formulieren, die es zu untersuchen und zu beantworten gilt:

Wie lassen sich die Vorteile des Lean Development in der agilen Software Entwicklung einsetzen und damit agile Prozesse weiter verbessern?

Dabei beschränkt sich die Bearbeitung der Fragestellung nicht nur auf die Erstellung eines Konzepts, um Lean Development einzubeziehen, sondern auch auf dessen praktische Anwendung in besagtem österreichischen IT-Unternehmen, mit dem im Rahmen dieser Diplomarbeit eine Zusammenarbeit stattfindet.

Die Ingenieur-wissenschaftliche Leistung im Zuge der Diplomarbeit liegt sowohl in der Erstellung eines Konzepts für die Integration von Kanban-Elementen in Scrum als auch in der praktischen Erprobung derselben in einem österreichischen Unternehmen sowie der Analyse der gewonnenen Praxiserfahrungen.

Das erwartete Ergebnis ist ein praxistaugliches Konzept zur Verbindung von Kanban und Scrum. Wie diese Kombination aussehen wird, ist zum jetzigen Zeitpunkt nur in Umrissen erkennbar; angestrebt wird jedoch die Anpassung oder Erweiterung eines Scrum-Prozesses um Kanban-Elemente.

1.3 Methodisches Vorgehen

Die Durchführung der Diplomarbeit wird in vier Phasen stattfinden. Die jeweiligen Phasen sind von den Ergebnissen der Vorgängerphasen abhängig, eine parallele oder überschneidende Bearbeitung ist daher nicht möglich, sondern es muss die Abfolge eingehalten werden.

- **Phase I: Theoretische Analyse von Scrum und Kanban**
  Basierend auf vorhandener Literatur sollen Gemeinsamkeiten und Unterschiede sowie jeweilige Vor- und Nachteile der beiden Prozesse ermittelt werden, um die Erstellung eines Konzepts für eine mögliche Kombination in Phase II zu ermöglichen.

- **Phase II: Erarbeitung des Konzepts zur Kombination von Scrum und Kanban**
  Dieses Konzept soll auf zweierlei Quellen zurückgreifen können: Einerseits die Ergebnisse der Recherche aus Phase I und daraus resultierende eigene Überlegungen, andererseits die Befragung von Mitarbeitern des österreichischen IT-Unternehmens zur Gewinnung praktischer Erkenntnisse.

- **Phase III: Überprüfung der Praxistauglichkeit des Konzepts in einem österreichischen IT-Unternehmen**
  Das in Phase II erarbeitete Konzept wird in besagtem österreichischen Unternehmen präsentiert und Anwendung finden. Für diesen Praxisteil ist ein Zeitraum von einem Monat vorgesehen. In dieser Zeit werden Erfahrungen mit der Anwendung des Konzepts und damit mit der Einbindung von Kanban ins Tagesgeschäft gesammelt.
  Weiters werden auch in dieser Phase Geschäftsführer und Mitarbeiter befragt werden, um einen hinreichenden Einblick in die praktische Erprobung des Konzepts zu erhalten.

- **Phase IV: Analyse der Erfahrungen und Ableitung der Beantwortung der Fragestellung**
  Die in Phase III gewonnenen Erkenntnisse werden in Hinblick auf Verbesserungsmöglichkeiten des in Phase II erstellten Konzepts analysiert. Das daraus entstehende überarbeitete Konzept wird als endgültige Beantwortung der Fragestellung dienen.

Ein Anspruch an diese Diplomarbeit ist es daher, aktuelle Erkenntnisse aus vorhandenen Quellen (Phase I) mit Ergebnissen der Befragung von Scrum-erfahrenen Mitarbeitern zu vereinen (Phase II), in der Praxis zu erproben (Phase III) und dadurch einen modifizierten Scrum Prozess zu erhalten, der sinnvoll und praxisrelevant um Kanban-Elemente erweitert wurde (Phase IV).

1.4 Aufbau der Arbeit

Kapitel 2 gibt einen Überblick über den theoretischen Hintergrund dieser Arbeit. Konkret werden verschiedene Arten von Vorgehensmodellen erläutert, um die Eingliederung von Scrum und Kanban zu verdeutlichen. Anschließend wird neben der bereits erfolgten allgemeinen Vorstellung eine genaue Analyse auf Gemeinsamkeiten und Unterschiede hin von Scrum und Kanban durchgeführt, da diese beiden Modelle den Hauptbestandteil dieser Arbeit darstellen. Als letzten Punkt des Kapitels sollen anhand des „State of the Art“ auch aktuelle Forschungserkenntnisse in den theoretischen Hintergrund miteinfließen. Die hier durchgeführte Analyse gilt als theoretische Grundlage für die Erstellung des Konzepts in Kapitel 4.

In Kapitel 3 wird die praktische Grundlage für das in Kapitel 4 zu erstellende Konzept erfasst. Diese setzt sich zusammen aus Interviews mit der Geschäftsführung und den Mitarbeitern sowie der Analyse vorhandener interner Projektdaten des österreichischen IT-Unternehmens (ITU).

Kapitel 4 analysiert die Ergebnisse aus dem theoretischen (Kapitel 2) und praktischem Hintergrund (Kapitel 3). Aufgrund dieser Erkenntnisse wird dann ein Konzept erstellt, das eine konkrete Verbindung von Scrum und Kanban erhält.

Kapitel 5 setzt sich mit der praktischen Erprobung des Konzepts in dem österreichischen IT-Unternehmen (ITU) auseinander.

Die Ergebnisse werden schließlich in Kapitel 6 diskutiert. Es soll an dieser Stelle auch das Konzept aufgrund des Feedbacks aus Kapitel 5 verbessert werden. Das verbesserte Konzept stellt schließlich eine endgültige Beantwortung der vorliegenden Fragestellung im Rahmen dieser Diplomarbeit dar.

Abschließend fasst Kapitel 7 die wesentlichen Erkenntnisse zusammen und gibt einen Ausblick in die Zukunft.

1.5 Praxisphase

Für die Durchführung der Praxisphase findet eine Zusammenarbeit mit einem österreichischen IT-Unternehmen statt (vgl. Phasen II & III in Kapitel 1.3). Es wird in der restlichen Arbeit kurz als „ITU“ referenziert werden, da die Zusammenarbeit anonym erfolgt.

Bei diesem Unternehmen handelt es sich um einen Anbieter von Individualsoftware. Ein bereits kurz vor der Fertigstellung stehendes Projekt mit einer Laufzeit von zwei Jahren dient als Grundlage der praktischen Untersuchung, sodass Erfahrungen in Form von Interviews, Protokollen und weiteren Erhebungen eingeholt werden können. An dem Projekt arbeiten ungefähr 15 Leute.

Die Erfahrungen aus der ITU spiegeln sich in Kapitel 3 „Praktische Erkenntnisse aus der ITU“ sowie auch in Kapitel 5 „Praktische Erprobung des Konzepts“ wider.

2 Theoretischer Hintergrund

In diesem Kapitel werden die theoretischen Grundlagen erläutert. In den Kapiteln 2.1 bis 2.4 werden die verschiedenen Arten von Vorgehensmodellen inklusive eines Beispielmodells jeder der vier Arten beschrieben, um darzustellen, wo sich Scrum respektive Kanban in die Landschaft der Vorgehensmodelle eingliedern.

In Kapitel 2.5 werden schließlich Gemeinsamkeiten und Unterschiede zwischen Scrum und Kanban analysiert. Dies ist eine wichtige Voraussetzung, um ein praxisnahes Konzept für die Kombination von Scrum und Kanban entwerfen zu können und damit dem Anspruch dieser Diplomarbeit einen Schritt näher zu kommen.

2.1 Linear: Wasserfallmodell

Als "linear" werden jene Vorgehensmodelle bezeichnet, die nicht iterativ sind, das heißt die einzelnen Phasen der Softwareentwicklung werden sequentiell abgearbeitet, wobei es jedoch zu kleinen Überlappungen kommen kann. Die Erläuterung zum Wasserfallmodell basiert großteils auf [BrBe11].

2.1.1 Ursprung

Das Wasserfallmodell wurde von [Royc70] definiert und wird trotz bekannter Schwächen (vgl. 2.1.2 Prozessüberblick) immer noch häufig angewandt. Es wurde damals konzipiert, um der schlichten „Code and Fix“-Methode entgegenzuwirken (vgl. [Benn95]) und eine geordnete Struktur in die Software Entwicklung einzuführen.

Es ist die Basis für viele effektivere Prozesse. [Royc99] betrachtet es gar als Benchmark für konventionelle Software Prozesse. Zu den Erweiterungen des Wasserfallmodells zählen unter anderem das Spiralmodell (vgl. [Boeh88]) und das V-Modell (vgl. [Bröh93]). Im Spiralmodell beispielsweise werden die Phasen mehrfach spiralförmig durchlaufen, wodurch es nicht mehr zu den linearen, sondern zu den iterativen Vorgehensmodellen zu zählen ist.

2.1.2 Prozessüberblick

Das Wasserfallmodell durchläuft sequentiell alle Phasen von „Requirements“ bis hin zu „Maintenance“. Der Output einer spezifischen Phase ist gleichzeitig Input für die nachfolgende Phase. Es kann daher eine neue Phase erst gestartet werden kann, wenn die vorangehende beendet ist. In der Praxis jedoch werden geringfügige Überlappungen der Phasen akzeptiert. Bestimmte Kriterien, die das Ende und den Anfang einer Phase bestimmen, dienen der Sicherstellung, dass eine Phase ordnungsgemäß beendet worden ist, bevor die nächste begonnen wird.

Die folgende Abbildung zeigt die einzelnen Phasen des Wasserfallmodells:

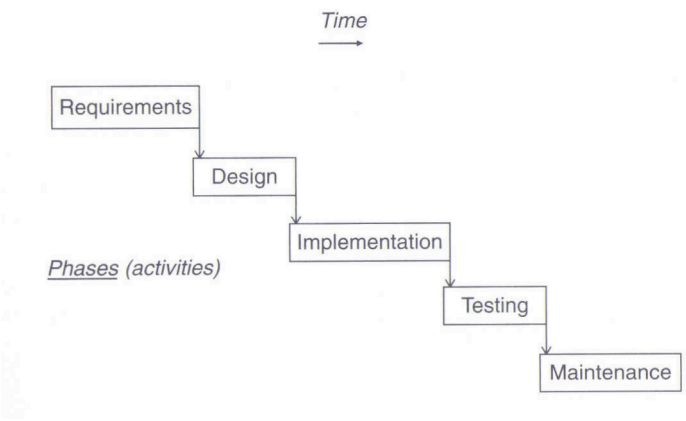


Abbildung 1: Einfaches Wasserfallmodell [BrBe11]

In der Praxis jedoch ist nach jeder durchschrittenen Phase ein Feedback an die vorangehende unumgänglich, weswegen das Wasserfallmodell eher in dieser Form Anwendung findet:

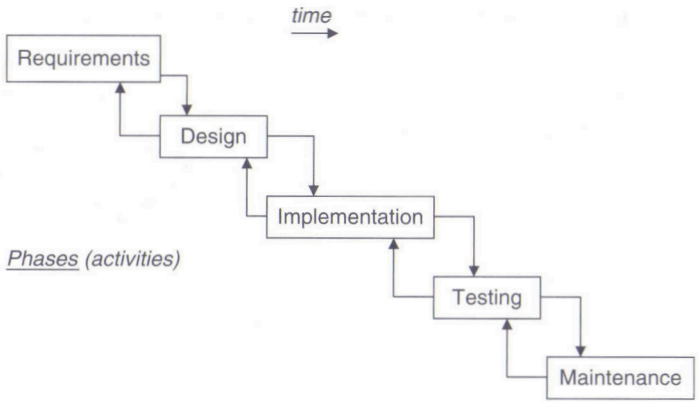


Abbildung 2: Erweitertes Wasserfallmodell [BrBe11]

Als einfaches Beispiel sei „Testing“ und „Maintenance“ genannt. Wenn der Kunde in der Wartungsphase ein Problem findet, muss gegebenenfalls die Testabdeckung verbessert werden. Das Feedback soll jedoch auf einander folgende Phasen beschränkt sein.

Das Wasserfallmodell eignet sich aufgrund der guten Planbarkeit durch klar abgegrenzte Phasen typischerweise gut für stabile und genau beschriebene Anforderungen und Abläufe [ScBi10]. Dies ist jedoch zugleich auch das Hauptproblem: Da bereits zu einem frühen Zeitpunkt Verpflichtungen eingegangen werden müssen, ist es schwer, auf Änderungen des Kunden zu reagieren [Somm07]. [Lehm80] kritisiert, dass es grundsätz-

lich nicht möglich ist, so systematisch vorzugehen, wie es das Wasserfallmodell gemäß seiner Definition verlangt.

2.1.3 Rollen, Meetings und Artefakte

Zu Beginn einer neuen Phase finden Verifikations- und Validierungsreviews statt, um die ordnungsgemäße Fertigstellung der vorangehenden Phase zu überprüfen [BrMc00]. Sobald eine Phase fertig gestellt ist, werden Artefakte erstellt, wie beispielsweise ein Spezifikationsdokument der Anforderungen am Ende der ersten Phase. Für das Wasserfallmodell sind keine konkreten Rollen definiert.

2.2 Iterativ: Rational Unified Process (RUP)

Da der lineare Ansatz, wie ihn das Wasserfallmodell beschreibt, sehr problembehaftet und unflexibel ist, wird in vielen modernen Softwareprojekten ein iterativer Ansatz verfolgt. "Iterativ" bedeutet, dass innerhalb der verschiedenen Phasen eines Software Projekts mehrere Iterationen durchlaufen werden, die alle ein Release zur Folge haben - ein sogenanntes Produktinkrement [BeRa03]. Abbildung 3 veranschaulicht diesen Vorgang:

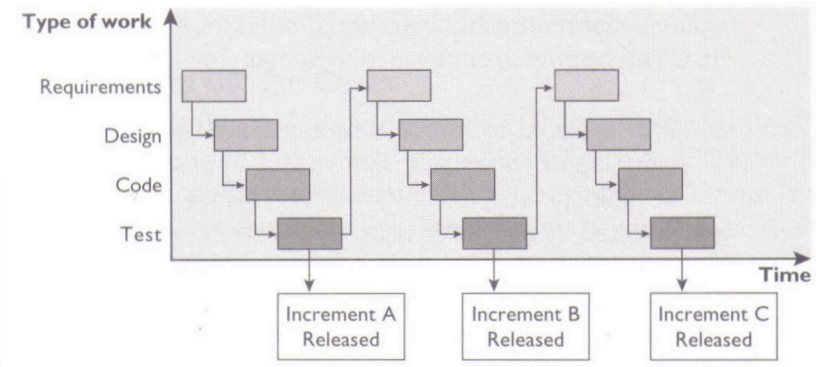


Abbildung 3: Iteration vs. Inkrement [BeRa03]

2.2.1 Ursprung

Dieser Prozess ist über mehrere Jahre hinweg aus diversen objektorientierten Ansätzen entstanden und wurde von der Firma Rational Software entwickelt, der drei der bedeutendsten objektorientierten Methodentheoretiker angehören: Ivar Jacobson, Grady Booch und Jim Rumbaugh [Vers00]. Zu den Standardwerken des Rational Unified Process gehört unter anderem [Kruc00].

2.2.2 Prozessüberblick

Der Prozessüberblick stützt sich großteils auf [KuDi07].

Der Rational Unified Process ist ein vergleichsweise komplexes und umfassendes Vorgehensmodell, das als Methodik die Unified Modeling Language (UML) unterstützt.

Es ist ein Phasenmodell, das in seinen einzelnen Phasen verschiedene Workflows positioniert, die iterativ durchlaufen werden. Es sind folgende vier Phasen definiert:

- Inception
- Elaboration
- Construction
- Transition

Zu diesen vier Phasen orthogonal stehen insgesamt neun Workflows. Diese unterteilen sich in sechs „Core Workflows“ und drei „Core Supporting Workflows“ [Vers00].

„Core Workflows“:

- Business Modeling
- Requirements
- Analyse & Design
- Implementation
- Test
- Deployment

„Supporting Workflows“:

- Configuration & Change Management
- Project Management
- Environment

Abbildung 4 zeigt die Zusammensetzung des RUP Prozesses (von [IBM10]):

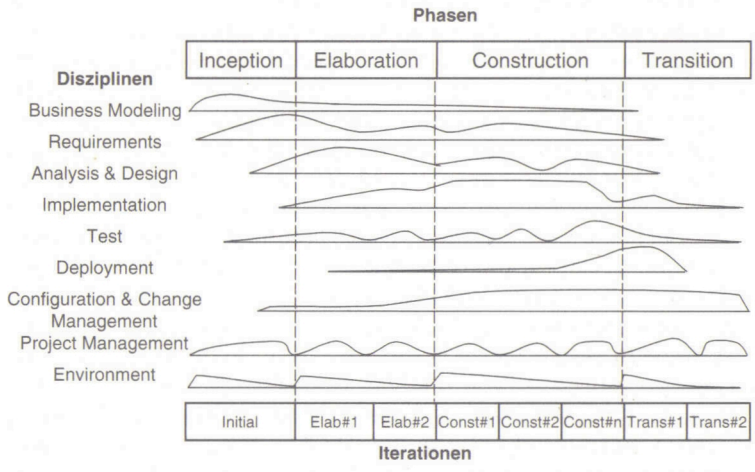


Abbildung 4: Überblick Rational Unified Process [IBM10]

Diese Übersicht über den RUP wird auch Hügel-Diagramm genannt. In Form von Erhöhungen („Hügel“) zeigt sich, ob einzelne Workflows in den verschiedenen Phasen

beziehungsweise Iterationen wichtig sind. So ist beispielsweise der Workflow „Business Modeling“ in der Inception-Phase sehr wichtig, während er in der Transition-Phase keine Rolle mehr spielt. Die Bezeichnungen „Initial“ bis „Trans #2“ am Fuße der Abbildung stellen die einzelnen Iterationen dar. Die Anzahl der Iterationen kann von Projekt zu Projekt variieren.

Die vier Phasen beschreiben das zeitliche Voranschreiten des Projekts. Innerhalb dieser Phasen können 1 bis n Iterationen vollzogen werden. Am Ende jeder Iteration steht ein Meilenstein, für den zu erfüllende Ziele definiert werden.

Es folgt eine Beschreibung der vier Phasen nach [EsMe03].

Inception
Die erste Phase dient der Vorbereitung und konzentriert sich auf die Ziele des Projekts. Es wird eine Projektvision zusammen mit der Risikoabschätzung erarbeitet. Weiters sollen auch bereits die Use Cases des Systems in groben Zügen bekannt sein und eine Abschätzung von Kosten und Nutzen des Projekts erfolgen. Als Abschluss der Phase steht der Meilenstein „Life Cycle Objective“, für dessen Erreichung die Lebenszyklus-Ziele des Projekts klar definiert sein müssen.

Elaboration
In dieser Phase erfolgt die Ausarbeitung des Entwicklungsplans. Ziel ist es die Anforderungen zu stabilisieren, eine Architektur zu definieren und die höchsten Risiken aus dem Weg zu räumen. Weiters soll ein Use Case-Modell der Anwendungsfälle in dieser Phase weitgehend komplettiert werden.

Construction
In der Konstruktionsphase steht das Entwickeln und Testen der Systemkomponenten im Mittelpunkt. Das System wird basierend auf der vorher erstellten Architektur entwickelt. Diese Phase dauert im Normalfall am längsten innerhalb eines Projekts. In ihr findet der Großteil der Implementierung statt, wie Abbildung 4 entnommen werden kann.

Transition
Die letzte Phase ist die Übergangsphase: Das Produkt geht zum Kunden über. Es erfolgt ein Test des Produkts vor Ort sowie Fehlerbehebung und Schulung von Anwendern. Am Ende steht das Product Release mit dem das Projekt abgeschlossen ist und das Produkt abgenommen wird.

Durch diese Zusammensetzung integriert der Rational Unified Process die folgenden sechs „Best Practices“ (nach [Vers02]):

- Iterative Softwareentwicklung
- Anforderungsmanagement
- Verwendung komponentenbasierter Architekturen
- Visuelle Softwaremodellierung
- Verifizierte Softwarequalität
- Kontrolliertes Change Management

2.2.3 Rollen, Meetings und Artefakte

RUP ist vielmehr ein Prozess-Framework als ein reiner Prozess. Das heißt, dass ein Projekt eine Instanz dieses Prozess-Frameworks darstellt. RUP beschreibt eine Reihe von Rollen, Meetings und Artefakte. Jedoch kann es nicht Anspruch an ein Projekt sein alle beschriebenen Konstrukte des Frameworks umzusetzen; viel mehr sollen für das Projekt sinnvolle Teile von RUP ausgesucht werden [BeRa03]. Durch den Framework-Charakter ist RUP daher bis zu einem gewissen Grad unabhängig von Aspekten wie Projektgröße, eingesetzte Programmiersprache, Domäne oder Vertragsmodell [HrRu09].

Es sei an dieser Stelle auf [IBM10] verwiesen, die im Internet eine detaillierte und aktualisierte Informationsquelle über RUP bereitstellen, da eine Auflistung aller Rollen, Meetings und Artefakte den Rahmen sprengen würde und an der Aufgabenstellung dieser Arbeit vorbeigeht.

2.3 Agil: Scrum

Agile Softwareentwicklung ist die Antwort darauf, dass selbst mit den ausgereiftesten Planungen nur selten qualitativ hochwertige Software entwickelt werden kann. Ein wesentlicher Grund dafür sind permanente Änderungen, die sich im Laufe eines Projekts ergeben [CaKl05]. Es wurde daher versucht, „leichtgewichtige“ Methoden zu entwickeln, zu denen auch Scrum zählt. Um die verschiedenen leichtgewichtigen Methoden (Scrum, Crystal, XP u.a.) auf einen gemeinsamen Nenner zu bringen, wurde das „Agile Manifesto“ [Beck01] beschlossen, das den agilen Gedanken beschreibt (vgl. Folgekapitel 2.3.1).

Die Beschreibung zu Scrum in diesem Kapitel stützt sich großteils auf [Pich08].

2.3.1 Ursprung

Scrum (dt. „Gedränge“) ist ein agiles Managementframework zur Softwareentwicklung und unterliegt dem „Agile Manifesto“ [Beck01]:

„Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on
the right, we value the items on the left more.“

Das vorliegende Zitat zeigt, dass Agilität bei Scrum im Vordergrund steht: Der Mensch als Individuum und die zwischenmenschliche Interaktion und Kommunikation stehen im Mittelpunkt und sollen festgefahrene Prozesse und striktes Vorgehen nach Plan ersetzen, wie es bei linearen Prozessen der Fall ist (vgl. Kapitel 2.1.1). Weiters rückt auch die extensive Dokumentation in den Hintergrund (vgl. Kapitel 2.1.2). [High04] bemängelt in diesem Zusammenhang fehlende Kommunikation und Zusammenarbeit bei nicht-agilen Prozessen: *„Documentation is not a substitute for interaction“*.

2.3.2 Prozessüberblick

An dieser Stelle sei ein Überblick über die einzelnen Bestandteile von Scrum gegeben:

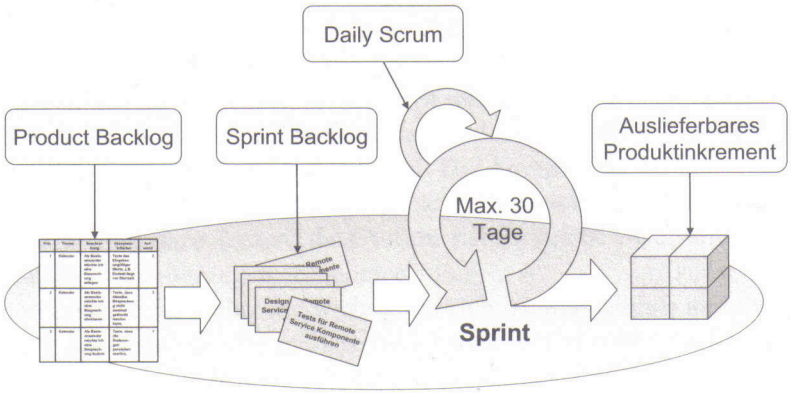


Abbildung 5: Scrum Überblick [Pich08]

Scrum ist ein empirischer Prozess und als solcher werden Arbeitsweise und Produkt regelmäßig angepasst. Abbildung 5 zeigt die wesentlichen Bestandteile des Scrum Prozesses, die nun näher erläutert werden.

In Scrum werden drei Rollen definiert:

- Der Product Owner repräsentiert die Endkundenbedürfnisse auf Entwicklungsseite.
- Der Scrum Master agiert als Coach zum Scrum Prozess und vermittelt zwischen Team und Product Owner.
- Das Team setzt Anforderungen in auslieferbare Produktinkremente um.

Initial werden vom Product Owner Anforderungen in der Form von „User Stories“ definiert, die zur Erreichung des Projektziels umgesetzt werden müssen. Diese Sammlung von User Stories wird „Product Backlog“ genannt. Eine User Story ist eine Anforderung aus Sicht des Benutzers formuliert (in einer bestimmten Rolle), wie beispielsweise „Als Basisanwender möchte ich eine Besprechung anlegen“ [Pich08].

Scrum ist ein iterativer Prozess, der auf „Sprints“ (ein Zyklus von maximal 30 Tagen) basiert. Zu Beginn eines Sprints findet ein Sprint-Planungsmeeting statt. In diesem wählt das Team umzusetzende Anforderungen aus dem Product Backlog für den „Sprint Backlog“ aus. Die ausgewählten Anforderungen sollen dann im Zuge des Sprints in ein auslieferbares Produktinkrement umgesetzt werden. Zusätzlich gibt es tägliche kurze Meetings, die „Daily Scrums“, um den aktuellen Fortschritt und etwaige Hindernisse zu besprechen. Am Ende des Sprints werden die Ergebnisse im „Sprint Review“ vom Product Owner geprüft und abgenommen. Im Anschluss daran findet die „Sprint Retrospektive“ statt in der das Team über seine Zusammenarbeit und den Prozess reflektiert. Die in diesem Rahmen erbrachten Verbesserungsvorschläge werden im nächsten Sprint-Planungsmeeting berücksichtigt, soweit dies möglich ist.

Dass Scrum ein agiler Prozess ist und die Prinzipien des Agilen Manifests [Beck01] verinnerlicht, verdeutlicht die folgende Gegenüberstellung in Tabelle 1:

Prinzipien des „Agile Manifesto“	Umsetzung in Scrum
Individuals and Interactions	Team erstellt Sprint Backlog Team organisiert sich großteils selbst Daily Scrums
Working Software	Produktinkrement pro Sprint
Customer Collaboration	Product Owner Sprint Planungsmeeting Sprint Review Meeting
Responding to Change	Daily Scrums Sprint Planungsmeeting Sprint Retrospektive

Tabelle 1: Prinzipien des Agile Manifesto in Scrum

2.3.3 Rollen

Scrum beschränkt sich auf die Definition von lediglich drei Rollen: Product Owner, Scrum Master und das Team. Der Product Owner übernimmt viele der klassischen Projektmanagement-Tätigkeiten, ohne jedoch das Team zu führen. Der Scrum Master ist für einen reibungslosen Prozessablauf zuständig. Das Team organisiert sich in Scrum selbst, verpflichtet sich jedoch innerhalb eines Sprints eine gewisse Anzahl an User Sto-

ries umzusetzen. Diese Verpflichtung bezeichnet man auch als das "Commitment" des Teams.

Product Owner
Der Product Owner vertritt die Kunden- beziehungsweise Endbenutzerinteressen. Er definiert die Anforderungen für das „Product Backlog“ und priorisiert sie. Damit ist er die bestimmende Rolle, welche die Reihenfolge der Umsetzung der Anforderungen festlegt.

Der Product Owner ist weiters für den Erfolg des Projekts verantwortlich. „Yahoo!“ beispielsweise bezeichnete die Rolle des Product Owners als „single wringable neck“ [Pich08] (dt.: „der alleinig umzudrehende Hals“). Dies verdeutlicht, dass der Product Owner für Erfolg, aber auch Misserfolg, verantwortlich ist.

Im Gegensatz zu traditionellen Ansätzen arbeitet er mit dem Team über den gesamten Projektverlauf hin und es werden gemeinsam mit dem Team laufend Anforderungen verfeinert und Arbeitsergebnisse am Ende jedes Sprints überprüft.

Der Product Owner bündelt die Interessen verschiedener Gruppen (Kunde, Marketing, Service, ...) und muss auf deren Grundlage das Product Backlog kontinuierlich bearbeiten: Neue Anforderungen hinzufügen beziehungsweise vorhandene verfeinern, priorisieren oder löschen. Er versucht den Mehrwert aus Kundensicht zu bestimmen; regelmäßige Treffen mit dem Kunden sind somit ein zwingendes Erfordernis.

[Pich08] definiert konkret folgende Aufgabenbereiche für den Product Owner:
- Anforderungsbeschreibung und –management
- Releasemanagement und Return on Investment
- Enge Zusammenarbeit mit dem Team
- Stakeholder-Management

Scrum Master
[ScBe02] beschreibt die Hauptaufgabe des Scrum Masters als *„The Scrum Master is responsible for the success of Scrum“.* Während der Product Owner also für den Erfolg des Projekts verantwortlich ist, so trägt der Scrum Master die Verantwortung für einen funktionierenden Scrum Prozess.

Laut [Pich08] kommen folgende Aufgaben dem Scrum Master zu:
- Scrum als Vorgehensmodell im Team etablieren
- Das Team unterstützen
- Die direkte Zusammenarbeit von Product Owner und Team sicherstellen
- Hindernisse beseitigen

- Entwicklungspraktiken verbessern helfen
- Führen durch Dienen

Zusammengefasst agiert der Scrum Master als Rahmengestalter für einen funktionierenden Scrum Prozess. Er agiert jedoch nicht in der Rolle einer Führungskraft, sondern unterstützt das Team, so gut er kann. Der Scrum Master muss außerdem darauf achten, dass das Team die Qualität nicht herabsetzt, um die Produktivität zu steigern [Schw08]. [Pich08] beschreibt die Philosophie folgendermaßen:*„Als Scrum Master verfügen Sie zwar über Einfluss, aber über keine Autorität bezüglich der Arbeitsorganisation im Team.“*. [Green02] spricht von einem *„Servant-Leader“*, also eine Führungsfigur ohne Autorität, die dem Team dient und es unterstützt.

Team
Das Team führt alle Arbeiten zur Umsetzung der Anforderungen in auslieferbare Produktinkremente aus, das heißt die Softwareentwicklung ist vollständig teambasiert und alle arbeiten eng zusammen – vom Architekten bis zum Tester. Wenn beispielsweise ein Feature nicht ausgeliefert werden kann, weil es nicht getestet wurde, so ist das nicht die Schuld der Tester, sondern ist vom Team als Ganzes zu verantworten, da es selbstverwaltend agiert. Es ist daher interdisziplinär besetzt und autonom: es muss das Sprint-Ziel ohne wesentliche externe Abhängigkeiten erreichen können.

Allem voran ist es bevollmächtigt, eigenständig zu entscheiden, wie viele Anforderungen es innerhalb des nächsten Sprints umsetzen kann. Wenn es sich festgelegt hat, verpflichtet (engl.: „commit“) es sich die ausgewählten Anforderungen innerhalb des Sprints umzusetzen.

2.3.4 Meetings

Scrum beschreibt eine Reihe verschiedener Meetings, die den agilen Charakter des Prozesses reflektieren, da die Personen und deren Interaktion im Vordergrund stehen. Im folgenden werden alle Meetings kurz beschrieben und ihre Aufgabe innerhalb von Scrum erläutert (vgl. [DoKl05]).

Projektplanung
Hier werden Mitarbeiter, Programmiersprache, Entwicklungswerkzeuge und Konventionen festgelegt. Weiters wird eine erste Version des Product Backlogs durch den Product Owner erstellt. Dies ist auch jener Zeitpunkt, zu dem ein gemeinsames Verständnis bei allen beteiligten Parteien über ihre Rolle in diesem Projekt hergestellt wird [Glog08].

Sprint Planning Meeting
Am Beginn jedes Sprints wird ein Planungsmeeting durchgeführt. Es wird entschieden, welche Ziele beziehungsweise User Stories aus dem Product Backlog in diesem Sprint bearbeitet werden. Das Team legt einvernehmlich fest, wie diese Ziele erreicht werden sollen. Im Gegensatz zu traditioneller Planung wird nicht mit einzelnen Entwickler-Ressourcen geschätzt, sondern mit dem Team als Ganzes [Wird09]. Alle Ziele werden in Form zu erledigender Aufgaben im Sprint Backlog festgehalten.

Daily Scrum (auch Stand Up-Meeting)
In diesem täglichen Meeting beschreiben alle Teammitglieder kurz ihren aktuellen Status. Die Dauer ist auf maximal 15 Minuten beschränkt. Konkret stellt der Scrum Master folgende drei Fragen, die von jedem Teammitglied beantwortet werden müssen (nach [DoKl05]):

- Was hast du gestern getan?
- Was machst du heute?
- Wirst du durch irgendetwas behindert?

Werden konkrete Hindernisse identifiziert, hält der Scrum Master diese schriftlich fest (vgl. „Impediment Chart“ aus Kapitel 2.3.4). Seine unmittelbare Aufgabe besteht nun in der raschen Beseitigung dieser Hindernisse [Pich08].

Sprint Review Meeting
Am Ende des Sprints wird ein Sprint Review Meeting abgehalten. Beim Review-Meeting präsentiert das Team das im Laufe des Sprints erstellte Produktinkrement. Teilnehmer dieses Meetings sind im besten Fall Management, Kunden, Benutzer und der Product Owner [ScBe02].

Der Scrum Master koordiniert das Sprint Review Meeting. Es handelt sich dabei um ein sehr informelles Meeting, weshalb keine Präsentationen vorbereitet werden dürfen. Einzig und allein das Produktinkrement steht im Vordergrund.

Im Zuge des Meetings sollen sich die Teilnehmer das nötige Wissen für das nächste Sprint-Planungsmeeting aneignen und Stärken und Schwächen des aktuellen Produktinkrements kennenlernen.

Hier wird weiters überprüft, welche Aufgaben aus dem Sprint Backlog umgesetzt worden sind (optimalerweise alle). Diese werden im Zuge des Meetings bei zufriedenstellendem Ergebnis vom Product Owner abgenommen.

Sprint-Retrospektive
Die Retrospektive soll unmittelbar nach dem Review stattfinden. Inhalt ist es über die Zusammenarbeit im Team und die Anwendung des Prozesses innerhalb des vergangenen Sprints zu reflektieren und Verbesserungspotentiale zu besprechen [Schw04]. Ziel ist eine Steigerung der Produktivität, der Leistungsfähigkeit des Teams und der Softwarequalität [DeLa06].

Mit der Retrospektive gilt der Sprint offiziell als abgeschlossen.

2.3.5 Artefakte

Die Artefakte in Scrum unterstützen den Prozess, beschränken sich jedoch auf ein absolutes Minimum. Product Backlog und Sprint Backlog stellen das Rückgrat von Scrum dar, wie der Name bereits verrät. Auch der Burndown-Chart darf in keiner Scrum-Anwendung fehlen, um den aktuellen Fortschritt innerhalb des Sprints sichtbar zu machen. Zuletzt wird noch der „Impediment Chart“ und der „End-of-Sprint Chart“ beschrieben, die jedoch eher als weitere nützliche Hilfsmittel zu sehen sind (von [Pich08]) als als fixer Bestandteil jedes Scrum-Prozesses.

Product Backlog
Das Product Backlog wird in der Planungsphase vom Product Owner angelegt. Es ist eine Liste von Anforderungen, die im fertigen Produkt enthalten sein sollen. Im Gegensatz zu dem klassischen Pflichtenheft ist der Product Backlog jedoch dynamisch, das heißt bewusst unvollständig gehalten [ScBe02]. Dies spiegelt den agilen Gedanken wider. Er wird ständig durch den Product Owner verfeinert. Neue Anforderungen können in jeder Phase des Projekts hinzukommen oder entfernt werden.

Weiters sind die im Product Backlog enthaltenen Anforderungen nach Dringlichkeit beziehungsweise Wichtigkeit priorisiert.

