Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterarbeit

Matthias Holsten

IT-basierte Unterstützung von agilen Softwareentwicklungsprozessen

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Matthias Holsten

IT-basierte Unterstützung von agilen Softwareentwicklungsprozessen

Masterarbeit eingereicht im Rahmen der Masterprüfung

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

Betreuender Prüfer: Prof. Dr. Luck
Zweitgutachter: Prof. Dr. Zukunft

Eingereicht am: 13. Februar 2015

Matthias Holsten

Thema der Arbeit
IT-basierte Unterstützung von agilen Softwareentwicklungsprozessen

Stichworte
verteiltes Pair Programming, verteilte Teams, Werkzeugkette, Agile Softwareentwicklung

Kurzzusammenfassung
Agile Softwareentwicklung wird bereits seit einigen Jahren betrieben und ist sehr verbreitet. Allerdings ist die IT-Unterstützung von Entwicklern nicht immer zufriedenstellend. Insbesondere unter erschwerten Rahmenbedingungen, wie zum Beispiel im Falle verteilter Teams, kann die IT-Unterstützung optimiert werden. Die vorliegende Arbeit beschreibt in einem ersten Schritt, wie die IT-Unterstützung von Entwicklern bisher durchgeführt wurde, um sich in einem zweiten Schritt der Optimierung für den Spezialfall verteilter Teams zu widmen.

Matthias Holsten

Title of the paper
IT-based support of agile software development processes

Keywords
distributed pair programming, distributed teams, toolchain, agile software development

Abstract
Agile Software development has become a common tool in the software developing industry. Nevertheless under certain conditions the IT-support may be unsatisfying, as in the case of distributed teams. This thesis shows in a first step how software development has been implemented so far. In a second step it will focus on distributed teams and how to optimize the software development procedure in this special case.

Inhaltsverzeichnis

1 Einleitung

Die agile Softwareentwicklung ist nicht mehr als jung zu bezeichnen. Ihre Anfänge lassen sich bis in die 1990er Jahre zurückverfolgen. Der erste Durchbruch kam 1999 durch das Buch *Extreme programming explained*[Bec99] von Kent Beck. Ein weiterer Vertreter der agilen Softwareentwicklung ist das *Framework Scrum*. Dieses ist auch nicht viel jünger als *Extreme Programming* (XP). Das erste Buch zu Scrum erschien 2 Jahre nach Kent Becks *Extreme programming*-Anstoß, Ken Schwaber und Mike Beedle veröffentlichten *Agile Software Development with Scrum*[SB01].

Trotz der langen Zeit, in der diese Methoden schon eingesetzt werden, erhalten nicht alle Entwickler eine optimale Unterstützung in ihrer Arbeit. Das hat verschiedene Gründe. Der erste ist, dass jeder Entwickler individuelle Ansprüche an die IT-Unterstützung in einem agilen Prozess stellt und ganz eigene Bedürfnisse mitbringt. Ein Zweiter Grund ist, dass die Regeln und die IT-Unterstützung in neuen Team-Verbünden unterschiedlich sind. Als dritter Grund ist zu nennen, dass jedes Unternehmen und jedes Projekt seine eigenen Rahmenbedingungen und Spielregeln besitzt.

Diese Arbeit wird den Fokus auf Softwareentwickler, die sich in besonderen Rahmenbedingungen bewegen, setzen. Dabei wird die Fragestellung behandelt: Wie können Softwareentwickler in einem Scrum Team am effizientesten arbeiten und wie kann die derzeitige Technik sie dabei am besten unterstützen. Diese Fragestellung wird erarbeitet am Beispiel eines Deutschen Unternehmens mit der Besonderheit, dass Teams auf zwei Standorte verteilt arbeiten.

Verteilte Teams bringen eine besondere Brisanz in den agilen Prozess, denn dieser lebt von der direkten schnellen Kommunikation, bei dem die Entwickler möglichst nahe beieinander sitzen. *So wurden durch das Fehelen von persönlicher Kommunikation einige Schwierigkeiten in der Prozess Koordination festgestellt* (bspw.: [KMAM10], [SSR09], [SSK+09]).[AIG12]

Im zweiten Kapitel wird der Wandel des Softwareentwicklungsprozesses, vom eher starren Wasserfall Modell zur agilen Softwareentwicklung, vorgestellt. Im anschießenden Kapitel werden Arbeiten eingeführt, mit dessen Hilfe die Prozessunterstützung des in Kapitel 4 beschriebenen Beispiel-Prozesses verbessert werden soll. Abschließend stellt Kapitel 5 die Analyse und die Verbesserung der Prozessunterstützung dar.

2 Vorgehensmodelle

Diese Arbeit beschäftigt sich mit der Unterstützung von *agilen Softwareentwicklungsprozessen*. Für ein besseres Verständnis führt dieses Kapitel schrittweise in die agile Entwicklung ein. Dabei werden zuerst grundlegende Vorgehensmodelle der Softwareentwicklung vorgestellt und anschließend ein konkrekter agiler Ansatz, *Scrum*, näher betrachtet.

2.1 Vorgehensmodelle der Softwareentwicklung

Für die Erstellung von Softwareprodukten sind über Jahrzehnte verschiedene Vorgehensmodelle konzipiert und angewandt worden. Die Entwicklung von Software ist komplex, selten linear und wird durch viele Faktoren beeinflusst, die nicht immer mess- und steuerbar sind. Mit dem Fortschritt der Informationstechnologien haben sich auch die Vorgehensweisen von Softwareprojekten geändert. Nachfolgend werden drei grundlegende Modelle vorgestellt, die unter anderem das Fundament der agilen Vorgehen bilden.

2.1.1 Wasserfall-Modell

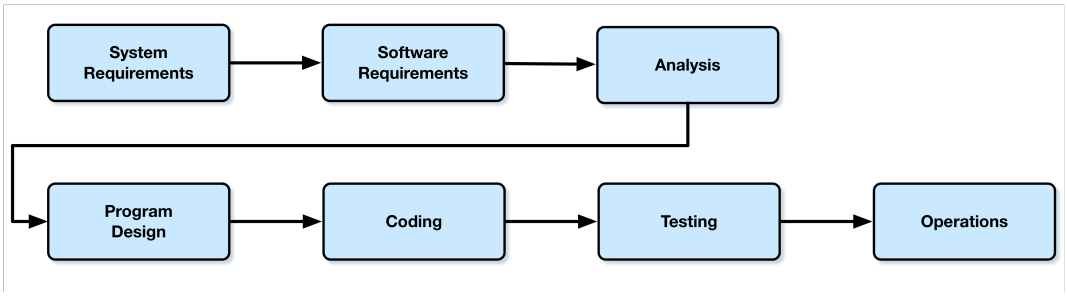

Abbildung 2.1: Wasserfall-Modell nach Royce[Roy87]

Das Wasserfall-Modell, oder auch Softwarelebenszyklus genannt, ist in seiner einfachen Art ein lineares Vorgehensmodell (siehe Abbildung 2.1). Dabei gibt es für jede Phase des Modells konkret definierte Start- und Endpunkte mit zugehörigen Ergebnissen (meist Dokumente). Royce definierte in seinem Modell grundlegende Entwicklungsaktivitäten [Som07]:

System Requirements – In dieser Phase werden die Dienstleistungen, Einschränkungen und Ziele des Systems zusammen mit dem Kunden erarbeitet und dokumentiert. Dabei entstehen das Lastenheft, die Projektkalkulation und der Projektplan.

Software Requirements – Hier wird die grobe Systemarchitektur spezifiziert. In einem Softwareentwurf werden grobe abstrakte Softwaresysteme beschrieben und ihre Beziehung zueinander festgelegt.

Analysis – In der Definitionsphase wird das sogenannte Pflichtenheft erstellt, in dem die Softwarearchitektur weiterhin verfeinert wird und in der Regel ein Vertragsbestandteil zwischen Kunde und Softwarelieferant ist. Weiterhin können Produktmodell, GUI-Modell und evtl. schon Benutzerhandbuch erstellt werden.

Program Design – Die Entwurfsphase erstellt ein detaillierten Softwareentwurf auf Basis der vorherigen Phase und beschreibt kleine Softwarekomponenten und dessen Beziehungen zueinander (bspw. durch Struktogramme).

Coding – Die entworfenen Softwarekomponenten werden nun implementiert und bereits durch einen Komponententest auf die Einhaltung der Spezifikationen geprüft.

Testing – Die einzelnen Programmteile werden zusammengeführt und als Ganzes getestet. Nach dem Test wird das Softwaresystem an den Kunden ausgeliefert.

Operations – Das fertige Produkt wird installiert und zum Gebrauch freigegeben. Zum Betrieb gehört auch die Wartung der Software, in der Fehler behoben werden oder die Implementierung verbessert wird. Neu entdeckte Anforderungen führen meist dazu, dass alle oder einige Phasen erneut durchlaufen werden müssen.

Als Vorteil ist mit Sicherheit die Einfachheit des Modells zu nennen. Sowohl dem Kunden, als auch dem Entwicklungsteam ist dieses Vorgehen einfach erklärt. Jede Phase ist präzise zu anderen Phasen abgegrenzt und bietet somit eine einfache Planung und Kontrolle. Sofern die Anforderungen von Anfang bis Ende stabil bleiben und eine klare Abschätzung der Kosten und des Umfangs erfolgt, kann das Wasserfall-Modell effizient sein [Wik15].

Vor allem die starre Aufteilung des Projekts in die verschiedenen Phasen stellt aber auch den größten Nachteil des Wasserfallmodells dar. Die frühe Verpflichtung erschwert in späten Phasen Anforderungswünsche des Kunden zu ändern oder neue Anforderungen aufzunehmen. Außerdem werden früh „eingeschleppte“ Fehler in jede weitere Phase mitgenommen und können das Problem vergrößern. Die Behebung solcher Fehler kann oftmals nur noch durch provisorische Lösungen behoben werden oder führen sehr spät zu sogenannten *Big Bangs* [Som07].

2.1.2 Evolutionäre Entwicklung

Die evolutionäre Softwareentwicklung orientiert sich nicht nur namentlich an *Charles Darwins* Evolutionstheorie. In Anlehnung an die Evolutionsbiologie werden die Anforderungen des Kunden nach und nach spezifiziert und ggf. auch verändert. Dafür wird eine *Anfangsimplementierung* erstellt, die vom Kunden bzw. dessen Benutzer ausprobiert und kommentiert werden kann. Die aus diesem Review gewonnenen Änderungswünsche formen neue Spezifikationen, mit denen das bestehende Softwaresystem verfeinert wird und eine neue testbare Version entsteht. Es werden so viele Versionen erstellt, bis ein akzeptiertes System entstanden ist und das endgültige System ausgeliefert werden kann. Das Vorgehen, aus dem initialen Prototypen eine marktreife Software zu erstellen, wird auch *explorative Entwicklung* genannt [Som07].

Ian Sommerville [Som07] erachtet dieses Vorgehen für kleine bis mittlere Systeme (bis zu 500.000 Codezeilen) für sinnvoll. Eine große Schwierigkeit dieses Ansatzes ist nämlich die Erstellung einer stabilen, gut strukturierten und wartbaren Systemarchitektur. So kann das System immer schwieriger und nur mit großem Aufwand erweitert oder geändert werden.

2.1.3 Inkrementelle Entwicklung

Die Entwicklung von Software unterscheidet sich grundsätzlich von anderen Projekten. Das Projektziel ist, beispielsweise anders als beim Bau eines Hauses, häufig bei Projektanfang noch unscharf und ungenau. Das führt dazu, dass Veränderungen an der erstellten Software unumgänglich sind. So können sich während des Projekts oder des Softwarelebenszykluses

die Prioritäten der Geschäftsführung ändern, durch neue Technologien können Anpassungen erforderlich werden oder die Prozesse des Kunden haben sich verändert.

Beim Wasserfall-Modell muss der Kunden sehr früh die beauftragte Software spezifizieren - noch vor dem ersten Softwaredesign. Bei nachträglichen Änderungen an der Software kann dies Anpassungen an der Spezifikation selbst, dem Design und der Implementierung bedeuten. Die evolutionäre Entwicklung bietet eine Verschiebung der Spezifikation an, erhöht damit aber auch die Wahrscheinlichkeit einer schlechten Systemstruktur. Somit wird das System immer komplexer, schwer wartbar und möglicherweise fehleranfälliger.

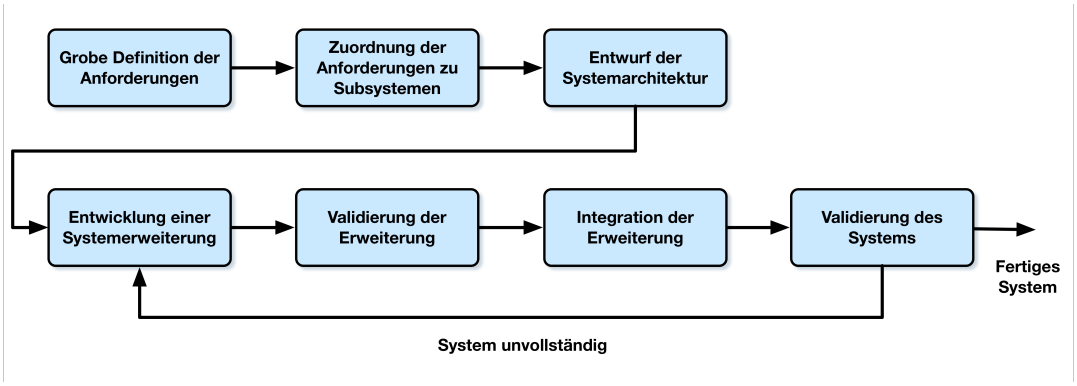

Abbildung 2.2: Inkrementelles Vorgehen

Die inkrementelle Entwicklung bietet eine Lösung zwischen diesen beiden Ansätzen. Sie bietet eine Strategie zur Termin- und Ablaufplanung womit einzelne Systemteile in unterschiedlicher Geschwindigkeit geplant und entwickelt werden können und sofort in das Gesamtsystem integeriert sind. Funktionen, die dem Kunden sehr wichtig sind, werden zuerst erstellt. Die bereits fertigen Systemteile, die *Inkremente*, können ähnlich wie ein Prototyp fungieren und dem Kunden und Projektteam erste Erfahrungen mit dem System liefern. Auf dieser Grundlage kann der Kunde die ausstehenden Systemteile neu spezifizieren und seine Anforderungen anpassen. Die Spezifikation des Gesamtsystems ist also erst abgeschlossen, wenn das letzte Inkrement spezifiert wurde. Bereits fertige Teile können nachträglich nicht mehr geändert werden. Die Phasen (bzw. *Iterationen*), in denen die Inkremente erstellt werden, werden in der Regel zeitlich und nicht funktionell festgelegt. Die Wikipedia [Wik14a] unterteilt das inkrementelle Vorgehensmodell in die folgenden, informellen Phase:

- *„Die Einleitungsphase ermittelt Projektrahmen, Risikofaktoren und (funktionelle wie nicht-funktionelle) Anforderungen in geringer aber ausreichender Tiefe, um eine grobe Schätzung des Arbeitsaufwandes zu erlauben.*

- *Die Ausarbeitungsphase liefert eine funktionsfähige Architektur, die die höchsten Risikofaktoren entschärft und die nicht-funktionellen Anforderungen erfüllt.*

- *Die Konstruktionsphase füllt die Architektur schrittweise mit produktionsreifem Code aus, der durch Analyse, Entwurf, Umsetzung und Prüfung der funktionellen Anforderungen entsteht.*

- *Die Übergangsphase überführt das System endgültig in den Produktiveinsatz.“*

Da das Projekt „*organisch*“ [Wik14a] wächst, die Spezifikation aber häufig ein Teil des Vertrags zwischen Auftraggeber und Auftragnehmer ist, kann dieser Umstand zu einem Problem der inkrementellen Entwicklung werden. Besonders große Kunden (bspw. Behörden) können meist nur schwer ihre Prozesse den agilen Softwareentwicklungsprojekten anpassen.

Agile Softwareentwicklung oder ganzheitliche agile Entwicklungsprozesse basieren auf der grundlegenden Idee der inkrementellen Entwicklung. Dabei wird zusätzlich berücksichtigt, dass Methodiken und Prozesse besonders schlank, einfach und damit flexibel sind. Diese Grundsätze, die sogenannten *Agilen Werte*, bilden dabei das Fundament und wurden im *Agile Manifesto* verankert (siehe Abschnitt 2.2.1). *Scrum* ist einer der bekanntesten agilen Ansätze und wird im folgenden Abschnitt beschrieben.

2.2 Scrum

Scrum wird als *Framework* bezeichnet und nicht als Vorgehensmodell, da es im Kern nur wenige, dafür aber notwendige Regeln vorgibt, dadurch ist Scrum sehr flexibel einsatzbar. Im Kern besteht das *Framework* aus drei Rollen, fünf Aktivitäten und drei Artefakte.

Der Ansatz von Scrum ist empirisch, inkrementell und iterativ. Er beruht auf der Erfahrung, dass viele Entwicklungsprojekte zu komplex sind, um in einen vollumfänglichen Plan gefasst werden zu können.[Wik14b]. Wie schon in der Evolutionären- und Inkrementellen Entwicklung wird bei Scrum iterativ entwickelt.

In Zyklen (genannt *Sprints*) werden Inkremente angefertigt, die dem Kunden vorgelegt werden. An diesen Inkrementen kann der Kunde die Anforderungen weiter spezifizieren. Der Unterschied zu den vorangegangenen Vorgehensmodellen besteht darin, dass die Inkremente auch an den Kunden als lauffähige Programme ausgeliefert werden können, sobald sie einen Mehrwert für ihn bieten.

Die grobe Planung geschieht im *Prduct Backlog*, sie wird zyklisch angepasst und verbessert. Die Planung für einen *Sprint*, die im *Sprint Backlog* gehalten wird, bleibt für den Zeitraum des *Sprints* stabil.

In Scrum werden meist die Anforderungen in so genannten *User Stories* definiert. Sie beschreiben eine Anforderung aus der Sicht eines Benutzers. Es wird in Umgangssprache beschrieben welche Funktionalität der Nutzer sich wünscht und warum er sie benötigt. Werden Anforderungen auf einer höheren Abstraktionebene beschrieben, werden sie *Epic* genannt.

2.2.1 Agile Manifesto

Als kleinster Nenner der agilen Vorgehensmodelle wurde das *Agile Manifest* von Ken Schwaber, Jeff Sutherland 2001 entwickelt.[KB14]

- Menschen und Interaktionen sind wichtiger als Prozesse und Werkzeuge.

- Funktionierende Software ist wichtiger als umfassende Dokumentation.
- Zusammenarbeit mit dem Kunden ist wichtiger als die ursprünglich formulierten Leistungsbeschreibungen.
- Eingehen auf Veränderungen ist wichtiger als Festhalten an einem Plan.

Es bedeutet, dass eher auf die Menschen und ihre Bedürfnisse eingegangen werden soll, als sich an Prozesse und Werkzeuge zu klammern, dass eine seitenlange Dokumentation nichts nützt, wenn die dazugehörende Software nicht funktioniert, dass die Änderungswünsche eines Kunden eine Software verbessern, auch wenn diese Anforderung nicht in der Leistungsbeschreibungen vorgesehen waren und Änderungen immer wilkommen sind.

2.2.2 Rollen

Das Scrum Team umfasst drei Rollen den *Product Owner*, den *Scrum Master* und das Entwicklungsteam. Außerhalb des Scrum Teams existieren die Stakeholder, sie haben Einfluss auf das Produkt und die Entwicklung wird für sie transparent dargelegt.

Der *Product Owner* fokussiert seine Arbeit auf den wirtschaftlichen Erfolg des Produktes, um diesen zu erreichen, erstellt und priorisiert er die Anforderungen im *Produkt Backlog*. Zur Unterstützung bei dieser Aufgabe kann er die Stakeholder und das Entwicklungsteam mit einbinden. Er muss die Wünsche und Bedürnisse der Stakeholder verstehen und sie fürs Team verständlich in *User Stories* umwandeln.

Das Entwicklungsteam setzt die von *Product Owner* priorisierten Anforderungen um. Bei der Arbeit ist das Team selbstorganisierend, kein anderer hat das Recht ihnen bei der Umsetzung der *User Stories* Vorschriften zu machen, nicht der *Product Owner* und auch nicht der *Scrum Master*. Das Entwicklungsteam trägt jedoch die Verantwortung vereinbarte Qualitätsstandards einzuhalten.[KS13]

Der *Scrum Master* ist als Coach für den Prozess und die Beseitigung von Hindernissen verantwortlich.[Wik14b] Er achtet darauf, dass die Scrum-Regeln eingehalten werden, hat jedoch keine Führungsgewalt über das Team.

2.2.3 Sprint

Ein Sprint ist ein zeitlich definierter Abschnitt, indem eingeplante Aufgaben vom Entwicklungsteam umgesetzt werden. Jeder Sprint sollte dieselbe Länge besitzen, meist zwischen zwei und vier Wochen. Das hilft dem Team einen Takt zu geben und gibt nach kurzer Zeit eine Sicherheit bei der Planung.

Vor Beginn eines Sprints wird ein *Sprint Planning* Meeting abgehalten, in diesem werden die vom *Product Owner* priorisierten *User Stories* geschätzt und dem *Sprint Backlog* hinzugeführt. Das Entwicklungsteam entscheidet wie viele Aufgaben in das *Sprint Backlog* kommen, es verständigt sich jedoch darauf diese *User Stories* alle zu bearbeiten.

Jeden Tag zur selben Uhrzeit findet ein höchstens 15 minütiges Meeting statt. Es wird *Daily Scrum* oder *Stand-up* genannt. Die Entwicklungsteam-Mitglieder besprechen Aufgaben, und zwar ihre bisher erfüllten, die für den kommenden Tag und ob sie etwas in der Produktivität am letzten Tag behindert hat.

Nach Beendigung des Sprints werden zwei weitere Aktivitäten durchgeführt, das Review und die Sprint-Retrospektive. Im Review wird dem *Product Owner* und den Stakeholdern vorgestellt, welche Leistungen im Sptint erbracht wurden. Dazu werden die umgesetzten *User Stories* direkt an der entwickelten Software vorgeführt. Die Stakeholder und der *Product Owner* sind nun in der Lage einzuschätzen, ob ihre Anforderungen zur Zufriedenheit umgesetzt wurden oder es Änderungswünsche gibt.

Die Sprint-Retrospektive beschäftigt sich mit dem Entwicklungsteam und ihrer Arbeitsweise. Das Entwicklungsteam schaut rückblickend auf den letzten Sprint mit dem Fokus, ob sie ihre Arbeitsweise optimieren kann. Dabei unterstützt sie der *Scrum Master*, er ist später auch dafür verantwortlich darauf zu achten, dass die neu getroffenen Vereinbarungen eingehalten werden.

2.2.4 Pair Programming

Im *Pair Pprogramming* arbeiten zwei Entwickler zusammen an einer Aufgabe. Dabei sitzen sie nebeneinander an einem Rechner und wechseln sich beim Tippen ab. Dieses Vorgehen führt zu

einem Wissensaustausch und effizienter Kommunikation, bei der beide Entwickler das gleiche Bild der Aufgabe und des Codes bekommen.

Obwohl die Methode, mit zwei Entwicklern an einer Aufgabe zu arbeiten ineffizient erscheint, wurde gezeigt, dass die Methode für komplexe Aufgaben schnellere und bessere Ergebnisse liefert.

3 Related Work

Vergleichbare Arbeiten waren nur schwer zu finden. Es wurden einzelne Aspekte, die in dieser Arbeit betrachtet werden sollen, nicht dargestellt. So wurden nun Arbeiten ausgewählt, die stellvertretend für einzelne Aspekte stehen.

Ein Großteil der Arbeiten beschreibt, wie die Arbeitsweise von agilen verteilten Teams im Gegensatz zu nicht verteilten Teams angepasst werden müsste. Aus diesem Bereich wurde eine Arbeit ausgewählt, die 2008 mit Zahlen belegen konnte, dass ein verteiltes *Scrum* Team gleich effizient wie ein *Scrum* Team an einem Standort arbeiten kann. Das Werk *Fully Distributed Scrum: The Secret Sauce for Hyperproductive Offshored Development Teams*[SSRR08] wurde in der Firma *Xebia* erarbeitet und beschreibt ein Outsourcing Projekt nach Indien, bei dem die Hälfte des Teams in Indien und die andere Hälfte in den Niederlanden sitzt.

Als Zweites wurde ein Werk gewählt, das sich mehr auf das Zusammenspiel von Werkzeugen fokussiert hat. Es versucht eine Möglichkeit aufzuzeigen, die Kommunikation und den Überblick über das Projekt zu verbessern. Die Arbeit zeigt, dass in technischen Systemen Statusübergänge im Prozess erfolgen, die unbemerkt bleiben. Diese Statusübergänge sollen dem Team illustrieren, in welchem Stadium sich andere, gerade bearbeitete *Stories* befinden. Diese Arbeit soll zeigen, wie Tool-Unterstützung ausgereizt werden kann.

3.1 A Holistic Approach to Developing a Progress Tracking System for Distributed Agile Team

Das Team um A. Sultan von der Universität Cardiff, beschreibt in ihrer Arbeit *A Holistic Approach to Developing a Progress Tracking System for Distributed Agile Team*[AIG12] einen Ansatz, um Auswirkungen von technischen Einflüssen auf den Prozessfortschritt in agilen verteilten Teams darzustellen und bei der Durchführung zu unterstützen. Die Arbeit wird als Related Work angesehen, da das Zusammenspiel der Werkzeuge in dieser Arbeit beispiellos ist. Es wird ein System beschrieben, das die Werkzeuge so miteinander verbindet, dass Prozess-Schritte komplett automatisiert sind. Worunter auch Teile der Kommunikation fallen. A. Sultan et al. gehen bei ihrem Ansatz davon aus, dass die einzelnen Team-Mitglieder in unterschiedlichen Zeitzonen leben und arbeiten, daher eine synchrone Kommunikation nur schwer realisierbar ist.

3.1.1 Einführung und Idee

Als Begründung für die Arbeit gilt, dass in einem agilen Prozess, der sehr viele Technologiebausteine besitzt, diese Bausteine Einfluss auf den Prozessfortschritt ausüben. Ein Team, das an einem Standort sitzt, kommuniziert über diese Einflüsse direkt, bspw. im *Stand-up* Meeting. In verteilten Teams, die nicht auf synchrone Kommunikation zurückgreifen können, ist das nicht ausreichend. So zitiert Sultan et al. Mira et al. mit den Worten:

> *„emails and instant messages proved to be insufficient for maintaining the daily communication as dictated by the agile values“* [KMAM10]

Diese fehlende Kommunikation möchte der beschriebene Ansatz technologiebasiert abfangen. Es wird eine Software dargestellt, die die Ereignisse der Werkzeuge verarbeitet und die Entwickler benachrichtigt, wenn sie von diesem Ereignis betroffen sind. Als Beispiel wird angeführt, dass eine Änderung im Source Code die Erstellung eines Akzeptanztests erfordert. Auf diese Bedingung könnte von einem Projektmanagement-Werkzeug nicht hingewiesen werden, weiles den Zusammenhang nicht prüfen kann.

Unit-Testing (UT)	Acceptance-Testing (AT)	Continuous Integration (CI) & Releasing	Source Code Versioning
-Create a new UT -Update existing UT -Delete UT -Run UT	-Create a new AT -Update existing AT -Delete AT -Run AT	-Perform integration -Releasing	-Create an artefact -Modify an artefact -Delete an artefact

Tabelle 3.1: Beeinflussung des Prozesses durch die Technik (aus [AIG12] Seite 504)

3.1.2 Konzept und Umsetzung

Das Konzept der Arbeit basiert darauf, dass bestimmte technische Ereignisse überwacht werden, aus denen Schlüsse gezogen und die Prozess-Steuerung gelenkt wird. Die Schritte, die zu überwachen sind, werden in der Tabelle 3.1 dargestellt.