Sprint Backlog
In jedem Sprint-Planungsmeeting übernimmt das Team Anforderungen aus dem Product-Backlog in den Sprint-Backlog. Das Team unterteilt die Anforderungen in Tasks, die während dem Sprint abgearbeitet werden. Der Sprint Backlog kann während dem Sprint nur so insofern verändert werden, als dass neue Subtasks hinzukommen [ScBe02].

Es ist das Artefakt, das die Anforderungen enthält, zu denen sich das Team für die Umsetzung im aktuellen Sprintverpflichtet hat.

Burndown Chart

Der Burndown Chart ist ein grafisches Hilfsmittel zur Überwachung der Aktivitäten innerhalb eines spezifischen Sprints [DoKl05]. Er zeigt den Fortschritt der Arbeit und wie sich die Aufwände im Sprint Backlog von Tag zu Tag entwickeln [ScBe02]. Abbildung 6 illustriert einen beispielhaften Burndown Chart.

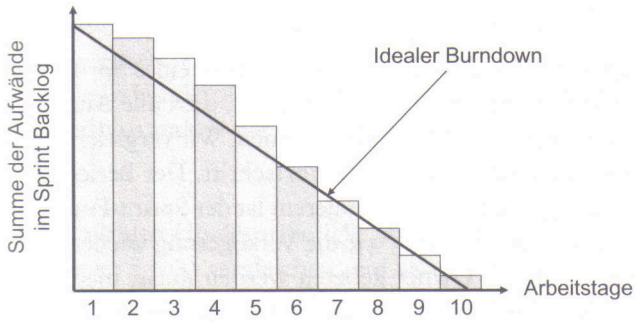


Abbildung 6: Sprint Burndown Chart [Pich08]

Der ideale Burndown beschreibt den idealen Sprint-Fortschritt [Pich08]. Anhand der Summe der Aufwände (y-Achse) kann abgelesen werden, ob der reale Fortschritt über oder unter dem idealen Burndown liegt.

Impediment Chart

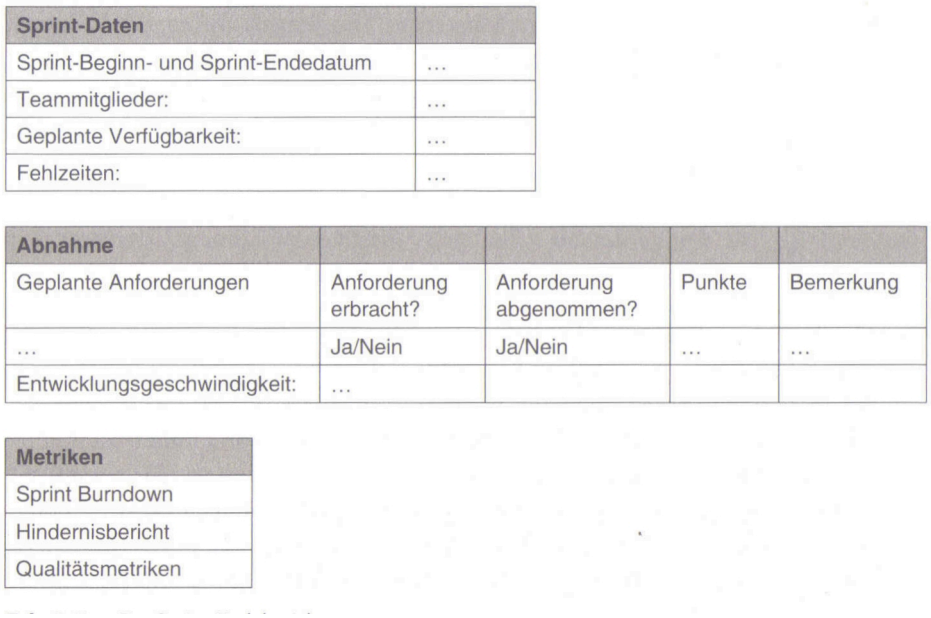
Der Impediment Chart (dt.: Hindernis-Diagramm) enthält eine kurze Beschreibung der aufgetretenen Hindernisse zusammen mit dem Datum, an dem das Hindernis identifiziert wurde, und dem Datum, an dem es behoben wurde [Pich08].

Er sollte immer am Ende des Daily Scrums durch den Scrum Master aktualisiert werden. Eine ausstehende Beseitigung von Hindernissen sind darauf basierend zu gegebener Zeit zu eskalieren.

End-of-Sprint Chart

Der Sprint-Endbericht enthält die wesentlichen Rahmendaten des Sprints und dokumentiert, wie viele der geplanten Anforderungen erfolgreich umgesetzt und vom Product Owner abgenommen worden sind [Cohn05]. Außenstehenden wird damit die Möglichkeit gegeben, kurz und bündig eine Aussage über die Qualität des Sprints zu erhalten [Pich08].

So kann ein End-of-Sprint Chart beispielsweise aussehen (Abbildung 7):



Sprint-Daten	
Sprint-Beginn- und Sprint-Endedatum	...
Teammitglieder:	...
Geplante Verfügbarkeit:	...
Fehlzeiten:	...

Abnahme				
Geplante Anforderungen	Anforderung erbracht?	Anforderung abgenommen?	Punkte	Bemerkung
...	Ja/Nein	Ja/Nein	...	...
Entwicklungsgeschwindigkeit:	...			

Metriken
Sprint Burndown
Hindernisbericht
Qualitätsmetriken

Abbildung 7: End-of-Sprint Chart [Pich08]

2.4 Lean: Kanban

Unter „lean“ (dt.: schlank) versteht man einen Entwicklungssansatz, der sich einzig darauf beschränkt einen Wert für den Endbenutzer zu erzeugen. Alle Ausgaben für andere Ziele werden als Verschwendung („waste“) angesehen und sind daher zu eliminieren.

2.4.1 Ursprung

Kanban als Vorgehensmodell geht auf die Automobilindustrie zurück. Toyota hat es nach dem zweiten Weltkrieg in den 50er Jahren entwickelt und 1988 veröffentlicht (vgl. Neuauflage [Ohno09]). Unter dem Überbegriff der „schlanken Produktion“[3] stellte es einen Kontrast zur damals üblichen Massenproduktion in der Automobilindustrie dar, die in den USA und Europa in dieser Zeit zu stagnieren begann [WoJo92].

Toyota beobachtete, dass die Stückkosten bei kleinen Losgrößen niedriger waren als bei sehr großen Mengen. Dies hatte zwei große Vorteile gegenüber der Massenproduktion [Wojo92]:

- Die Bestandsführung für riesige Lagerbestände, wie sie in der Massenproduktion üblich war, wurde beseitigt (Beginn der „Just in Time“ Produktion).
- Fehler konnten bei nur kleinem Teilevorlauf unmittelbar entdeckt und ausgebessert werden.

[3] In der Fachliteratur auch „Lean Management“

Diese Vorgehensweise verlangte jedoch auch äußerst qualifizierte und hoch motivierte Arbeitskräfte, da Fehler vorausschauend erkannt werden mussten und auf Eigeninitiative hin behoben, bevor eine Störung eintrat. Ohne diese Maßnahme bestand die Gefahr, dass die ganze Fabrik zum Stillstand kommt [WoJo92].

Diese und weitere Überlegungen, die damals die Automobilindustrie revolutionierten, können fast 1:1 in der Softwareentwicklung angewandt werden. [PoPo04] hat sich eingehend mit der Umsetzung der Lean-Prinzipien aus der Automobilindustrie befasst und diese auf die Softwareentwicklung umgelegt.

Tabelle 2 zeigt eine Gegenüberstellung der von [PoPo04] identifizierten Grundsätze für „Lean Software Development“ und ihren Ursprung im Toyota Produktionssystem [Ohno09].

Lean Software Development [PoPo04]	Toyota Produktionssystem [Ohno09]
Eliminate Waste	Umfassende Analyse der Verschwendung
Amplify Learning	Die Bedeutung des Verstehens
Decide as Late as Possible	Just-in-Time
Deliver as Fast as Possible	Just-in-Time
Empower the Team	Die Macht der Fähigkeit des einzelnen und der Teamarbeit Die Fähigkeit zur Stabübergabe
Build Integrity In	Die Einrichtung eines Fertigungsflusses
See the Whole	Kanban-Board

Tabelle 2: Gegenüberstellung Lean Software Development und Toyota Produktionssystem

2.4.2 Prozessüberblick

Kanban an sich ist ein sehr schlanker Prozess, wie es der Überbegriff „Lean Development“ (dt.: schlanke Entwicklung) bereits vermuten lässt. Es sind nur drei Punkte zu beachten (nach [KnSk10]):

Workflow visualisieren

Es wird ein Kanban-Board benutzt, um den Workflow für alle sichtbar zu machen. Die Arbeit wird in einzelne Arbeitspakete aufgeteilt und jedes Arbeitspaket auf ein Kärtchen geschrieben (oder alternativ elektronisch als Ticket in einem Ticket-Management-

System). Das Kanban-Board besteht aus mehreren Spalten, die diese Arbeitspakete durchlaufen müssen, wie zum Beispiel:

„To Do“ → „Development“ → „Test“ → „Release“ → „Done“

Work in Progress (WIP) limitieren
Für die einzelnen Spalten des Kanban-Boards werden Limits eingeführt; das heißt die maximale Anzahl an Arbeitspaketen ist begrenzt. Flaschenhälse können aufgedeckt werden und der Durchfluss der Arbeitspakete („Lead Time“) wird verbessert, da aktiv an Lösungen gearbeitet werden muss, um die Limits nicht zu überschreiten. Weiters kann versucht werden, rechtzeitig entgegen zu wirken, wenn sich die Arbeitspakete in einer Spalte dem Limit nähern.

Lead Time messen
Als „Lead Time“ (oft auch „Cycle Time“ genannt) wird die Durchschnittszeit verstanden, die für den Abschluss eines Arbeitspakets benötigt wird. Gemeint ist damit die durchschnittliche Zeit, die ein Arbeitspaket braucht, um alle Spalten zu durchlaufen.

Der Prozess wird dahingehend optimiert, die „Lead Time“ so klein und vorhersehbar wie möglich zu halten.

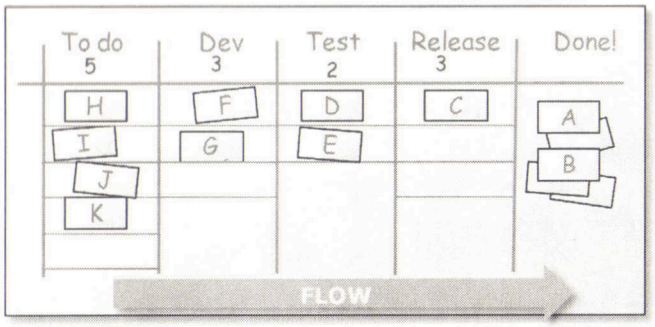


Abbildung 8: Lead Time im Kanban-Board [KnSk10]

2.4.3 Rollen, Meetings und Artefakte

In Kanban werden keinerlei Rollen oder Meetings explizit vorgeschrieben. Dies heißt jedoch nicht, dass keine hinzugefügt werden können oder sollen. Beispielsweise findet das „Daily Meeting“ aus Scrum sehr häufig auch in Kanban Anwendung.

Als einziges Artefakt beschreibt Kanban das Kanban-Board, das bereits im Überblick (vgl. Kapitel 2.4.2) erläutert wurde. Genau wie Rollen oder Meetings, können auch wei-

tere Artefakte hinzugefügt werden. Es sollte jedoch beachtet werden, dass jede Erweiterung gut begründet werden sollte, da Kanban ein schlanker (engl.: „lean“) Prozess ist.

2.5 Analyse Scrum und Kanban

An dieser Stelle folgt die Analyse der Gemeinsamkeiten und Unterschiede von Scrum und Kanban, die als Grundlage für die Erstellung des Konzepts dient (vgl. Kapitel 4). Diese Analyse basiert auf [KnSk10].

2.5.1 Gemeinsamkeiten

Gemeinsamkeit 1: Beide sind leichtgewichtige und adaptive Prozess-Tools
Sowohl Scrum als auch Kanban sind Prozess-Werkzeuge. Sie sind ein Mittel, um einen Task oder einen bestimmten Zweck zu erfüllen.

Aufgrund der Verschiedenheit von Scrum und Kanban ist es schwer zu sagen, welches der beiden Prozessmodelle der Vorzug zu geben ist. Jedes für sich verfolgt einen eigenen Ansatz und kann in verschiedenen Projektumfeldern das richtige Tool sein. Es gibt kein allgemein „richtiges“ Tool, aber es mag ein gutes Tool für ein gewisses Projektumfeld geben.

Was beide jedoch als Prozess-Tool oder Vorgehensmodell gemeinsam haben, ist, dass sie beide leichtgewichtig und adaptiv sind. Sie beschreiben beide weitaus weniger Rollen und Artefakte als traditionelle Ansätze (vgl. Kapitel 2.2 Iterativ: Rational Unified Process).

Gemeinsamkeit 2: Beide limitieren den Work in Progress
Beide Modelle limitieren den Work in Progress, jedoch auf verschiedene Weise. Kanban limitiert ihn direkt über das WiP-Limit, Scrum hingegen indirekt über die Sprint-Iterationen (vgl. Unterschied 4 in Kapitel 2.5.2 „Unterschiede“).

Gemeinsamkeit 3: Beide sind empirisch
Sowohl Scrum als auch Kanban verfügen nicht über direkte Kontrollmöglichkeiten, um den Prozess zu beeinflussen. Aber es gibt indirekte Kontrollmöglichkeiten, die den Prozess und die damit verbundenen Kapazitäten, „Lead Time“, Qualität und Schätzgenauigkeiten beeinflussen können.

[KnSk10] weist auf folgende indirekte Kontrollmöglichkeiten hin:

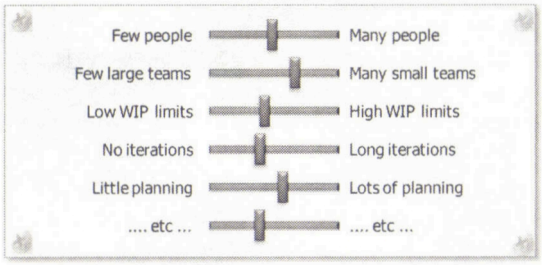


Abbildung 9: Indirekte Kontrollfaktoren [KnSk10]

Scrum und Kanban sind beide empirisch, das heißt man muss mit dem Prozess und den zur Verfügung stehenden indirekten Kontrollmöglichkeiten experimentieren und auf die vorhandene Arbeitsumgebung zuschneiden.

Gemeinsamkeit 4: Beide benutzen ein Board zur Veranschaulichung des Workflows

In beiden Prozessen werden Items auf einem Board verfolgt, lediglich die Handhabung des Boards ist eine andere (vgl. Unterschied 4 & 6 in Kapitel 2.5.2 „Unterschiede“). In Scrum beschränkt man sich zumeist auf die Unterteilung: „Backlog“ → „In Progress“ → „Done“. In Kanban kann es mehrere Aufteilungen geben. Ausschlaggebend jedoch ist die Benutzung von Work in Progress Limits je Spalte, die es in Scrum nicht gibt.

Gemeinsamkeit 5: Beide basieren auf inkrementeller Entwicklung

Für beide Prozesse wird die umzusetzende Arbeit in kleinere Arbeitspakete heruntergebrochen, die zu regelmäßigen (Scrum & Kanban) oder „On-Demand“ (nur Kanban) Produktinkrementen zusammengesetzt werden. Der Ansatz, Arbeitspakete herunterzubrechen ist der gleiche, jedoch passiert dies aus verschiedenen Gründen (vgl. Unterschied 9 in Kapitel 2.5.2 „Unterschiede“).

Gemeinsamkeit 6: Beide erlauben das gleichzeitige Arbeiten an mehreren Produkten

Das Backlog in Scrum heißt zwar „Product Backlog“, jedoch bedeutet das nicht, dass alle darin enthaltenen Features Teile desselben Produkts sein müssen. Die Features des Backlogs sind priorisiert und können einfach abgearbeitet werden, ungeachtet dessen, zu welchem Produkt das Feature gehört. Genauso ist es möglich, in Kanban einen Backlog zu haben, der Features verschiedener Produkte enthält.

Sowohl in Scrum als auch in Kanban können somit Features verschiedener Produkte gleichzeitig bearbeitet werden. In Scrum bietet sich zusätzlich die Option, die Sprints

produktweise aufzuteilen, beispielsweise ein Sprint für „Produkt A“, der nächste für „Produkt B“. Wie bereits erwähnt ist jedoch auch die gleichzeitige Abarbeitung innerhalb eines Sprints möglich.

Gemeinsamkeit 7: Beide sind „lean“
Scrum und Kanban sind beide Pull-Systeme, was dem „Just in Time“-Prinzip des Lean Development entspricht. „Pull“ bedeutet, dass das Team bestimmt, wann das nächste Arbeitspaket begonnen wird und es ihnen nicht von außen vorgeschrieben wird („Push“).

Beide basieren auf kontinuierlicher und empirischer Prozess-Optimierung, was dem Lean-Gedanken entspricht. Als Beispiel in Scrum sei hierfür die Retrospektive genannt [Pich08].

Gemeinsamkeit 8: Beide sind agil
Das „Agile Manifesto“ und deren Umsetzung von Scrum wurde bereits in Tabelle 1 erläutert. Auch Kanban setzt auf seine Art die Prinzipien des „Agile Manifesto“ (vgl. [Beck01]) um. Auch in Kanban stehen Individuen und Interaktion im Vordergrund, da das Kanban-Board Flaschenhälse unbarmherzig aufzeigt und damit Interaktion forciert, sowohl um Probleme zu lösen als auch um sie das nächste Mal gar nicht erst entstehen zu lassen. Der Punkt „Working Software“ soll durch einen funktionierenden Workflow gewährleistet werden, der in Kanban den zentralen Bestandteil darstellt. Die letzten beiden Punkte „Customer Collaboration“ und „Responding to Change“ sind Kanbans große Stärken. Es gibt keine Iteration und daher kann wesentlich schneller auf Änderungen in der Priorisierung reagiert werden, sei diese nun von Kundenseite oder intern propagiert.

Letztendlich hängt eine prinzipgetreue Erfüllung des Agile Manifesto sehr stark von der praktischen Umsetzung von Kanban ab, da es sehr viele Freiheiten lässt.

2.5.2 Unterschiede

Unterschied 1: Kanban ist adaptiver
Kanban beschreibt nur drei Regeln, während Scrum neun Regeln beschreibt. Unter „Regeln“ werden alle Vorschriften zusammengefasst, die ein Prozess vorgibt. Rollen, Meetings oder Artefakte sind hierzu konkrete Beispiele. In Kanban gibt es lediglich drei Regeln: „Work in Progress limitieren“, „Lead Time messen und optimieren“ und „Abfall vermeiden“, das heißt, alles zu vermeiden, was nicht direkt eine Wertschöpfung mit sich bringt (wie zum Beispiel exzessive Vorausplanung). Die folgende Abbildung zeigt einen Überblick an Vorschriften in Softwareprozessen von „RUP“ (über 120 Regeln) bis „Do Whatever“ (0 Regeln).

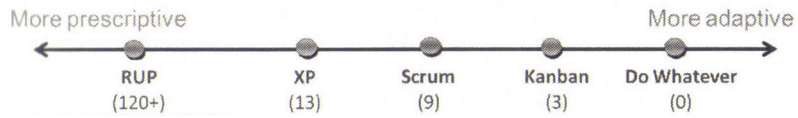


Abbildung 10: Prozessregeln [KnSk10]

Unterschied 2: Scrum schreibt Rollen vor, Kanban nicht
In Scrum gibt es „Product Owner", „Scrum Master" und das „Team". Kanban beschreibt keinerlei Rollen. Das bedeutet jedoch nicht, dass Kanban ohne Rollen auskommt. Es können Rollen von beispielsweise Scrum hinzugefügt werden. Irgendeine Form von Rollen muss es immer geben, jedoch schreibt Kanban nicht vor, welche das sind.

Unterschied 3: Scrum beschreibt zeitlich begrenzte Iterationen
Diese zeitlich begrenzten Iterationen werden „Sprints" genannt. Die Länge eines Sprints kann variieren. Generell ist es Sinn und Zweck über eine längere Zeitperiode, die Sprintlänge nicht zu variieren, sobald sie festgelegt ist.

In Kanban sind keine zeitlich begrenzten Iterationen vorgeschrieben. Es kann frei entschieden werden, wann Planung, Prozessverbesserung oder ein Release gemacht wird. Diese Aktivitäten sind in Kanban sowohl auf regelmäßiger Basis als auch „on demand" möglich.

Unterschied 4: Scrum limitiert WIP per Iteration, Kanban per Status des Workflows
Beide limitieren den Work in Progress (WIP). Kanban tut dies auf eine direkte Weise, indem es Limits angibt für jeden möglichen Status.

Scrum hingegen begrenzt den Work in Progress nur indirekt durch Iterationen. Durch die festgelegte Menge von Arbeitspaketen pro Iteration ergibt sich damit auch ein implizites Limit für den Work in Progress, da maximal alle für die Iteration definierte Arbeitspakete pro Status möglich sind. Es gibt also auch in Scrum ein (implizites) Limit für den Work in Progress.

Unterschied 5: Scrum lässt innerhalb einer Iteration keine Änderungen zu
In Kanban sind neue Arbeitspakete zulässig, sofern das Work in Progress Limit dies erlaubt.

In Scrum hingegen können neue Arbeitspakete erst in den nächsten Sprint einbezogen werden, da das Team sich bereits auf die Erfüllung von gewissen Arbeitspaketen inner-

halb der Iteration festgelegt hat und diese nicht ausgetauscht oder verändert werden dürfen.

Unterschied 6: Reset des Scrum-Boards nach jeder Iteration
Wenn ein Sprint in Scrum zu Ende geht, werden die Items aus dem Board typischerweise entfernt. Das Kanban-Board hingegen ist persistent, da es auf dem Pull-System und einem kontinuierlichen Workflow basiert, und nicht auf Iterationen. Bei physischen Kanban Boards müssen die fertigen Items jedoch ebenfalls nach einer gewissen Zeit aus praktischen Gründen entfernt werden, um Platz für weitere Items zu schaffen.

Unterschied 7: Scrum schreibt funktionsübergreifende Teams vor
Ein Scrum Team ist funktionsübergreifend, das heißt es beinhaltet alle Fähigkeiten, die es braucht, um die Arbeitspakete eines Sprints umzusetzen. In Kanban hingegen sind funktionsübergreifende Teams optional.

Unterschied 8: Das Scrum-Board gehört zu einem Team, das Kanban-Board zu einem Workflow
Ein Scrum-Board gehört genau einem Scrum-Team. Es wird benutzt, damit es das Commitment für diesen Sprint visualisieren kann. Ein Kanban-Board muss nicht gezwungenermaßen zu einem bestimmten Team gehören, es ist jedoch immer einem Workflow untergeordnet.

Unterschied 9: Scrum will kleine Arbeitspakete passend für Iteration, Kanban für bessere Lead Time
In Scrum werden Arbeitspakete so heruntergebrochen, dass sie in einen Sprint passen. Ist ein Arbeitspaket zu groß, versucht der Product Owner Wege und Mittel zu finden, um es in kleinere Teile aufteilen zu können – bis es schließlich in einen Sprint passt.

Es gibt in Kanban keine explizite Regel Arbeitspakete möglichst klein aufzuteilen. Demnach ist es möglich, dass auf demselben Kanban Board Arbeitspakete von stark variierender Bearbeitungszeit zu finden sind. In Kanban möchte das Team jedoch die Lead Time der Arbeitspakete möglichst gering halten und damit einen guten Workflow gewährleisten. Um das zu erreichen, muss die Arbeit ebenfalls in möglichst kleine und ungefähr gleich große Arbeitspakete heruntergebrochen werden.

Unterschied 10: Scrum schätzt den Durchfluss an Arbeitspaketen je Iteration
In Scrum schätzen Teams den Arbeitsaufwand von jedem Arbeitspaket, zu dessen Umsetzung sie sich verpflichten. Zählt man diese Arbeitspakete am Ende der Iteration zusammen, erhält man den Durchfluss an Arbeitspaketen (engl.: velocity). Die Kenntnis der Velocity ermöglicht es, realistische Planungen für weitere Sprints zu machen. Die folgende Abbildung verdeutlicht das Prinzip des Durchflusses („Velocity“):

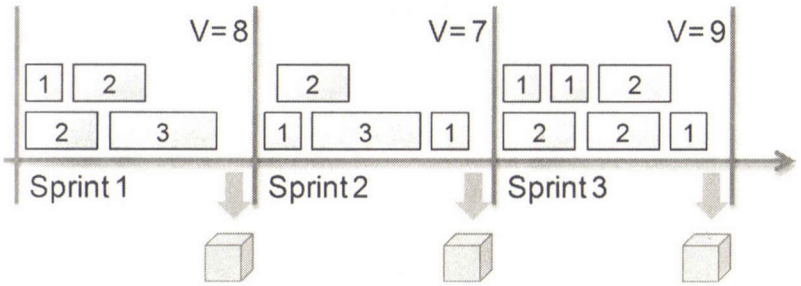


Abbildung 11: Sprint Velocity [KnSk10]

In Kanban ist keine Schätzung vorgeschrieben und auch nicht leicht einzuführen, da es nicht mit Iterationen arbeitet.

Unterschied 11: Scrum schreibt einen priorisierten Backlog vor
In Scrum muss der Product Backlog immer priorisiert und entsprechend sortiert werden. In Kanban gibt es keine Regel, die die Priorisierung des Backlogs vorschreibt oder verbietet. Daher ist Priorisierung in Kanban optional.

Unterschied 12: Scrum schreibt Meetings vor
In Scrum gibt es Daily Standup-Meetings, Sprint-Planungsmeeting, Sprint-Retrospektive und das Review-Meeting. Kanban schreibt keine bestimmten Meetings vor. Alle Meetings sind optional.

Unterschied 13: Scrum schreibt einen Burndown Chart vor
In Scrum werden Sprint Burndown Charts benutzt, um den Fortschritt des Sprints zu verfolgen beziehungsweise den Status zu überprüfen. In Kanban sind keinerlei Charts vorgeschrieben, jedoch optional einsetzbar.

Unterschied 14: Team in Scrum hat Commitment je Iteration
Ein Scrum-Team verpflichtet sich zur Umsetzung von Features innerhalb eines Sprints. Da es in Kanban kein Commitment für das Team gibt, gibt es auch keine Iterationen. Kanban konzentriert sich vielmehr auf einen kontinuierlichen und funktionierenden Workflow. Zeit spielt auch in Kanban eine große Rolle: statt zu kontrollieren wie viele Features in einem Sprint umgesetzt werden können, wird die Cycle Time eines Features gemessen, um Vorhersagen zu machen und Vergleiche treffen zu können.

2.5.3 Zusammenfassung

An dieser Stelle sind die Gemeinsamkeiten und Unterschiede der beiden vorangehenden Kapitel als Überblick zusammengefasst.

Gemeinsamkeiten

- Beide sind leichtgewichtige und adaptive Prozess-Tools
- Beide limitieren den Work in Progress
- Beide sind empirisch
- Beide benutzen ein Board zur Veranschaulichung des Workflows
- Beide basieren auf inkrementeller Entwicklung
- Beide erlauben das gleichzeitige Arbeiten an mehreren Produkten
- Beide sind „lean“
- Beide sind agil

Unterschiede

Scrum	Kanban
Ist adaptiver als traditionelle Prozesse (9 Regeln)	Ist noch adaptiver (3 Regeln)
Beschreibt drei Rollen (PO/SM/Team)	Beschreibt keinerlei Rollen
Arbeitet in zeitlich begrenzten Iterationen	Arbeitet mit kontinuierlichem Workflow
Limitiert WIP indirekt per Iteration	Limitiert WIP direkt per Workflow-Status
Keine Änderungen innerhalb einer Iteration	Änderung ständig möglich, sofern WIP Limits es erlauben
Board-Reset nach jeder Iteration	Kein Board-Reset (kontinuierlicher Workflow)
Schreibt funktionsübergreifende Teams vor	Keine Vorschrift bezüglich Team-Zusammensetzung
Scrum-Board gehört zu einem Team	Kanban-Board gehört zu einem Workflow
Scrum teilt in kleine Arbeitspakete passend für Iteration	Kanban teilt in kleine Arbeitspakete, um Lead Time zu verbessern
Scrum kann den Durchfluss an Arbeitspaketen (Velocity) pro Sprint schätzen	In Kanban ist schätzen schwieriger, da es keine Iterationen gibt
Scrum schreibt einen priorisierten Backlog vor	In Kanban gibt es beliebige oder auch gar keine Priorisierungsschemata
Scrum schreibt Meetings vor	Kanban schreibt keinerlei Meetings vor
Scrum schreibt die Benutzung eines Burndown-Charts vor	Kanban schreibt keinerlei Charts vor
Team hat Commitment je Iteration	Team hat keinerlei Commitment

Tabelle 3: Unterschiede zwischen Scrum und Kanban

2.6 State of the Art

In diesem Unterkapitel werden einzelne aktuelle Paper zum Thema „Lean Management in der IT“ näher untersucht, um weitere Möglichkeiten für das zu entwickelnde Konzept (Kapitel 4) zu erforschen. Es folgt eine Zusammenfassung dieser Papers respektive deren Kernaussage. Ihre Relevanz für das Konzept wird schließlich in Kapitel 4.3 untersucht.

2.6.1 „SPI-LEAM“-Methode

Ein sehr aktuelles Paper zu Lean Management in der agilen Softwareentwicklung ist [PeWo10]. Es besagt, dass ein großes Umdenken notwendig ist, um „Lean Management“ in vorhandene Prozessstrukturen zu integrieren. Als einer der größten Fehler hierbei wird genannt, dass die Arbeitsweise zu schnell und abrupt verändert wird. Die Umstellung hingegen sollte kontinuierlich und inkrementell erfolgen.
Um eine solche Umstellung zu vollziehen, wird in [PeWo10] die Methode „SPI-LEAM“ (Software Process Improvement through the Lean Measurement) vorgestellt. Die „SPI-LEAM“ baut auf dem Quality Improvement Paradigm (QIP, vgl. [Basi85] und [BaGr94]) auf und erweitert dieses um die Prinzipien der Lean Softwareentwicklung. Der wichtigste Bestandteil der Methode die Unterteilung so genannter „Inventories“ der Softwareentwicklung in verschiedene Dimensionen:

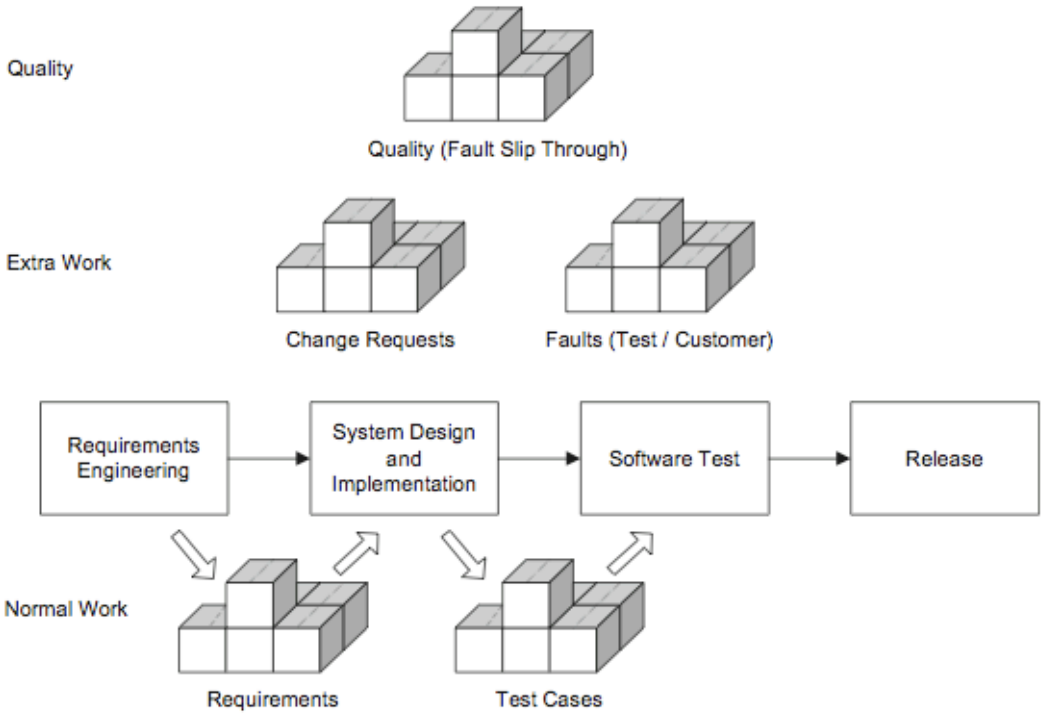


Abbildung 12: Inventories und Dimensionen [PeWo10]

[PeWo10] beschreibt folgende Inventories:

- Requirements
- Test Cases
- Change Requests
- Faults

Jedes dieser Inventories kann wiederum Subinventories umfassen. Es ist wichtig, den Stand der Inventories regelmäßig zu überprüfen. Zudem wird versucht, optimale Inven-

tory-Stände (zum Beispiel Anzahl Requirements) zu finden, um einen effizienten Workflow zu gewährleisten, was wiederum dem Lean-Gedanken entspricht.

2.6.2 To Pull or Not to Pull

[JaSu09] beschäftigt sich mit „Push"- und „Pull"-Methoden in der Softwareentwicklung und untersucht die zwei häufigsten Vertreter der „Push"-Methoden, sowie mögliche „Pull"-Alternativen.

Es werden folgende „Push"-Methoden untersucht:

- „Requirements-First" Entwicklung
- „Bottom-Up" Entwicklung

Requirements-First

Fast alle Software Entwicklungsprozesse beginnen mit dem Sammeln von Anforderungen (engl.: „Requirements"). Manche Ansätze versuchen alle Anforderungen vorab zu erfassen, in anderen werden die Anforderungen iterativ ermittelt. In beiden Fällen werden die Anforderungen zum Entwicklungsprozess „gepusht".

Als alternatives Pull-Konzept schlägt [JaSu09] „Goal-driven Software Development" (vgl. [ScPi06]) vor. Hierbei werden Ziele gemeinsam definiert, noch bevor Anforderungen erstellt werden. Diese Ziele werden weiters benutzt, um priorisierte Anforderungen von den Stakeholdern zu verlangen („pull"). Indem auch Techniker über die Ziele Bescheid wissen, können Probleme frühzeitig besprochen und Ideen eingebracht werden, wie die Ziele erreicht werden können (Kosten, Alternativen, technische Anforderungen, etc.).

Das Ergebnis ist ein Entwicklungsprozess, in dem Anforderungen nicht der Startpunkt der Implementierung sind, sondern Geschäftsziele auf die technischen Möglichkeiten abgestimmt werden, um eine optimale Lösung zu finden.

Bottom-Up

Bei der „Bottom Up"-Entwicklung werden verschiedene Komponenten entwickelt, die später zusammengefügt werden. Es ist weniger komplex, diese einzelnen Komponenten zu designen, implementieren und zu testen. Dieser Ansatz ist jedoch sehr schwierig handzuhaben, wenn sich Anforderungen ändern, da oft verschiedene Annahmen bei der Entwicklung der einzelnen Komponenten getroffen werden.

Als „Pull"-Alternative nennt [JaSu09] die „Top Down"-Variante „Test-driven Development" (vgl. [Beck03]): die Erstellung von Testfällen vor der Entwicklungsphase. Treibende Kraft für die Testfälle sind dabei die Geschäftsziele. Die Testfälle fordern

(„pull“) dann den kleinsten Codeumfang, der nötig ist, um den jeweiligen Testfall und damit das Geschäftsziel zu erfüllen.