Anhand des eines Beispiels aus der Arbeit (Abbildung 3.1) kann am besten die Arbeitsweise des Systems gezeigt werden. Das Beispiel zeigt Prozessschritte des *check-in* in die Versionsverwaltungssoftware. Anstatt aktiv ein *check-in* durchzuführen, wird ein Antrag auf ein *check-in* gestellt. Das System prüft die Bedingungen, ob diese zulässig sind oder der Mitarbeiter nacharbeiten muss und führt den *check-in* durch, wenn die Bedingungen erfüllt sind.

Folgende Schritte (zu verfolgen in Abbildung 3.1 aus [AIG12]) führt das System durch:

- Es prüft, ob der neue Quellcode mit Unit-Tests abgedeckt ist.
- Es prüft, ob alle Unit-Tests korrekt durchlaufen.
- Es prüft, ob der Quellcode auf einem älteren Stand basiert und nun erst integriert werden muss.
  - Falls ja, prüft es, ob Konflikte bei der Integration auftreten würden.
  - Falls nein, integriert es den Quellcode.
- Es erhöht die Versionsnummer.

- Es prüft, ob andere *Stories*, die kurz vor dem Abschluss stehen (gerade im Akzeptanz- oder Unit-Test Status) durch die Integration beeinflusst werden.
  - Interagiert mit dem Entwickler, falls andere *Stories* beeinflusst werden.
  - Benachrichtigt alle Entwickler und Tester über die Quellcode-Änderungen die ihre *Story* beeinflusst.
- Benachrichtigt alle betroffenen Entwickler, falls Bedingungen nicht eingehalten wurden und Verbesserungen durchgeführt werden müssen.

3.1.3 Fazit

Die Arbeit beschreibt, wie eine Verbindung von Softwareentwicklungswerkzeugen den Softwareentwicklungsprozess für agile verteilte Teams unterstützt und vereinfacht. In Bezug auf diese Arbeit ist besonders die hohe Kommunikation zu betrachten, die das entworfene System sogar eigenständig vornimmt. Die Entwickler werden in folgenden Fällen benachrichtigt:

- Wenn sie keine Unit-Tests geschrieben haben.
- Wenn sie keine Akzeptanztests geschrieben haben.
- Wenn Tests aufgrund ihrer Änderung nicht mehr fehlerfrei laufen.
- Wenn der Quellcode nicht integrierbar ist.
- Wenn Abhängigkeiten zu anderen User Stories bestehen.
- Wenn User Stories mit Abhängkeiten zur eigenen bearbeitet wurden.
- Wenn eine User Story abgeschlossen wurde.

Die automatischen Prüfungen, ob Tests vorhanden sind, erwirtschaften einen erheblichen Mehrwert des Systems. Es spart Zeit bei einem Review des Quellcodes. Trotz der technischen

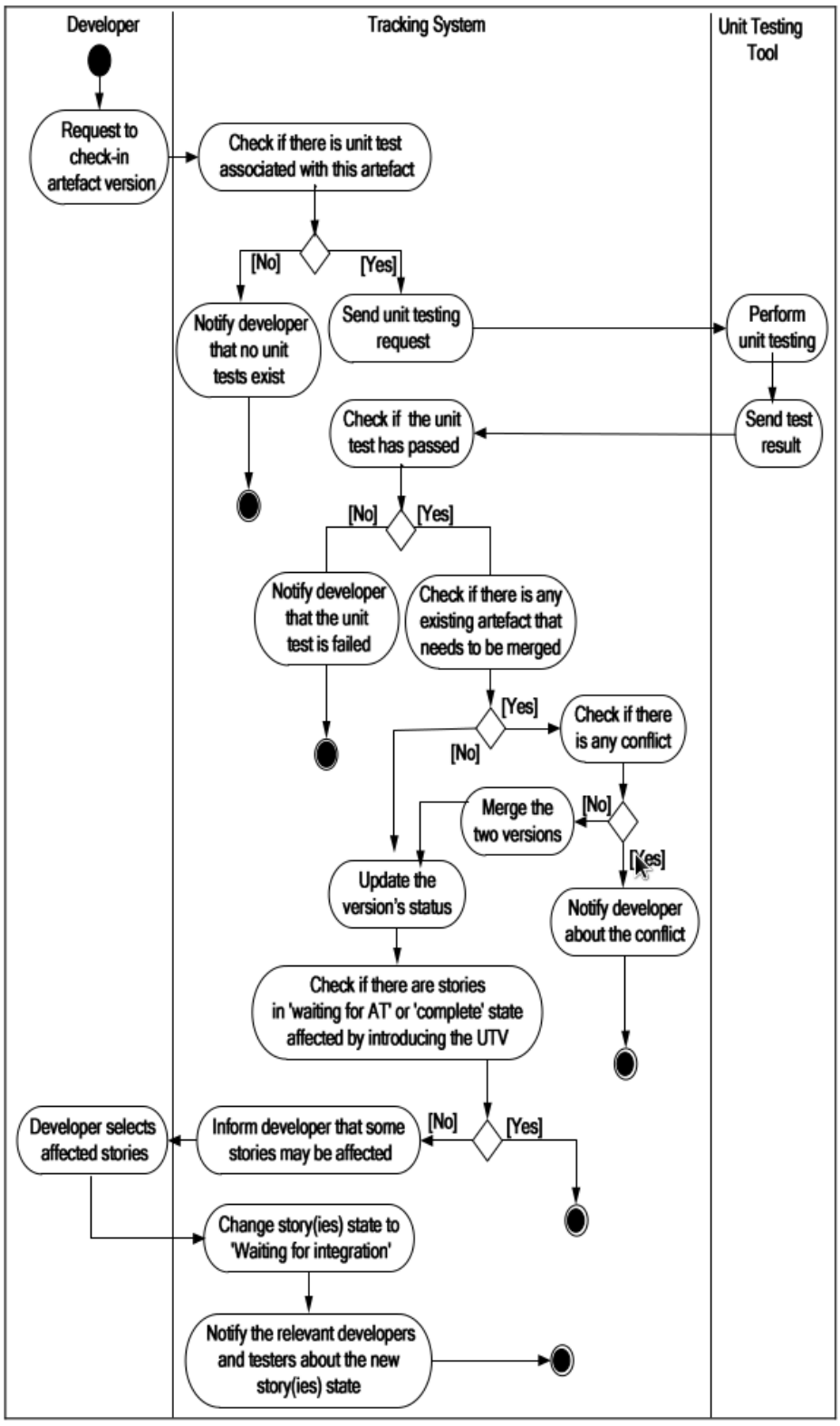

Abbildung 3.1: Tracking System der Arbeit [AIG12]

Prüfung ist der Wissensaustausch über Code-Reviews nicht zu vernachlässigen, da dieser nicht zu automatisieren ist.

Jedoch sehe ich noch ein Verbesserungspotential dieser Arbeit in der Kommunikation. Die Flut an E-Mails, die ein solches System bei einer etwas größeren Gruppe verursachen kann, schmälert die Übersichtlichkeit eventuell in kurzer Zeit. Wie Hodges in [Hod13] schreibt, ist E-Mail ein schlechtes Medium zur Kollaboration.

3.2 Fully Distributed Scrum: The Secret Sauce for Hyperproductive Offshored Development Teams

Die Arbeit *Fully Distributed Scrum: The Secret Sauce for Hyperproductive Offshored Development Teams* wurde von vier Mitarbeitern der Firma *Xebia* aus den Niederlanden geschrieben. Die Firma *Xebia* startete 2006 mit einem Projekt, das zur Hälfte mit niederländischen und zur anderen Hälfte mit indischen Entwicklern begonnen wurde[SSRR08]. Nach sechs Wochen (drei Sprints) zogen die indischen Entwickler in ihre Heimat zurück. So arbeitete das Team verteilt weiter. Die Produktivität stieg weiter an, und das Team wurde vergrößert bis es aus drei verteilten Teams und ein Team, dass nur an einem Standort tätig war, bestand. *Xebia* ließ später, nach Abschluss dieses Projekts, in diesen verteilten Teams weitere Projekte durchführen.

Es soll nicht auf die gesamte Arbeit eingegangen werden, nur die Werkzeugunterstützung des Projekts wird untersucht. Die Faktoren um Outsourcing spielen bei der Betrachtung keine Rolle.

3.2.1 Einführung und Idee

Nach dem Vorbild von anderen Unternehmen (SirsiDynix (U.S.) und Exigen Services (Russland) [Jon00]) hat es *Xebia* geschafft, Outsourcing produktiv zu nutzen und die Produktivität mit Zahlen zu belegen. Dabei arbeiteten drei *Scrum* Teams komplett verteilt, die wiederum in einem *Scrum of Scrums*[1] koordiniert wurden.

Es gab für alle Teams ein gemeinsames *Product Backlog* und vier – separat für jedes Team – *Sprint Backlogs*. Auch Teams, die verteilt gearbeitet haben, arbeiteten jeweils mit einem *Sprint Backlog*. Die Absprachen geschahen in den täglichen 15 Minuten dauernden *daily Scrum Meetings*. Diese wurden zum Arbeitsbeginn der niederländischen Teams abgehalten. Ein „verteiltes-“*Pairing* der Team-Mitglieder wurde nicht erwähnt.

[1] http://en.wikipedia.org/wiki/Scrum_(software_development)Scrum_of_scrums

3.2.2 Konzept und Umsetzung

Die Meetings in dieser Konstellation wurden immer im gesamten Team abgehalten. Dazu wurden entweder Videokonferenzen via Skype oder in separaten Meetingräumen, die mit Videokonferenz Equipment ausgerüstet wurden, genutzt. Zur Anzeige des Fortschritts brachte *Xebia* in den Meetingräumen digitalen *Burndown* Charts an. In den Teamräumen wurden die *Burndown* Charts täglich neu ausgedruckt und an die Wand gehängt[SSRR08].

In den Teams hat sich herausgestellt, dass wegen der Entfernung die direkte Kommunikation für die Teamgemeinschaft sehr wichtig ist. Besonders die Mimik des Gegenübers bei einer Videokonferenz zu sehen, schafft Nähe. Auch das *Planing Poker* (Kapitel 2.2.4) wurde über eine Videokonferenz realisiert. Die Diskussion in der Gruppe ist die Basis vom *Planing Poker*[Coh05].

Zur technischen Verwaltung des *Product* und *Sprint Backlog* wurde *ScrumWorks*[2] genutzt. Dies ist eine Web basierte Plattform um *Scrum* Projekte zu planen und zu steuern.

Zur Wissensverbreitung wurde ein gemeinsames Wiki und Mailinglisten verwandt. Digitale Whiteboards (Smartboards) unterstützten die gemeinsame Diskussion und halfen Architekturen zu entwerfen. Selbstverständlich wurde ein gemeinsames *Source Code Repository* und CI-System genutzt, um den Source Code zu verteilen und zu verifizieren bzw. zu bilden.

3.2.3 Fazit

Xebia hat gezeigt, dass ein verteiltes *Scrum*-Team die gleiche Leistung bieten kann wie ein Team, dass an einem Standort sitzt. Dabei sind die technischen Hilfsmittel, die sie zusätzlich, auf Grund des höheren Kommunikationsaufwands genutzt haben, recht gering. Für die direkte Kommunikation wurden Videokonferenzräume inklusive *Burndown chart* eingerichtet, und für die *Stand-up* Meetings wurde Skype genutzt. Als wichtiges Hilfsmittel für die verteilte Arbeit ist auch das Digitale Whiteboard zu sehen, über das die Team-Mitglieder zwischen den Standorten Zeichnen und Diskutieren konnten.

[2] http://www.collab.net/products/scrumworks

Die gemeinsame Nutzung eines Source Code Verwaltungssystems, eines Continuous Integration Servers, sowie eines Wikis wäre auch bei einem Team an einem Standort notwendig gewesen.

3.3 Zusammenfassung

Es wurden hier zwei sehr unterschiedliche Arbeiten betrachtet. Welche Erkenntnisse können aus diesen gezogen werden?

1. Projekte in verteilten Teams, die Agil arbeiten, können genau so effizient sein, wie Teams an einem Standort.

2. Die Spanne an technischer Unterstützung, die einem Team geboten werden kann, ist immens. Sie fängt an, bei den gleichen Werkzeugen, die auch ein Team, dass nicht verteilt arbeitet, benötigt, und endet bei der vollkommenen Verbindung aller Entwicklungswerkzeuge, um verborgene Fortschritte zu illustrieren.

In allen Arbeiten, die ihre Werkzeuge vorgestellt haben, wurden Wikis, Source Code Verwaltungssysteme, *Continuous Integration* Systeme und digitale *Scrum Bords* (*Product Backlog*, *Sprint Backlog*, *Taskboard*) genannt. Bis auf das digitale *Scrum Board*, sind dieses auch die Werkzeuge, die in einem nicht verteilten Team genutzt werden. Als Erweiterung für verteilte Teams ist ein Videokonferenz-System sinnvoll. Die Studie von Xebia[SSRR08] beschreibt, dass die direkte Kommunikation von Angesicht zu Angesicht sehr wichtig ist. Sofern gemeinsame Architekturen entwickelt werden, ist eine technische Unterstützung notwendig.

Die Statusübergänge, die im Verborgenen liegen und nur schwer von Mitarbeitern wahrgenommen werden, sichtbar zu machen ist ein Gewinn bringender Ansatz. Das perfekte Maß an Aufmerksamkeit zu schaffen ist jedoch bei diesem Ansatz die Kunst. Werden die Entwickler in einer zu hohen Frequenz auf einen Statusübergang hingewiesen, würde sehr wahrscheinlich der Effekt abgeschwächt. Daher ist die Benachrichtigungsart zu überdenken.

Laut dem Artikel von Flor[Flo06] kann die direkte Zusammenarbeit an Aufgaben, gerade in verteilten Teams, einen sehr hohen Mehrwert schaffen. So kann damit die Teamgemeinschaft und die Wissensverbreitung vorangetrieben werden. Jedoch wurde in diesen beiden Arbeiten *Pairprogramming* im verteilten Team nicht erwähnt. Die Arbeiten von Flor[Flo06] und von Schenk et al.[SPS14] empfehlen dieses und beschäftigen sich mit dem Thema.

4 Fallbeispiel

In diesem Kapitel wird das Fallbeispiel definiert, anhand dessen die Verbesserungsvorschläge in dieser Arbeit dargestellt werden. Die Rahmenbedingungen, das Unternehmen und die Mitarbeiterstruktur werden betrachtet, um die Problemstellung herauszuarbeiten. Eine entscheidende Rolle bei der Analyse spielt auch die Umsetzung des *Scrum*-Prozesses im Unternehmen und die bisher eingesetzten IT-Lösungen.

4.1 Rahmenbedingungen

Das Fallbeispiel handelt von einem realen Unternehmen in der Versicherungsbranche, das namentlich nicht genannt wird. Um das Beispiel besser zu verdeutlichen und die Darstellung zu erleichtern, wird das Unternehmen in der Arbeit *Dummy Versicherungs Gruppe* (Abgekürzt: *DVG*) genannt. Der Prozess und die Prozess-Verbesserungsvorschläge sollen jedoch exemplarisch für ähnlich strukturierte Unternehmen und Prozesse stehen und anwendbar sein.

4.1.1 Das Unternehmen

Die Versicherung *DVG* ist ein international tätiges Unternehmen mit zwei Direktionen in deutschen Großstädten. An beiden Standorten der Hauptverwaltung sind je über 2000 Mitarbeiter beschäftigt. Im IT-Bereich arbeiten ca. 350 Mitarbeiter, deren Aufgabenfelder sich in die Entwicklung von Softwarelösungen und den Betrieb der IT-Lösungen unterscheiden. Einige Projekte werden standortübergreifend entwickelt.

4.1.2 IT-Projekte

In der Versicherungsbranche wurde schon früh auf Digitalisierung gesetzt, bei der *DVG* ist das älteste noch laufende Programm aus den 1970er Jahren. Die Techniken aus dieser Zeit werden teilweise noch heute eingesetzt und unterstützt.

Die Softwareentwicklung dieses Unternehmens kann daher in zwei „Entwicklungsschwerpunkte“(Techniken) untergliedert werden, die Programmierung für den IBM Manframe (auch Host genannt) und für dezentrale (Client-Server basierte) Softwaresysteme. Auf dem Host werden Programme vorwiegend in den Programmiersprachen Cobol und PL/I implementiert. Für die Programmierung auf Client-Server-Systemen ist als Haus-Standard Java gesetzt.

Da vorwiegend Projekte, die für die Plattform Client-Server entwickeln, Scrum nutzen, wird in dieser Arbeit nur Scrum betrachtet. Die Trennung der Softwareentwicklung auf Host und

Client-Server-Systemen, so wie die Schwierigkeiten unaufgeschlossene Entwickler in dieses Vorgehen zu integrieren wird außer acht gelassen.

Projekte bzw. Abteilungen, die Software entwickeln, arbeiten, um die fachlichen Anforderungen umsetzen zu können, dabei mit den verschiedenen Fachabteilungen aus dem Kerngeschäft (Versicherungen) eng zusammen.

4.1.3 Welche Vorgehensmodelle werden eingesetzt?

Das Unternehmen führt in Hinsicht auf die Vorgehensmodelle eine zweigleisige Strategie. Die Mehrzahl der Projekte werden nach einem phasenorientierten Plan gelenkt. Seit 2009 wird zusätzlich eine Implementierung von *Scrum* im Haus eingesetzt. Um das *Scrum*-Vorgehen auf die Bedürfnisse und Rahmenbedingungen der Firma anzupassen und sie im Unternehmen zu verbreiten, wurde 2009 ein zweijähriges Projekt ins Leben gerufen.

Des Weiteren wird seit der Einführung von *Scrum* ein „Mischvorgehen“ in bestimmten Projekten praktiziert. So plant und arbeitet das Entwicklungsteam agil, jedoch in Richtung Auftraggeber wird ein phasenorientiertes Vorgehen gelebt. Das bedeutet, ein Release der Software wird zu Anfang geplant und mit dem Auftraggeber besprochen, der zur Testphase das erste Mal den neuen Softwarestand zu sehen bekommt. Die vereinbarten Aufgaben für das Release werden im Entwicklungsteam jedoch nach *Scrum* zerschnitten, inkrementell geplant und implementiert.

4.1.4 Stakeholder

In der *DVG* mit über 10.000 Mitarbeitern im In- und Außendienst sind Verantwortlichkeiten in viele Kompetenzbereiche aufgeteilt. Somit ist die Anzahl der Stakeholder für Projekte beträchtlich. Dieses kann durch Ideen, Anregungen und Vorlagen zur Beschleunigung des Projektes führen. Es werden jedoch auch viele, teils gegensätzliche Anforderungen und Ansprüche an ein Projekt gestellt. In diesem Abschnitt werden die verschiedenen Stakeholder und ihre Positionen kurz beschrieben.

Datensicherheit

Die Abteilung Datensicherheit ist für die Erstellung und Umsetzung der IT-Sicherheitsrichtlinie verantwortlich. Sie entscheidet nach welchen Normen und Vorgehen in Fragen der IT-Sicherheit gehandelt wird. In der *DVG* ist sie Ansprechpartner zu Fragen der IT-Sicherheit im Unternehmensverbund und somit bei fast jedem Projekt Stakeholder.

Die Abteilung Datensicherheit ist in Projekten als Ratgeber und als Sicherheitsvorgaben liefernde Instanz zu betrachten. Sie achtet auf die Einhaltung der Richtlinen und weist auf Verstöße hin.

Datenschutz

Der Datenschutz bearbeitet die Festlegungen und Weiterentwicklungen von Datenschutzgrundsätzen für das Unternehmen, Grundlage hierfür ist das Bundesdatenschutzgesetz (BDGS)[1]. Die Abteilung überwacht und unterstützt bei der Einhaltung datenschutzrechtlicher Vorschriften.

Die Abteilung Datenschutz hat gemäß den Datenschutzgrundsätzen ein Mitbestimmungsrecht bei der Erstellung oder Anschaffung von Programmen, die personenbezogene Daten bearbeiten.

Eine weitere Aufgabe ist die Genehmigung und Überwachung von Testdaten. Falls Daten zum Test von Anwendungen aus der Produktion in ein Testsystem übertragen werden sollen, muss der Datenschutz zustimmen. Somit ist der Datenschutz in vielen Projekten involviert.

Unternehmensarchitekt

Der Unternehmensarchitekt ist für die Umsetzung der Architekturmanagement-Ziele zuständig. Er berät die Projekte im Hinblick auf eine harmonische Integration in die IT-Architektur des Unternehmens.

[1] Das Gesetz ist im Internet unter diesem Link zu finden: http://www.gesetze-im-internet.de/bdsg_1990/index.html

Marketing

Die Marketing-Abteilungen in der *DVG* sind für das interne und externe einheitliche Auftreten, die Werbung und die Kommunikation mit den Außendienstpartnern verantwortlich. Sie hat zwei Funktionen bei der Softwareentwicklung. Sie dient als Schnittstelle zu den Maklern, die mit der entwickelten Software Versicherungsprodukte verkaufen sollen. Die Marketing-Abteilungen geben die Anforderungen und Probleme der Makler an das Projekt weiter.

Als zweite Funktion erstellen sie selber Anforderungen an die Software im Bezug auf Design und Verhalten. Auf Grund mehrerer Marketing-Abteilungen existieren auch unterschiedliche Anforderungen zum Design einer Anwendung in der *DVG*. Auf Projekte, die eine ausschließlich im Unternehmen genutzte Anwendung entwickeln, haben die Marketing-Abteilungen weniger Einfluss. Software, die für die Öffentlichkeit oder für den Einsatz im Außendienst produziert wird, ist zwangsläufig das Engagement größer.

4.1.5 Verteilte Teams

Die *Dummy Versicherungs Gruppe* ist in zwei Hauptstandorte unterteilt. Der IT-Bereich ist fast gleichmäßig auf die beiden Standorte aufgegliedert. Sowohl Abteilungen als auch Projekte arbeiten über die Standortgrenzen hinweg. Dieses ist in der reinen Scrum-Lehre nicht so vorgesehen. Wie in Abschnitt 2.2 erwähnt, ist es wünschenswert, wenn das Team gemeinsam in einem Raum sitzt. Somit besteht ein besseres Teamgefüge und eine wesentlich ausgeprägtere Kommunikation.

4.2 Scrum im Unternehmen

Im Gegensatz zum reinen *Scrum* wurden die Grundregeln etwas an die Unternehmensbelange der *DVG* angepasst. So existieren Vorgaben und Richtlinien, die nicht vom *Scrum* Framework vorgesehen sind. Beispielsweise ist es laut Unternehmensvorgabe Pflicht ein Fachkonzept zu schreiben. Dieses gehört nicht zu den herkömmlichen Artefakten von *Scrum* und muss in das *Scrum*-Vorgehen im Konzern mit integriert werden, wie in Abbildung 4.1 zu sehen ist. Des Weiteren werden Projekte grundsätzlich noch mit einem Projektleiter ausgestattet. Dieser soll im *Scrum*-Prozess die Rolle das *Product Owner* übernehmen.

Im Gegensatz zu der Teamgöße im *Scrum*, die zwischen 3 und 9 Teammitgliedern liegen sollte, kommt es in der Praxis häufig zu Abweichungen. Nur selten wird ein Projekt in Teams dieser Größe eingeteilt. In der *DVG* sind auch *Scrum*-Teams mit bis zu 20 Mitarbeitern vorhanden.

4.2.1 Meetings

In Scrum hat die Kommunikation einen hohen Stellenwert. Die Anpassung, die die *DVG* am Scrum-Prozess durchgeführt hat, ändert dieses Grundkonzept nicht. Der Prozess wurde sogar durch ein Meeting, den Anforderungsworkshop, erweitert. Dieser wird in Kapitel 4.2.2 beschrieben. Durch die verteilten Standorte wird der Aufwand, die Anzahl an Meetings durchzuführen, steigen.

Es hat sich jedoch in allen Teams durchgesetzt, dass die Sprint-Meetings in einem persönlichen Treffen durchführt werden. Die Produktivität, die durch die persönlichen Kontakte gefördert wird, wiegen den Zeitverlust, der durch die Fahrt zwischen den Standorten entsteht, wieder auf.

Neben den vom Prozess vorgesehenen Meetings, werden zur zusätzlichen Kommunikation weitere Meetings einberufen. Sowohl die regulären als auch die zusätzlichen werden über den Lotus Notes Kalender geplant.

Da Lotus Notes in einer älteren Version eingesetzt wird, fehlt bei der Planung über die Tätigkeit der Teammitglieder häufig eine gute Übersicht. Es werden zwar für die Teams Gruppenkalen-

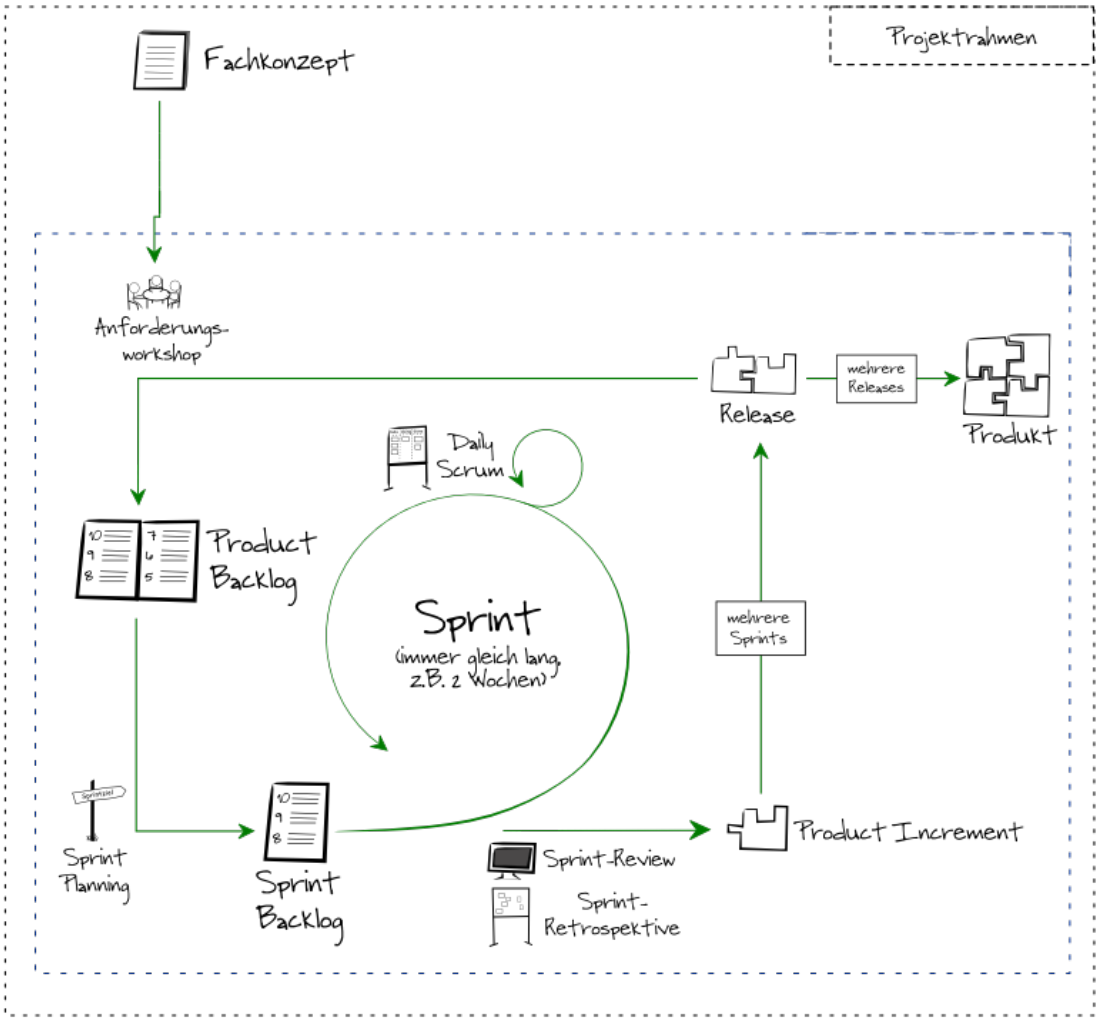

Abbildung 4.1: Scrum Umsetzung im Unternehmen

der erstellt, dem man entnehmen kann, wann es Zeiten gibt, an denen alle Mitglieder noch Kapazitäten frei haben. Wichtige Informationen jedoch bleiben verborgen, wenn bspw. der Kalender „leere“ Termine (genannt Blocker) enthält, die nur deswegen eingetragen wurden, damit die Zeiten nicht von anderen, außerhalb des Teams agierenden Mitarbeitern, belegt werden können. Die Information, dass es sich um einen Blocker handelt, wäre für das Team wichtig, weil innerhalb des Teams trotzdem zu diesen Zeiten Absprachen angesetzt werden könnten. Lotus Notes zeigt im Gruppenkalender jedoch nur, ob der Mitarbeiter Zeit hat oder nicht.