2.6.3 Issues and Advantages of Using Agile and Incremental Practices

[PeWo09] untersucht vorhandene State of the Art-Paper und konsolidiert Vor- und Nachteile der agilen Softwareentwicklung:

ID	Advantages	Model
A01	Better knowledge transfer due to better communication and frequent feedback from each iteration.	XP/XP/XP/SC&XP
A02	Customers are perceived by programmers as very valuable allowing developers to have discussions and get early feedback.	XP/SCRUM/XP/XP
A03	Pair programming facilitates learning if partners are exchanged regularly.	XP
A04	Process control, transparency, and quality are increased through continuous integration and small manageable tasks.	XP
A05	XP is very much technical-driven empowering engineers and thus increases their motivation.	XP
A07	Small teams and frequent face-to-face meetings (like planning game) improves cooperation and helps getting better insights in the development process.	XP(modification)
A08	The social job environment is perceived as peaceful, trustful, responsible, and preserving quality of working life.	XP
A09	Customers appreciate active participation in projects as it allows them to control the project and development process and they are kept up to date.	XP
A10	Developers perceive the job environment as comfortable and they feel like working more productive using pair programming.	XP
A11	Student programmers perceive the quality of code higher using pair programming	XP

Abbildung 13: Vorteile Inkrementelle Agile Entwicklung [PeWo09]

ID	Issues	Model
I01	Realizing continuous testing requires much effort as creating an integrated test environment is hard for different platforms and system dependencies.	XP(modification)
I02	Architectural design does not have enough focus in agile development leading to wrong design decisions.	gen./gen.
I03	Agile development does not scale well.	gen.
I04	Pair programming is perceived as exhaustive and inefficient.	XP/XP/XP
I05	Team members need to be highly qualified to succeed using agile.	XP
I06	Teams are highly coherent which means that the communication within the team works well, but inter-team communication suffers.	XP/SC&XP
I07	The empowerment of engineers makes managers afraid initially, and thus requires sufficient training of managers.	XP
I08	Implementation starts very early, thus technical issues are raised too early from a management point of view.	XP
I09	On-site customers have to commit for the whole development process which puts them under stress.	XP
I10	From the perspective of students, pair programming is not applicable if one partner is much more experienced than the other.	XP

Abbildung 14: Nachteile Inkrementelle Agile Entwicklung [PeWo09]

2.6.4 Weitere

Lean Development ist ein aktives Forschungsgebiet, wie zahlreiche weitere aktuelle Paper zeigen. Konkrete Beispiele sind [Ambl09] „Scaling Agile Software Development Through Lean Governance“ oder [Green10] „Enterprise Scrum: Scaling Scrum to the Executive Level“, das ebenfalls den Lean Gedanken anspricht. Da diese jedoch als Basis für die Erstellung des Konzepts in Kapitel 4 nicht relevant sind, wird auf sie hier nicht weiter eingegangen.

3 Praktische Erkenntnisse

Für diese Diplomarbeit wurden neun Interviews mit Mitarbeitern in einem österreichischen IT-Unternehmen geführt (vgl. „Anhang“). In Folge wird das Unternehmen der Einfachheit halber als „ITU“ bezeichnet. In der ITU wurden in der Vergangenheit sowohl Erfahrungen mit Scrum als auch Kanban gesammelt. Weitere Informationen zur ITU können Kapitel 1.5 entnommen werden.

Interviewpartner waren folgende Personen:

- „GF1“: Geschäftsführung/Kaufmännische Leitung
- „GF2“: Geschäftsführung/Technische Leitung
- „SM“: Scrum Master/Entwickler
- „EW1“, „EW2“ und „EW3“:Entwickler:
- „LQS“: Leitung Qualitätssicherung
- „QS“: Mitarbeiter Qualitätssicherung
- „Analytiker“: Analyse/Architektur

Im Zuge der Interviews wurde versucht, positive und negative Aspekte in der praktischen Anwendung von Scrum beziehungsweise Kanban zu erfragen. Diese werden in den Kapiteln 3.2 und 3.3 näher erläutert. In Kapitel 3.1 folgen allgemeine Informationen zum Projektverlauf.

3.1 Projektverlauf

Die Interviews beziehen sich vor allem auf ein großes Projekt, da hier sowohl Scrum als auch Kanban Anwendung gefunden haben (in chronologischer Reihenfolge). In den Interviews wird es als „Projekt A“ referenziert.

An dieser Stelle ist zum leichteren Verständnis der Projektverlauf chronologisch dargestellt:

Datum	Prozess
Mitte 2008	Projektstart mit klassischem Projektmanagement und anfänglichen Scrum-Strukturen
Ende 2008	Umstieg auf reinen Scrum-Prozess ohne Projektmanager, sondern nur Scrum Master und Product Owner
Ende 2009	Umstieg auf Kanban in der Projektendphase

Tabelle 4: Projektverlauf „Projekt A“

3.1.1 Lead Time

Die folgende Abbildung zeigt einen Überblick über das Projekt in Form der durchschnittlichen Durchlaufzeiten (engl.: Lead Time) von Tickets.

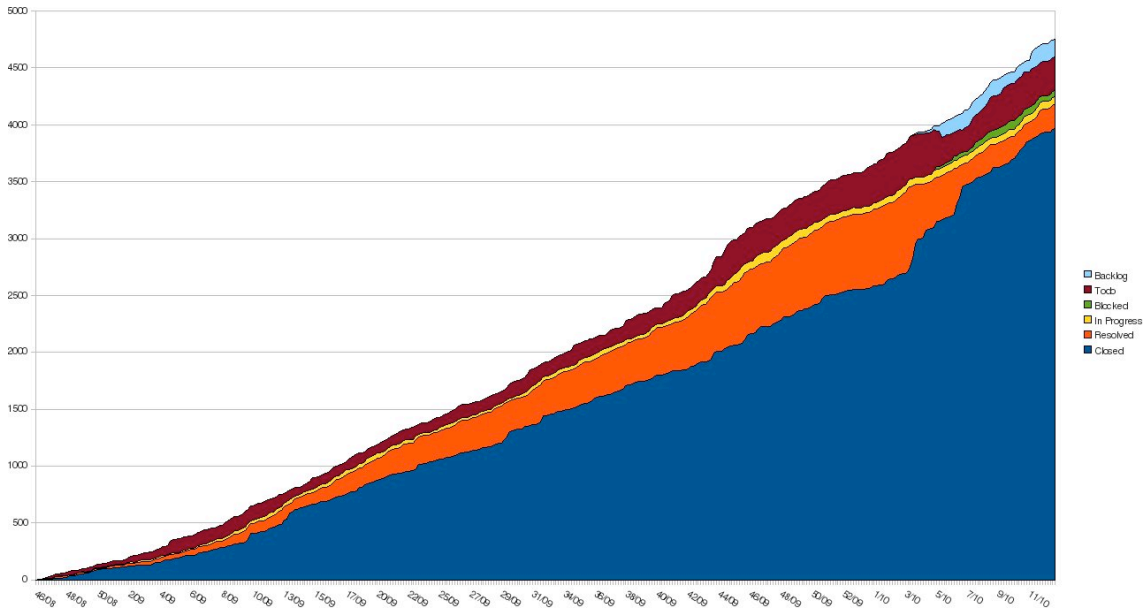


Abbildung 15: Lead Time der Tickets in „Projekt A“

Die y-Achse in Abbildung 15 zeigt die Gesamtanzahl an Tickets, die x-Achse beschreibt die Zeit in der Form „Kalenderwoche/Jahr“. Legt man eine horizontale Linie in den Graphen, das heißt parallel zur x-Achse, so kann man entlang dieser Linie mit Hilfe der farblichen Status-Unterscheidungen die durchschnittliche Durchlaufzeit eines Tickets ablesen. Je kürzer der Anteil an dieser imaginären horizontalen Linie pro Status ist, desto kürzer ist auch die Lead Time in diesem Status.

Tabelle 5 zeigt eine Auswertung der Lead Time für Scrum in 500er Schritten nach Anzahl der Ticket. Es gab zu dieser Zeit vier Status:

- „To Do“: Tickets aus Sprint Backlog
- „In Progress“: Tickets in Bearbeitung
- „Resolved“: Fertig bearbeitete Tickets
- „Closed“: Als fertig verifizierte Tickets

„Closed“ wurde in die Auswertung nicht einbezogen, da eine Lead Time für geschlossene Tickets sinngemäß nicht möglich ist.

Anzahl Tickets	To Do	In Progress	Resolved	Gesamt
500	11 Tage	4 Tage	11 Tage	26 Tage
1000	9 Tage	3 Tage	21 Tage	33 Tage
1500	15 Tage	2 Tage	27 Tage	44 Tage
2000	14 Tage	3 Tage	41 Tage	58 Tage
2500	14 Tage	3 Tage	43 Tage	60 Tage
3000	20 Tage	5 Tage	49 Tage	74 Tage
Durchschnitt	**14 Tage**	**3 Tage**	**32 Tage**	**49 Tage**

Tabelle 5: Lead Time je Status (Scrum)

Ab einer Ticket-Anzahl von ca. 3850 wurde Kanban eingeführt, daher startet die Auswertung für Kanban erst an dieser Stelle. Da Kanban kürzer in Verwendung war, wurden kleinere Schritte gewählt. Über „Resolved“ kann ab Ticket-Anzahl 4150 keine Aussage mehr gemacht werden, da weitere Daten in Abbildung 15 nicht verfügbar sind.

Für das Kanban-Board wurden folgende Spalten benutzt:

- „Backlog“: Tickets im Backlog
- „To Do“: Zur Bearbeitung freigegebene Tickets
- „Blocked“: Extern durch Auftraggeber blockierte Tickets
- „In Progress“: Tickets in Bearbeitung
- „Resolved“: Fertig bearbeitete Tickets
- „Closed“: Als fertig verifizierte Tickets

„Closed“ wurde wiederum nicht in die Auswertung einbezogen.

Anzahl Tickets	Backlog	To Do	Blocked	In Progress	Resolved	Gesamt
3700	-	28 Tage	3 Tage	6 Tage	17 Tage	54 Tage
3850	4 Tage	25 Tage	7 Tage	7 Tage	9 Tage	52 Tage
4000	12 Tage	14 Tage	6 Tage	3 Tage	13 Tage	48 Tage
4150	7 Tage	17 Tage	3 Tage	4 Tage	-	-
4300	7 Tage	16 Tage	4 Tage	5 Tage	-	-
4450	6 Tage	18 Tage	5 Tage	5 Tage	-	-
4600	6 Tage	17 Tage	6 Tage	-	-	-
Durchschnitt	**7 Tage**	**19 Tage**	**5 Tage**	**5Tage**	**13 Tage**	**49 Tage**

Tabelle 6: Lead Time je Status (Kanban)

Zu Beginn gab es in Kanban noch keine „Backlog“-Spalte.

Beim Vergleich von Scrum und Kanban fällt auf, dass die Einführung der „Backlog“-Spalte die „To Do“-Spalte nicht entlastet hat, was auf eine ungenaue Abgrenzung schließen lässt. Auch die „Blocked“-Spalte hat keinen Einfluss auf „To Do“ und „In Progress“.

Die Lead Time der „Resolved“-Spalte hat sich durch Kanban deutlich verbessert (vgl. Kapitel 3.3.2).

Ansonsten lässt sich die Lead Time betreffend nicht mehr aussagen. Ein Grund hierfür ist sicherlich auch, dass Kanban nicht so lange in Verwendung war und ein neuer Prozess immer eine gewisse Anlaufzeit benötigt.

3.1.2 Work in Progress

In der folgenden Abbildung 16 wird die Aufteilung des Work in Progress prozentuell dargestellt. Die „Closed“-Spalte ist ausgeblendet, da sie keinen Einfluss auf diese Metrik hat.

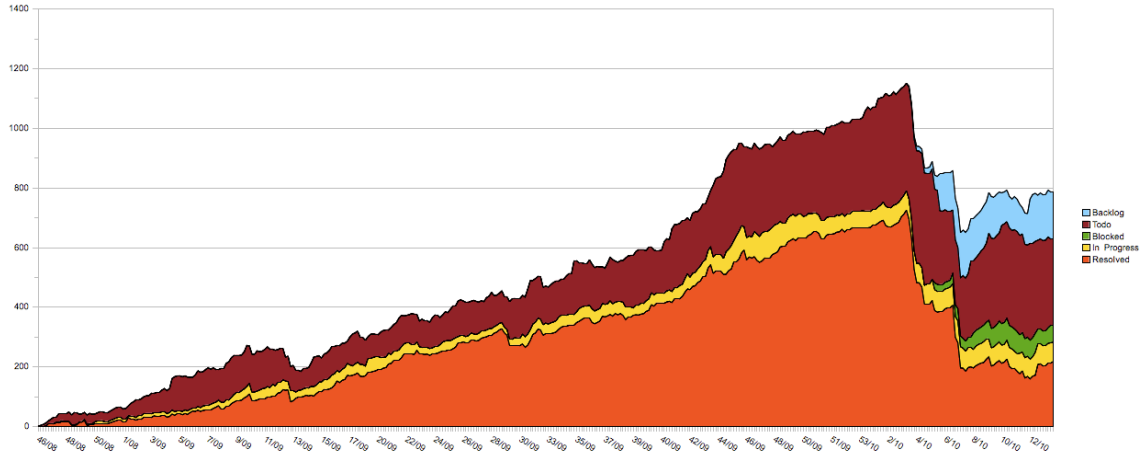


Abbildung 16: Verteilung des Work in Progress in „Projekt A“

Die y-Achse beschreibt wiederum die Anzahl an Tickets und die x-Achse wiederum die Zeit mit der Einheit „Kalenderwoche/Jahr“. In dieser Grafik sind jedoch im Vergleich zu Abbildung 15 die „Closed“-Tickets ausgeblendet. Dadurch lässt sich zwar die Lead Time nicht mehr auslesen, dafür ist der Überblick über die Verteilung des Work in Progress besser gegeben.

Hier kann gut erkannt werden, dass die „Resolved“-Spalte sehr ausgelastet war, das heißt Tickets wurden zwar erledigt („Resolved“-Spalte), jedoch nicht geschlossen („Closed“-Spalte nach Re-Test eines „Resolved“-Tickets), was einen großen Risikofaktor birgt, wenn Tickets wieder geöffnet werden müssen nach einem Re-Test. Zeitweise vereinnahmte die „Resolved“-Spalte 50% des gesamten Work in Progress. Diese Tatsache konnte durch das Kanban-Board aufgedeckt werden. Geeignete Gegenmaßnahmen bewirkten eine Verringerung des Anteils an „Resolved“-Tickets ab März 2010 (vgl. Abbildung 16 Wert der x-Achse „3/10“).

In vielen Interviews wurde der Umstand angesprochen, dass der Backlog nicht mehr gefüllt war und deshalb Scrum nicht mehr möglich war. Dieser Punkt ist hier leider nicht klar ersichtlich. Es scheint, als würde sich das frühere „To Do“ lediglich in „Backlog“ und „To Do“ aufteilen. Der allgemeine Abfall in der Höhe bei „3/10“ geht vor allem auf die Verringerung der Anzahl der „Resolved“-Tickets zurück. Demnach muss vielmehr die Qualität der vorhandenen Backlog-Tickets gesunken sein, als rein die Quantität.

Weiters zeigt sich hier auch, dass bis zu 10% des Work In Progress extern blockiert waren („Blocked“-Spalte). Diese Zahl erscheint niedrig. Jedoch ist zu bedenken, dass damit ungefähr gleich viele Tickets blockiert sind wie „In Progress“, was deutlich mehr über das beträchtliche Ausmaß der externen Blockierung gegen Ende des Projekts aussagt.

Die folgende Abbildung zeigt die einzelnen Spalten als 3D-Ansicht.

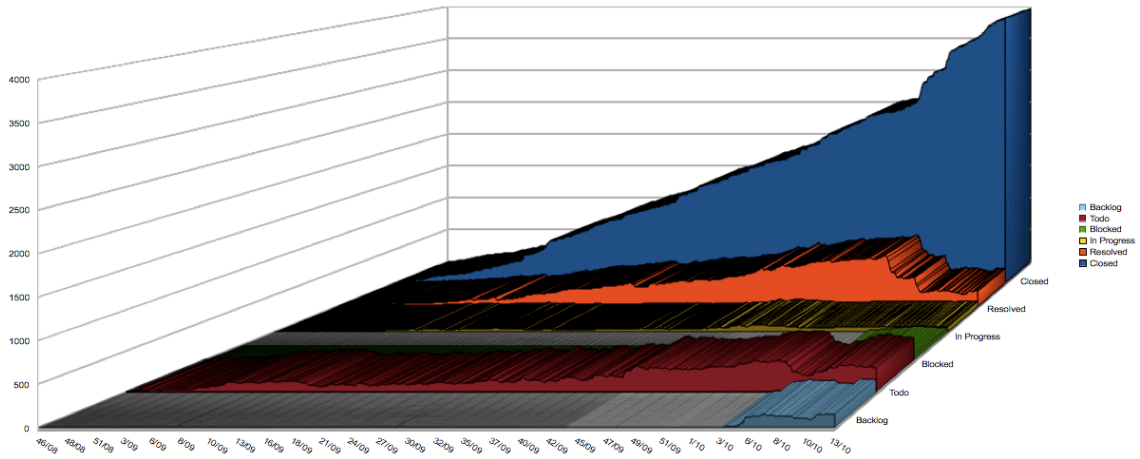


Abbildung 17: Einzelne Darstellung der Status in „Projekt A“

Die y-Achse beschreibt wiederum die Anzahl an Tickets, die x-Achse wiederum die Zeit in der Form „Kalenderwoche/Jahr“ und die z-Achse weist die verschiedenen Status aus.

Hier bestätigt sich im Vergleich zu Abbildung 16, dass die „Resolved“ Ticket-Anzahl im Zuge der Kanban Einführung deutlich reduziert werden konnte. In dieser Darstellung ist besser ersichtlich, dass in dieser Zeit auch die Ticket-Anzahl der „To Do“-Spalte gesunken ist, was aus Abbildung 16 noch nicht hervorgegangen ist, jedoch hier deutlich zum Vorschein kommt. Dies bestätigen auch die Aussagen aus den Interviews.

3.2 Scrum

Scrum wurde in der ITU eingeführt, nachdem es Probleme mit klassischen Projektmanagement-Strukturen gab (diese Probleme spiegeln sich auch in den Interviews wider). Der agile Prozess sollte die Zufriedenheit im Team sowie auch die Produktivität steigern.

3.2.1 Ablauf

Der Scrum-Prozess bei „Projekt A“ sieht wie folgt aus:

- Scrum Master ist Mitglied des Entwicklungsteams und verantwortlich für alles Administrative rund um den Scrum-Prozess. Dazu gehört:
  - Organisation der Meetings (Planung, Review, Retrospektive)
  - Führung eines Protokolls pro Sprint, das den Verlauf beschreibt
- Da es die Komplexität des Projektes verlangt, ist die Rolle des Product Owners aufgeteilt auf:
  - „GF2“ als Product Owner für die Entwicklung
  - „LQS“ als Product Owner für die Qualitätssicherung

    - „Analytiker“ übernimmt den Analyse-Teil der Product Owner-Rolle als eigenen Full Time Job.
- „GF2“ gibt in seiner Rolle als Product Owner die Prioritäten vor, das Team committet sich aufgrund dieser Prioritäten zu Features innerhalb eines Sprints und erstellt eigenständig entsprechende Tasks.
- Die Qualitätssicherung arbeitet ebenfalls parallel zur Entwicklung im gleichen Sprint Rhythmus von zwei Wochen, da sich ihre Prioritäten mit denen der Entwickler großteils decken beziehungsweise von diesen abhängen.
- Es finden täglich zur gleichen Zeit Daily Meetings statt mit einer Dauer von maximal 15 Minuten. Jeder Entwickler beziehungsweise Mitarbeiter der Qualitätssicherung vermittelt kurz und bündig seinen aktuellen Status und weist vor allem auch auf etwaige Blockierungen hin.
- Das Planungsmeeting findet am Anfang des Sprints statt. Inhalt ist die Besprechung von Priorisierungen, das Commitment zu Features für den nächsten Sprint sowie die Schätzung der Aufwände. Die Qualitätssicherung hält ein eigenes Planungsmeeting ab, nachdem die Prioritäten mit der Entwicklung abgeklärt worden sind.
- Am Ende jedes Sprints findet ein Review-Meeting statt in dem der Product Owner („GF2“) die entwickelten Features abnimmt und besprochen wird, was geschafft wurde und was nicht. Die Qualitätssicherung nimmt ebenfalls daran teil.
- Die Retrospektive wird im Normalfall direkt nach dem Review-Meeting abgehalten und bietet Entwicklern sowie der Qualitätssicherung die Möglichkeit über den Prozess zu reflektieren und Probleme beziehungsweise Verbesserungsmöglichkeiten zu diskutieren.
- Es gibt ein Task Board, das sich auf drei Spalten beschränkt: „To Do“, „In Progress“ und „Done“.

3.2.2 Vorzüge

Die folgenden Punkte wurden von den Mitarbeitern im Zuge der Interviews genannt:

Bevollmächtigung des Teams

Nach ursprünglich klassischem Projektmanagement in der ITU wurde die Bevollmächtigung und Eigenverantwortung im Zuge des Scrum-Prozesses sehr gut angenommen und hat die Zufriedenheit gesteigert. Es wurden nur noch Priorisierungen durch den Product Owner vorgegeben.

Klar definierte Feature-Größen

Es wurden Features in Scrum so granuliert, dass sie innerhalb eines zwei Wochen-Sprints umgesetzt werden können. Gemäß den Richtwerten für die Granularisierung sollten sich auf Features beziehende Tasks ein Zeitbudget von maximal acht Stunden benötigen.

Gute Abschätzbarkeit
Durch den Sprint-Rhythmus ist es unumgänglich, genügend Features im Backlog zu haben, sodass die nächsten Sprints versorgt sind. Dieser Fakt hat den positiven Effekt, dass das Projekt nicht Gefahr läuft, durch mangelnde Vorausplanung unabschätzbar zu werden:

„Ja, es hat sehr gut funktioniert. Wir hatten Tasks, die gut definiert waren. Wir hatten eine sehr gute Sicht auf die Zeitschätzung und Scrum war wirklich perfekt als Prozess für die Produktion."

(Zitat aus Interview mit Entwickler „EW2")

Einfaches Reporting möglich
Aufgrund der Sprint-Iterationen ist es einfach, vergleichbare Reports zu erhalten. Dementsprechend hat es sich in „Projekt A" auch gut bewährt, die Sprintlängen über den ganzen Projektverlauf hin nicht zu verändern, um Vergleichswerte zu bewahren.

Wenig Prozess-Overhead
Der Prozess-Overhead hat sich durch die agile Vorgehensweise in Scrum im Vergleich zum klassischen Projektmanagement deutlich verringert. Die Organisation übernahm das Team in Eigenregie. Einzige Regel war die Einhaltung des Commitments pro Sprint.

Leistungsnachweis visualisiert in Review-Meeting
Aufgrund der Iterationen auf Sprint-Basis ist es leicht, die erledigte Arbeit nachzuvollziehen, zu überprüfen und auch untereinander zu vergleichen. Weiters stellt der erfolgreiche Abschluss eines Sprints auch ein gewisses Erfolgserlebnis dar. Dieser Leistungsnachweis wird auch immer für alle im Review-Meeting deutlich gemacht, wodurch auch Management und Planung vereinfacht wird:

„Für uns aus Management-Sicht war der Prozess auch sehr in Ordnung, weil er ist sehr transparent und sehr klar."

(Zitat aus Interview mit Geschäftsführer „GF2")

Retrospektive bietet Möglichkeit über den Prozess als solchen zu reflektieren
Es ist nicht nur von großem Vorteil, dass es mit der Retrospektive ein eigenes Meeting in jedem Sprint gibt, um über Verbesserungen im Prozessablauf nachzudenken, sondern auch, dass alle anderen Meetings davon befreit werden können und damit das Produkt an sich immer im Vordergrund stehen kann:

„So konnte man im Daily Meeting sagen: ‚Wir reden nicht mehr vom Prozess, sondern machen das am Ende vom Sprint'. Das war sehr gut, dass es schon so definiert war."

(Zitat aus Interview mit Entwickler „EW2)

Optimalerweise pendelt sich der Prozess so ein, dass alle zufrieden sind:

„Irgendwann einmal hat der ‚SM' in seiner Scrum Master-Rolle in einer Retrospektive die Frage gestellt: ‚Und wie ist der Prozess, was sollten wir noch ändern, wie geht's uns eigentlich, sind wir zufrieden?' und irgendwann einmal nach eben diesen 6-8 Monaten war es so, dass einfach keine Antwort mehr gekommen ist."

(Zitat aus Interview mit Geschäftsführer „GF2")

Qualitätssicherung kann im gleichen Sprint-Rhythmus arbeiten

Diese Vorgehensweise hat sich gut bewährt, da die Qualitätssicherung von der Planung und Priorisierung der Entwicklung teilweise abhängig ist. Eine Abhängigkeit hat sich daraus ergeben, dass Tests erst durchgeführt werden können, wenn die Implementierung vollzogen ist. Der gleiche Sprint-Rhythmus hat Entwicklung und Qualitätssicherung dazu bemächtigt, auf ein gemeinsames Ziel zusammen hinzuarbeiten, was sich sehr gut bewährt hat.

Aufteilung der Product Owner-Rolle auf mehrere Personen gut bewährt

Die Rolle des Product Owners wurde auf drei Personen aufgeteilt, wie bereits in Kapitel 3.2.1 erwähnt. Diese Aufteilung auf die drei Aufgabengebiete Analyse, Entwicklung und Qualitätssicherung hat es ermöglicht, die Komplexität der Product Owner-Rolle zu bewältigen.

Geschäftsführer und Product Owner für die Entwicklung „GF2" beschreibt den Erfolg mit der Aufteilung in „Projekt A" folgendermaßen:

„Außerdem ist es auch von der Kommunikation zum Projektpartner oder Auftraggeber hin ganz gut. Es ist gut, wenn du Emotionen trennen kannst. Der fachliche Part ist hochgradig emotionslos, während der Part, der dann Prioritäten und Termine... und um Budgets diskutiert und manchmal auch streiten muss durchaus auch emotional werden kann.
Ganz wichtig ist, das glaube ich zumindest, dass diese beiden Bereiche getrennt sind, dass selbst wenn es im projektorganisatorischen und kommerziellen Umfeld Spannungen gibt, soll zumindest noch die Facharbeit ungehindert möglich sein und das ist schwer, wenn das einer [Anm.: eine Person] ist."

(Zitat aus Interview mit Geschäftsführer „GF2")

3.2.3 Probleme

Scrum benötigt stets ausreichend gefülltes Backlog
Wenn der Backlog zu wenig gefüllt ist, was in „Projekt A“ in der Wartungsphase der Fall war, kann das Team sich nicht mehr innerhalb eines Sprints committen, was Scrum ad absurdum führt.

Entwickler „EW3“ beschreibt die Situation folgendermaßen:

„Das Commitment hat insofern [Anm.: zu Ende des Projekts hin] nicht funktioniert, weil die Vorlaufzeiten nicht gepasst haben. Für ein Commitment braucht man ein gefülltes Backlog für zwei Wochen oder wie lange halt der Sprint ist und das haben wir nicht gehabt.
Immer wenn eine neue Spezifikation reingekommen ist, dann haben wir die abgearbeitet. Aber wir haben nie so viel Spec [Anm.: Spezifikation] vor uns gehabt, dass wir zwei Wochen hätten planen können. Das hat von der Vorlaufzeit her einfach nicht gepasst.“

(Zitat aus Interview mit Entwickler „EW3)

Der zweite Grund, warum das Team Schwierigkeiten hatte sich zu committen, war eine externe Blockade vorhandener Features, was eine korrekte Planung unmöglich gemacht hat:

„Der Hauptgrund war aus meiner Sicht, dass wir, also das gesamte Team, aber hauptsächlich Development, sehr oft blockiert waren, weil einfach Inputs und Rückmeldungen von Auftraggeberseite her offen waren. Die haben uns so stark blockiert, dass das eigentliche Commitment nicht erfüllt werden hat können und dafür aber andere Tasks abgearbeitet wurden. Das war der Grund warum wir aus meiner Sicht dann auf Kanban gewechselt sind.“

(Zitat aus Interview mit Leiter der Qualitätssicherung „LQS“)

Unfähigkeit auf Änderungen schnell zu reagieren
Durch das Commitment auf einen Sprint, konnte man nicht auf kurzfristige Änderungen reagieren, sondern nur anlässlich eines neuen Planungsmeetings.

Dieser Punkt zeigte sich gleich zu Beginn in der Architekturphase des Projekts, wie Entwickler „EW2“ beschreibt:

„Es war auch eine Architekturphase, es war vieles unklar und wir haben noch nicht wirklich ein Release gehabt. Scrum funktioniert wirklich besser, wenn es produktive Arbeit gibt.“

(Zitat aus Interview mit Entwickler „EW2“)

Ähnliche Probleme gab es zu Projektende, da Scrum mit kurzfristigen Change Requests und Bugfixing nicht gut zurechtkommt:

„Da habe ich schon das Gefühl, dass Scrum da zu statisch ist für das. Also dass ich dann mit irgendwelchen abstrakten Tasks arbeite und da steht dann beim Entwickler ‚Bugfixing' so ungefähr oder ‚Warten auf Change Requests'. Das ist nicht der Sinn von Scrum."

(Zitat aus Interview mit „Analytiker")

Auch Geschäftsführer „GF2" schildert seine große Unzufriedenheit in dieser Sache:

„Dadurch ist dann eigentlich die Planung immer ad absurdum geführt worden, weil immer wieder haben sich dann doch ein paar Tasks wieder ent-blockiert und dann haben wir gemerkt, dass wir de facto trotz agilem Vorgehen zu wenig agil sind, weil einfach das Commitment war irrsinnig schwer für das Team einzuhalten."

(Zitat aus Interview mit Geschäftsführer „GF2")

Scrum-Sprint erfordert gewisse Ruhe im Projektumfeld

Dieser Punkt hat sich im Zuge des Projektverlaufs herausgestellt. Vor allem zu dem Zeitpunkt, als viele Änderungen von Kundenseite verlangt wurden, obgleich viele Features blockiert waren. Geschäftsführer „GF1" beschreibt das Problem folgendermaßen:

„Weil der Scrum lebt ja davon, dass in der Phase, wo der Scrum läuft und das ist – habe da auch schon mit vielen anderen Leuten gesprochen – sicherlich eines der Schlüsselfaktoren, das die Leute, die an dem Ding arbeiten Ruhe haben."

(Zitat aus Interview mit Geschäftsführer „GF1")

Aufgrund des vorliegenden Problems konnten Sprints nicht mehr sinnvoll durchgeführt werden, da das Commitment auf Teamseite aufgrund von häufigen Änderungen nicht eingehalten werden konnte.

Scrum hat für Entwickler nicht unbedingt wenig Prozess-Overhead

Scrum Master „SM" relativiert, dass Scrum fast gar keinen Prozess-Overhead hat:

„Wenig Overhead in Scrum... wenn man es jetzt richtig betrachtet, hat man doch jeden Tag eine Viertelstunde Meeting und zu Sprint-Beginn und Ende auch ein paar Stunden und für den durchschnittlichen Entwickler ist wahrscheinlich schon mehr Overhead als in klassischem Projektmanagement. Man muss sich auch aktiv einbringen."

(Zitat aus Interview mit Scrum Master „SM")

Scrum unterstützt Multi-Projektmanagement nicht oder nicht gut

Wenn gesagt wurde, wir brauchen jetzt was aus dem anderen Projekt, dann sind wir einfach dazu übergegangen und haben ‚Projekt A' derweil hinter uns gelassen, sozusagen. Ich weiß nicht, ob Scrum das erlaubt.

(Zitat aus Interview mit Entwickler„EW1")

Laut Entwickler „EW1" ist Scrum also nicht nur zu statisch für sich ändernde Prioritäten innerhalb eines Projektes, sondern auch für Multi-Projektmanagement schlechter geeignet, da sich Prioritäten zwischen den Projekten ändern können und es zu Problemen kommt, wenn Mitarbeiter gleichzeitig an mehreren Projekten arbeiten.

3.3 Kanban

Kanban hat Scrum zu Projektende hin in der Wartungsphase abgelöst, als es keine oder nur mehr wenige neue Features zu implementieren gab, beziehungsweise vorhandene Features häufig extern durch den Auftraggeber blockiert waren. Der Scrum Master beschreibt die Hauptmotivation wie folgt:

„Wir haben gesagt: Wir können uns nicht mehr committen irgendwas fertig zu kriegen in diesen zwei Wochen, aber das, was wir machen, das wollen wir in der richtigen Reihenfolge tun und auch mit einer möglichst guten Durchlaufzeit."

(Zitat aus Interview mit Scrum Master „SM")

3.3.1 Ablauf

Der Kanban-Prozess wurde bei ITU so umgesetzt:

- Die Planung beschränkt sich nun auf die Vorgabe von Prioritäten. Aufwandsschätzung tritt im Vergleich zu Scrum in den Hintergrund. Dafür wurde die Frequenz des Planungsmeetings von zwei Wochen auf einen wöchentlichen Rhythmus erhöht. In diesen Priorisierungsmeetings wurde auch regelmäßig das Kanban-Board betrachtet und der Status diskutiert sowie in weiterer Folge Limits für den Work in Progress angepasst, sofern nötig und sinnvoll.
- Die Erstellung der Releases geschieht nun ebenfalls wöchentlich.
- Das Scrum Task Board („To Do", „In Progress" und „Done") wurde folgendermaßen zu einem Kanban-Board erweitert:
    - „Backlog"
    - „To Do"
    - „In Progress"
    - „Resolved"
    - „Closed"

  - „Blocked“[4]
- Sprints wurden gänzlich abgeschafft. Da es dadurch kein Commitment mehr gibt, gibt es auch keine Review-Meetings mehr, um dieses zu überprüfen.
- Retrospektive findet nur mehr unregelmäßig statt.
- Daily Meetings werden in derselben Form weitergeführt.

3.3.2 Vorzüge

Konzentration auf das aktuell Wichtigste

Die große Erleichterung bestand in Kanban durch den Wegfall der Sprints. Es war damit möglich, ungestört an den aktuell wichtigsten Tasks zu arbeiten, ohne ein Commitment im Hinterkopf haben zu müssen, dessen Erfüllung durch Blocker beeinträchtigt wird. In diesem Fall wird einfach zum nächst wichtigen nicht blockierten Task übergegangen.

Geschäftsführer „GF1“ beschreibt die Umstellung wie folgt:

„Diese Unruhe war schlagartig weg mit Kanban, weil es genau dort ansetzt, dass man sagt: „Man lässt es zu, dass in einer Produktionsphase Unruhe drinnen ist und dass sich Anforderungen ändern. Wir schrauben einfach nur ein paar andere Parameter, nämlich der Aufwand, wie viel reinläuft, was rauskommt, wie viel liegen bleibt. Und mit dem war dann schlagartig Ruhe.“

(Zitat aus Interview mit Geschäftsführer „GF1“)

Weniger Overhead, mehr direkte Arbeit

„Vordergründig ist es wahrscheinlich gut aufgenommen worden, weil es den Prozess vereinfacht hat und es gab weniger Meetings und mehr direkte Arbeit, weniger Overhead.“

(Zitat aus Interview mit Scrum Master „SM“)

Aus dem Planungsmeeting ist ein reines Priorisierungsmeeting geworden. Das Reviewmeeting wurde abgeschafft, da es keine Zyklen mehr dafür gibt. Retrospektive gibt es ebenfalls nicht mehr.

Mehr Flexibilität und schnelleres Reaktionsvermögen auf Umpriorisierungen

Das nicht vorhandene Commitment in Kanban erleichtert eine Reaktion auf Umpriorisierungen. Es muss nicht mehr auf den nächsten Sprint gewartet werden, um reagieren zu können:

[4] Die „Blocked“-Spalte operiert außerhalb des normalen Workflows, um externe Blockierungen auszuweisen, wird jedoch für gewöhnlich von einem Arbeitspaket nicht durchlaufen.

„Ich habe nur gemerkt, dass weniger gemauzt wird, wenn ich mit Änderungen gekommen bin. Dass es offenbar weniger aufwendig war, die Planung etwas zu verändern."

(Zitat aus Interview mit „Analytiker")

Entwickler „EW2" bestätigt diese Beobachtung auch von Entwicklerseite her:

„Anstatt zu warten, dass das Sprint-Ende kommt und wir dann die Tasks planen können... es hat uns mehr Flexibilität gegeben, dass wir einfach mit Tasks anfangen konnten, wenn wir mit anderen fertig waren. Nur das hat wirklich geholfen."

(Zitat aus Interview mit Entwickler „EW2")

Häufigere Priorisierungen
Mit dem Fall des Zwei-Wochen-Sprints ging man dazu über, Priorisierungen häufiger zu überarbeiten, was mehr dem Charakter von Kanban entspricht. Wöchentliche Priorisierungsmeetings dienten dafür als Forum.

Kürzeres Releaseintervall
Da man an kein Sprintintervall mehr gebunden war, wurde der Zyklus von zweiwöchigen Releases (ein Sprint) auf wöchentliche Releases verkürzt. Durch die häufige Releaseerstellung wurde immer mehr darauf geachtet, dass der Vorgang selbst möglichst wenig Zeit in Anspruch nimmt und semi-automatisiert vonstatten gehen kann. Dieser Weg wurde mit Scrum begonnen und mit Kanban weitergeführt.

Visualisierung des Work Flows
Durch die Einführung des Kanban-Boards war ein besserer Gesamtüberblick über das Projekt gegeben. Vor allem von Management-Seite her hat sich der Überblick verbessert.

„Man hat besser gesehen, wo es Probleme gibt. Also wenn was geblockt ist und das extrem kritisch ist, dann sieht man das gleich und kann was unternehmen."

(Zitat aus Interview mit Scrum Master „SM")

Auch die Qualitätssicherung zeigt sich mit der Übersicht in Kanban zufrieden, die sich im Vergleich zu Scrum noch weiter verbessert hat:

„Was man halt sehr schön sieht ist: ‚Das ist jetzt der Backlog, das was wir vorhaben und das andere, daran arbeiten wir gerade'. Das ist von der Übersichtlichkeit her sehr gut."

(Zitat aus Interview mit Leiter der Qualitätssicherung „LQS")

Visualisierung von Flaschenhälsen
Durch die Visualisierung des Work Flows können auch Flaschenhälse leicht identifiziert werden. Als Flaschenhals gilt eine Spalte, in der sich mehr Arbeitspakete befinden

als das Work in Progress-Limit zulässt. Das Überschreiten des Limits bietet demnach einen Ansatzpunkt über die Ursache der Verzögerung nachzudenken.

Visualisierung von externen Blockern
Durch die hinzugefügte Spalte „Blocked“ konnten extern blockierte Tickets als solche ausgewiesen werden. Dies hat geholfen interne Blockierungen, die durch Work in Progress-Limits aufgezeigt werden können, und externe Blockierungen zu unterscheiden.

„Aber das Einführen der ‚Blocked’-Spalte war schon ein Vorteil, weil es damit managementtauglicher wurde zu zeigen: Hier sind wir in dieser Liste von 80 Issues blockiert und können nicht weiterarbeiten.“

(Zitat aus Interview mit Scrum Master „SM“)

Limitierung des Work in Progress
Die Limitierung des Work in Progress zeigt an, wie viele Arbeitspakete sich innerhalb eines Arbeitsschrittes (z.B. „In Progress“) befinden dürfen. Dadurch wird der Ressourceneinsatz beschränkt. Als Vorteil ergibt sich eine bessere Visualisierung des tatsächlichen Arbeitsfortschritts, da eine Überschreitung des WiP-Limits auf Probleme im Workflow hinweist beziehungsweise genau aufzeigt, an welcher Stelle des Workflows es zu Wartezeiten und Problemen kommt.

Prioritäten aus Kundensicht kommunizieren gut bewährt
Die Prioritäten des Kunden dem Team transparent zu machen, hat sich in Kanban im Priorisierungsmeeting als sehr effizient erwiesen. Dadurch erhält das Team mehr Hintergrundinformationen zur Priorisierungsfindung und kann diese besser nachvollziehen:

„Im Endeffekt gebe ich da [Anm.: in den Priorisierungsmeetings] nur mehr, genau wie in den Planungsmeetings, die Priorisierung vor und gebe auch gleichzeitig - nachdem es jetzt wöchentlich ist - gleichzeitig mehr Feedback, weil das auch noch Team-Wunsch war, aus dem Projektstatus-Meeting Feedback mit dem Kunden.
(...) Deshalb werden dann die frischen Priorisierungen vom Kunden am nächsten Tag gleich ans Team weiter delegiert.“

(Zitat aus Interview mit Geschäftsführer „GF2“)

Risikominimierung durch Verringerung der Ticket-Anzahl in der „Resolved“-Spalte
In Scrum gab es nur eine „Done“-Spalte, die sowohl „Resolved“ als auch „Closed“-Tickets beinhaltete. Diese wurde in Kanban in zwei eigene Spalten „Resolved“ und „Closed“ aufgeteilt. Dies verhindert, dass „Resolved“-Tickets in der Masse verschwinden und niemals wirklich abgeschlossen werden.

Die Limitierung des Work In Progress hat weiters zu der Erkenntnis geführt, dass sehr viele „Resolved“-Tickets nicht geschlossen werden („Closed“-Spalte). „GF2“ be-

schreibt dieses Problem wie folgt:

„Wir haben uns zuvor immer damit zufrieden gegeben, wenn die Tickets „Resolved“ wurden [Anm.: jedoch nicht „Closed“]. Wir haben uns dann angeschaut, wie viel „Resolved“-Tickets wir überhaupt in dem Projekt haben und sind dann etwas erschrocken, dass es da einige 100 Tickets gibt... und die eigentlich jederzeit wieder explodieren können.“

(Zitat aus Interview mit Geschäftsführer „GF2“)

Die Verbesserung schlägt sich unmittelbar in der kürzeren Lead Time nieder, wie ein Vergleich von Tabelle 5 und Tabelle 6 im vorangehenden Kapitel zeigt sowie auch in einer Minimierung des Risikos, dass abgeschlossen geglaubte Tasks wieder geöffnet werden müssen, wenn ein Re-Test nicht erfolgreich verläuft.

Umstieg von Scrum relativ einfach
Der Umstieg von Scrum auf Kanban konnte fließend vollzogen werden. Es gibt nach wie vor Prioritäten, jedoch keine zeitliche Vorgabe mehr und damit kein Commitment. Das Planungsmeeting wurde zu einem reinen Priorisierungsmeeting, das Standup-Meeting blieb erhalten, jedoch Review und Retrospektive fielen weg. Weiters wurde das vorhandene Scrum-Board um einige Spalten erweitert und Work in Progress Limits eingeführt.

Eingliederung eines Spinoff-Projekts möglich
Mit der Hilfe von Kanban wurde es möglich, ein Spinoff-Projekt in „Projekt A“ einzugliedern. Die erfolgreiche Eingliederung unterstreicht die hohe Flexibilität, die Kanban bietet, selbst während einer laufenden Projektentwicklung.

Geschäftsführer „GF1“ beschreibt die Situation, wie folgt:

„Noch dazu mussten wir noch ein Projekt eingliedern, das sehr chaotisch war und das eigentlich ein Paradebeispiel ist, wie man Software nicht entwickeln soll. Und selbst das haben wir noch mit einem Wirkungsgrad umgesetzt, der beeindruckend ist und das haben wir nur können, weil wir Kanban eingesetzt haben. (...) Ich bin überzeugt, dass das (Anm.: Kanban) der Grund ist, warum die Projekte nicht von der Tischkante runtergefallen sind.“

(Zitat aus Interview mit Geschäftsführer „GF1“)

3.3.3 Probleme

Folgende Probleme konnten identifiziert werden:

Kanban-Board war nicht allgegenwärtig
Der Visualisierungsbonus durch das Kanban-Board war nur einmal die Woche beim Priorisierungsmeeting gegeben, ansonsten wurde darauf nicht sehr viel Wert gelegt. Dies begründete sich darin, dass viele Mitarbeiter eigene Filter für das elektronische Board benutzten, sodass nur für sie „wichtige“ Tickets sichtbar waren (wie zum Bei-

spiel nur „Qualitätssicherung“-Tickets). Dies führt aber am Ziel des Kanban-Boards vorbei, einen allgegenwärtigen Überblick schaffen. Demnach wurde das *gesamte* Kanban-Board nur einmal wöchentlich im Priorisierungsmeeting betrachtet.

Weniger Vorausplanung
Aus einigen Interviews ging hervor, dass durch die Umstellung auf Kanban, anders als in Scrum, nicht mehr so weit in die Zukunft geplant wurde, was auf das Fehlen der Sprints zurückzuführen ist.

Geschäftsführer „GF2“ äußert hierzu Bedenken:

„Wo ‚GF1’ und ich uns dann überlegt haben, ob das die richtige Entscheidung war, wir haben nicht mehr das Gefühl, dass so zielgerichtet entwickelt wird. Das Problem ist, dass man in dieser Wartungsphase auch schwer ein Ziel definieren kann, außer dass man fertig werden will.“

(Zitat aus Interview mit Geschäftsführer „GF2“)

Weniger Aufwandsschätzung
Seit der Einführung in Kanban wird die Aufwandsschätzung ungenauer betrieben. Vermutlich hängt dies auch mit dem Fehlen der Sprints respektive deren Planung zusammen. Durch die fehlende Aufwandsschätzung kann der Prozess schwer planbar und steuerbar wirken. Doch dieser Eindruck täuscht: durch Metriken wie die „durchschnittliche Cycle Time“ kann sehr wohl eine Vorausplanung gemacht werden und eine qualifizierte Schätzung abgegeben werden, ähnlich wie es in Scrum möglich war.

Volumen schwer abschätzbar
„Das Commitment war dadurch [Anm.: durch die Einteilung in Sprints in Scrum] auch klarerweise gegeben und stärker und im Nachhinein kann auch die Base ermittelt werden: ‚Sind wir noch immer so schnell wie am Anfang oder wie viel können wir abarbeiten?’ und bei Kanban ist das mehr eine Metrik über: ‚Wie viele Tickets hat man abgearbeitet?’, was aber über das Volumen überhaupt nichts aussagt.“

(Zitat aus Interview mit Leiter der Qualitätssicherung „LQS“)

Der Leiter der Qualitätssicherung „LQS“ vermisst in Kanban Aussagen über das Ticketvolumen, was Kontrolle und Planung erschwert. Ähnlich dem vorigen Punkt sind jedoch auch in Kanban Aussagen über das Volumen möglich. Die Möglichkeiten zur Verbesserung von Schätzungen sollen im Zuge des Konzepts aufgezeigt werden (vgl. Kapitel 4).

Kein größeres Bild bei der Bearbeitung des Tasks mehr vorhanden
Dadurch dass sich alle Arbeitspakete in einem Workflow befinden, fehlt die Übersicht auf einzelne Features. Auch wird die „Closed"-Spalte immer größer und niemals entleert, wie beispielsweise zuvor in Scrum mit jedem Sprint.

„Es ist halt wirklich nur ein Haufen von Tickets, die man einigermaßen organisiert abarbeitet, aber wenn du das tun musst, dann reicht das auch."

(Zitat aus Interview mit Scrum Master „SM")

Es werden demnach nur mehr Tasks abgearbeitet, jedoch fehlt der Blick für das größere Gesamtbild. Auf die Frage hin, ob fehlender Zusammenhang der Tasks bei Kanban der Hauptpunkt sei, der im Vergleich zu Scrum fehlt, erwiderte ein Mitarbeiter der Qualitätssicherung „QS" folgendes:

„Ich würde sagen das ist der Hauptpunkt, dass die Tasks nicht mehr so genau bearbeitet werden. Man nimmt einen Task, bearbeitet ihn und schließt ihn."

(Zitat aus Interview mit Mitarbeiter der Qualitätssicherung „QS")

Abhängigkeiten zwischen Entwicklungs- und Qualitätssicherungstasks am Kanban-Board schwer erkenntlich
Die Aussagekraft des Kanban-Boards wurde von Seiten der Qualitätssicherung bemängelt. Wenn auf der Entwicklerseite Tasks blockiert sind, können diese Blockierungen auch Tasks der Qualitätssicherung blockieren. Diese Abhängigkeiten sind jedoch nicht sofort am Kanban-Board ersichtlich.

Ein weiterer Grund ist, dass das Board auf Qualitätssicherungsseite nicht so gut gepflegt wird wie auf Entwicklungsseite.

Review fehlt
Weiters ist aus den Interviews hervorgegangen, dass das Reviewmeeting fehlt beziehungsweise in dieser Form nicht mehr stattfindet, und dadurch der Blick auf die Arbeit der anderen sowie auch auf die eigene im Vergleich zu Scrum ins Hintertreffen gerät.

Da es kein Review mehr gibt, ist auch ein höheres Maß an Vertrauen in Kanban notwendig:

„Man braucht schon Vertrauen, auch von Management-Seite, also Kanban ist hochgradig selbst organisierend, noch mehr als Scrum, weil es eben dieses Commitment nicht gibt. Du hast nur dieses Work in Progress Limit und du willst in Kanban ja einige wenige Dinge möglichst schnell durch die Pipeline jagen."

(Zitat aus Interview mit Geschäftsführer „GF2")

Erfolgserlebnis des Sprint-Abschlusses fehlt
Aus den Interviews geht hervor, dass einige Mitarbeiter bei Kanban das Fehlen eines Erfolgserlebnisses am Ende eines Sprints bemängeln, da es in Kanban keine Iteration dieser Art gibt.

Ein Mitarbeiter der Qualitätssicherung beschreibt das Problem folgendermaßen:

„In Kanban hab ich diese Kontrolle, was ich die letzten zwei Wochen gemacht habe, habe ich verloren. Jetzt hab ich nur mehr ein Board und die fertig gestellten Tasks kommen auf die Seite, aber was habe ich in den letzten zwei Wochen gemacht? Dieses Erfolgserlebnis fehlt."

(Zitat aus Interview mit Mitarbeiter der Qualitätssicherung „QS")

Keine Retrospektive mehr
Da es kein Review-Meeting mehr gibt, gibt es auch für die Retrospektive im Prozess keinen vorgesehenen Platz mehr, die üblicherweise nach dem Review-Meeting stattgefunden hat. Die Retrospektive ist im Scrum-Prozess sehr gut angekommen, da sie Raum bietet, über den Prozess per se zu reflektieren.

Verständigung in Standup-Meetings schwieriger
Da es kein Planungsmeeting mehr gibt, ist es schwieriger nachzuvollziehen, an welchen Tasks die Kollegen gerade arbeiten. Zu wissen, wer, woran und wieso arbeitet, kann für einen funktionierenden Workflow jedoch sehr wichtig sein. Das Kanban-Board sollte eigentlich diese Informationen bieten, daher soll dem Problem im Konzept in Kapitel 4 näher auf den Grund gegangen werden.

Flaschenhälse konnten nicht immer beseitigt werden
Vorhandene Flaschenhälse konnten zwar durch das Kanban-Board visualisiert werden; eine Beseitigung ist jedoch nicht immer möglich, da es sich großteils um externe Blockierungen handelt. Die Visualisierung von Blockierungen und der Umgang mit ihnen ist in Kanban jedoch ein zentraler Bestandteil und wird ebenfalls in das Konzept mit einfließen.

Kanban hat wenige Regeln, jedoch müssen diese eingehalten werden
„Also ich finde, bei so einem Prozess [Anm.: Kanban] müssen alle dran arbeiten, damit das funktioniert. Das hat uns einfach chaotisch gemacht."

(Zitat aus Interview mit Entwickler „EW1")

Entwickler „EW1", aber auch einige andere Mitarbeiter, bemängelten in den Interviews, dass Kanban fast schon zu viel Freiheit lässt beziehungsweise dazu einlädt, die wenigen vorhandenen Regeln nicht ausreichend ernst zu nehmen und zu befolgen. Dies ist natür-

lich am Ziel von Kanban vorbei. Da es so wenige Regeln gibt, ist es umso wichtiger diese zu befolgen. Ansonsten ist man sehr schnell bei dem von [KnSk10] benannten „Do Whatever"-Prozess angelangt, der zu keinerlei Ziel führt.

Weniger Notwendigkeit sich mit dem Backlog auseinander zu setzen
Der Product Owner (oder wie man diese Rolle bei Kanban nennen möchte) muss nun selbst darauf achten, den Backlog auf aktuellem Stand zu halten. Bei Scrum wurde diese Notwendigkeit automatisch dadurch unterstützt, dass ein gefülltes und aktualisiertes Backlog für den nächsten Sprint vorhanden sein muss:

„Aber das Kanban glaube ich gibt noch weniger Notwendigkeit sich mit dem Backlog auseinander zu setzen und beim Scrum bist du gezwungen durch den Prozess und durch die Meetings, das alle zwei Wochen zu tun."

(Zitat aus Interview mit Geschäftsführer „GF2")

Limits schwierig anzupassen
Es gab Schwierigkeiten die Limits anzupassen, da hierfür eine konstante Anzahl von Projektmitarbeitern benötigt wird. Es kommt daher beispielsweise zu Verfälschungen, wenn ein Mitarbeiter auf Urlaub geht und unklar war, wie damit umgegangen werden soll. Die Anpassung der Limits basierend auf Feedback aller Teile des Prozesses und in regelmäßigen Abständen wird ebenfalls Gegenstand des Konzepts sein.

Gemeinsame WiP Limits für Entwicklung und Qualitätssicherung nicht zielführend
Der Leiter der Qualitätssicherung bemängelt weiters, dass die gleichen Limits für die Entwicklung und die Qualitätssicherung benutzt wurden, was nicht zielführend ist, jedoch bedingt durch eine Tool-Schwäche war:

„Man kann das in Issue Tracking Tool nur einmal einstellen. Bei ‚Blocker' zum Beispiel 50 und mit unserem QA-Team (Anm.: allein) werden wir das nie sprengen. Bei den Entwicklern kann das sehr leicht geschehen, da sie ein größeres Team sind."

(Zitat aus Interview mit Leiter der Qualitätssicherung „LQS")

Limit für „To Do" Spalte zu groß
Wie das folgende Zitat beschreibt, war das Limit für die „To Do"-Spalte zu groß gewählt, sodass es demnach gleichzeitig zu viele priorisierte Issues gab, was nicht zielführend ist:

„Die ‚To Do'-Spalte hat als maximale Anzahl 40 Issues, fast ein bisschen zu groß. Also man hat schon einen Überblick, aber bei 40 Issues... die, die unten stehen, bleiben dann meistens auch unten."

(Zitat aus Interview mit Scrum Master)

Einführung der „Backlog"-Spalte hat „To Do" Spalte nicht entlastet

Wie aus Tabelle 5 und Tabelle 6 entnommen werden kann, ist die Lead Time der „To Do"-Spalte nicht geringer geworden, obwohl nun eine eigene Spalte für den Backlog eingerichtet wurde und damit in der „To Do"-Spalte nur mehr hoch priorisierte Tickets verbleiben sollten. Das unterstreicht den vorangehenden Punkt, dass die „To Do"-Spalte überfüllt war.

Einführung der „Blocked"-Spalte hat „To Do"- bzw. „In Progress"-Spalte nicht entlastet

Die „Blocked"-Spalte wurde eingeführt, um externe Blocker explizit zu kennzeichnen. Diese Tickets sind zuvor in „To Do" beziehungsweise „In Progress" verweilt, jedoch ist die Lead Time dieser beiden Spalten nicht gesunken, was eine erhöhte Anzahl von Blockern vermuten lässt. Diese Annahme deckt sich auch mit Aussagen aus den Interviews:

„Die Liste [Anm.: an externen Blockern] ist einfach immer länger geworden und wir haben gemerkt, dass da jetzt ein Stau beim Kunden und in der Analyse noch ist."

(Zitat aus Interview mit Geschäftsführer „GF2")

3.4 Generelle Erkenntnisse

Dieses Kapitel befasst sich mit generellen Beobachtungen zum Projektverlauf, das heißt mit Angelegenheiten, die gut oder schlecht funktioniert haben, jedoch nicht unmittelbar mit Scrum oder Kanban in Verbindung stehen.

Analyse über den ganzen Projektverlauf hin wichtig

„Wenn man den [Anm.: den Analytiker] schon direkt in den laufenden Prozess integriert, dann kann man es sich auch erlauben, agiler zu sein. Dann muss man nicht ein fertiges Pflichtenheft haben und dann wird umgesetzt ohne dass ein Analytiker dabei ist, sondern man macht das zu einem laufenden Prozess. Das hat durchgehend in jeder Ebene nur Vorteile."

(Zitat aus Interview mit Geschäftsführer „GF1")

Es hat sich demnach in „Projekt A" sehr gut bewährt, die Analyse-Rolle über das ganze Projekt hin zu führen. Die Analyse-Rolle wurde zu einem Schlüsselfaktor für die agile Entwicklung in „Projekt A", jedoch agierte der „Analytiker" außerhalb des Prozesses aufgrund starker Abhängigkeit zur Kundenverfügbarkeit:

„Dadurch, dass ich soviel mit dem Kunden arbeiten muss, hat es überhaupt keine Möglichkeit gegeben, das an die Sprints anzupassen. Weil das oft auftaucht, ich lese die Requirements und stelle fest, es fehlt die Hälfte oder ist nicht up to date, dann haben wir quasi den Zyklus von diesem Arbeitsschritt abbrechen müssen und ich habe mit was anderem angefangen.
Das hat überhaupt nicht zusammengepasst, mit dem, was programmiert wird. Wir haben diese Sprints einfach nicht gehabt.“

(Zitat aus Interview mit „Analytiker“)

Agile Entwicklung beginnt in der Projektanbahnungsphase

Aus dem Interview mit dem Geschäftsführer „GF1“ geht sehr anschaulich anhand eines Beispiels hervor, dass der agile Ansatz in der ITU bereits in der Projektanbahnungsphase gelebt wird und dort auch enorm wichtig ist:

„Für uns als Pauschalanbieter ist es eine enorm wichtige Sache ein Projekt am Anfang so zu hinterfragen, dass einfache Aufgaben überbleiben und der große Vorteil ist, das auf beiden Seiten zu tun – also zwischen Auftraggeber und Auftragnehmer.
Zum Beispiel das neue Projekt – Projektanfrage: die haben ein Budget von Hausnummer 70 000, wir schauen uns das an und kommen auf 300 000, wir liegen um das 3,5fache auseinander. Was tut man in so einem Fall? Fangen dann zwei Kaufleute zum streiten an und der eine sagt ‚viel zu viel’ und der andere ‚viel zu wenig’ und dann trifft man sich irgendwo in der Mitte und entwickelt es wahrscheinlich um 180 000.

Totaler Blödsinn. Was man wirklich tun sollte ist, dass man einfach sagt: ‚Was habt ihr verstanden von diesen Anforderungen, wie kommt ihr auf diese 70 000? Was haben wir verstanden darunter, wie kommen wir auf diese 300 000?’.
Da fängt in Wirklichkeit schon der Scrum an: ‚Wir haben die und die Komponenten, die und die Auswirkungen, es muss so und so skalieren, muss so und so ausschauen’, und dann brichst du genau diesen Backlog schon herunter. Dann bleibt eine Liste von Aktivitäten über und auf die schauen beide Seiten, Auftraggeber und Auftragnehmer.

Wenn die Liste gut ist, dann fängst du an und sagst: ‚Das dauert 2 Tage, das dauert 1 Tag, das kannst streichen, usw.’. Dann bleibt eine konsolidierte Aufwandsliste über von beiden Seiten und dann wird maximal noch über den Stundensatz gestritten. Genau dort findet sich dann die Wahrheit. Das ist das allumfassende Geheimnis von einer erfolgreichen Pauschalprojektentwicklung, nämlich dass wir so herangehen bevor wir anfangen zu entwickeln. Wenn wir anfangen, das ist schon fast ein Backlog. Diese Projektanbahnungsrolle ist eigentlich schon ein Großteil von der Software-Analyse, die man da macht und die man auch machen muss bevor man beginnt zu entwickeln.
So sind wir zum Scrum gekommen. Das ist der Grund, warum wir das so durchgehend leben, weil wir vom ersten Kundengespräch an eigentlich schon die Analyse und die gesamte Projektbesprechung in diese Richtung hinbringen.“

(Zitat aus Interview mit Geschäftsführer „GF1“)

Flexibilität durch Umpriorisierungen
Sowohl in Scrum als auch in Kanban ist es möglich, Priorisierungen im Laufe des Projekts zu verändern mit dem Ergebnis einer enormen Flexibilität. In Scrum ist dies je Sprint möglich, in Kanban sogar noch flexibler, da das „Pull"-Prinzip umgesetzt ist.

Enge Zusammenarbeit zwischen Entwicklung und Qualitätssicherung hat für beide Seiten Vorteile
Sowohl in Scrum als auch Kanban arbeiteten Entwicklung und Qualitätssicherung sehr eng zusammen. Dies konnte vor allem durch die täglich gemeinsam abgehaltenen Standup-Meetings sichergestellt werden, sowie auch durch das gemeinsame Planungsmeeting (Scrum) respektive Priorisierungsmeeting (Kanban).

Ein Mitarbeiter der Qualitätssicherung „QS" beschreibt die Zusammenarbeit folgendermaßen:

„Das war schon ziemlich eng zusammen. Zum Beispiel: es kommen dauernd Change Requests vom Auftraggeber und durch die Daily Meetings haben wir gut verfolgen können, wo sie [Anm.: die Entwickler] arbeiten und was sie ändern, sodass wir die Testfälle dann auch anpassen. Dass es eng war, hat unsere Arbeit schon sehr viel leichter gemacht."

(Zitat aus Interview mit Mitarbeiter der Qualitätssicherung „QS")

Autorität und Akzeptanz bei agilen Prozessen ebenfalls wichtig
Geschäftsführer „GF2" spricht aus persönlicher Erfahrung mit „Projekt A", dass die Rollenakzeptanz bei agilen Prozessen von zentraler Bedeutung ist, da es keine strikte hierarchische Rangordnung gibt, sondern Eigeninitiative gefragt ist.

„Das ist eine unserer Grundphilosophien: Wir verteilen eigentlich nie Kompetenzen, sondern die kommen immer aus dem Team. Das ist ähnlich wie bei der Apache Group, diese Open Source Community, die uns am besten gefällt. Dort ist es so: Der, der am besten in einem Gebiet ist und auch die meiste Arbeit leistet, hat auch am meisten zu sprechen."

(Zitat aus Interview mit Geschäftsführer „GF2")

Agile Prozesse erfordern Selbstdisziplin
Es hat in „Projekt A" Probleme mit der Umstellung von klassischem Projektmanagement auf Scrum gegeben. Dieses Problem hat sich aber weniger spezifisch auf Scrum bezogen, sondern generell auf agile Prozesse. Agile Prozesse erfordern viel Eigeninitiative, aber auch die Übernahme von Verantwortung.

Letztendlich ist es nicht möglich, einen Prozess umzustellen, wenn diese Umstellung nicht auch von allen Mitarbeitern getragen wird. So gab es vor allem Probleme bei den täglichen Standup-Meetings:

„Wir haben einen im Team gehabt, der schwer diesen kurzen organisatorischen Abgleich mit ewigen technischen Diskussionen trennen konnte. Das Team hat lange gebraucht, damit umzugehen und letztendlich hat das auch zu einer Trennung geführt, der war einfach in diesen Prozess nicht integrierbar.
Das klingt jetzt wahrscheinlich relativ hart und ist es auch, aber man muss auch das Team entsprechend ‚staffen' (Anm.: von engl. ‚staff') und aufstellen."

(Zitat aus Interview mit Geschäftsführer „GF2")

Auch Entwickler „EW2" beschreibt Probleme mit der Umstellung auf Scrum aufgrund individueller Widerstände gegen das Konzept:

„Das ist das Problem mit Scrum eigentlich. Es ist besser, wenn die Leute wirklich im Team arbeiten können. Weil auch der Scrum Master ist nur ein Rolle und kein Begriff von Autorität. Es gibt Leute, die haben Schwierigkeiten so zu arbeiten und stellen Hierarchien auf."

(Zitat aus Interview mit Entwickler „EW2")

4 Konzept zur Verbindung von Scrum und Kanban

In der Praxis kann es oft sinnvoll sein, Prozesse und Tools zu kombinieren, um der individuellen Arbeitsweise eines Projektteams zu entsprechen und damit die Vorteile verschiedener Paradigmen für sich nutzen zu können:

„Do not develop an attachment to any one weapon or any one school of fighting."
(Miyamoto Musashi, Samurai des 17. Jahrhunderts)

In diesem Kapitel soll nun ein Konzept entwickelt werden, das Verknüpfungsmöglichkeiten für Scrum und Kanban aufweist. Kapitel 4.1 erläutert die genaue Vorgehensweise zur Herleitung des Konzepts.

4.1 Herleitung

Input für die Entwicklung des Konzepts bietet sowohl die theoretische Basis (vgl. Kapitel 2), die praktische Basis (vgl. Kapitel 3) als auch [Ande10], [KnSk10] und [Lada08], die sich alle mit der Umsetzung von Kanban respektive Scrum beschäftigen.

Da dieses Konzept sehr praxisnah entwickelt wird, werden zuerst in Kapitel 4.2 die Ergebnisse der praktischen Untersuchung präsentiert, die in Form von Interviews stattgefunden hat. Es werden sowohl Vor- als auch Nachteile von Scrum respektive Kanban aufgelistet, die im Tagesgeschäft der ITU aufgetreten sind und sich in den Interviews mit den Mitarbeitern widergespiegelt haben. In weiterer Folge werden auch Verbesserungsvorschläge genannt, die im Laufe der Interviews von Mitarbeitern der ITU gemacht wurden.

In Kapitel 4.3 sollen schließlich identifizierte Probleme (vgl. Tabelle 7: „Vor- und Nachteile von Scrum (praktische Anwendung der ITU)" respektive Tabelle 8: „Vor- und Nachteile von Kanban (praktische Anwendung in der ITU)") im Rahmen des Konzepts gelöst werden.

Lösungsmöglichkeiten:

- Vorhandene Verbesserungsvorschläge aus den Interviews (vgl. Kapitel 4.2.4)
- Kapitel 2.6 „State of the Art"
- Literatur [Ande10], [KnSk10] und [Lada08]

Die vorhandenen Verbesserungsvorschläge aus den Interviews sollen mit Hilfe der Literatur hinsichtlich ihrer Umsetzbarkeit überprüft werden sowie Widersprüche aufgelöst werden.

Abgesehen von den vorhandenen Verbesserungsvorschlägen sollen auch neue Vorschläge gebracht werden, die sich auf das Kapitel 2.6 „State of the Art“ sowie [Ande10], [KnSk10] und [Lada08] stützen.

Als Ergebnis dieser Herleitung soll schließlich in Kapitel 4.4 ein konsolidiertes Konzept entstehen, das sowohl praktische Erfahrungen als auch theoretische „Best Practices“ miteinander vereint.

4.2 Ergebnis praktische Basis

In diesem Kapitel finden sich Vor- und Nachteile von Scrum (Kapitel 4.2.1) respektive Kanban (Kapitel 4.2.2), wie sie in den Interviews genannt wurden. In Kapitel 4.2.3 werden aufgetretene Probleme zu konsolidierten Problemen zusammengefasst.

In Kapitel 4.2.4 folgt eine Auflistung und Diskussion der in den Interviews genannten Verbesserungsvorschläge.

Vorweg zu nehmen sei an dieser Stelle, dass in den Interviews der allgemeine Konsens darin bestand, dass Scrum in einer ruhigen Projektsituation sehr gut funktioniert (ausreichend gefülltes Backlog für kommenden Sprint, keine häufigen Change Requests, wenig externe Blockierungen), während Kanban sich gut für unruhige Projektsituationen eignet (häufige Umpriorisierungen und externe Blockierungen).

4.2.1 Vor- und Nachteile Scrum

Zuerst folgt eine Auflistung der Vor- und Nachteile von Scrum, so wie sie von den Mitarbeitern der ITU im Zuge der Interviews beschrieben wurden.

Scrum Vorteile	Scrum Nachteile
Bevollmächtigung des Teams	Scrum benötigt stets ausreichend gefülltes Backlog
Klar definierte Feature-Größen	Unfähigkeit auf Änderungen schnell zu reagieren
Gute Abschätzbarkeit	Scrum-Sprint erfordert gewisse Ruhe im Projektumfeld
Einfaches Reporting möglich	Scrum hat für Entwickler nicht unbedingt wenig Prozess-Overhead
Wenig Prozess-Overhead	Scrum unterstützt Multi-Projektmanagement nicht oder nicht gut
Leistungsnachweis visualisiert in Review-Meeting	
Retrospektive bietet Möglichkeit über den Prozess als solchen zu reflektieren	
Qualitätssicherung kann im gleichen Sprint-Rhythmus arbeiten	
Aufteilung der Product Owner-Rolle auf mehrere Personen gut bewährt	

Tabelle 7: Vor- und Nachteile von Scrum (praktische Anwendung der ITU)

4.2.2 Vor- und Nachteile Kanban

Die nachfolgende Tabelle zeigt die Vor- und Nachteile der praktischen Anwendung von Kanban in der ITU, so wie sie aus den Interviews hervorgehen.

Kanban Vorteile	Kanban Nachteile
Konzentration auf das aktuell Wichtigste	Kanban-Board war nicht allgegenwärtig
Weniger Overhead, mehr direkte Arbeit	Weniger Vorausplanung
Mehr Flexibilität und schnelleres Reaktionsvermögen auf Umpriorisierungen	Weniger Aufwandsschätzung
Häufigere Priorisierungen	Volumen schwer abschätzbar
Kürzeres Releaseintervall	Kein größeres Bild bei der Bearbeitung des Tasks mehr vorhanden
Visualisierung des Work Flows	Abhängigkeiten zwischen Entwicklungs- und Qualitätssicherungstasks am Kanban-Board schwer erkenntlich
Visualisierung von Flaschenhälsen	Review fehlt
Visualisierung von externen Blockern	Erfolgserlebnis des Sprint-Abschlusses fehlt
Limitierung des Work in Progress	Keine Retrospektive mehr
Prioritäten aus Kundensicht kommunizieren gut bewährt	Verständigung in Standup-Meetings schwieriger
Risikominimierung durch Verringerung der Ticket-Anzahl in der „Resolved“-Spalte	Flaschenhälse konnten nicht immer beseitigt werden
Umstieg von Scrum relativ einfach	Kanban hat wenige Regeln, jedoch müssen diese eingehalten werden
Eingliederung eines Spinoff-Projekts möglich	Weniger Notwendigkeit sich mit dem Backlog auseinander zu setzen
	Limits schwierig anzupassen
	Gemeinsame WiP Limits für Entwicklung und Qualitätssicherung nicht zielführend
	Limit für „To Do“ Spalte zu groß
	Einführung der „Backlog“-Spalte hat „To Do“ Spalte nicht entlastet
	Einführung der „Blocked“-Spalte hat „To Do“- bzw. „In Progress“-Spalte nicht entlastet

Tabelle 8: Vor- und Nachteile von Kanban (praktische Anwendung in der ITU)

4.2.3 Vor- und Nachteile von Scrum und Kanban konsolidiert

Generell kann beobachtet werden, dass Kanban unter den Mitarbeitern der ITU mehr polarisiert hat als Scrum, da sowohl mehr Vorteile als auch Nachteile genannt wurden. Dieser Fakt mag daher kommen, dass Scrum bereits lange Zeit in Verwendung war und Kanban erst seit kurzem, und die Stärken und Schwächen von Kanban daher noch vergleichsweise unbekannt sind in der ITU.

Die identifizierten Vorteile von Scrum (vgl. Tabelle 7) können wie im Folgenden kategorisiert werden, wobei *„Bevollmächtigung des Teams“* und *„Klar definierte Feature-Größen“* nicht einbezogen werden, da sie ebenfalls in Kanban vorhanden sind und daher für die Gegenüberstellung keinen Mehrwert bringen. Der Vorteil *„Aufteilung der Product Owner-Rolle auf mehrere Personen gut bewährt“* wird ebenfalls nicht in die Konsolidierung einbezogen, da er nichts mit dem Vergleich von Scrum und Kanban zu tun hat.

Sprint-Rhythmus vereinfacht Planung

- Gute Abschätzbarkeit
- Einfaches Reporting möglich
- Leistungsnachweis visualisiert in Review-Meeting
- Qualitätssicherung kann im gleichen Sprint-Rhythmus arbeiten

Reflexion ist Teil des Prozesses

Retrospektive bietet die Möglichkeit über den Prozess als solchen zu reflektieren und ihn dadurch zu verbessern.