Wünschenswert bei einer Terminplanung wäre es auch Rahmenbedingungen zu sehen, beispielsweise einen Releaseplan. Direkt vor einer Veröffentlichung eines Releases ein Meeting anzusetzen, ist meist unerwünscht.

4.2.2 Anforderungsworkshop

Der Anforderungsworkshop ist kein reines *Scrum* Meeting. Er ist auch ohne den Einsatz von *Scrum* sehr sinnvoll und ist von *Scrum* auch nicht vorgeschrieben. In der *DVG* wurde er jedoch in den *Scrum*-Prozess mit aufgenommen, um ein besseres Verständnis der Anforderungen und einen effizienten Weg der Beschreibung der *Epics* bzw. *Stories* zu erlangen.

Am Anforderungsworkshop sollten der *Product Owner*, der *Scrum Master*, ein Teamvertreter und eventuell weitere Stakeholder teilnehmen. Das Fachkonzept wird im Workshop in *Epics* oder *Stories* zerlegt. Diese werden so weit besprochen und beschrieben, dass jedes Teammitglied den Inhalt der Aufgabe versteht und es in dieser Runde keine Missverständnisse über die Anforderungen bestehen. Anschließend werden die Epics priorisiert.

4.2.3 Rollen

Um die Aufgaben der Teams besser zu strukturieren, wurden weitere Rollen eingeführt, die nicht ursprünglich zum *Scrum*-Prozess gehören. Bis auf die Rolle des Projektleiters ist die Besetzung dieser Rollen nicht verpflichtend. In kleinen Projekten kann eine Person auch mehrere Rollen auf sich vereinen.

Projektleiter

Wie schon erwähnt, wird jedes Projekt mit einem Projektleiter ausgestattet. Daher wurde in der *DVG* vereinbart, dass er die Rolle des *Product Owners* einnimmt. Somit verliert er im Gegensatz zu einem herkömmlichen Projektleiter die Aufgaben der Feinplanung, er kann nur noch bestimmen, in wecher Reihenfolge die Aufgaben abgearbeitet werden, aber nicht wie viel Aufgaben in einen Sprint kommen. Die Vorgabe der technischen Mittel und die Vorgabe bzw. Überprüfung der Einhaltung des Vorgehens fallen dem Team bzw. dem *Scrum Master* zu.

Build-Manager

Im Scrum-Prozess ist die *Kontinuierliche Integration* ein bedeutender Baustein. Aus diesem Grund wurde die Rolle des Build-Managers in den *Scrum*-Prozess der *DVG* aufgenommen. Seine Aufgaben sind die Build-Prozesse und Jobs des Projektes zu erstellen, zu verwalten und zu warten.

Lösungsarchitekt

Der Lösungsarchitekt ist verantwortlich für die Umsetzung der Ziele eines Projekts durch Entwurf geeigneter Lösungsarchitekturen auf der Basis der Referenzarchitektur des Unternehmensarchitekten. Er ist Schnittstelle zum zentralen Architekturmanagement.

4.3 Derzeitige IT-Unterstützung im Scrum-Prozess

In dem Projekt, das *Scrum* in die *DVG* eingeführt hat, wurde die Idee des *Lean software development* gelebt. Was bedeutet, es wurde kein Werkzeug angeschafft, dass den *Scrum*-Prozess in seiner Gesamtheit abdeckt, sondern jeder einzelne Prozessschritt wurde mit einem passenden Werkzeug unterstützt. In diesem Abschnitt wurden zuerst die Werkzeuge und dann ihre Funktionen beschrieben. Anschließend skizziere ich den Prozess und die Integration der Werkzeuge.

4.3.1 Einführung vorhandener Tools

Viele Werkzeuge bestanden schon vor der Einführung von *Scrum* im Unternhmen. Sie werden auch für weitere Aufgaben verwendet. Ich fokussiere mich hier auf die Funktionen, die für den *Scrum*-Prozess von Belang sind.

Jira

Jira, ein Produkt der Firma Atlassiana[2], ist eine webbasierte Anwendung für das Fehler- und Anforderungsmanagement, vorwiegend in der Softwareentwicklung. Der Einsatz von Jira ist in der IT der *DVG* für die Aufgabenfelder Fehler- und Anforderungsmanagement vorgeschrieben.

Jira unterstützt in so genannten Tickets die Erstellung, Planung, Bearbeitung und Dokumentation von Aufgaben oder Fehlerfällen. Die Identifikation eines Tickets wird über eine Projektzuordnung und eine eindeutige Nummer bewerkstelligt. Es enthält mindestens eine Zusammenfassung, einen Typen, eine Priorität und eine Beschreibung. Des Weiteren können einem Ticket noch mehr Informationen anhaften, wie beispielsweise Dokumente angehängt oder das Ticket mehreren Komponenten, einer Versionen oder einer *Epic* zugeordnet werden. Jede Änderung an einem Ticket, sei es der Status oder der Inhalt, wird vom System festgehalten und kann so jederzeit nachvollzogen werden.

[2] https://de.atlassian.com/software/jira

Tickets in Jira haben immer einen Status, der den Zustand beschreibt, in dem sich das Ticket befindet. Die Status für Agile Projekte wurden dem Vorgehen angepasst und damit der *Workflow* vereinfacht. Für Agile Projekte kann ein Ticket in vier Status sein (siehe Abbildung 4.2), **Offen**, **in Arbeit**, **Gelöst** und **Geschlossen**. Der *Workflow* für herkömliche Jira-Projekte hat zehn Status.

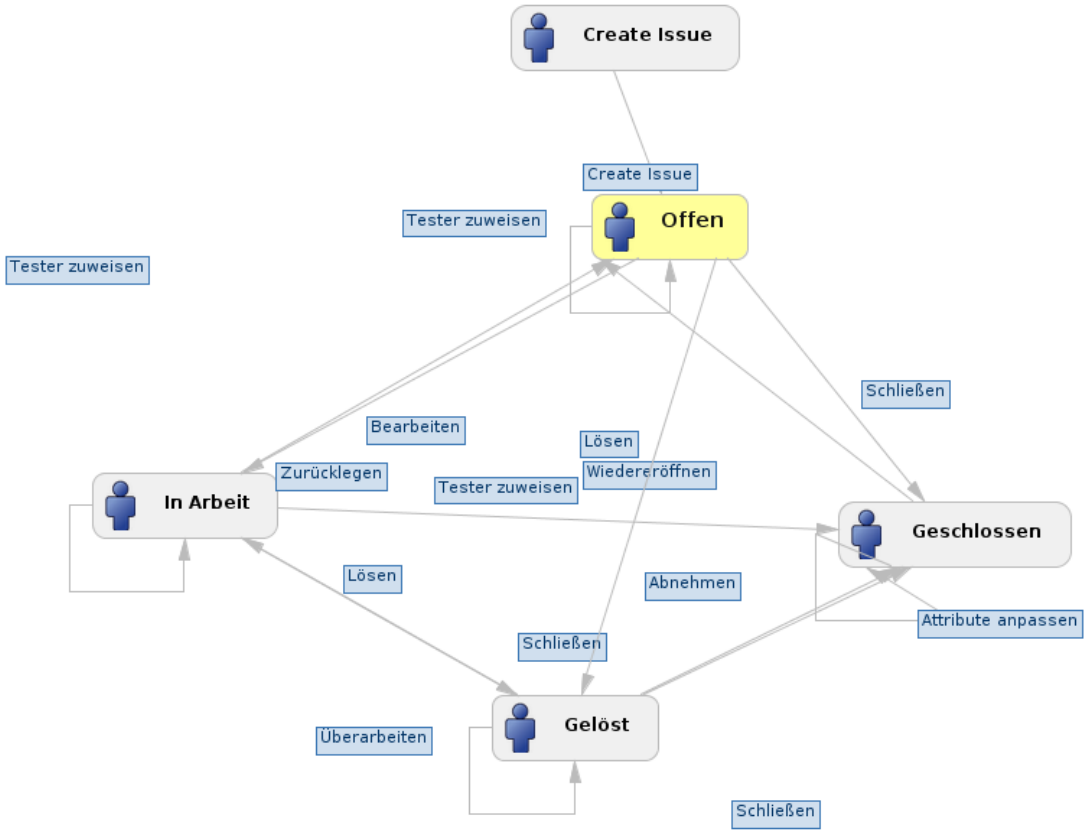

Abbildung 4.2: Jira *Agiler Workflow*

Von Jira wird seit 2011 eine Version in der *DVG* eingesetzt, die eine Erweiterung für agile Teams bietet. Früher hieß diese Erweiterung Greenhopper, jetzt in der eingesetzten Jira-Version 5.2 heißt sie *Jira Agile*. Sie bietet ein digitales *Scrum Board* mit einem *Product Backlog*, einem *Taskboard* und Statistiken, die über die Sprints ausgewertet werden können, bspw. ein *Burndown chart*. In den meisten agilen Teams wird Jira mit dieser Erweiterung genutzt.

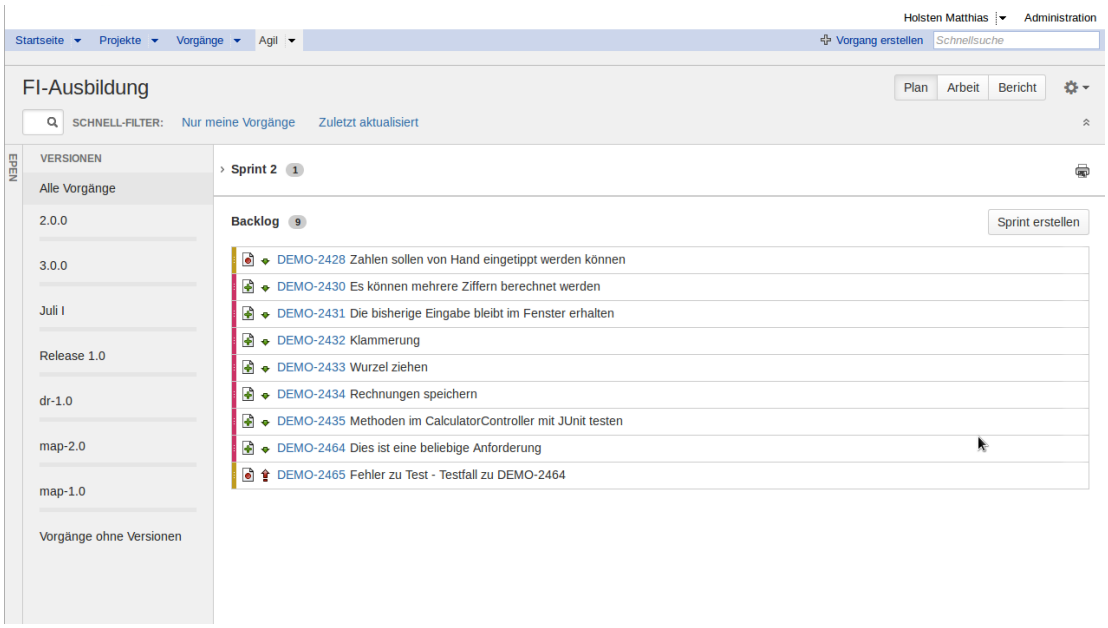

Abbildung 4.3: Jira Planing Board

Im *Planing Board* (siehe Abbildung 4.3) werden die Tickets erstellt, priorisiert, Versionen oder Epen zugeordnet. Das *Planing Board* unterstützt die Arbeit des *Product Owner* und das Team während des *Sprint-Meetings*. Das *Taskboard* ersetzt das Papier Board an der Wand, und bildet die Arbeit eines Sprints ab. Es lassen sich wie auch beim Papier-Äquivalenten die Tickets von links nach rechts über das Board hängen, dieses geschieht per *drag & drop*. Jede Spalte repräsentiert einen Jira Status. Links ist **Offen** und Rechts **Geschlossen** (siehe Abbildung 4.4).

Confluence

Confluence[3] ist eine Wiki-Software, die von der Firma Atlassian vertrieben wird. Sie deckt die herkömmlichen Funktionen einer Wiki-Software ab, ist jedoch für den Einsatz in Unternehmen optimiert. Beispielsweise können die Artikel in Bereiche, *Spaces* genannt, eingeteilt werden. Jeder dieser Bereiche kann mit eigenen Nutzerbeschränkungen versehen werden, falls dieses gewünscht ist.