Die identifizierten Nachteile von Scrum (vgl. Tabelle 7) können wie folgt kategorisiert werden. Der Vorteil *„Weniger Prozess-Overhead“* widerspricht dem Nachteil *„Scrum hat für Entwickler nicht unbedingt wenig Prozess-Overhead“*, daher wird dieser Punkt weder als Vorteil noch als Nachteil einbezogen. Der Nachteil *„Scrum unterstützt Multi-Projektmanagement nicht oder nicht gut“* wurde nicht in die Konsolidierung einbezogen, da Multi-Projektmangement außerhalb des Fokus dieser Arbeit liegt und Scrum und Kanban sich laut Literatur (vgl. [KnSk10]) beide für Multi-Projektmanagement eignen.

Scrum funktioniert nur in ruhigem Projektumfeld

- Scrum benötigt stets ausreichend gefülltes Backlog
- Unfähigkeit auf Änderungen schnell zu reagieren
- Scrum-Sprint erfordert gewisse Ruhe im Projektumfeld

Die identifizierten Vorteile in Kanban (vgl. Tabelle 8) können wie folgt kategorisiert werden:

Sehr hohe Flexibilität

- Konzentration auf das aktuell Wichtigste
- Mehr Flexibilität und schnelleres Reaktionsvermögen auf Umpriorisierungen
- Häufigere Priorisierungen
- Kürzeres Releaseintervall
- Eingliederung eines Spinoff-Projekts möglich

Visualisierung verbessert

- Visualisierung des Work Flows
- Visualisierung von Flaschenhälsen
- Visualisierung von externen Blockern
- Prioritäten aus Kundensicht kommunizieren gut bewährt
- Limitierung des Work in Progress
- Risikominimierung durch Verringerung der Ticket-Anzahl in der „Resolved"-Spalte

Leichtgewichtiger Prozess

- Weniger Overhead, mehr direkte Arbeit
- Umstieg von Scrum relativ einfach

Die identifizierten Nachteile in Kanban (vgl. Tabelle 8) können wie folgt kategorisiert werden:

Schätzung und Planung ist schwierig

- Weniger Vorausplanung
- Weniger Aufwandsschätzung
- Volumen schwer abschätzbar
- Weniger Notwendigkeit sich mit dem Backlog auseinander zu setzen

Kontrolle des Projektfortschritts nicht ausreichend gegeben

- Kanban-Board war nicht allgegenwärtig
- Kein größeres Bild bei der Bearbeitung des Tasks mehr vorhanden
- Review fehlt
- Erfolgserlebnis des Sprint-Abschlusses fehlt
- Verständigung in Standup-Meetings schwieriger

Gefahr „zu agil“ zu werden

- Keine Retrospektive mehr
- Kanban hat wenige Regeln, jedoch müssen diese eingehalten werden

Unstimmigkeiten zwischen Entwicklung und Qualitätssicherung

- Abhängigkeiten zwischen Entwicklungs- und Qualitätssicherungstasks am Kanban-Board schwer erkenntlich
- Gemeinsame WiP Limits für Entwicklung und Qualitätssicherung nicht zielführend

Schwierigkeiten mit WiP Limits

- Flaschenhälse konnten nicht immer beseitigt werden
- Limits schwierig anzupassen
- Limit für „To Do“ Spalte zu groß
- Einführung der „Backlog“-Spalte hat „To Do“ Spalte nicht entlastet
- Einführung der „Blocked“-Spalte hat „To Do“- bzw. „In Progress“-Spalte nicht entlastet

Es folgt abschließend eine Matrix in Tabelle 9, um die konsolidierten Vor- und Nachteile von Scrum respektive Kanban zusammenzufassen, so wie sie aus den Interviews mit Mitarbeitern der ITU hervorgegangen sind:

	Scrum	Kanban
Vorteile	Sprint-Rhythmus vereinfacht Planung Reflexion ist Teil des Prozesses	Sehr hohe Flexibilität Visualisierung verbessert Leichtgewichtiger Prozess
Nachteile	Scrum funktioniert nur in ruhigem Projektumfeld	Schätzung und Planung ist schwierig Kontrolle des Projektfortschritts nicht ausreichend gegeben Gefahr „zu agil“ zu werden Unstimmigkeiten zwischen Entwicklung und Qualitätssicherung Schwierigkeiten mit WiP Limits

Tabelle 9: Vor- und Nachteile von Scrum und Kanban konsolidiert

In der konsolidierten Matrix in Tabelle 9 zeigt sich auch, dass Kanban in den Interviews mehr polarisiert hat, da sowohl mehr Vorteile als auch Nachteile genannt wurden.

4.2.4 Verbesserungsvorschläge aus Interviews

An dieser Stelle seien die Vorschläge aus den Interviews genannt. Sie stellen neben den Vor- und Nachteilen von Scrum beziehungsweise Kanban die praktische Grundlage für das zu erstellende Konzept dar.

Verbesserungsvorschläge	Interviewpartner
Maßkriterium für Unruhe im Sprint finden	Geschäftsführer „GF1“
Steigende Länge der Planungs- und Reviewmeetings & Team-Unsicherheit könnte als Indikator für Unruhe dienen	Geschäftsführer „GF2“
Retrospektive wieder öfter durchführen	Scrum Master „SM“
WiP-Limits öfter anpassen	Scrum Master „SM“
Scrum ist gut geeignet für die Architekturphase, da sie chaotisch ist und man durch den Sprint vorausplanen muss	Entwickler „EW2“
Scrum-Werkzeuge wie das Daily Meeting können auch in Kanban nützlich sein	Entwickler „EW2“
Mehr darauf achten die Lead Time zu verringern und das Reporting diesbezüglich zu vereinfachen	Entwickler „EW2“
Es besteht die Möglichkeit, dass nur die Qualitätssicherung auf Scrum zurückwechselt, sofern sich die Situation für sie schneller beruhigt	Leiter der Qualitätssicherung „LQS“
Möglichkeit eine Übersicht zu bekommen in Kanban wäre ein Filter auf das „Closed“-Datum von Tickets	Leiter der Qualitätssicherung „LQS
Ein Burndown-Chart oder vergleichbare Metriken wären in Kanban wünschenswert	Leiter der Qualitätssicherung „LQS
Die Qualitätssicherung braucht eigene WiP-Limits, da das Board gemeinsam benutzt wird.	Leiter der Qualitätssicherung „LQS
Auch in Kanban ausreichend schätzen	Mitarbeiter der Qualitätssicherung „QS“
Ist ein Development-Task blockiert, so sollten auch davon abhängende Qualitätssicherung-Tasks auf „Blocked“ gesetzt werden	Mitarbeiter der Qualitätssicherung „QS“

Tabelle 10: Verbesserungsvorschläge aus Interviews

4.3 Problemlösungen

In diesem Kapitel sollen Lösungen für die in Tabelle 9 identifizierten Probleme („Nachteile“) erarbeitet werden, die in das Konzept miteinfließen.

Die nachfolgende Tabelle stellt die konsolidierten Probleme den in Interviews gebrachten Lösungsvorschlägen gegenüber.

Konsolidierte Probleme	Lösungsvorschläge
Scrum: Scrum funktioniert nur in ruhigem Projektumfeld	Maßkriterium für Unruhe im Sprint finden*(aus Interview mit Geschäftsführer „GF1“)* Steigende Länge der Planungs- und Review-meetings & Team-Unsicherheit könnte als Indikator für Unruhe dienen *(aus Interview mit Geschäftsführer „GF2“)*
Kanban: Schätzung und Planung ist schwierig Kanban: Kontrolle des Projektfortschritts nicht ausreichend gegeben	Mehr darauf achten die Lead Time zu verringern und das Reporting diesbezüglich zu vereinfachen *(aus Interview mit Entwickler „EW2“)* Möglichkeit eine Übersicht zu bekommen in Kanban wäre ein Filter auf das „Closed“-Datum von Tickets *(aus Interview mit Leiter der Qualitätssicherung „LQS“)* Ein Burndown-Chart oder vergleichbare Metriken wären in Kanban wünschenswert *(aus Interview mit Leiter der Qualitätssicherung „LQS“)* Auch in Kanban ausreichend schätzen *(aus Interview mit Mitarbeiter der Qualitätssicherung „QS“)*
Kanban: Gefahr „zu agil“ zu werden	Retrospektive wieder öfter durchführen *(aus Interview mit Scrum Master „SM“)* Scrum-Werkzeuge wie das Daily Meeting können auch in Kanban nützlich sein *(aus Interview mit Entwickler „EW2“)*
Kanban: Unstimmigkeiten zwischen Entwicklung und Qualitätssicherung	Es besteht die Möglichkeit, dass nur die Qualitätssicherung auf Scrum zurückwechselt, sofern sich die Situation für sie schneller beruhigt *(aus Interview mit Leiter der Qualitätssicherung „LQS“)* Ist ein Development-Task blockiert, so sollten auch davon abhängende Qualitätssicherung-stasks auf „Blocked“ gesetzt werden *(aus Interview mit Mitarbeiter der Qualitätssicherung „QS“)*
Kanban: Schwierigkeiten mit WiP Limits	WiP-Limits öfter anpassen *(aus Interview mit Scrum Master „SM“)* Die Qualitätssicherung braucht eigene WiP-Limits, da das Board gemeinsam benutzt wird. *(aus Interview mit Leiter der Qualitätssicherung „LQS“)*

Tabelle 11: Lösungsvorschläge für konsolidierte Probleme

Weiters sollen an dieser Stelle auch die Papers aus Kapitel 2.6 „State of the Art“ auf ihre Nützlichkeit hin für das zu erstellende Konzept untersucht werden.

Die in Kapitel 2.6.1 vorgestellte "SPI-LEAM"-Methode (Software Process Improvement through the Lean Measurement) aus [PeWo10] ist für das vorhandene Praxisbeispiel zu aufwendig und geht am Ziel vorbei, da Kanban in der ITU bereits eingeführt ist.

Auch die in Kapitel 2.6.2 erläuterten Paradigmen "Goal-driven Software Development" und "Test-driven Software Development" aus [JaSu09] sollen nicht näher betrachtet werden. Beide Paradigmen sind Pull-Systeme genau wie Kanban, jedoch ist der Fokus dieser Arbeit darauf gesetzt Scrum und Kanban zu vereinen und nicht Kanban durch andere Pull-Systeme zu ersetzen.

Kapitel 2.6.3 wurden die Vor- und Nachteilen agiler Softwareentwicklung zusammengefasst, die [PeWo09] untersucht hat. Dies sind jedoch allgemeine Erkenntnisse agiler Software Entwicklung und lassen sich nicht gut auf die vorhandene Problematik abbilden (da auch ein weiterer agiler Prozess, Extreme Programming „XP“, miteinbezogen wurde) oder sind bereits bekannt.

4.4 Detaillierte Erläuterung des Konzepts

Scrum und Kanban können beide als Prozess-Werkzeuge mit unterschiedlichen Stärken und Schwächen gesehen werden [KnSk10]. Daher soll an dieser Stelle ein Konzept geschaffen werden, das die Stärken beider Prozesse für sich nutzt.

Dies soll geschehen, indem die beiden Prozesse einander folgen, sodass jeweils der für die aktuelle Projektphase am besten geeignete Prozess zum Einsatz kommt beziehungsweise Kanban um Scrum-Elemente erweitert wird.

Es folgt die Aufteilung des Einsatzes von Scrum respektive Kanban entsprechend den Projektphasen:

Architekturphase: Scrum oder Kanban?
Hier sind sowohl Scrum als auch Kanban denkbar. Bisher wurde für die Architekturphase in der ITU Scrum eingesetzt, der Vorteil hierfür wird von Entwickler „EW2“ wie folgt beschrieben (vgl. Tabelle 10):

„Scrum ist gut geeignet für die Architekturphase, da sie chaotisch ist und man durch den Sprint vorausplanen muss.“

(Zitat aus Interview mit Entwickler „EW2“)

Die Planung auf den Zeitraum eines Scrum-Sprints soll also dabei behilflich sein, eine gewisse Ruhe in die Architekturphase zu bringen und die Planung zu forcieren.

Jedoch könnte sich auch Kanban sehr gut für diese Phase eignen, da es hier normalerweise häufig Änderungen gibt und Kanban sehr flexibel bezüglich Umpriorisierungen ist.

Die Entscheidung diesbezüglich soll zu diesem Zeitpunkt offen gelassen werden und später im Zuge der praktischen Erprobung in der ITU nachgereicht werden, damit eine fundierte Entscheidung getroffen werden kann.

Produktionsphase: Scrum
Wie aus den Interviews übereinstimmend hervorgeht, hat Scrum überaus gut für die Produktionsphase funktioniert, daher soll Scrum hierfür ohne weitere Anpassungen auch beibehalten werden.

Stellvertretend für die allgemeine Meinung in der ITU zur Nützlichkeit von Scrum in der Produktionsphase sei folgendes Zitat genannt:

„Ja, es hat sehr gut funktioniert. Wir hatten Tasks, die gut definiert waren. Wir hatten eine sehr gute Sicht auf die Zeitschätzung und Scrum war wirklich perfekt als Prozess für die Produktion.“

(Zitat aus Interview mit Entwickler „EW2“)

Projektendphase: Kanban
Gegen Ende des Projekts hin kann es sein, dass sich Prioritäten häufiger ändern als während des Projektverlaufs, beispielsweise aufgrund von Termindruck. Um mit Prioritätsänderungen flexibler umgehen zu können, soll Kanban in dieser Phase zum Einsatz kommen und Scrum ablösen.

Die folgenden Unterkapitel befassen sich mit Ausnahme von Kapitel 4.4.1 damit, wie der Einsatz von Kanban verbessert werden kann.

4.4.1 Wechsel von Scrum auf Kanban

Dieses Kapitel beschäftigt sich mit dem konsolidierten Problem *„Scrum funktioniert nur in ruhigem Projektumfeld“*. Der Lösungsvorschlag basiert auf den Interviews mit Geschäftsführern „GF1“ *(„Maßkriterium für Unruhe im Sprint finden“)* und „GF2“ (*„Steigende Länge der Planungs- und Reviewmeetings und Team-Unsicherheit könnte als Indikator für Unruhe dienen“*).

In den Interviews wurde bemängelt, dass Scrum nicht gut mit häufigen Priorisierungsänderungen umgehen kann. Daher soll in der Projektendphase Kanban eingesetzt werden. Es ist sehr schwierig (sofern überhaupt möglich), den Zeitpunkt des Wechsels von Scrum zu Kanban zu bestimmen. Daher soll, wie von Geschäftsführern „GF1“ und

„GF2“ in den Interviews vorgeschlagen (vgl. Tabelle 10), die Teamunruhe als Faktor dienen. Da Teamunruhe kein messbarer Wert ist, soll sie vielmehr in der Sprint-Retrospektive beziehungsweise den Planungsmeetings beobachtet werden.

Folgende Faktoren können Gründe für Unruhe im Team sein:

- Zu wenig gefülltes Backlog für Sprintplanung
- Häufige Umpriorisierungen innerhalb eines Sprints
- Häufige interne oder externe Blockierungen

Diese Phase muss nicht zwingend am Projektende auftreten, sondern kann auch früher hervorgerufen werden (beispielsweise durch externe Blockierungen). Es ist die Aufgabe des Scrum Masters die Stimmung im Team einzuschätzen und gemeinsam mit dem Team und Product Owner zu entscheiden, ob und wann ein Wechsel auf Kanban sinnvoll ist.

4.4.2 Das „richtige“ WiP Limit finden

Dieses Kapitel befasst sich mit dem konsolidierten Problem *„Schwierigkeiten mit WiP Limits“* und basiert auf [KnSk10].

Um den Work in Progress (WiP) limitieren zu können, müssen alle Tasks gleich groß sein. Alternativ können Tasks auch mit Story Points bewertet werden [KnSk10]. In diesem Fall basiert das WiP Limit nicht auf Taskanzahl, sondern auf Story Points. Der Nachteil dabei besteht jedoch in dem Umstand, dass das Kanban-Board dadurch in seiner Handhabung komplizierter wird.

Für die Größe der WiP-Limits gibt es keine Faustregel, nur durch Experimentieren kann man sich dem idealen Limit annähern:

„Kanban says you should limit WIP. So what should the limit be? Don’t know, experiment.“ [KnSk10]

Grundsätzlich gilt jedoch:

- Ist das WiP-Limit zu niedrig, sind häufig Mitarbeiter ohne Aufgabe.
- Ist das WiP-Limit zu hoch, werden Probleme erst sehr spät entdeckt, da man eher zu einem neuen Task übergeht, wenn der alte blockiert ist als sich um das Problem zu kümmern.

Sowohl ein zu niedriges als auch ein zu hohes WiP-Limit wirkt sich negativ auf die durchschnittliche Cycle Time aus. Ziel ist es, durch Experimentieren mit den WiP-Limits, eine Balance zu finden.

Wird das WiP Limit verändert, so müssen Auswirkungen auf folgende Punkte beobachtet werden (und eventuell weitere):

- Kapazität
- Cycle Time (auch Lead Time genannt)
- Qualität
- Vorhersehbarkeit

Weiters soll jedes Mal, wenn ein Arbeitspaket die „Done"-Spalte erreicht, die durchschnittliche Cycle Time aktualisiert werden [KnSk10].

Der wichtigste Input für Änderungen des Work in Progress Limits ist eine funktionierende Feedback-Loop (vgl. Kapitel 4.4.3).

Als äußerster Zeitrahmen für das Anpassen der „WiP"-Limits könnte der „Kanban-Sprint" dienen (vgl. Kapitel 4.4.10).

4.4.3 Feedback Loop verbessern

Dieses Kapitel befasst sich mit dem konsolidierten Problem „*Gefahr zu ‚agil' zu werden*", „*Kein größeres Bild bei der Bearbeitung des Tasks mehr vorhanden*", basiert auf [KnSk10] und beinhaltet den Lösungsvorschlag von Scrum Master „SM" *(„WiP-Limits öfter anpassen"*).

[KnSk10] schlägt vor auch in Kanban die in agilen Prozessen üblichen Feedback Loops einzusetzen von XP's[5] „Pair Programming" bis hin zum Scrum-Sprint (vgl. Abbildung 18).

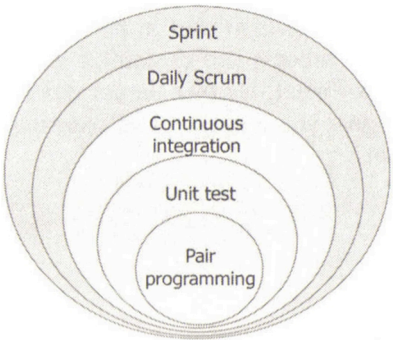


Abbildung 18: Agile Feedback Loop [KnSk10]

Innere Feedback Loops dienen einem anderen Zweck als die äußeren. Die inneren können Feedback innerhalb von Sekunden geben (Pair Programming), während es sich bei den äußeren um Wochen handelt (Sprint). Demnach ändert sich auch die Sichtweise:

[5] Es handelt sich bei der Abkürzung „XP" um den agilen Prozess „Extreme Programming".

*„Are we building the stuff **right**?“ [KnSk10]* (Pair Programming)
im Gegensatz zu:
*„Are we building the **right** stuff?“ [KnSk10]* (Sprint).

Die Länge der Feedback Loops kann variiert werden. Je länger jedoch die Feedback Loop, desto seltener gibt es Verbesserungen im Prozess. Zu kurze Feedback Loops hingegen bewirken Instabilität.

Weiters helfen funktionierende Feedback-Loops auch dem Problem entgegenzuwirken, dass der Überblick über die Arbeit der anderen verloren geht und schaffen Anreiz über WiP-Limits nachzudenken und diese gegebenenfalls anzupassen.

Als äußerste Feedback Loop kann auch in Kanban ein stark vereinfachter Sprint eingeführt werden (vgl. Kapitel 4.4.10).

4.4.4 Retrospektive einführen

Dieses Kapitel befasst sich mit den Problemen „*Scrum-Sprint erfordert gewisse Ruhe im Projektumfeld*“, „*Review fehlt*“, „*Erfolgserlebnis des Sprint-Abschlusses fehlt*“ und das konsolidierte Problem „*Gefahr ‚zu agil' zu werden*“, basiert auf [KnSk10] und beinhaltet den Lösungsvorschlag von Scrum Master „SM“ *(„Retrospektive wieder öfter durchführen“*).

Anschließend an das vorangehende Kapitel, soll auch eine Retrospektive eingeführt werden, um die Feedback Loop weiter auszubauen. Die Retrospektive ermöglicht es über den Prozess als solchen zu reflektieren und ihn daher zu verbessern sowie die allgemeine Zufriedenheit zu erhöhen.

Es sollten zumindest folgende Fragen behandelt werden [KnSk10]:

- Welche Spalten im Kanban-Board sollten wir hinzufügen oder streichen? Wo gibt es Probleme?
- Was sollten die Kanban-Limits je Spalte sein?
- Wie streng werden die Kanban-Limits eingehalten? Wie streng sollen sie eingehalten werden?

Generell sollte die Retrospektive Raum bieten, auf jegliche Probleme im Prozess hinweisen zu können und Lösungen dafür zu besprechen.

Weiters muss für die Retrospektive ein konkreter Zeitpunkt festgesetzt werden. Diesen Zeitpunkt könnte beispielsweise das Ende eines „Kanban-Sprints“ darstellen (vgl. Kapitel 4.4.10).

4.4.5 Analyse mehr integrieren

Die Analyse ist genauso ein Bestandteil des Prozesses wie Entwicklung und Qualitätssicherung. Bisher ist in der ITU die Analyse gesondert behandelt worden, die außerhalb des Prozesses agiert, da sie sich nicht mit Scrum-Sprints vereinbaren lässt aufgrund großer Kundenabhängigkeit.

So wurde im vorliegenden Projekt in der ITU mit sehr umfangreichen Spezifikationspaketen gearbeitet. Man kann die Analyse besser in den Prozess einbinden, wenn man die Spezifikation in kleinere Teile unterteilt, beispielsweise auf Requirements- oder Featureebene [Lada08].

Man kann den Spezifikationsfortschritt beispielsweise in *„Spezifikation in Arbeit (in Wochen)“* und *„Prozentsatz fertig gestellter Spezifikationen"* messen. Je größer der erste Wert ist, das heißt je größer das Spezifikationspaket, desto kleiner ist voraussichtlich der zweite Wert [Lada08].

„For organizations that carry months' worth of specs at a time, that second number [Anm.: 'Prozentsatz fertig gestellter Spezifikation'] can quickly converge to zero. So don't do that! If you're carrying more than a few weeks worth of detailed specifications at a time, ask yourself... why? What are you going to do with them? Specification inventory is a liability just like any other kind of inventory.“ [Lada08]

Dies könnte auch der Grund sein, warum es im vorliegenden Projekt in der ITU so große Probleme gegen Ende des Projektverlaufs hin gegeben hat und im Endeffekt zu Kanban gewechselt wurde. Die externen Blockierungen auf Kundenseite haben sich gegen Ende des Projekts hin vervielfacht und fallen bei großen Spezifikationspaketen viel mehr ins Gewicht, da im schlimmsten Fall der gesamte Workflow zum Stillstand kommt. Demnach bringen umfangreiche Spezifikationspakete auch ein viel größeres Projektrisiko mit sich als kleine, in der Größenordnung von Stunden bis Tagen (im Gegensatz zu Wochen und Monaten).

4.4.6 Aufwandsschätzung in Kanban

Dieses Kapitel befasst sich mit den konsolidierten Problemen *„Schätzung und Planung ist schwierig“*, *„Kontrolle des Projektfortschritts nicht ausreichend gegeben“*, basiert auf [KnSk10] und beinhaltet den Lösungsvorschlag des Mitarbeiters der Qualitätssicherung „QS“ *(„Auch in Kanban ausreichend schätzen“*).

[KnSk10] beschreibt die Möglichkeit einer Aufwandsschätzung in Kanban. Zu ihrer Realisierung ist es jedoch überaus wichtig, WiP-Limits richtig einzusetzen und anzupassen (vgl. Kapitel 4.4.2) und daher auch eine funktionierende Feedback-Loop zu implementieren (vgl. Kapitel 4.4.3).

In so einem Fall kann nicht nur die durchschnittliche Cycle Time eines Items bestimmt werden, sondern auch ähnlich dem Scrum-Sprint die „Velocity“ (engl.: Geschwindigkeit im Sinne von Durchsatz) an Items. Diese „Velocity“ kann in Items pro Zeiteinheit gemessen werden, also beispielsweise fertig gestellte Items pro Woche. Durch diesen Durchschnittswert lässt sich daher auch in Kanban eine Schätzung einführen und die Planung erleichtern.

Es kann auch in Kanban Planungspoker durchgeführt werden, um neue Tasks gemeinsam zu schätzen. Häufig kommt es jedoch vor, dass das Wissen im Team zu ungleich verteilt ist, eine Person die Führung übernimmt und/oder Team Mitglieder wenig Interesse an Planung haben, sondern möglichst schnell zur Bearbeitung ihrer Tasks zurückkehren wollen. [KnSk10] beschreibt daher zwei Möglichkeiten, mit denen Planungspoker verbessert werden kann:

Shifting and Reviewing

1. Jeder User Story soll ein Senior Mitglied (d.h. eine Person die sich mit dieser Story gut auskennt) und ein Junior Mitglied zugewiesen werden. Dieses Paar soll die User Story schätzen. Die Aufteilung in Senior/Junior Paare hilft das Wissen zu verteilen.
2. Die restlichen Team Mitglieder entscheiden, welche User Story sie mitschätzen wollen, wobei maximal 4 Leute eine User Story schätzen sollen.
3. Jedes Schätzungsteam unterteilt die User Story in Tasks und schätzt diese gemeinsam als Team.
4. Dann tauschen jeweils zwei Teams Stories aus und überprüfen die Ergebnisse des jeweilig anderen Teams, wobei jeweils ein Mitglied des ursprünglichen Teams zurückbleibt, um die Taskaufteilung zu erklären.

Das Planungsmeeting soll ungefähr 45 Minuten dauern und durch die Abwechslung des *„Shifting and Reviewing“* sollte die Konzentration auf einem höheren Niveau gehalten werden können als bei regulärem Planungspoker.

Senior High Level Pass, then Estimation

Für diese Herangehensweise besprechen erst zwei Senior Mitglieder die User Stories und deren mögliche architekturelle Lösungen. Als Ergebnis sollten sich die zwei Senior Mitglieder bereits vor dem Planungspoker für jede User Story auf eine grobe architekturelle Lösung geeinigt haben. Danach wird mit dem gesamten Team ein reguläres Planungspoker durchgeführt, das die erarbeiteten Lösungen der Senior Mitglieder als Startpunkt nimmt. Diese Vorgehensweise soll ebenfalls helfen das Wissen besser zu verteilen.

Welche Metriken gibt es in Kanban?

In Kanban lässt sich mehr als nur die Cycle Time messen. [KnSk10] beschreibt eine Tabelle von möglichen Metriken für Kanban. Es soll gemeinsam mit der ITU besprochen beziehungsweise praktisch erprobt werden, welche Metriken sich am besten eignen:

Metrik Kandidat	Vorteil	Nachteil
Cycle Time (Lead Time)	Leicht zu messen. Keine Schätzung nötig. Startet und endet bei Kunde.	Keine Aussagen über das Volumen.
Total Velocity (aggregiert über alle Work Types)	Ein grober, aber simpler Indikator für den Grad der Verbesserung.	Keine Aussagen über Delivery Dates.
Velocity pro Work Type[6]	Präziser als „Total Velocity“.	Muss gemessen werden von Kunde bis zur Delivery, um nützlich zu sein. Aufwendiger zu messen als Total Velocity.
Länge der Queues[7]	Indikator für Fluktuation der Nachfrage. Leicht visualisierbar.	Keine Aussage darüber, ob Schwierigkeiten mit Nachfrage oder Kapazität. Eine Queue von 0 könnte auf Überkapazität hinweisen.

Tabelle 12: Metriken für Kanban (vgl. [KnSk10])

4.4.7 Burndown-Chart und Cumulative Flow Diagram

Dieses Kapitel befasst sich mit den Problemen *„Review fehlt“*, *„Erfolgserlebnis des Sprint-Abschlusses fehlt“*, dem konsolidierten Problem *„Schätzung und Planung ist schwierig“*, basiert auf [KnSk10], [Lada08] und beinhaltet den Lösungsvorschlag des Leiters der Qualitätssicherung „LQS“ *(„Ein Burndown-Chart oder vergleichbare Metriken wären in Kanban wünschenswert“)* sowie von Entwickler „EW2“ *(„Mehr darauf achten die Lead Time zu verringern und das Reporting diesbezüglich zu vereinfachen“)*.

[6] „Work Types“ können zum Beispiel sein: Release, Support, Unplanned, Project A, Project B, ...

[7] Als „Queue“ werden Spalten im Board bezeichnet, die nicht wertschöpfend sind und Wartezeit visualisieren. Demnach sollten Queues gering gehalten werden.

Burndown-Chart
In Kanban sind keinerlei Charts vorgeschrieben, jedoch auch keine verboten. Demnach kann unter anderem auch der aus Scrum bekannte Burndown-Chart (vgl. Kapitel 2.3.5) eingesetzt werden [KnSk10].

Vorstellbar wäre zum Beispiel die Einführung des Burndown-Charts basierend auf dem Release Zyklus, da es keine Sprints gibt. Dies könnte helfen, ein gewisses Review und Feedback zu erzeugen und zu visualisieren, und damit auch das Erfolgserlebnis zu erhöhen. Der große Unterschied zum Scrum-Burndown ist jedoch, dass es kein Commitment für diesen Burndown-Chart gibt.

Eine andere Möglichkeit wäre die in Kapitel 4.4.10 vorgeschlagene Einführung eines „Kanban-Sprints". Diese im Rhythmus von ein bis zwei Wochen künstlich eingeführte Iteration hätte folgenden Vorteil:

Es kann ein Burndown-Chart erstellt werden, jedoch im Gegensatz zu Scrum setzt sich der geschätzte Aufwand des Charts für die nächste Iteration nicht aus den Tasks zusammen, sondern aus den Durchschnittswerten der letzten „Kanban-Sprints".

Auf diese Weise kann mit dem „imaginären" Burndown-Verlauf eine grobe Abweichung zu den letzten Wochen im Verlauf eines „Kanban-Sprints" festgestellt werden.

Cumulative Flow Diagram
Ein Beispiel für das Cumulative Flow Diagram findet sich in Abbildung 15. [Lada08] schreibt Folgendes über die das Verhältnis von Burndown-Chart und Cumulative Flow Diagram:

„(...) we will start to become less concerned with burndown and more concerned with cycle time, as burndown is the effect and cycle time is the cause. (...) If cycle time is under control, then burndowns are predictable and uninteresting."[Lada08]

[Lada08] argumentiert, dass Burndown-Charts uninteressant werden, sobald die Cycle Time unter Kontrolle ist, da das Cumulative Flow Diagram über die aktuelle Situation besseren Aufschluss gibt. Es soll daher mit der ITU gemeinsam besprochen werden, welche Diagramm-Art zum Einsatz kommen soll.

4.4.8 Board-orientiertes Standup Meeting

Dieses Kapitel befasst sich mit den Problemen *„Kanban-Board war nicht allgegenwärtig"*, *„Kein größeres Bild bei der Bearbeitung des Tasks mehr vorhanden"* und *„Verständigung in Standup-Meetings schwieriger"*, basiert auf [KnSk10] und beinhaltet den

Lösungsvorschlag von Entwickler „EW2“ *(„Scrum-Werkzeuge wie das Daily Meeting können auch in Kanban nützlich sein“*).

Das Standup Meeting hat sich grundsätzlich in der ITU als sehr nützlich bewiesen und soll auf jeden Fall auch in Kanban beibehalten werden. Jedoch wurde in den Interviews bemängelt, dass die Verständigung schwieriger gewesen sei als in Scrum. [KnSk10] berichtet, dass in Kanban viele Teams ein Board-orientiertes Standup Meeting eingeführt haben. Dies könnte auch in der ITU für mehr Klarheit sorgen.

Die Vorteile sind, dass das Standup Meeting sich mehr auf Flaschenhälse und andere sichtbare Probleme richtet. Es ist demnach nicht unbedingt nötig, dass jede Person im Standup Meeting seinen Fortschritt berichtet, was von Vorteil sein kann, wenn sich mehrere Teams ein Board teilen, wie es in der ITU zwischen Entwicklung und Qualitätssicherung der Fall ist.

[Ande10] beschreibt weiters große Vorteile von physischen Boards gegenüber elektronischen Ticket Management Systemen:

„The physical board had a huge psychological effect compared to anything we got from the electronic tracking tool we used at Microsoft. By attending the standup each day, team members were exposed to a sort of time-lapse photography of the flow of work across the board. (...) the team became much more focused on issue resolution and maintaining flow. Productivity jumped dramatically.“ [Ande10]

[Ande10] argumentiert wiederum ebenfalls, dass elektronische Ticket Management Systeme für Reporting und Metriken sehr nützlich sind. Beide Möglichkeiten haben erhebliche Vor- und Nachteile, beides parallel zu betreiben, bedeutet Mehraufwand. Die endgültige Entscheidung im Zuge dieses Konzept soll gemeinsam mit der ITU besprochen werden.

4.4.9 „Team Impediments“-Liste für Product Owner

Dieses Kapitel befasst sich mit dem Problem *„Flaschenhälse konnten nicht immer beseitigt werden“*.

[KnSk10] beschreibt im Zuge eines Praktikbeispiels die Einführung einer *„Team Impediments“*-Liste (engl.: Team Hindernisse) für den Manager, den in unserem Fall der Product Owner darstellt. Ziel ist es dabei auch den Product Owner in die WiP-Limits einzubeziehen. Beispielsweise kann das Limit eingeführt werden, dass das Team dem Product Owner maximal zwei „Team Impediments“ gleichzeitig zuweisen kann, die dieser daraufhin zu lösen versucht.

[KnSk10] beschreibt die Regeln für die *„Team Impediments“*-Liste, wie folgt:

1. Der Product Owner kann jederzeit an zwei Hindernissen arbeiten.
2. Wenn beide Slots voll sind, kann das Team nur ein neues Hindernis hinzufügen, wenn ein weniger wichtiges entfernt wird.
3. Das Team entscheidet, wann ein Hindernis als gelöst gilt.

Die Einführung dieser Liste hat den Vorteil, dass aufgetretene Probleme nicht übersehen werden können, sondern für beide Seiten visualisiert werden. Weiters dreht sich die Verantwortlichkeit hier um: Der Product Owner verpflichtet sich die Hindernisse zu beseitigen.

4.4.10 Kanban-Sprint einführen

Dieses Kapitel befasst sich mit den konsolidierten Problemen *„Schätzung und Planung ist schwierig“*, *„Kontrolle des Projektfortschritts nicht ausreichend gegeben“*, *„Kanban: Gefahr ‚zu agil’ zu werden“* und *„Schwierigkeiten mit WiP Limits“*.

Auch wenn in Kanban grundsätzlich alles einem Workflow folgen sollte ohne Unterbrechung, so ist dies auch der Hauptpunkt, der von vielen Mitarbeitern der ITU bemängelt wurde. Es werden dadurch Schätzung, Planung und Kontrolle des Projektfortschritts außerordentlich erschwert. Auch das Erfolgserlebnis für die Mitarbeiter fällt weg und Kanban bietet keinen Raum über den Prozess zu reflektieren.

Viele dieser Probleme wurden bereits in den vorangehenden Kapiteln adressiert. Um diese Lösungen jedoch auch in die Tat umsetzen zu können, muss es dafür ein vorgesehenes Zeitfenster geben. Im Rahmen des Konzepts soll daher ein 1- oder 2-wöchiger „Kanban-Sprint“ eingeführt werden, für den es jedoch keinerlei Commitment und keine Planung gibt.

Dieser Kanban-Sprint kann folgende Vorteile bieten:

- Ermöglichung der Einführung von Burndown-Charts in Kanban zur besseren Fortschrittskontrolle (vgl. Kapitel 4.4.7)
- Festgesetzter, periodischer Zeitpunkt für Retrospektive als äußerste Feedback Loop und im Zuge dessen Anpassung der „WiP“-Limits (vgl. Kapitel 4.4.2, 4.4.3 und 4.4.4)

Es gab in der ITU bereits in Kanban ein wöchentliches Kanban Priorisierungsmeeting. Dies soll aber neben Priorisierungsupdates nun auch folgende Punkte beinhalten [KnSk10]:

- Aktualisierung von Boards und Charts (z.B. Burndown-Chart)
- Rückblick über die vergangene Woche (entspricht Retrospektive)
- WiP Limit anpassen, wenn nötig (als äußerster Teil der Feedback Loop)

- Neues Projekt in Tasks aufteilen und schätzen, wenn nötig

Es ist also nicht nur wichtig, die kommende Woche zu planen, sondern auch die vorangehende zu besprechen, um den Prozess weiter verbessern zu können in Bezug auf Planung und Schätzung.

Weiters ist der „Kanban-Sprint“ auch ein guter Anstoßpunkt für Product Owner, sich mit dem Backlog auseinanderzusetzen.

[Lada08] besagt bezüglich Iterationen weiters, dass es keinerlei Grund gibt in einem funktionierenden Kanban System die Planungs- und Releasezeitspanne gleichsetzen müssen. Bezüglich der Planung in Kanban ist ebenfalls Kapitel 4.4.14 von großer Wichtigkeit.

4.4.11 Limit für „To Do“-Spalte stets gering halten

Dieses Kapitel befasst sich mit dem Problem *„Limit für ‚To Do’Spalte zu groß“*, *„ Einführung der ‚Backlog’-Spalte hat ‚To Do’ Spalte nicht entlastet“* und basiert auf [Lada08].

Die WiP-Limits für die „To Do“-Spalte waren bisher in der ITU zu hoch. Diese Spalte sollte nur hoch priorisierte Items des Backlogs enthalten, die unmittelbar bearbeitet werden sollen. In der ITU war diese Spalte jedoch häufig überfüllt, was am Sinn der „To Do“-Spalte vorbeiführt:

„Die ‚To Do’-Spalte hat als maximale Anzahl 40 Issues, fast ein bisschen zu groß. Also man hat schon einen Überblick, aber bei 40 Issues... die, die unten stehen, bleiben dann meistens auch unten.“

(Zitat aus Interview mit Scrum Master „SM“)

Eine Backlog-Spalte und eine „To Do“-Spalte zu haben, erleichtert es die Arbeitszuweisung und Priorisierung zu trennen, wobei die „To Do“-Spalte ein möglichst geringes Limit haben sollte:

„This enables us to decouple the process of assigning work from the process of prioritizing work, and it simplifies assignment. The ready queue [Anm.: „To Do“-Spalte] also has a kanban limit, and it should be a small limit, since its only purpose is to indicate which work item should be started next.“ [Lada08]

4.4.12 Umgang mit externen Blockierungen

Dieses Kapitel befasst sich mit dem Problem *„Einführung der ‚Blocked’-Spalte hat ‚To Do’- bzw. ‚In Progress’-Spalte nicht entlastet“* und basiert auf [KnSk10].

Laut [KnSk10] ist die Einführung einer eigenen Spalte, abseits des regulären Workflows, nicht sinnvoll, da diese Items häufig übersehen werden. [KnSk10] bringt als Beispiel eine „Overflow"-Spalte, in die Items gesetzt wurden, die das WiP-Limit überschritten haben.

ITU hat ähnlich diesem Prinzip eine „Blocked"-Spalte für externe Blockierungen eingeführt, da es im vorliegenden Projekt sehr viele externe Blockierungen von Kundenseite her gab. Diese Spalte ist jedoch mit Vorsicht zu gebrauchen, da sie tendenziell am Sinn von Kanban vorbeiführt, nämlich das Aufzeigen von Flaschenhälsen durch Erreichen des WiP-Limits.

Der Lösungsvorschlag im Zuge dieses Konzepts ist es daher eine „Blocked"-Spalte nach Möglichkeit zu vermeiden und diese keinesfalls beim Umstieg auf Kanban einzuplanen. Weiters sollte diese Spalte unter keinen Umständen für interne Blockierungen genutzt werden, da ansonsten der Einsatz von Kanban ad absurdum geführt wird.

4.4.13 Concurrency in Kanban

Dieses Kapitel befasst sich mit dem Problem *„Gemeinsame WiP Limits für Entwicklung und Qualitätssicherung nicht zielführend"*, *„Abhängigkeiten zwischen Entwicklungs- und Qualitätssicherungstasks am Kanban-Board schwer erkenntlich"*, basiert auf [Ande10] und auf dem Lösungsvorschlag des Leiters der Qualitätssicherung „LQS" *(„Die Qualitätssicherung braucht eigene WiP-Limits, da das Board gemeinsam benutzt wird"*).

[Ande10] schlägt eine Aufteilung der „Development"-Spalte vor, jedoch in vertikaler Weise (vgl. Abbildung 19), da um Requirements oder Features testen zu können, auch Testfälle für diese erstellt werden müssen, das heißt der „In Progress"-Teil der „Development"-Spalte wird vertikal unterteilt in *„Development In Progress"* und *„Test Development in Progress"*. Für jede dieser beiden Unterteilungen gibt es ein eigenes WiP-Limit. In die „Done"-Spalte sollte dann schließlich ein Feature oder Requirement stehen, das sowohl implementiert ist als auch erstelle Testfälle hat. Erst dann ist es bereit für die Testexekution. Dieser Punkt wird sehr gut durch das Board visualisiert.

Der Workflow ist daher:

1. Ein Requirement oder Feature wird spezifiziert in der „Analyse"-Spalte.
2. Das Requirement oder Feature wird aufgeteilt in einen Implementierungstask und einen Task für die Erstellung der Testfälle.
3. In der „Done"-Spalte werden die beiden Tasks wieder zu dem überfassenden Requirement oder Feature zusammengefügt. Das Requirement ist demnach bereit für die Testphase.

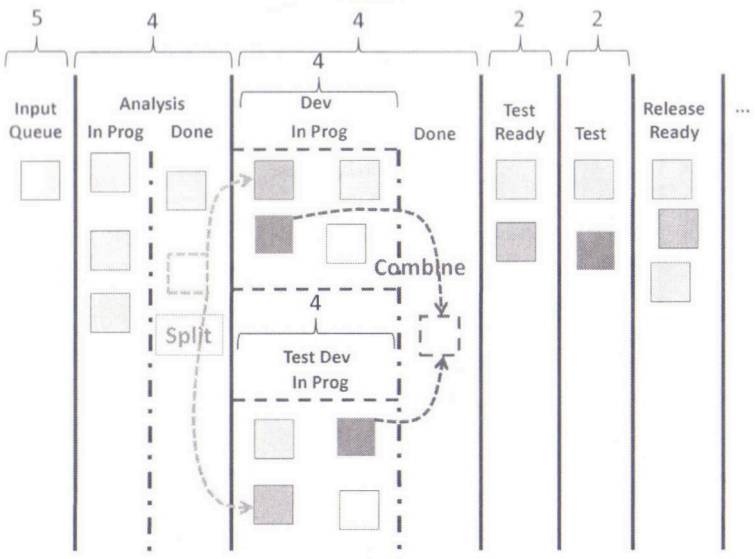


Abbildung 19: Gleichzeitige Aktivitäten innerhalb einer Spalte im Kanban-Board [Ande10]

Durch die vertikale Aufteilung und die WiP-Limits wird daher dafür gesorgt, dass Requirements für ein Release zeitnah implementiert und getestet werden und nicht eine der beiden Seiten stark hinterher ist. Weiters wird damit das Problem gelöst, dass die WiP-Limits in der „In Progress"-Spalte nichtssagend waren, da sie für Entwicklung und Qualitätssicherung gleichermaßen galten im vorangehenden Projekt der ITU.

Dadurch wird auch die Abhängigkeit zwischen Entwicklung und Qualitätssicherung deutlicher, was in den Interviews bemängelt wurde.

4.4.14 Backlog in Kanban richtig benutzen

Dieses Kapitel befasst sich mit dem Problem *„Weniger Notwendigkeit sich mit dem Backlog auseinander zu setzen"* und beruht auf [Lada08].
In Kanban geht es darum, den Workflow von „Waste" (engl.: Abfall) zu befreien. Daher gilt jede Tätigkeit als überflüssig, die nicht unmittelbar der Bereinigung oder Aufrechterhaltung eines funktionierenden Workflows dient.

Dazu zählt auch die Planung. Das Scrum-Planungsmeeting erzeugt in der Regel ein viel größeres Backlog als nötig ist. Entsprechend den Kanban-Regeln ist das ein höherer Inventarstand als erforderlich und daher „Waste". [Lada08] beschreibt den idealen Planungsprozess wie folgt:

„The ideal work planning process should always provide the development team with best thing to work on next, no more and no less. Further planning beyond this does not add value and is therefore waste." [Lada08]

[Lada08] schlägt das Konzept eines „Order Points“ für den Zeitpunkt der Planung in Kanban. Der „Order Point“ ist eine bestimmte Anzahl an Items im Backlog. Wird dieser Punkt unterschritten, so ist das der Auslöser neu zu planen. Mit Hilfe des „Order Points“ ist es möglich, gerade so viele Items bereit zu stellen, sodass das Entwicklungsteam an den wichtigsten Items weiterarbeiten kann, nicht mehr und nicht weniger.

Wird beispielsweise im Durchschnitt ein Item pro Tag abgeschlossen und die derzeitige Backloggröße ist zehn, so wird bei einem „Order Point“ von fünf ungefähr einmal die Woche ein Planungsmeeting stattfinden [Lada08].

4.4.15 Kanban Release Management

[Lada08] geht in Bezug auf das vorangehende Kapitel und das Konzept des „Order Points“ noch einen Schritt weiter:

„The same kind of logic can be applied to the release interval. There is an optimum batch size for releases and we should first try to find it, and then try to improve it. The result of our efforts will ultimately be features-on-demand.“[Lada08]

Das heißt: so wie der Backlog der Input für den Workflow ist und das Konzept des „Order Points“ angewandt werden kann, so ist das Release der Output des Workflows, das in entgegen gesetzter Weise zum Backlog erstellt werden kann, wenn ein bestimmter „Release Point“ (Feature Limit) erreicht ist, beispielsweise entsprechend eines wahrhaftigen Pull-Systems mit minimalem „Waste“.

Wie oft ein Release erstellt werden sollte, hängt auch mit den Kosten zusammen ein Release zu erstellen. Wenn die Releasekosten hoch sind, sollte weniger oft ein Release erstellt werden als wenn die Kosten niedrig sind. Daher sollte versucht werden das Releaseintervall so zu wählen, dass die Releasekosten gering gehalten werden können [Lada08].

4.4.16 Priorisieren in (und mit) Kanban

Dieses Kapitel befasst sich mit dem Problem *„Weniger Notwendigkeit sich mit dem Backlog auseinander zu setzen“* und beruht auf [Lada08].

Der Aussage von Kapitel 4.4.14 folgend, nämlich die Backlog-Größe gering zu halten, stellt [Lada08] eine Priorisierungsmethode vor, die wiederum selbst ein Kanban-Board benutzt, um Prioritäten zu finden.

Gründe für die Benutzung der Methode sind folgende zwei Beobachtungen:

1. *„If everything is high priority, then nothing is.“ [Lada08]*
2. *„Why should we care what the 37th work item from now will be, when we only need to know what the next on is?“ [Lada08]*

Als Lösung hierfür möchte man also ein geringes Set an Features haben, die effektiv priorisiert sind, das heißt nur eine geringe Anzahl von hohen Priorisierungen aufweisen. Um dies zu erreichen, kann folgende Methode eingesetzt werden von [Lada08] eingesetzt werden:

Progressive Priority Filtering
Hauptbestandteil dieser Methode ist ein Kanban-Taskplanungsboard mit folgenden Spalten (in Klammern befinden sich die WiP-Limits):

„Backlog“ → „Priorität 3“ (8) → „Priorität 2“ (4) → „Priorität 1“ (2) → „Done“

Durch dieses Board wird man gut visualisiert dazu angehalten, effektiv zu priorisieren, das heißt nur eine geringe Anzahl ist sehr hoch priorisiert („Priorität 1“), sowie auch ein überschaubares Backlog zu haben (in diesem Beispiel 8+4+2 = 14) und vermeidet damit die zwei größten Fehler beim Priorisieren.

Die „Backlog“-Spalte hier enthält Tasks, die abgearbeitet werden sollten, jedoch noch keine Priorität haben. Grundsätzlich darf jedes Item, das noch nicht in Bearbeitung ist, jederzeit ausgetauscht werden, sei es mit einem Item des Backlogs und mit einer anderen Priorität, jedoch sollten stets die WiP-Limits eingehalten werden. Das Limit für „Priorität 1“ muss genau der Arbeitskapazität von einem Tag entsprechen (bzw. des minimalen Planungsintervalls). Prioritäten 2 und 3 sollten sich entsprechend einer geometrischen Reihe erhöhen (z.B. 2, 4, 8) oder laut Fibonacci (z.B. 3, 5, 8), um die Unterschiede zwischen den Prioritätsniveaus zu verdeutlichen [Lada08].

Rückt ein fertig gestelltes Item in die „Done“-Spalte, sendet dies naturgemäß einen Planungsimpuls, da Kapazität frei geworden ist.

[Lada08] schlägt weiters vor, Tickets mit einem Ablaufdatum zu versehen, um zu verhindern, dass einige Tickets auf unbestimmte Zeit in „Priorität 3“ verweilen, und bei der Priorisierungsentscheidung sollten ältere Tickets bevorzugt werden oder entsprechend überdacht werden.

Auch das Team hat auf diese Weise einen guten Einblick in die Priorisierungsfindung.

4.4.17 Bug Handling

[Lada08] beschreibt zwei Optionen für Bug Handling in Kanban, die im Folgenden erläutert werden. Gegeben sei das Szenario, dass für ein Requirement oder Feature in der Testphase (Kanban-Board Spalte „Test“) ein Bug gefunden wird.

Reinsert defective WiP Upstream
In dieser Option wird bei Auftreten eines Bugs das Requirement bzw. Feature zurück in den Development-Prozess gesetzt, wo es wie jedes andere Requirement behandelt wird. Die Qualitätssicherung hat daher die Möglichkeit, das nächste Requirement entsprechend den WiP Limits zu testen.

Dies ist eine sanftere Herangehensweise und stört den Workflow nicht. Der Nachteil ist jedoch, dass der Bug wie ein reguläres Work Item behandelt wird, das heißt mitunter auf seine Bearbeitung warten muss, bis es das WiP Limit des Entwicklungsteams erlaubt. Das Auftreten eines Bugs wird hier als bekannte Variation des Workflows gesehen.

Shared Rework Station
In dieser Option bleibt das Requirement in der Test-Spalte und ein neues „Rework“-Item wird in die Entwickler-Spalte gesetzt. Die Tester arbeiten weiter an diesem Requirement oder anderen entsprechend ihrem WiP-Limit. Dies hat jedoch zum Ergebnis, dass die Tester leicht blockiert werden, da sie den Test des Requirements nicht (erfolgreich) abschließen können, bis der Bug behoben ist. Das „Rework“-Item zählt nicht für das WiP-Limit des Entwicklungsteams, da sich das Entwicklungsteam verpflichtet Kapazitäten frei zu geben, um den Bug umgehend zu beheben. Dies hat zur Folge dass die Kapazität des Entwicklungsteams insgesamt reduziert wird, da immer wieder Bugs den Vorzug gegenüber regulären Items gegeben werden muss.

Dem Bug wird hier erhöhte Wichtigkeit zugesprochen und er wird als Prozess-Fehler gesehen, der unmittelbare Aufmerksamkeit benötigt, da sonst das System Gefahr läuft zum Stillstand zu kommen. Diese Konfiguration kann sich gut für ein Projekt oder Team mit akuten Qualitätsproblemen eignen. Sobald dieses Problem jedoch unter Kontrolle ist, ist die erste Option zu bevorzugen.

4.4.18 Einführung von Queues

Da die Arbeitsschritte nicht fließend ineinander übergehen können, soll ein Buffer (auch als „Queue“ bezeichnet) eingefügt werden, wie von [Lada08] vorgeschlagen, um die Wartezeiten zwischen den Arbeitsschritten zu visualisieren. So soll beispielsweise aus

„Analyse“ → „Entwicklung“ → „Test“

„Analyse“ → „Queue“**→ „Entwicklung“→** „Queue“**→ „Test“**

werden, wobei die „Queues“ genauso ein Teil des Workflows sind wie die Kernarbeitsschritte Analyse, Entwicklung und Test. Die Queues visualisieren daher die Wartezeit eines Items zwischen den Bearbeitungsschritten. Limits für Queues unterscheiden sich von denen jener Arbeitsschritte, die wertschöpfend sind. Da die Queues eine Wartezeit visualisieren, sollen die Limits hierfür grundsätzlich gering sein, jedoch bei zu geringen Limits wird der Workflow ebenso ins Stocken geraten wie bei zu großen. Auch hier muss mit Limits experimentiert werden.

Besser ist jedoch den Buffer unter dasselbe WiP Limit miteinzubeziehen, beispielsweise „Analyse“-Spalte mit einem WiP-Limit von 4 unterteilt in „In Progress“ und „Done“. Der Vorteil liegt laut [Lada08] darin, dass Probleme im Workflow früher erkannt werden. Die erste Variante mit eigenen Queues verhält sich folgendermaßen, wenn ein Problem auftritt:

„If there is some kind of blockage downstream, first the completion queue will fill up, THEN the specify queue will fill up, THEN the specify process will stall. And when it stalls, it stalls all at once.“ [Lada08]

Die zweite Variante mit geteilten WiP-Limits hingegen ermöglicht eine schnellere Reaktion:

„The difference is that it begins stalling immediately, and when it does stall, it stalls with less WiP. For equivalent throughput, this pipeline should operate with fewer kanban and less variation in WiP, and therefore should have smoother flow and shorter lead times.“ [Lada08]

Dementsprechend ist die zweite Variante zu bevorzugen, da Probleme durch das gemeinsame WiP-Limit schneller bemerkt werden und damit die Reaktionszeit verkürzt wird.

4.4.19 Filter für „Closed“-Spalte

Dieses Kapitel basiert auf dem Vorschlag des Leiters der Qualitätssicherung „LQS“ *(„Möglichkeit eine Übersicht zu bekommen in Kanban wäre ein Filter auf das ‚Closed’-Datum von Tickets“*). Die „Closed“-Spalte bezeichnet die letzte Spalte des Workflows am Kanban-Board. Im vorgeschlagenen Kanban-Board (vgl. Kapitel 4.4.20) heißt diese Spalte „Production“.

Aus den Interviews ist mehrmals hervorgegangen, dass die „Closed“-Spalte in Kanban mit der Zeit überfüllt wird. Da es keine Iterationen gibt, wird diese Spalte auch niemals geleert, was die Übersicht verschlechtert. Abhilfe könnte der Vorschlag des Leiters der Qualitätssicherung „LQS“ sein, einen Filter auf das Datum bereitzustellen, zu dem das Ticket geschlossen wurde. Mögliche Intervalle wären zum Beispiel „x Wochen“ oder „seit dem letzten Release“.

4.4.20 Unterteilung des Kanban-Boards

Dieses Kapitel befasst sich mit der vorgeschlagenen Unterteilung beziehungsweise Zusammensetzung des Kanban-Boards. Es ist der letzte (auf Kanban bezogene) Verbesserungsvorschlag des Konzepts, da er viele Paradigmen miteinbezieht, die in den vorangehenden Kapiteln behandelt wurden.

Es folgt in Tabelle 13 der Vorschlag für die Unterteilung des Kanban-Boards entsprechend den Erkenntnissen der vorangehenden Kapitel. Es wurde überall, wo es sinnvoll erscheint, die Aufteilung in Spalten „In Prog“ (In Progress) und „Done“ vorgenommen, die sich das WiP-Limit teilen und damit Probleme im Workflow frühzeitig visualisieren (vgl. Kapitel 4.4.18).

<table>
<tr><th>Backlog</th><th>P3</th><th>P2</th><th>P1</th><th colspan="2">Analyse</th><th colspan="2">Development</th><th>Test</th><th>Stage</th><th>Production</th></tr>
<tr><td rowspan="2"></td><td rowspan="2"></td><td rowspan="2"></td><td rowspan="2"></td><td rowspan="2">In Prog</td><td rowspan="2">Done</td><td>Dev in Prog</td><td rowspan="2">Done</td><td rowspan="2"></td><td rowspan="2"></td><td rowspan="2"></td></tr>
<tr><td>Test Dev in Prog</td></tr>
</table>

Tabelle 13: Aufteilung des Kanban-Boards

Backlog/P3/P2/P1: Eingeführt entsprechend Kapitel 4.4.16 für zeitnahe, nachvollziehbare und effektive Priorisierung.

Analyse: Entsprechend Kapitel 4.4.5 in das Kanban-Board integriert, da Analyse genauso ein Teil des Workflows ist.

Development: Entsprechend Kapitel 4.4.13 auch vertikal unterteilt, um die Gleichzeitigkeit zwischen Entwicklung und Testerstellung für ein Feature oder Requirement zu forcieren und visualisieren.

Test: Eingeführt um die Qualitätssicherung besser als Teil des Workflows zu integrieren und visualisieren (vgl. Kapitel 4.4.13). Dieser Vorschlag wurde bereits vorab im Interview mit dem Leiter der Qualitätssicherung „LQS“ als unter Umständen sinnvoller Schritt verifiziert.

Stage / Production: Stage enthält alle Items, die bereit sind in das Release aufgenommen zu werden, und Production die tatsächlich bereits im Release enthaltenen Items. Production erweitert damit die frühere „Closed“-Spalte.

4.4.21 Wartung von Tasks im Ticket Management System

Dieses Kapitel basiert auf dem Vorschlag eines Mitarbeiters der Qualitätssicherung „QS“ *(„Ist ein Development-Task blockiert, so sollten auch davon abhängende Qualitätssicherungstasks auf „Blocked“ gesetzt werden“*). Dieses Problem hat generelle Relevanz und ist nicht direkt auf Kanban bezogen, aber da es aus den Praxisberichten der ITU kommt, soll es an dieser Stelle erwähnt werden, da es Verbesserungspotential für den vorliegenden Prozess bietet. Er wird jedoch nicht in das finale Konzept mit einfließen.

Aus den Interviews ist hervorgegangen, dass die Tasks der Entwickler gut gewartet waren, jedoch bei den Tasks der Qualitätssicherung Verbesserungspotential in Bezug auf Blockierungen besteht. Im speziellen sollten die Auswirkungen von blockierten Entwicklungstasks auf die Qualitätssicherung besser kommuniziert bzw. im Ticket Management System vermerkt werden. Eventuell ist es nötig hier Verantwortlichkeiten zu vergeben, beziehungsweise diese zu klären. Bis auf diesen Punkt gab es in den Interviews jedoch keinerlei angesprochene Mängel bezüglich der Wartung von Tickets.

4.4.22 Nicht einbezogene Lösungsvorschläge

Es wurde nur einer der Lösungsvorschläge aus den Interviews nicht ins Konzept einbezogen: „*Es besteht die Möglichkeit, dass nur die Qualitätssicherung auf Scrum zurück wechselt, sofern sich die Situation für sie schneller beruhigt“*. (aus Interview mit Leiter der Qualitätssicherung „LQS“, vgl. Tabelle 10).

Der Grund hierfür ist, dass mit diesem Konzept versucht wird, die Qualitätssicherung noch enger in den allgemeinen Entwicklungsprozess zu inkludieren (vgl. beispielsweise Kapitel 4.4.13 und 4.4.20). Wenn die Qualitätssicherung, wie vorgeschlagen, wieder alleine zu Scrum zurückkehren würde, so würde das insgesamt mehr Prozessaufwand ohne ersichtlichen Mehrwert bringen. Daher ist auch dieses Kapitel im Grunde genommen ein weiterer Vorschlag und Teil des Konzepts, nämlich die Qualitätssicherung möglichst eng in den Prozess integrieren und daher ebenfalls auf Kanban umsteigen zu lassen. Ansonsten verliert das Kanban-Board an Sinnhaftigkeit und Wert.

4.5 Probleme und Konzeptlösungen gegenübergestellt

In diesem Kapitel sollen überblicksmäßig anhand einer Tabelle die identifizierten Probleme den im Konzept gebrachten Lösungen gegenübergestellt werden, sowohl um die Übersichtlichkeit zu verbessern, als auch um zu überprüfen, ob alle konsolidierten Probleme und Teilprobleme dieser im Konzept behandelt wurden.

Konsolidiertes Problem	Teilproblem	Konzeptlösung
Scrum: Scrum funktioniert nur in ruhigem Projektumfeld	Scrum benötigt stets ausreichend gefülltes Backlog	Kapitel 4.1.1 Wechsel von Scrum auf Kanban
	Unfähigkeit auf Änderungen schnell zu reagieren	Kapitel 4.1.1 Wechsel von Scrum auf Kanban
	Scrum-Sprint erfordert gewisse Ruhe im Projektumfeld	Kapitel 4.1.1 Wechsel von Scrum auf Kanban Kapitel 4.4.4 Retrospektive einführen
Kanban: Schätzung und Planung ist schwierig	Weniger Vorausplanung	Kapitel 4.4.6 Aufwandsschätzung in Kanban Kapitel 4.4.7 Burndown-Chart / Cumulative Flow Diagram Kapitel 4.4.10 „Kanban-Sprint“ einführen
	Weniger Aufwandsschätzung	Kapitel 4.4.6 Aufwandsschätzung in Kanban Kapitel 4.4.7 Burndown-Chart / Cumulative Flow Diagram Kapitel 4.4.10 „Kanban-Sprint“ einführen
	Volumen schwer abschätzbar	Kapitel 4.4.6 Aufwandsschätzung in Kanban Kapitel 4.4.7 Burndown-Chart / Cumulative Flow Diagram Kapitel 4.4.10 „Kanban-Sprint“ einführen
	Weniger Notwendigkeit sich mit dem Backlog auseinander zu setzen	Kapitel 4.4.10 „Kanban-Sprint“ einführen Kapitel 4.4.14 Backlog in Kanban richtig benutzen Kapitel 4.4.16 Priorisieren in (und mit) Kanban
Kanban: Kontrolle des Projektfortschritts nicht ausreichend gegeben	Kanban-Board war nicht allgegenwärtig	Kapitel 4.4.8 Board-orientiertes Standup Meeting

	Kein größeres Bild bei der Bearbeitung des Tasks mehr vorhanden	Kapitel 4.4.3 Feedback Loop verbessern Kapitel 4.4.8 Board-orientiertes Standup Meeting
	Review fehlt	Kapitel 4.4.4 Retrospektive einführen Kapitel 4.4.7 Burndown-Chart / Cumulative Flow Diagram Kapitel 4.4.10 „Kanban-Sprint“ einführen
	Erfolgserlebnis des Sprint-Abschlusses fehlt	Kapitel 4.4.4 Retrospektive einführen Kapitel 4.4.7 Burndown-Chart / Cumulative Flow Diagram Kapitel 4.4.10 „Kanban-Sprint“ einführen
	Verständigung in Standup-Meetings schwieriger	Kapitel 4.4.8 Board-orientiertes Standup Meeting
Kanban: Gefahr „zu agil“ zu werden	Keine Retrospektive mehr	Kapitel 4.4.4 Retrospektive einführen Kapitel 4.4.10 „Kanban-Sprint“ einführen
	Kanban hat wenige Regeln, jedoch müssen diese eingehalten werden	Kapitel 4.4.3 Feedback Loop verbessern Kapitel 4.4.4 Retrospektive einführen
Kanban: Unstimmigkeiten zwischen Entwicklung und Qualitätssicherung	Abhängigkeiten zwischen Entwicklungs- und Qualitätssicherungstasks am Kanban-Board schwer erkenntlich	Kapitel 4.4.13 Concurrency in Kanban
	Gemeinsame WiP Limits für Entwicklung und Qualitätssicherung nicht zielführend	Kapitel 4.4.13 Concurrency in Kanban
Kanban: Schwierigkeiten mit WiP Limits	Flaschenhälse konnten nicht immer beseitigt werden	Kapitel 4.4.9 „Team Impediments“-Liste für Product Owner
	Limits schwierig anzupassen	Kapitel 4.4.2 Das „richtige“ WiP Limit finden
	Limit für „To Do“ Spalte zu groß	Kapitel 4.4.11 Limit für „To Do“-Spalte stets gering halten

	Einführung der „Backlog“-Spalte hat „To Do“ Spalte nicht entlastet	Kapitel 4.4.11 Limit für „To Do“-Spalte stets gering halten
	Einführung der „Blocked“-Spalte hat „To Do“- bzw. „In Progress“-Spalte nicht entlastet	Kapitel 4.4.12 Umgang mit externen Blockierungen

Tabelle 14: Gegenüberstellung Probleme und Konzeptlösung

5 Praktische Erprobung des Konzepts

Das im vorangehenden Kapitel entwickelte Konzept soll nun in der ITU in der Praxis erprobt werden. Dieser Praxisteil erfolgt in mehreren Schritten, die in Kapitel 5.1 näher erläutert werden. Als Abschluss des Praxisteils werden Erfahrungen mit dem Konzept in Kapitel 5.2 präsentiert, die als Grundlage für die abschließende Diskussion und Verbesserung des Konzepts in Kapitel 6 dienen.

5.1 Vorgehensweise

Die praktische Erprobung des Konzepts unterteilt sich in folgende Schritte:

1. Initiales Feedback der ITU auf das entwickelte Konzept
2. Auswahl der Verbesserungsvorschläge, die in der ITU umgesetzt werden
3. Praktische Erprobung der ausgewählten Verbesserungsvorschläge in der ITU
4. Erfassung (Kapitel 5.2) und Analyse (Kapitel 6.1) der praktischen Erfahrungen

Für die Durchführung der ersten beiden Schritte ist ein Zeitraum von 2 Wochen geplant. Anschließend folgt die praktische Erprobung über einen Zeitraum von knapp 2 Monaten, die in der Erfassung und Analyse der praktischen Erfahrungen ihren Abschluss findet. An dem Projekt, in dem das Konzept erprobt wird, sind knapp 10 Mitarbeiter beteiligt über die Laufzeit von einem Jahr. Es befindet sich derzeit in der Produktionsphase.

5.2 Feedback der ITU

Das Feedback der ITU bietet die Grundlage für die abschließende Verbesserung des Konzepts. Das Feedback unterteilt sich in drei Phasen:

- Initiales Feedback: Erste Eindrücke der ITU auf das Konzept
- Umzusetzende Verbesserungsvorschläge: Auswahl und Festlegung jener Verbesserungsvorschläge, die in der Praxis in der ITU erprobt werden sollen
- Praktische Erfahrungen: Erfassung der Erfahrungen am Ende der knapp zweimonatigen Praxisphase in der ITU

Die folgenden Unterkapitel 5.2.1, 5.2.2 und 5.2.3 beschreiben das erhaltene Feedback dieser drei Phasen.

5.2.1 Initiales Feedback

Dieses Kapitel beinhaltet das Feedback der beiden Geschäftsführer „GF1“ und „GF2“ der ITU, nachdem das Konzept zum ersten Mal präsentiert wurde. Jeder einzelne Vorschlag wurde behandelt und Unklarheiten beseitigt. Dieses initiale Feedback dient als Grundlage für die Auswahl umzusetzender Verbesserungsvorschläge (Kapitel 5.2.2). Tabelle 15 gibt die wichtigsten Argumente dieses Treffens wieder.

Vorschlag	Feedback „GF1“	Feedback „GF2“	Conclusio
Kapitel 4.4.1: Wechsel von Scrum auf Kanban	Ja, Kanban in Architekturphase interessant, weil: • Architekturphase zu chaotisch für Scrum • Wenig produziert in Architekturphase daher Scrum nicht sinnvoll	Ja, Kanban in Architekturphase interessant, weil: • Chaos kann man nicht durch Scrum bekämpfen. • Kein gutes Backlog wirklich möglich.	**Ja**, wäre interessant Kanban in der Architekturphase in einem der nächsten Projekte auszuprobieren. Weiterer Verlauf ist in Ordnung: • Produktionsphase: Scrum • Endphase: Kanban
Kapitel 4.4.2: Das „richtige“ WiP Limit finden	Ja, Kanban-Prozess mehr Struktur geben und gerichteter machen und Parameter öfter anpassen.	Ja, WiP über Feedback Loops ins Team zu bringen ist großartig.	**Ja**, eine funktionierende Feedback Loop ist entscheidend für funktionierende WiP-Limits und daher soll diese in Zukunft verbessert werden. *(Die beiden Vorschläge wurden gemeinsam besprochen)*
Kapitel 4.4.3: Feedback Loop verbessern			
Kapitel 4.4.4: Retrospektive einführen	Ja, sehr wichtig öfter zu machen.	Ja.	**Ja**, sehr wichtig regelmäßig(er) zu machen als Teil einer funktionierenden Feedback Loop.
Kapitel 4.4.5: Analyse mehr integrieren	Zweischneidiges Schwert. Fulltime-Analytiker dabei, aber im Prozess nicht gut integriert: nicht mehr möglich, nicht mehr gebraucht.	Wichtiges Thema. In „Projekt A“ Teilspezifikationsergebnisse nicht möglich, kommt extrem auf das Projekt und die Anforderung des Kunden an.	**Ja**, sofern es das Projekt erlaubt, Spezifikationspakete so gering wie möglich halten; ist aber nicht immer machbar aufgrund von Komplexität und/oder Kundeninteressen.
Kapitel 4.4.6: Aufwandsschätzung in Kanban	Ordentliche Schätzung hat mit Kanban aufgehört. – Fehler. Müssen wieder bessere Metriken und ein besseres Gefühl bekommen.	Modelle beide interessant, aber zuerst wichtig, dass wir generell wieder schätzen. Aber es wäre interessant beide auszuprobieren.	**Ja**, generell ist es wichtig, in Zukunft in Kanban jeden Task zu schätzen. Aber eventuell nur mit gewöhnlichem Planungspoker.

Vorschlag	Feedback „GF1“	Feedback „GF2“	Conclusio
Kapitel 4.4.7: Burndown-Chart und Cumulative Flow Diagram	Am meisten gefehlt hat die Cycle Time. Burndown bringt weniger als die Cycle Time, jedoch widerspricht sich beides nicht.	Ja, sehr wichtig Cycle Time automatisch anzupassen, um stabile Werte zu erhalten. Wichtig, jedoch keine Toolunterstützung vorhanden.	**Ja**, beide Diagramme wünschenswert und sehr wichtig, die Cycle Time genau zu beobachten in Kanban, jedoch muss Toolunterstützung geprüft werden.
Kapitel 4.4.8: Board-orientiertes Standup Meeting	Ja, auch physikalisches Board interessante Idee ohne Tool.	Ja, es hat uns geschadet, dass das „Big Picture“ verloren gegangen ist.	**Ja**, absolut wichtig das Board täglich im Auge zu behalten.
Kapitel 4.4.9: “Team Impediments“-Liste für Product Owner	Nein, kein klarer Vorteil ersichtlich im Vergleich zur „Blocked"-Spalte.	Nein, der Scrum Master behält alles im Auge und führt im Endeffekt eine ähnliche Liste.	**Nein**, keine Notwendigkeit zur Einführung gegeben derzeit.
Kapitel 4.4.10: Kanban-Sprint einführen	Ja, guter Punkt.	Ja, Sprint ohne Artefakte ist eine super Idee, um Parameteranpassung und Retrospektive in Kanban zu institutionalisieren.	**Ja**, soll auf jeden Fall ausprobiert werden.
Kapitel 4.4.11: Limit für „To Do“-Spalte stets gering halten	Ja, wichtig und spielt in Priorisierungskonzept hinein (vgl. Kapitel 4.4.14).	Ja.	**Ja**, wichtiger Punkt.
Kapitel 4.4.12: Umgang mit externen Blockierungen	Nein; das WiP Limit war nichts wert, da wir vom Kunden abhängig waren.	Nein; die „Blocked“-Spalte war bisher so groß, dass sie keinem mehr wehgetan hat. Hat nur Priorisierungsverlagerungen bewirkt, jedoch keine Abarbeitung.	**Nein**, die „Blocked“-Spalte wird voraussichtlich beibehalten werden aufgrund großer Kundenabhängigkeiten, die anders den Prozess zu sehr behindern würden.
Kapitel 4.4.13: Concurrency in Kanban	Nein; sehr interessant, aber wieder Problem mit Tool.	Nein; Konzepte gut, jedoch Tool-Support schlecht.	**Nein**, sehr interessantes Konzept, jedoch kein Tool-Support.