[3] https://de.atlassian.com/software/confluence

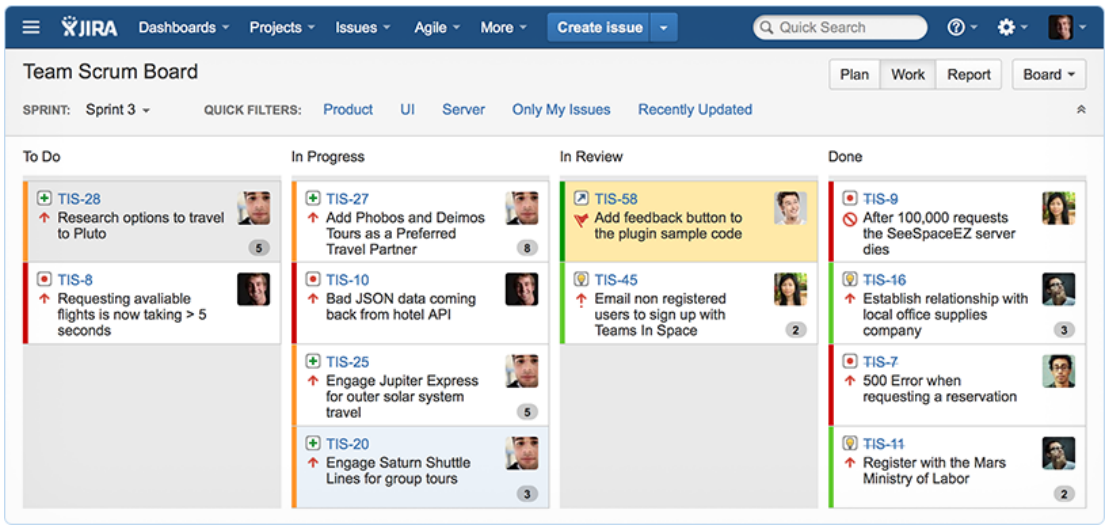

Abbildung 4.4: Jira Taskboard

In der *DVG* hat jeder Mitarbeiter automatisch Zugang zu Confluence. Es wird von ca. einem Viertel der Mitarbeiter genutzt. Hauptsächlich zur Dokumentation der Arbeit und zum Wissensaustausch. Fast jedes Projekt, das mit IT-Unterstützung neu gestartet wird, erhält einen eigenen Wiki-Bereich. Dabei sind nur wenige Bereiche im Unternehmen nicht von allen Mitarbeitern einsehbar. Confluence schafft so jetzt schon Transparenz im Unternehmen.

Jenkins

Jenkins ist ein Werkzeug zur *Build-Automation*. Es wird genutzt, um *Continuous Integration* durchzuführen und schnellstmögliches Feedback zu erhalten. Die Entwickler können über Unit-Tests und Code-Metriken die Qualität überprüfen. Jede Software, die in der *DVG* produktiv gesetzt wird, muss über die Jenkins *Build-Automation* compiliert werden.

Telefonkonferenz

Telefonkonferenzen sind kein IT-unterstütztest Werkzeug, jedoch werden sie häufig eingesetzt, um über die Standorte hinweg zu kommunizieren, in den *Scrum*-Teams mindestens einmal am

Tag. Zum *Stand-up Meeting* oder auch *Daly-Scrum* genannt, treffen sich alle Team-Mitglieder eines Standorts um ein Telefon und rufen die zweite Hälfte des Teams am anderen Standort an. Dieses hat sich ebenso um ein Telefon versammelt. So werden die Aufgaben und Unwägbarkeiten des letzten und des kommenden Tages besprochen.

Telefonkonferenzen werden in vielen Situationen genutzt, da der Fahrtaufwand zwischen den Standorten recht hoch ist. Es besteht auch die Möglichkeit einer Videokonferenz. Sie ist nur in wenigen Teams beliebt, da die Räumlichkeiten weit im Voraus gebucht werden müssen, was eine Videokonferenz sehr unflexibel macht, und der Mehrwert gegenüber der Telefonkonferenz gering ist.

4.3.2 Mapping Prozesse zu Tools

Als Initiator eines Scrum Prozesses (Projektes) steht meistens eine Anforderung. In der *DVG* muss ein Fachkonzept erstellt werden. Dieses enthält die fachlichen Anforderungen, die in einer Software umgesetzt werden sollen.

Fachkonzepte werden nach dem IEEE-Standard 830-98 beschrieben. Der Standard wird als Gliederungsvorlage angeboten. Die Ausformulierung geschieht meist in MS-Office, da das Dokument später auf einem Netzlaufwerk abgelegt werden muss. Im Anforderungsworkshop wird das Fachkonzept in fachlich sinnvolle, kleine Aufgaben (*Epics* oder *Stories*) unterteilt. Falls die Aufgaben konkret zu formulieren und klein genug für Jira-Tickets sind, werden sie direkt in dem Anforderungsmanagement-Werkzeug erfasst. Andernfalls werden die noch etwas gröber gefassten Aufgaben in Confluence dokumentiert.

Aus den in Jira erstellen Tickets füllt sich das *Product Backlog*. Häufig wird ein zusätzliches *Product Backlog* im Confluence geführt, dass die gröberen Anforderungen enthält. Im Sprint-Meeting werden während der Review-Phase Jira und die entwickelte Anwendung genutzt, um die Umsetzung der Tickets nachzuweisen und sie vom *Product Owner* abnehmen zu lassen. Der anschließende Planungsteil des Meetings wird wiederum von Jira unterstützt, in dem die vom Product Owner priorisierten Aufgaben sukzessiv geschätzt und im neuen Sprint eingeplant werden.

Die vom Team abgenommenen und eingeplanten Tickets bilden das Sprint Backlog, das die Grundlage für die tägliche Arbeit liefert. In der *DVG* ist als Entwicklungsumgebung (IDE) für

Java Eclipse zwar nicht als Standard gesetzt, jedoch die meist genutzte IDE. Der Quellcode wird in einem Subversion Repository gehalten. Im Jenkins werden Continuous Integration Jobs erstellt und die Qualität des Quellcodes überprüft, so wie die Produkte über Release-Jobs erstellt. Die Dokumentation der Arbeit geschieht dabei im Confluence oder in den Jira-Tickets.

Die Daily Stand-up's werden bei verteilten Teams mit Hilfe einer Telefonkonferenz unterstützt.

5 Verbesserung Prozess-Workflow

In Kapitel 4 wurde das Vorgehen in agilen Teams im Unternehmen *DVG* vorgestellt, wie die Teams arbeiten und mit welchen IT-Lösungen ihre Arbeit unterstützt wird. Im Kapitel 3 Related Work wurden Vorschläge aus anderen Arbeiten vorgestellt. In diesem Abschnitt soll analysiert werden, an welchen Stellen der Prozess der *DVG* noch Verbesserungspotentiale aufweist und wie die Mitarbeiter in ihrer täglichen Arbeit unterstützt werden können.

Es werden Lösungsvorschläge für diese Probleme dargestellt, wovon einige schon in der *DVG* getestet wurden und im Einsatz sind.

5.1 Analyse der Probleme

In der Analyse wurde zuerst der beschriebene Prozess aus Kapitel 4 Fallbeispiel mit den Arbeiten aus Kapitel 3 Related Work verglichen und die Erweiterungsmöglichkeiten des in der *DVG* verwendeten Prozesses herausgearbeitet.

In diesem Abschnitt werden die gefundenen Differenzen in Unterkapiteln erläutert. Im Kapitel 5.2 Verbesserungen werden anschließend Unterstützungsmöglichkeiten der Mitarbeiter durch die Erweiterung oder Veränderung der IT-Infrastruktur vorgestellt.

Um den Prozess besser zu verstehen und analysieren zu können, wurden mit einem Team Gespräche geführt. Diese Erkentnisse werden in den einzelnen unten aufgeführten Punkten erläutert und behandelt.

Wesentliche Defizite lassen sich in Kategorien zusammen fassen. So werden die Kategorien „Verteilte Teams“ und „Zusammenhängender Prozessverlauf“ gebildet, weil hier die größten Probleme auftreten.

5.1.1 Verteilte Teams

Eine besondere Herausforderung für die Zusammenarbeit der Mitarbeiter in der *DVG* stellt die räumliche Distanz einiger Teams dar. Sie ist der Hauptgrund, warum der Scrum-Prozess der *DVG* optimiert werden sollte. Dabei spielt die Kommunikation und die Wahrnehmung dem jeweiligen Teil-Team am anderen Standort eine besonders große Rolle.

Der Effekt, eines Großraumbüros, bei dem alle die Informationen als ein Grundrauschen im Vorübergehen mitbekommen, ist so nicht mehr für das Team erzielbar. Die beiden Teil-Teams treffen sich nicht einmal in der Kaffeeküche um Informationen auszutauschen (InfoToGO). Der Informationsaustausch kann nur über initiierte Kommunikation stattfinden. Damit dieser so häufig wie möglich gesucht wird, müssen alle technischen Barrieren aus dem Weg geräumt werden.

Sultan et al. betont die Relevanz der Kommunikation über die Standortgrenzen, um das Verständnis im Team füreinander zu fördern.

> *„Distributed agile teams find it difficult, with infrequent communications, to understand how the work of one team member at one site influences the work progress of another team member at a different site.“*[AIG12]

Kommunikation zwischen den Standorten

Wie in Abschnitt Meetings (siehe 4.2.1) beschrieben, existieren Hindernisse bei der Planung von initiierter Kommunikation. Es ist nicht einfach ersichtlich für ein Team-Mitglied, ob ein anderer Mitarbeiter Zeit zum Austausch oder zur Zusammenarbeit hat. Es stellt sich häufig die Frage: Ist er derzeit am Platz oder ist er gerade bei einem anderen Kollegen? Selbst ein Anruf klärt diese Frage nicht immer. Für die Mitarbeiter der *DVG* ist es verpflichtend, wenn sie ihren Platz verlassen, das Telefon umzustellen. Doch ein umgestelltes Telefon ist nicht aussagekräftig, ob jemand an seinem Platz sitzt. Ein Kontakt kommt nicht zustande und wichtige Absprachen können nicht erfolgen.

In den Gesprächen mit den Mitarbeitern wurde eine fehlende Übersicht über die Aktivitäten der Team-Mitglieder genannt. Besonders bei der Terminfindung für Meetings fiel dieses auf. Es ist nicht direkt ersichtlich zu welchem Zeitpunkt die Team-Mitglieder zur Verfügung stehen. Mehrere Absprachen wurden getroffen, um den in Lotus Notes geführten Gruppenkalender übersichtlicher zu gestalten. Dieser aggregiert jedoch nur die Zeiten der Kalender-Einträge von den Team-Mitgliedern. Es ist nicht möglich, in den Gruppenkalender Einträge, beispielsweise für gemeinsame Arbeitszeiten oder Realeas-Pläne, zu stellen, er zeigt nicht was die Mitglieder für Termine haben. Die getroffenen Absprachen erzielten nicht den gewünschten Erfolg.

Stand-up Meetings

Stand-up Meetings in verteilte Teams werden in der *DVG* mit Hilfe von zwei Telefonapparaten durchgeführt. Alle Team-Mitglieder eines Standorts stehen um das Telefon herum. Da die Freisprechfunktion des Telefons dafür gedacht ist, dass der Sprechende sich höchstens in einem Abstand von 60 cm befindet, hält der Sprecher den Telefonhörer in der Hand, der wird von Sprecher zu Sprecher weitergereicht.

Diese Konvention hat den positiven Effekt, dass derjenige mit dem Hörer ungestört sprechen kann und es selten zu Diskussionen in den *Stand-up* Meetings kommt. Jedoch ist mit dem ständig wechselnden Sprecher am Telefon eine große Unruhe verbunden und diejenigen, die gerade nicht sprechen, schalten gedanklich ab. Dieser Effekt wird dadurch verstärkt, dass die Zuhörer am anderen Standort den Sprecher nicht sehen und ihm so weniger Aufmerksamkeit zukommen lassen.

Die Unaufmerksamkeit könnte durch das Vorgehen laut Xebia (siehe Abschnitt 3.2), auch für *Stand-up* Meetings eine Videokonferenz abzuhalten, gelindert werden. Somit würden auch die Emotionen der Teams von den Standorten übertragen werden.

Pair Programming

Wissenstransfer zwischen den Standorten, besonders bei den Entwicklern des Quellcode, kann nur geschehen, wenn gemeinsam an Aufgaben gearbeitet wird. Nicht nur der Wissensaustausch wird beim *Pair Programming* (siehe Abschnitt 2.2.4) gefördert, sondern die Team-Mitglieder lernen sich besser kennen. Dadurch bauen sie Kommunikationshemmnisse ab und können besser beurteilen, wie sie ihre Fähigkeiten im Team einsetzen können. In der Zusammenarbeit werden Projektziele und Pläne immer wieder neu geschärft. Laut Flor[Flo06] sind diese guten Eingenschaften auch für verteilte Teams nutzbar. Dieser Mehrwert wird in der *DVG* derzeit nicht ausreichend ausgeschöpft.

Die Herausforderungen und Aufwände sind laut Flor[Flo06] nicht zu unterschätzen. Mit dem Vorgehen, Aufgaben über die Standort-Grenzen im *Pair Programming* zu lösen, könnte jedoch die Distanz der Teil-Teams verringert und Wissensinseln abgebaut werden.

Digitale Whiteboards

In der Arbeit von Sutherland et al.[SSRR08] wird zur Absprache zwischen den Mitarbeitern an den verschiedenen Standorten und besonders zur Konstruktion von Architekturen ein digitales *Whiteboard* eingesetzt. Diese Möglichkeit besteht in der *DVG* derzeit nicht.

Somit werden Konzepte oder Architekturen an einem Standort erstellt, im Wiki beschrieben und dem restlichen Team vorgestellt. In einer vorgegebenen Frist kann jedes Team-Mitglied

seine Änderungsvorschläge abgeben, die vom Ersteller in das Konzept oder die Architektur eingearbeitet werden.

In seltenen Fällen wird ein Viedeokonferenz-Raum für die Erarbeitung eines Konzeptes genutzt. In diesen Räumen fehlt die Möglichkeit etwas für beide Seiten sichtbar zu zeichnen. Darum verlängern Team-Mitglieder gegebenenfalls den Aufenthalt bei einem Sprint Meeting um einen Tag, damit sie mit den Kollegen dieses Standortes gemeinsam am Whiteboard konstruieren können. Dadurch wird ein nicht unerheblicher Aufwand an Zeit und Kosten verursacht, der beim Abwägen zwischen Nutzen und Aufwand dazu führt, dass ein gemeinsames Konstruieren verhindert wird. Das Vorhandensein der technischen Voraussetzungen würde die Kommunikation und den Wissenstransfer zur gemeinsamen Lösung eines Problems erheblich steigern.