Vorschlag	Feedback „GF1“	Feedback „GF2“	Conclusio
Kapitel 4.4.14: Backlog in Kanban richtig benutzen	Nein; wurde bisher informell gemacht. Vor-Priorisierungen (vgl. Kapitel 4.4.14) ohne Order Point-Mechanismus sollten ausreichend sein.	Nein; ebenfalls Vorzug für Priorisierungen (vgl. Kapitel 4.4.14); hat den gleichen Effekt.	**Nein**, Priorisierungskonzept (vgl. vgl. Kapitel 4.4.14) wird dem Order Point-Mechanismus vorgezogen.
Kapitel 4.4.15: Kanban Release Management	Nein; haben 1-wöchigen Releasezyklus der alle fertigen Features/Fixes enthält.	Nein; Kosten der Releaseerstellung ist fast Null, daher keine Änderung nötig.	**Nein**, keine Notwendigkeit Release-Management zu verändern, da automatisiert und daher kein Kostenaufwand.
Kapitel 4.4.16: Priorisieren in (und mit) Kanban	Ja, großartig und leicht umsetzbar.	Ja, stimme zu.	**Ja**, sollte auf alle Fälle probiert werden.
Kapitel 4.4.17: Bug Handling	Ja, derzeit wird eher wie in Modell 1 verfahren.	Ja; wenige Erfahrungen damit in Kanban, jedoch auch wenig Probleme derzeit damit.	**Ja**, Überlegung wert, jedoch wenige Probleme mit Bug Handling in der Vergangenheit.
Kapitel 4.4.18: Einführung von Queues	Ja, sicherlich mit mehr Aufwand verbunden, jedoch verhilft definitiv zu mehr Transparenz.	Ja, ist hilfreich.	**Ja**, gute Idee, die etwas Mehraufwand mit sich bringt, jedoch sicherlich zu mehr Transparenz verhilft.
Kapitel 4.4.19: Filter für „Closed“-Spalte	Ja, das Intervall seit dem letzten Release ist sicherlich das Wichtigste.	Ja, hätte auch gerne Release-Notes automatisiert aus dem Ticket-System – wichtige Information.	**Ja**, besonders Filter auf Release-Intervalle sicherlich wertvoll.
Kapitel 4.4.20: Unterteilung des Kanban-Boards	Ja, der Board-Vorschlag ist großartig, aber die Tool-Unterstützung dafür ist nicht überall gegeben, muss geprüft werden.	Ja, Backlog mit Priorisierungskonzept (vgl. Kapitel 4.4.14) ist sehr gut und auch „Stage“ und „Production“ werden wir sicher brauchen.	**Ja**, sehr interessant, Kanban weiter dahingehend auszubauen.

Tabelle 15: Zusammenfassung des initialen Feedbacks der ITU

Zu dem Vorschlag aus Kapitel 4.4.1 sei noch anzumerken, dass bezüglich des Umstiegs von Scrum auf Kanban in der ITU mittlerweile bereits Kriterien vorhanden sind. Diese lauten:

- Umstieg, wenn mehr geblockt ist, als gearbeitet werden kann.
- Umstieg, wenn die Durchlaufzeiten der Sprints sich verschlechtern.
- Umstieg, wenn sich Tickets schneller ändern als der Sprint zu Ende ist.

Wie Tabelle 15 entnommen werden kann, ist das erste Feedback sehr positiv ausgefallen: 15 aus 20 Verbesserungsvorschlägen wurden in der Erstbetrachtung als aussichtsreich befunden und werden als Grundlage für das nächste Treffen herangezogen. Die verbleibenden 5 Vorschläge werden aus den folgenden Gründen nicht weiter verfolgt:

- Keine Notwendigkeit vorhanden (Vorschläge 4.4.9, 4.4.12 und 4.4.15)
- Kein Tool-Support vorhanden (Vorschlag 4.4.13)
- Anderer Vorschlag bevorzugt (Vorschlag 4.4.16 wird 4.4.14 vorgezogen)

5.2.2 Umzusetzende Verbesserungsvorschläge

Eine Woche nach dem initialen Feedback (Kapitel 5.2.1) wurde in einem weiteren Treffen diskutiert, welche Vorschläge des Konzepts in den folgenden knapp 2 Monaten in der ITU praktische Anwendung finden werden.

Diesem Treffen wohnte neben den Geschäftsführern der ITU „GF1“ und „GF2“ auch der Scrum Master „SM“ bei, um die Praxistauglichkeit näher zu diskutieren.

Vorschlag	Feedback „GF1“	Feedback „GF2“	Feedback „SM“	Umsetzung?
Kapitel 4.4.1: Wechsel von Scrum auf Kanban	Nein, vorerst nicht in der Architekturphase, weil das Problem nicht bewusst ist, dass sie chaotisch ist. Ja zu restlichem Vorschlag.	Nein, wir setzen uns in der Architekturphase einfach Zeitlimits für gewisse technische Themen und die bis dahin ausgearbeitete Lösung wird genommen.	-	**Ja:** Scrum in Produktionsphase, Kanban in Wartungsphase **Nein:** Kein Kanban in Architekturphase n
Kapitel 4.4.2: Das „richtige“ WiP Limit finden	Ja.	Ja.	Ja, jedoch Problem möglich Kanban an Story Points zu orientieren, weil Transparenz leidet.	**Ja.**
Kapitel 4.4.3: Feedback Loop verbessern	Ja.	Ja.	-	**Ja.**
Kapitel 4.4.4: Retrospektive einführen	Ja.	Ja.	-	**Ja.**
Kapitel 4.4.5: Analyse mehr integrieren	Ja, der Prozessteil der Analyse, User Stories zu erstellen, soll besser in den Prozess integriert werden.	-	Ja, das setzt natürlich auch das richtige Projektumfeld voraus; auch der Punkt kleinere Spezifikationsteile definieren zu können.	**Ja**, weil es interessant ist, ob wir dadurch effizienter und agiler werden.
Kapitel 4.4.6: Aufwandsschätzung in Kanban	Ja, für alle Projekte vorgenommen, um die Kanban-Phase besser bewertbar zu machen.	Nein zu den beiden Planungsmodellen, da die nächsten Projekte zu klein dafür sind.	-	**Ja:** Aufwandsschätzung in allen Projekten. **Nein:** Die beiden Planungsmodelle sind zu umfassend derzeit.

Vorschlag	Feedback „GF1“	Feedback „GF2“	Feedback „SM“	Umsetzung?
Kapitel 4.4.7: Burndown-Chart und Cumulative Flow Diagram	-	-	Ja, Burndown-Chart sollten wir reaktivieren. Er stellt eine weitere Feedback-Schleife für die Schätzungen dar. Nein; der Informationsgehalt im Cumulative Flow Diagram war bisher zu gering im Vergleich zu dem Aufwand aufgrund mangelnder Toolunterstützung.	**Ja:** Burndown-Chart wird reaktiviert. **Nein:** Cumulative Flow Diagram zu aufwendig in der Erstellung aufgrund mangelhaften Toolsupports.
Kapitel 4.4.8: Board-orientiertes Standup Meeting	-	-	Ja, jedoch vielleicht zu aufwendig für die Standup-Meetings, eventuell daher nur in Retrospektive.	**Ja**, voraussichtlich jedoch nur in Retrospektive.
Kapitel 4.4.9: “Team Impediments“-Liste für Product Owner	Ja, Missverständnis letzte Woche: Verwechslung mit Liste für externe Blockierungen.	-	Ja, noch nicht sicher, ob mit Toolsupport oder außerhalb, jedoch spielt das keine große Rolle für das Konzept.	**Ja**, als Liste für Dinge, die das Team behindern, durchaus denkbar.
Kapitel 4.4.10: Kanban-Sprint einführen	Ja, bereits im letzten Meeting davon überzeugt. Super Idee, wollen wir unbedingt ausprobieren als Zeitpunkt für Retrospektive etc. ohne Commitment.	-	-	**Ja**.

Vorschlag	Feedback „GF1“	Feedback „GF2“	Feedback „SM“	Umsetzung?
Kapitel 4.4.11: Limit für „To Do“-Spalte stets gering halten	Ja, ebenfalls Frage der WiP-Limits.	-	-	**Ja.**
Kapitel 4.4.12: Umgang mit externen Blockierungen	-	-	Ja, im neuen Projekt sollen Issues solange im Workflow bleiben bis sie gelöst sind.	**Ja.**
Kapitel 4.4.13: Concurrency in Kanban	Nein, das Tool dafür fehlt.	Man kann es mit einer zusätzlichen Spalte auch lösen, auch wenn es dann von der Visualisierung her nicht so optimal ist.	Muss überlegt werden, ob wir das machen wollen.	**Nein**, da keine Tool-Unterstützung vorhanden.
Kapitel 4.4.14: Backlog in Kanban richtig benutzen	Ja, kein Problem das einzuführen bei internen Projekten - bei Kundenprojekten schwierig.	-	Ja, gibt sogar Tool-Unterstützung dafür. Bei Kundenprojekten funktioniert es nach Pull-Prinzip und nicht nach Push-Prinzip.	**Ja** bei internen Projekten. **Nein** bei Kundenprojekten.
Kapitel 4.4.15: Kanban Release Management	Ja, grundsätzlich machen wir im internen Projekt Releases, wenn uns das Feature-Set gefällt.	Nein, nicht möglich. Auch beim internen Projekt ist es ein Hybrid aus Termin und Feature-Set.	Nein, bei Kundenprojekten nicht möglich, da sie in kurzen Abständen Releases haben wollen.	**Nein**, kein reines Feature-Set Release möglich.
Kapitel 4.4.16: Priorisieren in (und mit) Kanban	Ja, für Management interessant. Würden es gerne bei internem Projekt ausprobieren.	-	Nein, aus Team-Sicht wahrscheinlich weniger interessant.	**Ja**, im internen Projekt.

Vorschlag	Feedback „GF1“	Feedback „GF2“	Feedback „SM“	Umsetzung?
Kapitel 4.4.17: Bug Handling	Ja, tendieren zu ersterem Modell.	Ja, teilweise möglich umzusetzen, wir haben derzeit in jedem Projekt einen hochgradig auf den Kunden angepassten Bug-Handling Prozess.	-	**Ja**, eines der beiden Modelle könnte im internen Projekt ausprobiert werden.
Kapitel 4.4.18: Einführung von Queues	Nein, wissen derzeit nicht, wie wir das konkret umsetzen sollen.	-	Ja, grundsätzlich gut das zu sehen. Vielleicht kann man Spalten, die die eigene Arbeit nicht betreffen, ausblenden, damit das Board nicht zu breit ist.	**Nein,** vorerst nicht.
Kapitel 4.4.19: Filter für „Closed“-Spalte	-	-	Ja, im Prinzip schon umgesetzt in einem Projekt.	**Ja**, sollte man für alle Projekte bereitstellen.
Kapitel 4.4.20: Unterteilung des Kanban-Boards	Ja, die Idee mit „Stage“ und „Production“ kann vielleicht auch im internen Projekt ausprobiert werden.	Ja, also Priorisierung wird ausprobiert (P3-P2-P1). Nein, vertikale Unterteilung von Development in „Test“ und „Dev“ wahrscheinlich technisch nicht möglich.	Das muss man sich im Detail anschauen. Es wird sicherlich nicht alles gleich umgesetzt, da es ja auch eine Änderung der täglichen Arbeit bedeutet.	**Ja:** • Priorisierungsteil • „Stage“ und „Production“ **Nein:** Vertikale Unterteilung aufgrund mangelnden Toolsupports nicht möglich.

Tabelle 16: Umzusetzende Verbesserungsvorschläge des Konzepts

Tabelle 16 zeigt die Auswahl an Vorschlägen des vorliegenden Konzepts, die in der ITU erprobt werden. Die Spalte „Umsetzung?“ gibt Aufschluss darüber, welche der 20 Vorschläge für einen praktischen Einsatz ausgewählt wurden.

Es ergibt sich folgende Auswahl:

- 11 Vorschläge wurden zur Umsetzung ausgewählt.
- Weitere 6 Vorschläge werden teilweise oder leicht abgewandelt umgesetzt.
- Die verbleibenden 3 Vorschläge des Konzepts werden nicht umgesetzt.

Im 2. Treffen nach Fertigstellung des Konzepts wurde eine endgültige Auswahl an Vorschlägen getroffen (vgl. Tabelle 16). Auf dieser Grundlage beginnt somit die knapp zweimonatige Praxisphase in der ITU. Erfahrungen daraus sind im Folgekapitel 5.2.3 festgehalten.

5.2.3 Praktische Erfahrungen

Nach knapp 2 Monaten praktischer Anwendung der ausgewählten Vorschläge in der ITU wurden Erfahrungen eingeholt, die die Grundlage für die Überarbeitung des Konzepts in Kapitel 6.1 darstellen.

Die gemachten Erfahrungen wurden im Zuge eines Treffens mit den Geschäftsführern der ITU „GF1“ und „GF2“ sowie dem Scrum Master „SM“ besprochen. Im Vergleich zu vorangehenden Tabellen 15 „Zusammenfassung des initialen Feedbacks der ITU“ und 16 „Umzusetzende Verbesserungsvorschläge des Konzepts“ sind in Tabelle 17 nur mehr die für die Praxis ausgewählten Kapitel zu finden (vgl. Kapitel 5.2.2 und Tabelle 16).

Vorschlag	Feedback „GF1“	Feedback „GF2“	Feedback „SM“	Conclusio
Kapitel 4.4.1: Wechsel von Scrum auf Kanban	-	-	-	Hierzu gab es kein neues Feedback, da in der zweimonatigen Testphase kein Wechsel von Produktiv- auf Wartungsphase stattgefunden hat.
Kapitel 4.4.2: Das „richtige“ WiP Limit finden	Der Zeitpunkt von 2 Monaten ist wahrscheinlich ein bisschen zu früh. Wir sind jetzt einmal soweit, dass wir Messwerte aufgesetzt haben, aber eben noch keine Maßnahmen. Das war jedoch auch nicht die Erwartungshaltung.	Im Prinzip wissen wir noch nicht, was wir mit den Limits tun sollen. Wir definieren sie, aber wenn die Spalte rot ist, ist sie rot, wenn die Spalte leer ist, ist sie leer.	Hat mäßig funktioniert. Es fehlt noch, die Limits überhaupt erst einzuhalten. Der zweite Schritt wäre dann erst sie entsprechend anzupassen. Gefühl, dass der Wille fehlt etwas dagegen zu tun.	Die WiP Limits sind in Betrieb, jedoch ist noch Verbesserungspotential gegeben in Bezug auf Maßnahmensetzung bei Überschreitung und regelmäßige Anpassung der Limits.
Kapitel 4.4.3: Feedback Loop verbessern	-	-	Wir haben derzeit Iterationsmeetings mit Reviews und Retrospektive. Es wird teilweise in der Retrospektive über Limits gesprochen.	Es werden wieder regelmäßige Reviews und Retrospektiven abgehalten als Teil des Kanban-Sprints (vgl. Kapitel 4.4.10).
Kapitel 4.4.4: Retrospektive einführen	-	-		
Kapitel 4.4.5: Analyse mehr integrieren	-	-	Die Analyse ist nun im Board vertreten. Wir bemühen uns kleinere, geschlossene Pakete zu machen, die man sehr gut in 2-3 Tagen umsetzen kann.	Die Analyse ist nun Teil des Kanban-Boards. Weiters wird entsprechend dem Vorschlag versucht, Spezifikationspakete klein zu halten.

Vorschlag	Feedback „GF1“	Feedback „GF2“	Feedback „SM“	Conclusio
Kapitel 4.4.6: Aufwandsschätzung in Kanban	Der Vorsatz ist, dass alles wieder geschätzt wird. Ich weiß nicht, wie weit wir in der Praxis sind, aber der Vorsatz ist da.	In allen neuen Projekten soll nur mehr in Story Points geschätzt werden.	Die Durchlaufzeit wird nicht berechnet, weil nicht klar ist, was man damit macht. Das Überticket wird in Story Points geschätzt. Wenn es dann tatsächlich fertig ist für die Umsetzung, werden die Subtickets geschätzt.	Dieser Vorschlag wurde im Endeffekt doch nicht umgesetzt, da weder die Modelle ausprobiert werden konnten noch Metriken gemessen werden.
Kapitel 4.4.7: Burndown-Chart und Cumulative Flow Diagram	-	-	Wir wollen jetzt beginnen die Charts zu beobachten, weil der erste Schritt natürlich sein muss, dass alle Tickets geschätzt werden, damit sinnvolle Daten für das Burndown-Chart bereitstehen.	Noch nicht im Einsatz, da zuerst regelmäßige Aufwandsschätzung betrieben werden muss.
Kapitel 4.4.8: Board-orientiertes Standup Meeting	-	-	Das Board-orientierte Meeting findet nicht im Standup statt, sondern im Iterationsmeeting, das heißt einmal die Woche.	Board-orientiertes Standup wurde als zu schwerfällig empfunden, daher nur im wöchentlichen Iterationsmeeting.
Kapitel 4.4.9: “Team Impediments“-Liste für Product Owner	-	Anfangsprobleme haben wir derzeit eher nicht, da die Projekte länger am Laufen sind. Demnach kein Bedarf.	Wurde nicht ausprobiert. Halte es grundsätzlich für eine gute Idee, wobei ich bisher in keinem der Projekte eine ausreichende Anzahl an Problemen gesehen hätte.	Wurde nicht eingesetzt, da in den vergangenen zwei Monaten keine Projekte neu gestartet sind und daher kein Bedarf gesehen wurde.

Vorschlag	Feedback „GF1“	Feedback „GF2“	Feedback „SM“	Conclusio
Kapitel 4.4.10: Kanban-Sprint einführen	-	-	Ja, es findet im Prinzip ein Scrum-Prozess statt ohne Commitment in den Sprints. Die Iterationen sind mittlerweile wöchentlich und es wird auch das Produktinkrement hergezeigt. Ein großer Teil ist ebenfalls die Retrospektive und das Board wird ebenfalls immer angeschaut.	Der Kanban-Sprint wurde mit Erfolg eingeführt. Es ist nun ein Zeitpunkt geschaffen, um Review und Retrospektive durchzuführen, sowie auch den aktuellen Status des Kanban-Boards gemeinsam zu überprüfen.
Kapitel 4.4.11: Limit für „To Do“-Spalte stets gering halten	-	Die „To Do“-Spalte gibt es eigentlich nicht mehr. Im Backlog gibt es gar kein Limit mehr.	-	Die „To Do“-Spalte ist nun sinngemäß umgewandelt worden in „Ready for Analysis“.
Kapitel 4.4.12: Umgang mit externen Blockierungen	-	Derzeit auch eine etwas ruhigere Zeit diesbezüglich.	Es gibt keine „Blocked"-Spalte mehr. Ergibt sich implizit aus dem Board; ein überschrittenes Limit bedeutet jedoch nicht „Stopp“, sondern „Alarm“.	Die „Blocked“-Spalte wurde entsprechend dem Vorschlag abgeschafft. Blockierungen werden nun implizit aus dem Board herausgelesen mit Hilfe der WiP Limits.
Kapitel 4.4.14: Backlog in Kanban richtig benutzen	Es ist schwer mit der Praxis zu vereinbaren aufgrund von Kundenabhängigkeiten.	Backlog haben wir kein Limit, da kommt einfach unkontrolliert rein, was im Projekt noch zu tun ist oder was am Horizont ist.	Order Point Limit ist schwer mit der Praxis zu vereinbaren aufgrund von Kunden und auch nicht unbedingt nötig.	Für das Order Point Konzept wird kein Bedarf gesehen, da die Backlogfüllung bisher intuitiv funktioniert hat ohne Mechanismus.

Vorschlag	Feedback „GF1“	Feedback „GF2“	Feedback „SM“	Conclusio
Kapitel 4.4.16: Priorisieren in (und mit) Kanban	-	-	Nein, das Konzept ist es nicht geworden, aber wir haben ein Ranking. Also es ist nicht auf z.B. 5 Levels beschränkt, sondern alle Tickets werden in Relation zueinander gereiht.	Das vorliegende Priorisierungskonzept wurde nicht umgesetzt, jedoch ein ähnliches Konzept: Stattdessen gibt es so viele „Stufen“ wie es Tickets gibt, da kein Ticket einem anderen gleichgestellt werden darf.
Kapitel 4.4.17: Bug Handling	-	Derzeit: Wenn ein Bug kommt, wird ein neues Ticket angelegt.	Das Feature, zu dem der Bug gehört, verbleibt in „Test“ oder es wird komplett abgeschlossen, je nach Größe des Fehlers. Wenn riesige Dinge fehlen, geht das Ticket in „Dev“ zurück.	Es wird derzeit eine Mischung aus den beiden Modellen des Konzepts angewandt.
Kapitel 4.4.19: Filter für „Closed“-Spalte	-	-	Es gibt eigene Filter je Iteration.	Es sind nun entsprechende Filter für alle Projekte bereitgestellt.
Kapitel 4.4.20: Unterteilung des Kanban-Boards	-	Das Team erkennt jetzt besser, woran eigentlich gerade gearbeitet wird und kann sich besser einstellen, was da in nächster Zeit auf einen zukommt.	Es hat den Vorteil gebracht, dass man keine persönlichen Filter mehr braucht, um die relevanten Tickets zu finden.	Das feiner granulierte Kanban-Board mit Einsatz von Queues hat sich gut bewährt und für mehr Klarheit und Überblick gesorgt.

Tabelle 17: Zusammenfassung des Praxis-Feedbacks der ITU

Einige Vorschläge waren zwar zur Umsetzung vorgesehen (vgl. Tabelle 16), sind jedoch im Endeffekt doch nicht umgesetzt worden (vgl. Tabelle 17). Gründe dafür sind unter anderem Gegenstand der Diskussion der Ergebnisse in Kapitel 6.

Abschließend zu diesem Kapitel wird die Statistik der tatsächlich in der ITU umgesetzten Vorschläge[8] angeführt:

- 13 Vorschläge wurden umgesetzt
- 7 Vorschläge wurden nicht umgesetzt

[8] Teilweise umgesetzte Vorschläge gelten hier ebenfalls als umgesetzt. Der Grad und die Art der Umsetzung wird in Kapitel 6 näher untersucht.

6 Diskussion der Ergebnisse

Dieses Kapitel befasst sich mit der Analyse der Ergebnisse der Arbeit. Besonderes Augenmerk liegt auf den mit dem Konzept gemachten Erfahrungen (vgl. Kapitel 5.2.3). In Kapitel 6.1 wird die praktische Umsetzung der Vorschläge durchleuchtet und daraus Verbesserungen des Konzepts abgeleitet. Kapitel 6.2 schließt die Arbeit mit einer Beantwortung der Fragestellung aufgrund der gewonnenen Erkenntnisse ab.

Einleitend sei folgendes Zitat als Resümee der Praxisphase der ITU erwähnt:

„*Und man kann schon auch sagen, dass in dem Projekt, wo wir uns am meisten damit [Anm.: mit den Vorschlägen des Konzepts] beschäftigt haben: Das ist ein super Projekt. Da haben wir von Auftraggeber- und Auftragnehmerseite ein super positives Feedback bekommen und das liegt wohl auch am Prozess."*

(Zitat Geschäftsführer "GF1" aus Abschluss-Meeting)

6.1 Verbesserung des Konzepts

Viele der Vorschläge wurden teilweise oder leicht modifiziert umgesetzt. Dies hat sich in der Praxis aus verschiedensten Gründen ergeben, wie Geschäftsführer „GF1" beschreibt:

„Das Tolle an dieser Arbeit ist, dass wir es in einer gelebten Praxis umsetzen. Das ist aber auch gleichzeitig schon wieder der Nachteil, weil wir uns wirklich viel damit beschäftigt haben - aber wir entwickeln ja auch Software nebenbei. Daher werden wir auch in weiteren 6 Wochen nicht so einen Stand haben, dass wir Wasserfallmodell-mäßig sagen: ‚So jetzt probieren wir die Idee aus und da haben wir die Ergebnisse'. Den Zustand werden wir nie haben. Es wird immer ein bisschen Work in Progress sein. Wir schrauben immer an den einen oder anderen Richtlinien herum. Die Infos, die wir bekommen haben, waren sehr, sehr wertvoll und wir haben vieles davon – vielleicht nicht 1:1, aber doch vom Gedankengang her - umgesetzt."

(Zitat Geschäftsführer „GF1" aus Abschluss-Meeting)

Wie dem Zitat entnommen werden kann, ist ein Teil der Umsetzung der Vorschläge an praktischen Problemen gescheitert wie Zeitmangel oder fehlende Tool-Unterstützung. Dennoch konnten 13 von 20 Vorschlägen umgesetzt werden. Die daraus gewonnenen Erfahrungen werden im Unterkapitel 6.1.1 analysiert. In Unterkapitel 6.1.2 werden ebenfalls die nicht umgesetzten Vorschläge in Hinblick auf das erhaltene Feedback untersucht.

6.1.1 Umgesetzte Vorschläge

Es soll für jeden umgesetzten Vorschlag bewertet werden, ob:

1. durch den Vorschlag eine Verbesserung in der ITU erzielt werden konnte.
2. der Vorschlag selbst aufgrund der gewonnenen Erfahrung verbessert werden kann.

Es folgt eine Auflistung und Analyse aller umgesetzten Vorschläge:

Vorschlag 4.4.1 Wechsel von Scrum auf Kanban

Der Vorschlag sieht folgende Aufteilung vor:

- Architekturphase: Scrum oder Kanban
- Produktionsphase: Scrum
- Endphase: Kanban

Für die Architekturphase wurde in der ITU ein eigener Mechanismus ins Leben gerufen an Stelle von Scrum oder Kanban:

„Wir setzen uns einfach Zeitlimits für gewisse technische Themen und die bis dahin ausgearbeitete Lösung wird genommen."

(Zitat Geschäftsführer „GF2" aus Meeting vor Praxisphase)

Die Produktionsphase mit Scrum und Endphase mit Kanban hat gut funktioniert. Der Umstieg von Scrum auf Kanban wurde nach folgenden Kriterien durchgeführt:

- Es gibt so viele Blockierungen, dass nicht mehr sinnvoll in Sprints gearbeitet werden kann, weil das Commitment nicht mehr möglich ist.
- Die Durchlaufzeiten der Sprints verschlechtern sich.
- Die Tickets ändern sich schneller als es die Sprint-Iteration erlaubt.

Bewertung des Vorschlags:

Die hier angeführten Umstiegskriterien werden in den Vorschlag aufgenommen. Die Frage, ob Scrum oder Kanban in der Architekturphase zu bevorzugen ist, kann derzeit nicht beantwortet werden, da hierzu keine konkreten Erfahrungen aus der Praxis eingeholt werden konnten. Weiters soll im Vorschlag deutlicher hervorgehoben werden, dass das vorliegende Konzept in allen Phasen eines Softwareprozesses zum Einsatz kommen kann. Die Aufteilung nach Phasen in diesem Vorschlag bezieht sich lediglich auf eine mögliche Kombination von „reinem" Scrum und Kanban unabhängig von den restlichen Vorschlägen.

Vorschlag 4.4.2 Das „richtige“ WiP Limit finden
Funktionierende WiP Limits zu finden ist ein kontinuierlicher Prozess. In der zweimonatigen Praxisphase in der ITU wurden Limits eingeführt, jedoch fehlt es noch an der Maßnahmensetzung und der regelmäßigen Adaptierung:

„Die Limits schaffen schon die Awareness, dass es hier oder da Probleme gibt, aber welche Maßnahmen man ergreifen sollte oder will ist noch undefiniert.
(Zitat Scrum Master „SM“ aus Abschluss-Meeting)

Bewertung des Vorschlags:
Der Vorschlag konnte zum Ende der Praxisphase noch nicht vollständig umgesetzt werden. Daher ist keine Ableitung für den Vorschlag möglich und er verleibt unverändert im Konzept.

Vorschlag 4.4.3 Feedback Loop verbessern
Es wurden alle vorgeschlagenen Feedback Loops umgesetzt: Von Pair Programming bis zum Sprint. Der Sprint wurde in Form eines einwöchigen Kanban-Sprints umgesetzt, der ohne Commitment einen Zeitpunkt für Review und Retrospektive schafft (vgl. Kapitel 4.4.10).

Bewertung des Vorschlags:
Der Vorschlag hat sich sehr gut bewährt. Teile davon wurden schon in der Vergangenheit in der ITU mit großem Erfolg eingesetzt. Der Einsatz eines Sprints ohne Commitment in Kanban als Zeitpunkt für Retrospektive und Review hat sehr großen Anklang gefunden. Aufgrund des positiven Feedbacks verbleibt der Vorschlag unverändert im Konzept.

Vorschlag 4.4.4 Retrospektive einführen
Die Retrospektive wird nun in der ITU wöchentlich abgehalten am Ende des Commitment-losen Kanban-Sprints (vgl. Kapitel 4.4.10). Die Einführung der Retrospektive wurde von den Mitarbeitern sehr begrüßt. Es sollte das Meeting jedoch auch genutzt werden, um über die Aufteilung des Kanban-Boards und die Anpassung sowie Einhaltung der Limits zu reflektieren.

Bewertung des Vorschlags: Verbleibt aufgrund des positiven Feedbacks unverändert im Konzept.

Vorschlag 4.4.5 Analyse mehr integrieren
Die Analyse-Tätigkeit hat bisher außerhalb des Boards agiert, sowohl in Scrum als auch in Kanban. Jetzt wurde sie aufgrund des Vorschlags in das Board integriert und ist damit überschaubarer geworden für alle Beteiligten. Es wird ebenfalls in neuen Projekten

versucht, mit kleineren Spezifikationspaketen zu arbeiten entsprechend dem Vorschlag, um das Projektrisiko zu minimieren.

Bewertung des Vorschlags: Verbleibt aufgrund des positiven Feedbacks unverändert im Konzept.

Vorschlag 4.4.8 Board-orientiertes Standup Meeting
Das Kanban-Board wurde in der ITU nicht in das Standup Meeting einbezogen, dafür wurde es einmal die Woche im Iterationsmeeting gemeinsam besprochen. Die neue Zusammensetzung des Boards hat dazu beigetragen, dass das Board für alle präsenter war und jeder damit ohne Einschränkungen (wie zum Beispiel persönliche Filter) arbeiten konnte. Dennoch soll der Vorschlag, das Board auch in das Standup Meeting einzubeziehen, beibehalten werden, da dies gerade in größeren Projektteams die interne Kommunikation wesentlich erleichtern kann.

Bewertung des Vorschlags:
Die Idee ein physisches Board zu benutzen wurde mit der ITU diskutiert, jedoch sprechen folgende zwei Vorteile für ein elektronisches Board:

- Einfachere Wartung: Neue Spalten, Limits, Tickets, etc.
- Automatisierte Reports und Metriken für aktuelle und historische Daten

Demnach wird der Vorschlag insofern aktualisiert, dass nur zu Beginn eines Kanban-Prozesses, das heißt wenn kein geeignetes Tool verfügbar ist, mit einem physischen Board gearbeitet werden kann. Es sollte jedoch mit der Zeit aufgrund der genannten Vorteile auf ein elektronisches Tool umgestiegen werden.

Vorschlag 4.4.10 Kanban-Sprint einführen
Den Commitment-losen Kanban-Sprint einzuführen hat in der ITU sehr gut funktioniert, da es damit einen regelmäßigen Zeitpunkt für Review und Retrospektive gibt. Er sollte jedoch überdies auch noch genutzt werden, um Board, Charts und WiP Limits zu aktualisieren sowie neue Tasks zu schätzen (entsprechend dem ursprünglichen Vorschlag).

Bewertung des Vorschlags:
Die Einführung des Kanban-Sprints hat sich sehr gut bewährt. Aufgrund der Praxiserfahrung in der ITU wird dem Vorschlag noch hinzugefügt, dass am Ende des Kanban-Sprints optional auch ein formloses Review stattfinden kann, das heißt die Präsentation des aktuellen Produktinkrements.

Vorschlag 4.4.11 Limit für „To Do“-Spalte stets gering halten
Durch die Überarbeitung des Kanban-Boards in der ITU ist aus der „To Do“ Spalte mittlerweile der treffendere Name „Ready for Analysis“ geworden. Diese hat WiP Li-

mits wie jede andere Spalte des Workflows. Es sollte entsprechend dem Vorschlag darauf geachtet werden, auch das Limit der „Ready for Analysis"-Spalte gering zu halten, da es lediglich aufweisen soll, welche Items als nächstes gestartet werden sollen.

Bewertung des Vorschlags:
Der Vorschlag wird in *„Limit der Ready Queue stets gering halten"* umbenannt, um sich von dem konkreten ITU-Fallbeispiel etwas zu entfernen und im allgemeinen Rahmen besser verständlich zu sein.

Vorschlag 4.4.12 Umgang mit externen Blockierungen
Die „Blocked"-Spalte, die es früher in der ITU gab, wurde im Zuge der Umsetzung des Vorschlags entfernt. Blockierungen – interne wie externe – zeigen sich nun implizit durch Überschreitung der WiP Limits am Kanban-Board.

Geschäftsführer „GF2" beschreibt die Vorgehensweise bei Blockierungen folgendermaßen:

„Wir haben uns dann [Anm.: nachdem es zu Blockierungen gekommen ist] eine Strategie überlegt, wie wir damit umgehen. Zum Beispiel war vor einigen Wochen die Qualitätssicherung das Bottle Neck und wir haben ein Ressourcenplanungsmeeting ins Leben gerufen. Wir haben erkannt, dass wir da wirklich einfach ein Ressourcenproblem haben."

(Zitat Geschäftsführer „GF2" aus Abschluss-Meeting)

Bewertung des Vorschlags:
Eine eigene „Blocked"-Spalte einzuführen würde den Kanban Prinzipien widersprechen und der Workflow würde durch eine solche Spalte unterbrochen werden. Der Vorschlag verbleibt unverändert im Konzept, da Blockierungen mit Hilfe der WiP Limits ausfindig gemacht werden können und sollen; dies stellt eines der Grundprinzipien in Kanban dar.

Vorschlag 4.4.17 Bug Handling
In der ITU kam keines der beiden Modelle gänzlich zum Einsatz. Vielmehr wurde eine Mischung der beiden eingesetzt:

„Das Feature, zu dem der Bug gehört, verbleibt in ‚Test' oder es wird komplett abgeschlossen, je nach dem, ob der Fehler abnahmeverhindernd ist oder nicht. Wenn riesige Dinge fehlen, wird das Ticket sicher auch in Development zurückgehen."

(Zitat Scrum Master „SM" aus Abschluss-Meeting)

Bewertung des Vorschlags:
Entsprechend der Abwandlung in der ITU wird dem Vorschlag eine dritte Alternativmöglichkeit hinzugefügt:

- Schwerwiegende Bugs: Das Feature wird zurück in „Development“ gesetzt. (entsprechend Modell A)
- Geringfügiger Bug: Das Feature verbleibt in der „Test“-Spalte. (entsprechend Modell B)

Diese Vorgehensweise stellt einen interessanten Ansatz dar, da sie bewirkt, dass kleine Bugs kurzfristig aufgrund blockierender WiP Limits in „Test“ abgeschlossen werden müssen; große Bugs erhalten mehr Zeit, indem sie zurück in „Development“ gesetzt werden.

Vorschlag 4.4.18 Einführung von Queues
Es wurden Queues für die drei Hauptbereiche des Workflows „Analyse“, „Development“ und „Test“ eingeführt. Die Queues wurden als eigene Spalten mit eigenen WiP Limits in das Board in der Form „Ready for x“ eingefügt (wobei „x“ den Bereich darstellt). Es konnte kein gemeinsames WiP Limit für die jeweilige „Ready for“-Spalte und „In Progress“-Spalte eines Bereichs aufgrund fehlender Tool-Unterstützung in der ITU eingeführt werden.

Dennoch haben die Queues (mit eigenem WiP Limit realisiert) für mehr Übersicht am Board gesorgt und einen realistischeres Abbild des Workflows geschaffen:

„Das Team erkennt, woran eigentlich gerade oder in nächster Zeit gearbeitet wird und man kann sich drauf einstellen, was da bald kommen wird.
(Zitat Geschäftsführer „GF2“ aus Abschluss-Meeting)

Bewertung des Vorschlags:
Auch wenn gemeinsame WiP Limits für Queue-Spalte und „In Progress“-Spalte in der ITU nicht zum Einsatz kommen konnten, so ist diese Variante dennoch vorzuziehen gegenüber getrennten WiP Limits. Der Vorschlag verbleibt daher unverändert im Konzept.