5.1.2 Zusammenhängender Pozessablauf

Ein gravierendes Defizit lässt sich im dezentralen und nicht verknüpften Prozessablauf finden. Es gibt keinen klaren Einstiegspunkt, um alle Daten eines Projekts einzusehen.

Das Fachkonzept wird auf einem Netzlaufwerk gelagert. Einige der Inhalte des Fachkonzepts wurden als *Epics* aufgearbeitet ins Wiki oder Jira gestellt.

Meist findet man eine kurze Beschreibung der *Epic* im Jira und eine ausführliche im Wiki. Die beiden Versionen können über einen eingefügten Link miteinander verbunden werden, jedoch ist das nur selten der Fall. Meist wird eine der beiden *Epic*-Beschreibungen gepflegt, in der anderen veralten die Informationen. Die Versionen des Wikis und Jiras (Confluence Version 3.5 und Jira Version 5.2), die in der *DVG* eingesetzt werden, bieten eine lose Verbindungsmöglichkeit. Im Wiki können Ticket-Zusammenfassungen angezeigt werden, im Jira fehlt leider die Möglichkeit Inhalte aus dem Wiki anzuzeigen, denn die Ausdrucksmöglichkeiten und Übersichtlichkeit im Wiki bei der Beschreibung einer *Epic*, sind sehr viel umfassender.

Weiterhin muss es nachvollziehbar und klar zu verfolgen sein, welche Quellcode-Änderungen zu einem Ticket durchgeführt wurden. Dieses ist derzeit mit einem Jira Plugin möglich, wie in Abbildung 5.1 zu sehen ist.

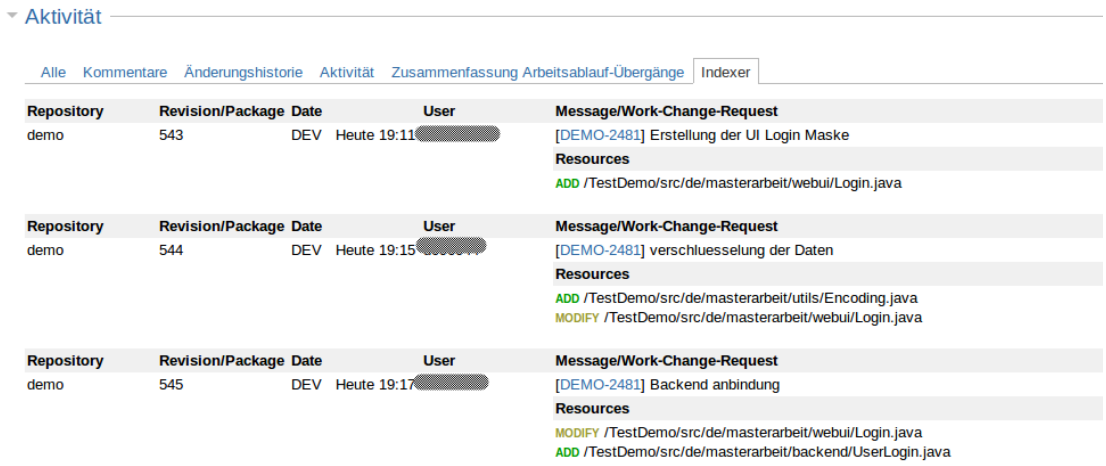

Abbildung 5.1: Visualisierung der Quellcode-Änderungen in Jira

Das Plugin informiert darüber, welcher Entwickler zu welchem Zeitpunkt eine Änderung am Quellcode vorgenommen hat. Möchte ein anderer Entwickler nun nachvollziehen, welche Änderungen genau durchgeführt wurden, muss er in ein Quellcodeverwaltungssystem wechseln, um einen Vergleich zwischen der Quellcode-Version, die in diesem Ticket steht und der Vorversion durchführen. Dieser Vorgang ist relativ aufwendig, wenn man bedenkt, dass eine Team-Absprache aussagt, dass bei jedem Ticket ein Quellcode-Review durchgeführt werden sollte.

Einen interessanten Ansatz, den Prozess durchgängiger zu gestalten, bietet auch die Arbeit *A Holistic Approach to Developing a Progress Tracking System for Distributed Agile Team*[AIG12] von Sultan et al.. Danach sollten die Mitarbeiter eines Teams jeden Fortschritt mitgeteilt bekommen. Sie werden nicht nur über offensichtliche Fortschrittsveränderungen informiert, sondern auch über die, die normalerweise im Verborgenen bleiben, beispielsweise die Integration von verändertem Quellcode. Wenn solche verborgenen Informationen offen dargestellt würden, hätte das Team eine viel schnellere Resonanz auf ihre Arbeit.

5.2 Verbesserungen

Die in Kapitel 5.1 analysierten Punkte werden in diesem Kapitel einzeln aufgegriffen und Vorschläge zur Verbesserung der IT-Unterstützung von Prozessen dargestellt. Sofern mehrere Werkzeuge für die IT-Unterstützung der *DVG* in Frage kommen, wird der Vorschlag favorisiert, der den Prozess optimal unterstützt.

Einige der foavorisierten Vorschläge wurden schon in der Praxis der *DVG* getestet, und können in der Bewertung vorgezogen werden.

5.2.1 Verteilte Teams

Das Zusammenarbeiten der verteilten Teams wurde in Kapitel 5.1.1 als große Herausforderung dargestellt. Die große Herausforderung besteht darin, die direkte Kommunikation zu fördern, um so das Teamgefüge zu unterstützen. Des Weiteren muss die direkte Zusammenarbeit zweier Mitarbeiter von verschiedenen Standorten so unterstützt werden, dass die Barrieren nicht erheblich größer sind, als würden sie an einem Standort sitzen.

Als erstes werden die Terminfindungsprobleme und die Kommunikation auf kürzestem Wege in den Fokus gesetzt. Daraufhin werden die Probleme bei den Stand-up Meetings betrachtet. Abschließend wird eine Lösungsmöglichkeit angeboten wie den Team-Mitgliedern in der direkten Zusammenarbeit an Tickets über die Standortgrenzen hinweg geholfen werden kann.

Kommunikation zwischen den Standorten

Als erstes Problem wurde analysiert, dass es nicht klar erkennbar ist, ob ein Team-Mitglied am anderen Standort zur Zeit zur Zusammenarbeit verfügbar ist. Wenn das Team-Mitglied am selben Standort ist, so kann als letzte Instanz, ein Gang zum entsprechenden Kollegen klären, ob er am Platz ist und ob er vielleicht nur vergessen hat die Umleitung aus dem Telefon zu nehmen.

Der Vorschlag, um dieses Problem zu beheben, kam direkt aus dem Team, als es für diese Arbeit befragt wurde. Ein Team-Mitglied schlug vor, einen Chat-Server einzurichten. So kann jedes Team-Mitglied einen Status setzten, wenn er den Platz verlässt, indem er mitteilt, wo er sich aufhalten wird. Falls dieses vergessen würde, so hat fast jeder Chat-Client die Möglichkeit automatisch nach einer bestimmten Zeit der Abwesenheit am Computer einen Statuswechsel vorzunehmen. Auch Herbsleb bestätigt die gute Kommunikation zwischen verteilten Teams über Chats[HAB+02].

Im Rahmen dieser Arbeit wurde in der *DVG* ein Xmpp-Server installiert. Das Team, aus dem die Idee stammt, testet seit dem die Nutzung mit den Chat-Client Programmen Pidgin[1] und Spark[2]. Sowohl die eingesetzte Server-Komponente *ejabberd*[3] als auch die beiden Client-Programme sind *Open-Source*-Projekte und frei verfügbar.

Der Chat wird in diesem Team viel genutzt und hilft bei einer direkten Kommunikation. So können auch Absprachen für Termine einfacher getroffen werden. Das Problem der fehlenden Übersicht über die Termine im Zusammenhang mit Metadaten, die jeder eintragen kann, bietet diese Lösung jedoch nicht.

Dafür wurde ein Plugin für *Confluence* gewählt, der Team-Kalender[4]. Er bietet viele Merkmale, die ein verteiltes Team unterstützen. So kann jeder Mitarbeiter, der Zugang zu Confluence hat, einen oder mehrere Kalender anlegen und ihn mit anderen Mitgliedenr gemeinsam verwalten. Weitere Funktionen des Team-Kalenders sind:

- In einen geteilten Kalender können alle berechtigten Personen Einträge einstellen.
- Terminkalender aus anderen Programmen wie Outlook oder Lotus Notes können importiert bzw. bei angebotener Schnittstelle synchronisiert werden.
- Einfache Integration von Jira Artefakten. Sprints, Release-Pläne und Fälligkeitsdaten von Tickets können automatisch im Kalender angezeigt werden.

Die Funktionalitäten bieten dem gesamten Team einen Überblick über die Termine jeden einzelnen Team-Mitglieds, so dass Termine effizient geplant werden können.

[1] https://www.pidgin.im/
[2] http://www.igniterealtime.org/projects/spark/
[3] http://www.process-one.net/en/ejabberd/
[4] https://de.atlassian.com/software/confluence/team-calendars

Stand-up Meetings

Als bessere Unterstützung für *Stand-up* Meetings wurde in der Arbeit der Firma Xebia[SSRR08] das Videokonferenz-System Skype genutzt. Dieses ist in der *DVG* nicht möglich, da interne Kommunikation das Firmennetzwerk nicht verlassen darf.

Für *Stand-up* Meetings in einen vorhandenen Videokonferenz-Raum zu wechseln ist schwer umsetzbar. Diese Räume sind sehr weit im Voraus ausgebucht und aufgrund der wenigen Videokonferenz-Räume im Unternehmen, würde der Weg zum Meeting länger dauern als das Meeting an sich.

Daher kann sich dieses Problem nicht ohne ein Investment in Equipment beheben lassen. Es sollte in jedem Team-Raum eine WebCam inklusive eines Raum-Mikrofons installiert werden, vor der sich die Teil-Teams zum *Stand-up* treffen. Als Software kann in diesem Fall wieder auf eine Open Source Lösung zurückgegriffen werden. Der Chat- und Videokonferenz-Server *Openfire*[5] im Zusammenspiel mit der Videokonferenz-Client-Software *jitsi*[6] bietet kostenlos eine Videokonferenz-Lösung, bei der alle Daten innerhalb des Firmennetzwerks bleiben und die die Kommunikation beim *Stand-up* Meeting verbessert.

Pair Programming

Pair Programming ist ein wesentlicher Bestandteil von agilen Softwareentwicklungs-Methoden. Es wurde in Abschnitt 2.2 vorgestellt. *Remote Pair Programming* ermöglicht das Zusammenarbeiten von zwei Team-Mitgliedern, die an unterschiedlichen Standorten sitzen. Gerade in einem verteilten Team kann es verhindern, dass spezielles Wissen nur an einem Standort gehalten und weiterentwickelt wird. Das würde bedeuten, dass nur bestimmte Entwickler ein Themengebiet bearbeiten und nicht bei einem Ausfall ersetzt werden könnten.

Das *DVG* Team, das exemplarisch für Versuche bereit stand, probierte erst mit vom Betriebssystem gegebenen Mitteln *Remote Pair Programming* umzusetzen. Microsoft bietet die Möglichkeit *Remoteunterstützung*[7] von einem anderen Netzwerk-Nutzer anzufordern.

[5] http://www.igniterealtime.org/projects/openfire/
[6] https://jitsi.org/
[7] http://windows.microsoft.com/de-de/windows/what-is-windows-remote-assistance

Zur Unterstützung wird der Bildschirm des Anforderers geteilt, so sehen beide dasselbe. Manuell kann der Unterstützer die Steuerung des Anforderer-PC's übernehmen, dazu muss der Anforderer eine Anfrage bestätigen.

Nach kurzer Testphase wurde diese Lösung verworfen. Die manuelle Übernahme der Steuerung war für eine Zusammenarbeit zu aufwendig. Eine Alternative ist die Integration in die Entwicklungsumgebung. In diesem Fall wurde Eclipse-Saros[8], ein Open Source Plugin gewählt. Es erfüllt die Ansprüche der Mitarbeiter. Schenk et al.[SPS14] setzt in ihrer Arbeit *Distributed-Pair Programming Can Work Well and Is Not Just Distributed Pair-Programming* auf Eclipse-Saros und beschreibt wie verteilte Teams erfolgreich mit dem Plugin arbeiten können.

Saros ist eine Erweiterung für die Entwicklungsumgebung *Eclipse*[9]. Sie ermöglicht ein *Remote Pair Programming*. Saros versucht den beiden Entwicklern das Gefühl zu vermitteln, dass sie an einem Computer säßen. Saros bietet auch einen Chat und ein digitales Zeichenbrett, auf dem die Entwickler kommunizieren und Ideen austauschen können.

Um zusammen an einer Aufgabe zu arbeiten, verbinden die Entwickler mit Hilfe von Saros die Entwicklungsumgebungen. Ab nun können sie sich gegenseitig *folgen*, jede Aktion wird in Echtzeit synchronisiert. Sobald einer der Entwickler anfängt zu tippen, wird die selbe Zeile bei dem Anderen eingefärbt und der Inhalt erscheint in Sekundenschnelle auf seinem Bildschirm. Beide Entwickler sind gleichberechtigt, es muss nicht erst die Steuerung angefragt oder übergeben werden, jederzeit können beide arbeiten.

Es besteht auch die Möglichkeit mit mehr als zwei Entwicklern gleichzeitig über Saros zu arbeiten. Dies wird jedoch nicht sehr häufig genutzt.

In Abbildung 5.2 ist das Eclipse Plugin im Einsatz zu sehen.