Vorschlag 4.4.19 Filter für „Closed“-Spalte
Der Vorschlag wurde in der ITU umgesetzt, das heißt es wurden Filter für alle Releases bereitgestellt. Mit ihrer Hilfe kann die Übersicht in der „Closed“-Spalte stets gewährleistet werden.

Bewertung des Vorschlags:
Der Vorschlag ist aus Problemen der ITU heraus entstanden und konnte mit der gewählten Vorgehensweise gelöst werden. Er hat aber auch generelle Praxisrelevanz für die Handhabung der „Closed“-Spalte und verbleibt daher unverändert im Konzept.

Vorschlag 4.4.20 Unterteilung des Kanban-Boards
Das Board wurde folgendermaßen in der ITU umgesetzt:

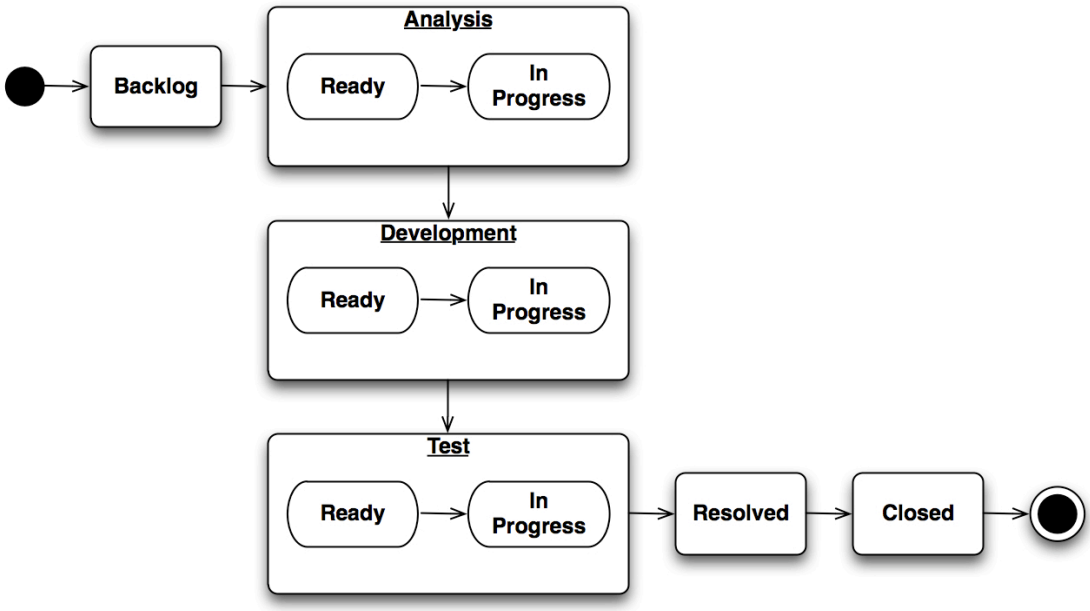


Abbildung 20: Neuer Workflow in der ITU als UML Zustandsdiagramm

Wie Abbildung 20 entnommen werden kann, kamen in der ITU, dem Konzept entsprechend, Queues zum Einsatz, wie bereits bei Vorschlag 4.4.18 besprochen. Die Queues tragen im vorliegenden Zustandsdiagramm den Namen „Ready“. Jeder Zustand des Diagramms stellt eine Spalte des Kanban Boards dar und verfügt daher über ein eigenes WiP Limit.

Die Rückmeldungen auf die neue Aufbereitung des Boards waren durchwegs positiv:

„Es hat den Vorteil gebracht, dass man keine persönlichen Filter mehr braucht, um die relevanten Tickets zu finden.“

(Zitat Scrum Master „SM“ aus Abschluss-Meeting)

„Das hat sich gut bewährt, finde ich, das feiner granulare Board.“

(Zitat Geschäftsführer „GF2“ aus Abschluss-Meeting)

Bewertung des Vorschlags:
Das Priorisierungskonzept aus Vorschlag 4.4.16 kam nicht zum Einsatz. Die Spalten „Stage“ und „Production“ des Vorschlags können mit den Spalten „Resolved“ und „Closed“ in der ITU verglichen werden. Die vorgeschlagene Concurrency ist ebenfalls nicht zum Einsatz gekommen, wie in Kapitel 5.2.3 erläutert wird. Der Vorschlag wird jedoch so im Konzept beibehalten, da er eine beispielhafte Zusammensetzung des Boards darstellt und der Anspruch nicht lautete, alle Vorschläge umzusetzen.

6.1.2 Nicht umgesetzte Vorschläge

Auch die nicht in die Praxis umgesetzten Vorschläge werden hier einer Analyse unterzogen. Es werden sowohl Gründe genannt, warum die Vorschläge nicht umgesetzt werden konnten, als auch mögliche Verbesserungen an den Vorschlägen untersucht werden.

Es folgt eine Auflistung und Analyse der nicht umgesetzten Vorschläge:

Vorschlag 4.4.6 Aufwandsschätzung in Kanban
Der Vorschlag wurde ursprünglich zur Umsetzung ausgewählt. Die beiden vorgeschlagenen Modelle wurden jedoch für kleinere Projektteams als zu aufwendig erachtet. Metriken wurden ebenfalls nicht umgesetzt, da zum Ende der Praxisphase auch noch keine entsprechenden Maßnahmen definiert waren, ähnlich dem Zustand in Vorschlag 4.4.2.

Bewertung des Vorschlags:
Der Vorschlag wurde zwar zur Umsetzung ausgewählt, jedoch nicht angewandt. Da es kein praktisches Feedback gibt, verbleibt er unverändert im Konzept.

Vorschlag 4.4.7 Burndown-Chart und Cumulative Flow Diagram
Der Burndown-Chart konnte nicht eingesetzt werden, da zuerst eine funktionierende und regelmäßige Aufwandsschätzung betrieben werden muss, bevor Abweichungen gemessen werden können.

„Wir wollen jetzt beginnen, die Charts zu beobachten, weil der erste Schritt natürlich sein muss, dass alle Tickets geschätzt werden, damit sinnvolle Daten für das Burndown-Chart bereitstehen.“

(Zitat Scrum Master „SM“ aus Abschluss-Meeting)

Auch das Cumulative Flow Diagram wurde nicht eingesetzt. Grund dafür war mangelnder Tool-Support.

Bewertung des Vorschlags:
Der Vorschlag verbleibt unverändert im Konzept, da der Burndown-Chart nicht praktisch getestet werden konnte und das Cumulative Flow Diagram nur aufgrund mangelnden Tool-Supports nicht in Verwendung ist.

Vorschlag 4.4.9 "Team Impediments"-Liste für Product Owner
Die „Team Impediments"-Liste wurde in der ITU nicht eingesetzt, da hier kein Bedarf gesehen wurde. Folgende zwei Gründe waren dafür ausschlaggebend:

1. In der zweimonatigen Praxisphase wurden keine neuen Projekte begonnen, in denen es zu Beginn unter Umständen vermehrt zu prozessbezogenen Problemen kommen kann.
2. Die laufenden Projekte konnten ohne Liste bewerkstelligt werden, da die Anzahl an Problemen überschaubar geblieben war.

Bewertung des Vorschlags:
Der Vorschlag soll dennoch im Konzept verbleiben, da der Einsatz der Liste nicht praktisch getestet wurde und in anderen Fällen als nützliche Schnittstelle zwischen Team und Product Owner fungieren kann.

Vorschlag 4.4.13 Concurrency in Kanban
Die ITU arbeitet nun mehr mit mehr Parent- und Subtickets als früher, jedoch eine wirkliche „Concurrency" (engl. für Nebenläufigkeit) ist Tool-bedingt nicht möglich.

Bewertung des Vorschlags:
Der Vorschlag konnte Tool-bedingt nicht umgesetzt werden. Das (theoretische) Feedback der ITU war jedoch sehr positiv und der Vorschlag verbleibt daher unverändert im Konzept.

Vorschlag 4.4.14 Backlog in Kanban richtig benutzen
Der Vorschlag wurde nicht umgesetzt, da dafür einerseits nicht zwingend ein Bedarf gesehen wurde und andererseits der Stand des Backlogs in der ITU bisher immer hochgradig kundenabhängig war. Weiters wurde darauf verwiesen, dass bisher auf intuitive Weise der Backlog-Stand überprüft wurde und ein Order Point Mechanismus keinen Mehrwert bringen würde. Als weiteres Argument ist anzuführen, dass der Order Point Mechanismus obsolet wird, sobald der Backlog einem angemessenen Priorisierungssystem unterworfen wird, wie es zum Beispiel der Vorschlag 4.4.16 darstellen könnte.

Bewertung des Vorschlags:
Der Vorschlag verbleibt im Konzept. Er gewährleistet eine ausreichende Beschäftigung mit dem Backlog. Es wird jedoch der Beschreibung des Vorschlags hinzugefügt, dass er

überflüssig wird, sobald eine regelmäßige Priorisierung (und damit Beschäftigung mit dem Backlog) betrieben wird.

Vorschlag 4.4.15 Kanban Release Management
Die Releaseplanung wurde in den Projekten großteils vom Kunden vorgegeben. Zusätzlich verfügt die ITU über geringe Releasekosten, da die Releaseerstellung automatisiert ist. Daher wurde auf die Umsetzung dieses Vorschlags verzichtet.

Bewertung des Vorschlags:
Da der Release Point Mechanismus (entgegengesetzt zum Order Point Mechanismus) nicht praktisch erprobt werden konnte, verbleibt er unverändert im Konzept.

Vorschlag 4.4.16 Priorisieren in (und mit) Kanban
Der vorgeschlagene Priorisierungsmechanismus wurde nicht eingesetzt, da der Einwand gebracht wurde, dass das Board nicht um weitere drei Spalten erweitert werden sollte; es würde dann bereits 12 Spalten umfassen. Es kam jedoch ein eigenes Priorisierungskonzept zum Einsatz:

„Wir haben ein Ranking. Also es ist nicht auf zum Beispiel 3 Levels beschränkt, sondern alle Tickets werden in Relation zueinander gereiht, das heißt kein Ticket darf mit einem anderen Ticket auf der gleichen Ebene sein.“

(Zitat Scrum Master „SM“ aus Abschluss-Meeting)

Bewertung des Vorschlags:
Das vorgeschlagene Priorisierungskonzept birgt Potential und verbleibt im Konzept. Es wird jedoch das in der ITU angewandte Konzept dem Vorschlag als Alternative hinzugefügt, da es dem Hauptgedanken des ursprünglichen Konzepts entspricht: die Schaffung einer Reihung der einzelnen Tasks auf der Grundlage einer klaren Priorisierung. Das Alternativkonzept ist sogar noch restriktiver, da es überhaupt keine Tasks auf der gleichen Priorisierungsebene erlaubt.

6.2 Abschließende Beantwortung der Fragestellung

Um den Prozess zur Beantwortung zu vervollständigen, wird zunächst der Bezug der Ergebnisse dieser Arbeit mit den Ergebnissen und Herangehensweisen der verwandten Arbeiten [Lada08] und [KnSk10] verglichen. Sie befassen sich ebenfalls beide mit dem Thema Scrum und Kanban; das heißt mit Unterschieden und Gemeinsamkeiten der beiden Prozesse.

Während die vorliegende Arbeit durchwegs positive Erfahrungen zur Verbindung von Scrum und Kanban aufzuweisen hat, so ist [Lada08] hier anderer Meinung. Er rät beispielsweise von Iterationen in Kanban gänzlich ab. Der im Konzept entwickelte “Kan-

ban-Sprint", das heißt ein Sprint ohne Commitment, hat sich jedoch in der Praxis als überaus nützlich und sinnvoll erwiesen. [Lada08] versucht entgegen dem Titel „Scrumban" Scrum-erfahrene Fachleute von Kanban zu überzeugen. Die vorliegende Arbeit hingegen hat sich gänzlich darauf konzentriert, Scrum und Kanban erfolgreich zu verbinden.

[KnSk10] verfolgt einen ähnlichen Ansatz wie diese Arbeit, das heißt auf die Stärken und Schwächen der beiden Prozesse einzugehen. Die vorliegende Arbeit geht jedoch noch einen Schritt weiter und hat aus den jeweiligen Vor- und Nachteilen eine Art Hybrid-Prozess geschaffen, der Elemente aus beiden Prozessen enthält.

Die Fragestellung dieser Arbeit, die in der Problemstellung (vgl. Kapitel 1.1) aufgeworfen wurde, lautet:

Wie lassen sich die Vorteile des Lean Development in der agilen Software Entwicklung einsetzen und damit agile Prozesse weiter verbessern?

Es wurde ebenfalls, wie auch schon bei [Lada08] und [KnSk10] versucht bei der Formulierung der Fragestellung sich Kanban von Scrum aus zu nähern, da Scrum dem neuen Prozess am nähesten kommt und sich in den vergangenen Jahren zu einem erfolgreichen und verbreiteten Prozessmodell entwickelt hat.

Wie die vorliegende Arbeit zeigt, liefert Kanban neue Praxis-relevante Ansätze für die Software Entwicklung. Die Ergebnisse dieser Arbeit unterstreichen (vgl. Kapitel 6.1), dass Kanban und Scrum nicht nur theoretisch sinnvoll kombiniert werden kann, sondern das erstellte Konzept auch in der Praxis zu Verbesserungen führen kann, ausgehend von den positiven praktischen Erfahrungen, die mit dem vorliegenden Konzept in der ITU gemacht wurden.

Als unmittelbare Antwort auf die Fragestellung kann das Kapitel 6.1 gesehen werden. Hier findet sich ein in Form von Vorschlägen aufbereitetes Konzept, dass entsprechend Erfahrungen aus der Praxis weiter überarbeitet wurde und sowohl Elemente aus Scrum als auch Kanban enthält. Dennoch soll das Konzept nicht als endgültige Beantwortung der Fragestellung betrachtet werden, sondern vielmehr als Schritt in die richtige Richtung und positiver Indikator. Erst die nächsten Jahre werden zeigen, wie gut sich Kanban(-Elemente) in der Praxis bewähren wird. Es kann jedoch festgehalten werden, dass es in der zweimonatigen Praxisphase in der ITU möglich war, große Erfolge mit dem vorliegenden Konzept zu erzielen.

7 Conclusio

Agile Software Entwicklung mit dem bekanntesten Vertreter Scrum ist mittlerweile aus der Prozesslandschaft nicht mehr wegzudenken. Das noch eher unbekannte Kanban verspricht Antworten auf Punkte zu geben, die sich bei Scrum im Laufe der Jahre als Schwächen herausgestellt haben. Es stellt daher einen interessanten Partner für eine Kombination der beiden Prozesse dar. Die folgenden Schwächen wurden im Zuge der vorliegenden Arbeit in Form von Interviews mit praxiserfahrenen Scrum-Teams identifiziert:

- Scrum benötigt stets ein ausreichend gefülltes Backlog
- Fehlende Möglichkeiten, auf Änderungen schnell zu reagieren
- Scrum-Sprint erfordert gewisse Ruhe im Projektumfeld

Aufgrund dieser Probleme wurde im Rahmen dieser Arbeit ein Konzept erarbeitet, um Scrum und Kanban zu verbinden und somit die Stärken beider Prozesse zu vereinen. Das Konzept wurde in Hinblick auf verfügbare State of the Art Erkenntnisse zu den beiden Vorgehensmodellen erstellt. Als weitere Informationsquelle wurden Interviews mit Scrum-erfahrenen Teams durchgeführt. Die Praxistauglichkeit des Konzepts wurde im Zuge einer Praxisphase festgestellt.

Die resultierenden Ergebnisse zeigen, dass Scrum und Kanban sinnvoll kombiniert werden können.

Es wurden WiP („Work in Progress“) Limits als Teil des Konzepts eingeführt. Durch diese Begrenzung der Task-Anzahl pro Bereich eines Workflows können Blockierungen und Engpässe frühzeitig identifiziert werden und Gegenmaßnahmen eingeleitet werden. Die Ergebnisse der vorliegenden Arbeit zeigen, dass für aussagekräftige WiP Limits ebenfalls funktionierende Feedback Loops eingesetzt werden müssen. Diese Feedback Loops sollen von verschiedener Dauer sein: Vom „Pair Programming“ (Feedback innerhalb von Sekunden) bis zum Sprint (Feedback innerhalb von Wochen). Da Kanban jedoch nicht wie Scrum mit Commitment arbeitet, wurde für die äußerste Feedback Loop im Zuge des Konzepts der Arbeit ein Commitment-loser „Kanban-Sprint“ entwickelt, der auch im dynamischen Kanban Prozess einen Platz für regelmäßige Retrospektiven und Reviews schafft. Weiters wird das Ende des Kanban-Sprints zur Überprüfung von Metriken benutzt (beispielsweise wöchentlich oder zweiwöchentlich) und WiP Limits werden entsprechend nach oben oder unten korrigiert.

Ein weiteres Ergebnis stellt die Einführung des Kanban-Boards dar, das jeden Schritt des Workflows visualisiert. In der Praxisphase wurde folgende Unterteilung des Boards gewählt:

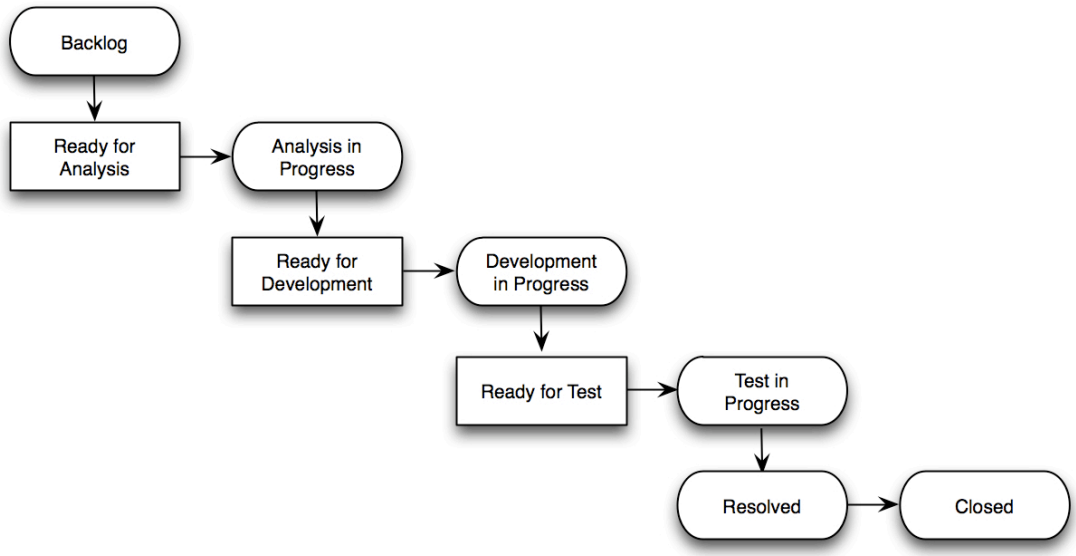


Abbildung 21: Kanban Workflow

Das Kanban-Board in Abbildung 21 zeigt den gesamten Lebenszyklus eines Arbeitspakets. Durch die Darstellung jedes Arbeitsschritts in einem erweiterten Kanban-Board verfügt jeder Projektmitarbeiter, egal ob Stakeholder, Product Owner, Analytiker, Entwickler oder Tester, zu jedem Zeitpunkt im Projekt über ein Abbild des aktuellen Projektstatus. Die Ergebnisse zeigen, dass dadurch Engpässe leichter behoben werden können und die Möglichkeit besteht sich entwickelnde Blockierungen frühzeitig zu entdecken und ihnen entgegen zu wirken.

Das Kanban-Board wurde ebenfalls durch „Queues“ erweitert, die in Abbildung 21 als Rechtecke dargestellt werden. Queues sind eigene Spalten des Boards, mit deren Hilfe natürliche Wartezeiten in einem Prozess visualisiert werden können. Beispielsweise kommt es zwischen Analyse und Entwicklung oder Entwicklung und Test in der Regel zu Wartezeiten, die bei vielen Prozessmodellen jedoch nicht beachtet werden. Die Visualisierung im Board als Teil des Workflows ist der erste Schritt, um diese Wartezeiten minimieren zu können. Das WiP Limit unterstützt diesen Vorgang, indem die Limits für Queues gering gehalten werden. Dadurch kommen überdurchschnittlich lange Wartezeiten zwischen den drei Bereichen „Analyse“, „Entwicklung“ und „Test“ schnell zum Vorschein und Gründe und Ursachen können ermittelt werden.

Weitere Erkenntnisse der Arbeit zeigen, dass Board-orientierte Meetings die Kommunikation effizienter machen können. Durch Betrachtung des Boards in täglichen Standup Meetings kann die aktuelle Projektsituation besprochen werden. Board-orientierte Standup Meetings können überdies auch in größeren Projektteams kurz und zielgerichtet bleiben. Auch die Retrospektive beziehungsweise das Review am Ende des Kanban-

Sprints kann durch die gemeinsame Betrachtung des Boards effizienter gestaltet werden.

Einige Aspekte konnten im Rahmen der vorliegenden Arbeit nicht behandelt werden: Multiprojektmanagement in Bezug auf die Kombination von Scrum und Kanban stellt beispielsweise einen weiteren interessanten Gesichtspunkt dar. Auch die Erweiterung des aktuellen Konzepts in Hinblick auf örtlich verteilte Projektteams würde eine aussichtsreiche Forschungsfrage darstellen.

Zusammenfassend lässt sich sagen, dass das erarbeitete Konzept der vorliegenden Arbeit zur Kombination von Scrum und Kanban richtungsweisend ist. Das Konzept kann grundsätzlich in beliebigen IT-Projekten Anwendung finden. Ein schneller Einstieg ist mit vorhandener Scrum-Erfahrung möglich. Die Erkenntnisse dieser Arbeit unterstreichen, dass Kanban und Scrum nicht nur theoretisch sinnvoll kombiniert werden können, sondern das erstellte Konzept auch in der Praxis zu messbaren Verbesserungen führt.

Literaturverzeichnis

[Ambl09] S.W. Ambler: Scaling Agile Software Development Through Lean Governance. In SDG '09 Proceedings of the 2009 ICSE Workshop on Software Development Governance, May 2009.

[Ande10] D.J. Anderson: Kanban. Successful Evolutionary Change for Your Technology Business. Blue Hole Press, 2010.

[BaGr94] V.R. Basili, S. Green: Software process evolution at the SEL. IEEE Software; Vol. 11, 1994.

[Basi85] V.R. Basili: Quantitative evaluation of software methodology. Tech. rep./ University of Maryland; TR-1519, 1985.

[BrBe11] E.J. Braude, M.E. Bernstein: Software Engineering. Modern Approaches. Wiley; 2. Auflage, 2011.

[Beck01] K. Beck: The Agile Manifesto. http://www.agilemanifesto.org, 2001.

[Beck03] K. Beck: Test Driven Development By Example. Addison-Wesley Professional, 2003.

[Benn95] E.M. Bennatan: Software Project Management. A Practicioner's Approach. McGraw-Hill; 2. Auflage, 1995.

[BeRa03] S. Bergström, L. Raberg: Adopting the Rational Unified Process. Success with the RUP. Addison-Wesley, 2003.

[Boeh88] B.W. Boehm: A Spiral Model for Software Development and Enhancement. In IEEE Computer, Mai 1988.

[BrMc00] W.J. Brown, H.W. McCormick III, S.W. Thomas: AntiPatterns in Project Management. Wiley, 2000.

[Bröh93] A.-P. Bröhl, W. Dröschel: Das V-Modell. Oldenbourg Verlag, 1993.

[Cohn05] M. Cohn: Agile Estimating and Planning. Prentice Hall, 2005.

[Cohn09] M. Cohn: Succeeding with Agile. Software Development Using Scrum. Addison-Wesley; 1. Auflage, 2009.

[DeLa06] E. Derby, D. Larsen: Agile Retrospectives: Making Good Teams Great. Pragmatic Programmers, 2006.

[DoKl05] C. Dogs, T. Klimmer: Agile Software-Entwicklung kompakt. Mitp-Verlag; 1. Auflage, 2005.

[EsMe03] A. Essigkrug, T. Mey: Rational Unified Process kompakt. Spektrum, 2003.

[Glog08] B. Gloger: Scrum. Produkte zuverlässig und schnell entwickeln. Hanser, 2008.

[Green02] R.K. Greenleaf: Servant Leadership. A Journey into the Nature of Legitimate Power and Greatness. Paulist Press; 25th Anniversary Edition, 2002.

[Green10] D.R. Greening: Enterprise Scrum. Scaling Scrum to the Executive Level. In HICSS '10 Proceedings of the 2010 43rd Hawaii International Conference on System Sciences, January 2010.

[HeBl10] W. Henning, W.G. Bleek: Agile Softwareentwicklung. Dpunkt; 2. Auflage, 2010.

[High04] J. Highsmith: Agile Project Management. Creating Innovative Products. Addison-Wesley; 1. Auflage, 2004.

[HrRu09] P. Hruschka, C. Rupp, G. Starke: Agility kompakt. Tipps für erfolgreiche Systementwicklung. Spektrum; 2. Auflage, 2009.

[IBM10|] IBM Rational Software, http://www.ibm.com/software/de/rational, 9.8.2010.

[JaSu09] A. Janes, G. Succi: To Pull or Not to Pull. In OOPSLA '09 Proceeding of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and application, October 2009.

[KnSk10] H. Kniberg, M. Skarin: Kanban and Scrum. Making the Most of Both. C4Media, 2010.

[Kruc00] P. Kruchten: The Rational Unified Process: An Introduction. Prentice Hall; English Edition, 2000.

[KuDi07] M. Kuhrmann, N. Diernhofer: Software Life Cycle. Management und Entwickung. Entwickler.press, 2007.

[Lada08] C. Ladas: Scrumban. Essays on Kanban Systems for Lean Software Development. Modus Cooperandi Press, 2008.

[Lehm80] M.M. Lehman: Programs, life cycles, and laws of software evolution. Proceedings of IEEE Special Issue on Software Engineering; Vol. 68, 1980.

[PeWo09] K. Petersen, C. Wohlin: A comparison of issues and advantages in agile and incremental development between state of the art and an industrial case. Journal of Systems and Software; Vol. 82, Issue 9, 2009.

[PeWo10] K. Petersen, C. Wohlin: Software process improvement through the Lean Measurement (SPI-LEAM) method. Journal of Systems and Software; Volume 83, July 2010.

[PoPo04] M. Poppendieck, T. Poppendieck: Lean Software Development. An Agile Toolkit.Addison-Wesley; 4. Auflage, 2004.

[Pich08] R. Pichler: Scrum. Agiles Projektmanagement erfolgreich einsetzen. Dpunkt; 1. Auflage, 2008.

[Royc70] W.W. Royce: Managing the Development of Large Software Systems: Concepts and Techniques. In IEEE WESCON, August 1970.

[Royc99] W.Royce: Software Project Management. A Unified Framework. Addison-Wesley; 4. Auflage, 1999.

[ScBe02] K. Schwaber, M. Beedle: Agile Software Development with Scrum. Prentice Hall, 2002.

[ScBi10] A. Schatten, S. Biffl, M. Demolsky et. al.: Best Practice Software-Engineering. Eine praxiserprobte Zusammenstellung von komponentenorientierten Konzepten, Methoden und Werkzeugen. Spektrum, 2010.

[Schw04] K. Schwaber: Agile Project Management with Scrum. Microsoft Press, 2004.

[Schw08] K. Schwaber: Scrum im Unternehmen. Microsoft Press, 2008.

[ScPi06] I. Schnabel, M. Pizka: Goal-Driven Software Development. SEW. IEEE Computer Society, 2006.

[Somm07] I. Sommerville: Software Engineering. Pearson; 8. Auflage, 2007.

[TaOh09] Taiichi Ohno: Das Toyota-Produktionssystem. Campus Verlag; Neuauflage, 2009.

[Vers00] G. Versteegen: Projektmanagement mit dem Rational Unified Process. Springer, 2000.

[Vers02] G. Versteegen: Software-Management. Beherrschung des Lifecycles. Springer, 2002.

[Wird09] R. Wirdemann: Scrum mit User Stories. Hanser, 2009.

[WoJo92] J.P. Womack, D.T. Jones, D. Roos: Die zweite Revolution in der Autoindustrie. Konsequenzen aus der weltweiten Studie aus dem Massachusetts Institute of Technology. Campus Verlag; 5. Auflage, 1992.

Anhang

Die Interviews beziehen sich häufig auf ein großes Projekt, das aus Vertraulichkeitsgründen als „Projekt A“ bezeichnet wird im Anhang.

Interview-Leitfäden

Die Interviews wurden semistrukturiert durchgeführt, das heißt mit Hilfe einer groben Gliederung von Punkten, wobei jedoch mehr auf ein fließendes Gespräch geachtet wurde als auf eine genaue Beantwortung aller Fragestellungen. Die Reihung der Interview-Leitfäden ist chronologisch entsprechend der Abhaltung der Interviews.

„Kick Off“ mit Geschäftsführer und Scrum Master

Wie sieht das Projektumfeld derzeit aus?

Wo und wie findet Scrum derzeit Anwendung?

Findet Kanban derzeit Anwendung?

Welche Projekte/Daten können für die Diplomarbeit genutzt werden?

Wie sieht die weitere Vorgehensweise aus?

Interview mit Scrum Master

Waren Sie von Anfang an bei „Projekt A“ dabei?

Was war Ihre genaue Aufgabe im Zuge von „Projekt A“?

War bei „Projekt A“ immer schon Scrum im Gespräch?

- Falls ja, warum?

Wie hat sich Scrum bewährt bei „Projekt A“?

- Positives?
- Negatives?

Wann und wie ist Kanban ins Gespräch gekommen?

Wie wurde der Übergang von Scrum zu Kanban vollzogen?

- Was wurde angepasst?
- Was kam neu?
- Gab es eine Änderung in den unterstützenden Tools?
- Haben sich die Standup-Meetings verändert?

Was sind die Erfahrungen mit Kanban?

- Positives?
- Negatives?

Wie war das Feedback beziehungsweise was hat sich geändert...

- ...für die Entwickler?
- ...für das Projektmanagement?
- ...für die Qualitätssicherung?
- ...für dich persönlich?

Wie wird es Ihrer Meinung nach in Zukunft weitergehen – wieder eine Mischung aus Scrum und Kanban?

Interview mit Geschäftsführer

Was war Ihre Funktion bei „Projekt A“? Waren Sie direkt involviert?

Wie war der Prozess bei „Projekt A“ zu Beginn und warum?

Wann wurde Scrum eingeführt und warum? Auf wessen Initiative hin?

Wie ist die Sicht der Geschäftsführung zu Scrum?

Wie wurde der Übergang von Scrum zu Kanban vollzogen?

Was sind die Erfahrungen mit Kanban?

- Positives?
- Negatives?
- Aus Sicht der Geschäftsführung?

Wie war das Feedback auf Kanban?

- Von Entwicklern?
- Vom Projektmanagement?
- Von der Qualitätssicherung?
- Persönliches Feedback?

Wie soll es in Zukunft weitergehen? Scrum, Kanban oder eine Mischform?

Interview mit Entwickler

Kurze Einführung in Diplomarbeits-Thema

Seit wann sind Sie bei „Projekt A“ dabei?

Wie hat der Software-Prozess zu Beginn ausgesehen bei „Projekt A“?

Wann und warum wurde Scrum eingeführt?

- Auf wessen Initiative hin? Wie groß war der Beitrag der Entwickler zur Umstellung?
- Was hat sich verändert bzw. verbessert? Rollen? Tools?

Ab wann hat es Probleme gegeben mit Scrum bei „Projekt A“?

- Warum?
- Auf wessen Initiative hin kam es zu einer weiteren Umstellung? Wie groß war der Beitrag der Entwickler zu dieser Umstellung?
- Was hat sich verändert bzw. verbessert? Rollen? Tools?

Wie hat sich das Kanban-Board bewährt?

Wie hat sich die Limitierung des Work in Progress bewährt?

Wie ist Ihr Feedback...

- allgemein zum „Projekt A“-Verlauf?
- zu Scrum?
- zu Kanban?

Was halten Sie in zukünftigen Projekten für die vernünftigste Lösung? (in Bezug auf die Verwendung von Scrum bzw. Kanban)

Interview mit Leitung Qualitätssicherung

Kurze Einführung in Diplomarbeits-Thema

Seit wann sind Sie bei „Projekt A“ dabei?

Was war Ihre Aufgabe im Zuge von „Projekt A“ und wie hat sie sich verändert?

Wie war die Qualitätssicherung zu Beginn?

Wie ist die Scrum-Einführung bei „Projekt A“ vor sich gegangen und wurde sie von der Qualitätssicherung gewünscht?

- Positives?
- Negatives?
- Veränderungen in der Qualitätssicherung?

Wann und wie ist Kanban ins Gespräch gekommen?

Wie wurde der Übergang von Scrum zu Kanban vollzogen?

Was sind die Erfahrungen mit Kanban und wie hat sich die Arbeit der Qualitätssicherung verändert?

Wie wird es in Zukunft weitergehen? Soll es von Seite der Qualitätssicherung Änderungen geben in Bezug auf Scrum/Kanban?

- Wenn ja: prozessbezogen oder genereller Natur?

Interview mit Mitarbeiter Qualitätssicherung

Kurze Einführung in Diplomarbeits-Thema

Seit wann sind Sie bei „Projekt A“ dabei?

Wie hat der Software-Prozess zu Beginn ausgesehen bei „Projekt A“?

Wie hat die Qualitätssicherung zu Beginn ausgesehen bei „Projekt A“?

Wurde die Scrum-Einführung von den Mitarbeitern der Qualitätssicherung gewünscht?

Wie sind die Erfahrungen mit Scrum in der Qualitätssicherung?

Wie war der Umstieg auf Kanban bei „Projekt A“? Gab es Änderungen für die Qualitätssicherung?

Was ist Ihr Feedback auf Scrum bzw. Kanban?

Welcher Prozess ist in Zukunft zu bevorzugen und was kann bei der Qualitätssicherung diesbezüglich verbessert werden?

Interview mit Analytiker

Kurze Einführung in Diplomarbeits-Thema

Seit wann sind Sie bei „Projekt A“ dabei?

Was ist Ihre Aufgabe bei „Projekt A“?

Wie ist die Aufteilung mit „GF2“ bzw. „LQS“ als Product Owner? Warum gibt es diese Aufteilung?

Wie hat sich Ihre Aufgabe entwickelt/verändert?

- bei klassischem Projektmanagement?
- bei Scrum?
- bei Kanban?

Wie direkt waren Sie in den Prozess involviert? Kam für Sie ein Punkt, wo Scrum nicht mehr funktioniert hat?

Was ist Ihr Feedback?

- zu Scrum?
- zu Kanban?
- generell zum Projektverlauf?

Denken Sie, dass Scrum bzw. Kanban in zukünftigen Projekten sinnvoll ist?

Interview mit Product Owner

Was war Ihre Funktion bei „Projekt A“? Product Owner und Eskalationsmanager?

Entwicklung von Scrum-Prozess:

- Auf wessen Initiative hin eingeführt?
- Wie eingeführt`
- Wann?
- Wie hat es sich bewährt?

Aufgeteilte Product Owner Rolle in „Projekt A“:

- Warum ist die Product Owner Rolle aufgeteilt in „Projekt A“?
- Wie funktioniert die Aufteilung mit „Analytiker“ und „LQS“?
- Warum ist die Aufteilung gut/schlecht so?
- In zukünftigen Projekten wieder aufgeteilt?

Wann und warum ist Kanban ins Gespräch gekommen?

- Wie wurde der Übergang vollzogen?
- Was hat sich verändert?
    - Für Projektmanagement?
    - Für Entwickler?
    - Für Qualitätssicherung?

    - Für die Rolle des Product Owners?
    - Wo am stärksten?
- Was sind die Erfahrungen mit Kanban?
- Wie hat sich das Kanban-Board bewährt?
- Wie hat sich die Limitierung des Work in Progress bewährt?
- Was ist das Feedback zu Kanban, dass du bekommen hast? Auch aus Sicht der Geschäftsführung?

Wie wird es in Zukunft weitergehen, Scrum oder Kanban?