1. Die farbigen Kreise an den Dateien zeigen, welcher User an einer Datei arbeitet.

2. Selektierter Text von einem anderen Entwickler.

3. Von einem anderen Entwickler geänderter Quellcode.

[8] http://www.saros-project.org/
[9] http://eclipse.org/
[10] http://www.saros-project.org/screenshots

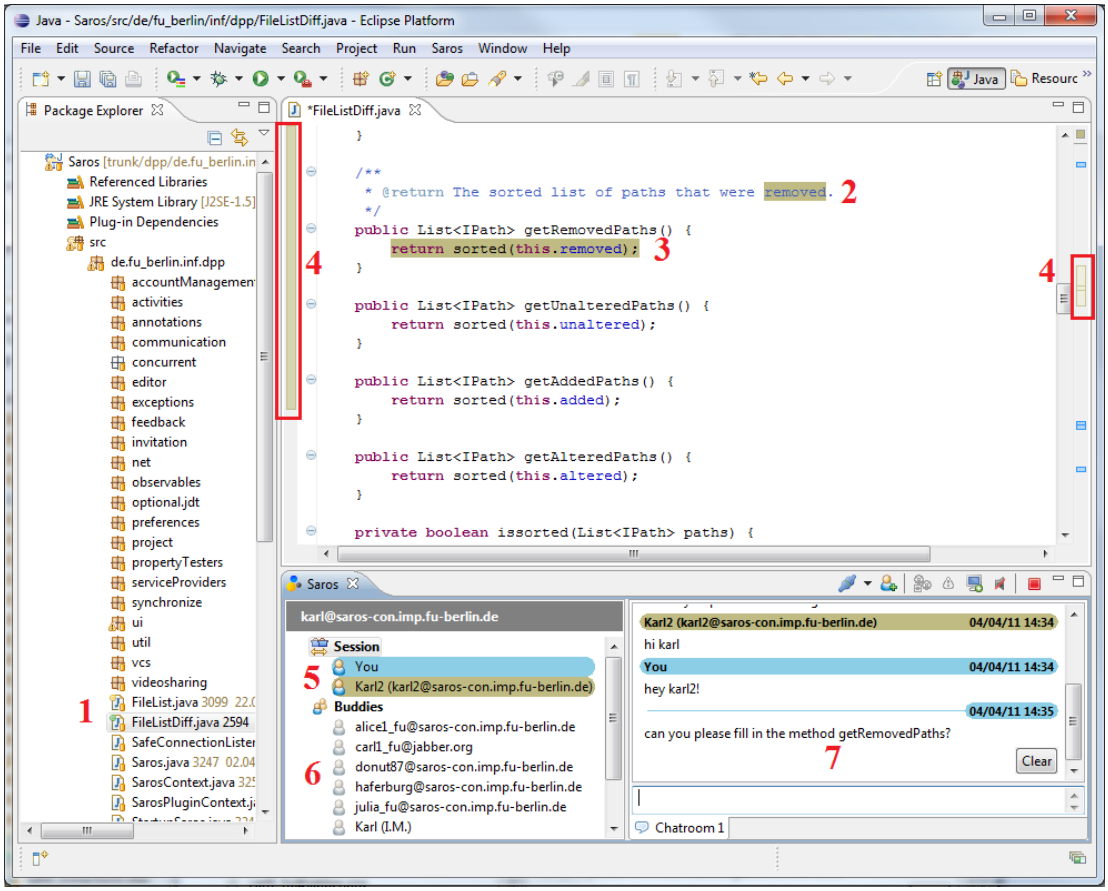

Abbildung 5.2: Pair Programming mit Saros (Bild von Saros-Projekt[10])

4. Der Sichtbereich des anderen Entwicklers, den er derzeit auf seinem Bildschirm sieht.

5. Entwickler die derzeit zusammen arbeiten. Die Farben an der rechten Seite des Editors zeigen welcher Entwickler an dieser Stelle eine Änderung vorgenommen hat.

6. Kontaktliste des Entwicklers

7. Ein Chat mit allen Mitgliedern der derzeitigen Session.

Digitale Whiteboards

Eine sehr gute Lösung für Kollaboration im Bereich des digitalen Whiteboards erstellt das Hasso-Plattner-Institut in Zusammenarbeit mit der *Stanford University*. Es nennt sich das Tele-Board[RG09], leider befindet es sich noch in der Prototypphase. Es bietet eine ausgezeichnete Unterstützung für verteilte Teams.

Zum Aufbau des Whiteboards gehört eine Kamera, die das Bild der Personen, die miteinander arbeiten, an den entgegengesetzten Standort überträgt. So werden Gesichtsausdrücke und Emotionen bei der Arbeit übertragen.

Es besteht nicht nur die Möglichkeit frei zu zeichnen, sondern es können auch vorgefertigte Formen, wie beispielsweise *Post-it's* oder Kreise eingefügt werden. IPads, IPhones oder Android-basierte Endgeräte können sich mit dem Tele-Board verbinden und Zeichnungen oder Texte übertragen. So sind die Bearbeitungsmöglichkeiten sehr vielfältig.

Jeder Veränderung auf dem Whiteboard wird gespeichert und kann später am Computer nachvollzogen und rekonstruiert werden. Die erzeugten Bilder können direkt am Computer weiterentwickelt und im Team geteilt werden.

5.2.2 Zusammenhängender Prozessverlauf

Die Übersichtlichkeit einer Projektdokumentation schwindet deutlich, wenn es keinen eindeutigen Einstieg gibt. Für Mitarbeiter, die neu zu einem Projekt stoßen, erschwert es die

Einarbeitung, aber auch für Mitarbeiter, die in mehreren Projekten arbeiten, kann es unübersichtlich sein, wo welche Daten abgelegt sind.

Die Aufbereitung der Daten in einem Wiki ist sehr flexibel. Es lassen sich sowohl grob strukturierte als auch fein ausformulierte Informationen gut veranschaulichen. Des Weiteren ist ein Wiki meist erweiterbar, so dass es Informationen aus anderen Werkzeugen anzeigen kann. Aus diesem Grund bin ich der Meinung, dass ein Wiki der beste Ort für einen Projektdokumentationseinstieg ist.

Das bedeutet, dass jede Anforderung im Wiki beschrieben sein sollte und von dieser Anforderungsbeschreibung alle anderen Informationen zu erreichen sind. Was muss von der Anforderung erreichbar sein:

- Die Jira-Tickets, mit denen die Anforderung umgesetzt wurde,
- die Quellcodeänderungen, die die Anforderung umsetzten,
- die Tests, die abdecken, ob die Anforderung korrekt umgesetzt wurde und
- die Dokumentation, wie die Software nach der Änderung zu nutzen ist.

In dem in der *DVG* eingesetzten Wiki, Confluence in der Version 3.5, muss so eine Anfoderungsbeschreibung manuell erstellt werden. Da in der *DVG* in naher Zukunft eine neue Version des Wikis (Version 5.0) eingesetzt wird, wird die Umsetzung für die neue Version beschrieben.

Confluence Version 5 bietet eine neue Funktion, in der vordefinierte Seiten ausgefüllt werden können. Diese vordefinierten Seiten werden *Blueprints* genannt. Eine der *Blueprints* ist eine Vorlage für eine Anforderung.

Ist das Confluence-System mit dem Anforderungsmanagementsystem Jira verbunden, kann eine Anforderung direkt mit einer Epic aus Jira verbunden werden. Mit dieser Verbindung wird in der Wiki-Anforderung eine Liste aller Tickets gezeigt, die unter der Jira-*Epic* gruppiert sind und somit zu der Anforderung gehören. Mit dieser Verbindung wird in der Jira-*Epic* ein Link zu der Beschreibung im Wiki automatisch hinzugefügt. Somit kann auch von der

Jira-*Epic* zurück zur beschriebung und allen daran hängenden Informationen gesprungen werden.

Die Anforderungsvorlage schlägt vor, dass Entwickler, Tester und weitere Stakeholder für die Anforderung definiert werden. Falls sich die Anforderung ändert, werden alle eingetragenen Personen benachrichtigt. Diese Funktion schafft eine Transparenz dem Kunden gegenüber, er weiß genau, wann an seiner Anforderung gearbeitet wurde.

Alle Anforderungen werden automatisch von Confluence in einer übersichtlichen Liste dargestellt und können so priorisiert werden, so eine Liste ist in Abbildung 5.3 zu sehen. Über einen Status ist zu erkennen, ob die Anforderung umgesetzt ist, ob an ihr gerade gearbeitet wird oder sie sich noch in einem Entwurfsstadium befindet.

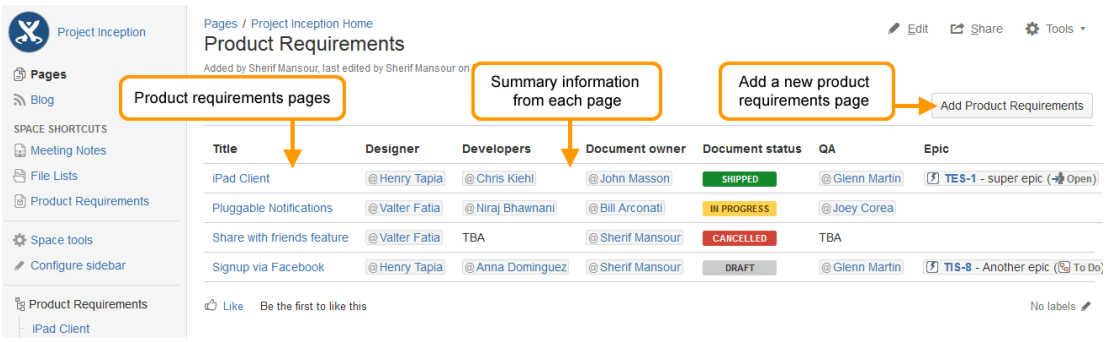

Abbildung 5.3: Alle Anforderungen auf einen Blick (Bild von Atlassion Website[11])

Da in der *DVG* alle Tests, die die Fachlichkeit von Anforderungen abdecken (Akzeptanztests genannt), mit dem Test-Werkzeug Test-Editor[12] entwickelt werden, stehen sie auf einem Webserver zur Verfügung. Jeder Test ist über eine URL erreichbar und kann so leicht zu der Anforderung im Wiki hinzugefügt werden.

[11] https://confluence.atlassian.com/display/CONF55/Product+Requirements+Blueprint
[12] http://testeditor.org

5.3 Zusammenfassung

Mit dieser Unterstützung wird besonders die tägliche Arbeit erleichtert. In der Analyse zeigte sich, dass besonders die Zusammenarbeit im verteilten Team Unterstützungsbedarf aufwies. Dem wurde überwiegend mit Kommunikationsunterstützung nachgekommen. So kann das Team näher zusammen rücken und sein Wissen austauschen.

Wie stellt sich die IT-Unterstützung im ganzen dar? Die Prozessbeschreibung beginnt in den groben Prozessschritten, angefangen mit der Übersicht über ein Release und wird immer feiner bis hin zur täglichen Arbeit.

Das Release

Als Einstiegspunkt für ein agiles Projekt in der *DVG* sollte stets das Wiki gelten. Bei Projektstart wird weiterhin in Word ein Fachkonzept vom Auftraggeber erstellt. Das Fachkonzept wird im Anforderungsworkshop in einzelne Anforderungen zerlegt, die jeweils in einer Confluence-Anforderung beschrieben und mit einer Jira-*Epic* verbunden werden.

Die *Epics* werden in Jira priorisiert und die am höchsten priorisierten *Epics* in *User Stories* zerteilt. Sie werden in Jira angelegt und bilden das *Product Backlog*.

Sobald das *Product Backlog* soweit gefüllt ist, dass für die ersten *Sprints* genug Aufgaben vorhanden sind und diese vom *Product Owner* priorisiert wurden, kann ein erstes Sprint-Planungsmeeting abgehalten werden. Im Sprint-Planungsmeeting geschieht das Schätzen der Tickets, diese werden in Jira, dem zu planenden Sprint, zugeordnet. Alle Tickets, die dem Sprint zugewiesen wurden, bilden das *Sprint Backlog*.

Die Erstellung jedes Produkt-Inkrements, die *Continuous Integration*, sowie die *Release*-Erstellung werden wie zuvor durch Jenkins unterstützt. Die Planung der *Releases* wird im Zusammenspiel zwischen Team-Kalender und der Jira Versionsplanung vorgenommen.

Der Sprint bzw. die tägliche Arbeit

Jegliche Terminfindung, sei es für Meetings, Telefonkonferenzen, Urlaub oder *Pair Programming* wird über den Team-Kalender in Confluence durchgeführt. Er gibt eine schnelle und

gute Übersicht über die Verfügbarkeit des Teams. Besteht der Bedarf zu schneller und direkter Kommunikation kann nun diese mittels eines Chats abgewickelt werden. Der Team-Kalender hat sich in der Praxis schon ausgezahlt und wird reichlich genutzt.

Das *Taskboard* von Jira ist die tägliche Anlaufstelle für die Aufgabenverteilung. Durch die Erweiterung der *Stand-up* Meetings durch WebCams und Mikrofone wurde die Kommunikation persönlicher gestaltet. Der erzielte Effekt ist, dass die Aufmerksamkeit steigt und die Teil-Teams sich besser wahrnehmen und ein Gemeinschaftsgefühl entsteht.

Besonders die standortübergreifende Arbeit unterstützt die Teamgemeinschaft und den Wissensaustausch. So können gemeinsam an einem digitalen Whiteboard Gedanken und Pläne erarbeitet und ausgetauscht werden.

Die Möglichkeit mit dem Eclipse-Plugin Saros über Standortgrenzen hinweg im Paar zu programmieren bietet neue Möglichkeiten in der Kollaboration. Der Sourcecode wird weiterhin in Subversion versioniert. Die nicht optimale Verbindung von Subversion zu Jira wurde in dieser Arbeit nicht betrachtet. Diese Lösung ist zwar nicht sehr übersichtlich, jedoch bietet sie die Möglichkeit nachzuvollziehen, welche Änderungen im Quellcode zu einem Ticket durchgeführt wurden. Diese Möglichkeit ist essentiell für die Nachvollziehbarkeit der Arbeit zu einer Anforderung und kommt der durchgängigen Projektdokumentation zugute.

6 Fazit

In dieser Arbeit wurde die Frage behandelt, wie Entwickler in einem verteilten Team, die nach einem agilen Vorgehensmodell arbeiten, am besten durch IT unterstützt werden können.

Es wurden die Arbeiten *A Holistic Approach to Developing a Progress Tracking System for Distributed Agile Team* und *Fully Distributed Scrum: The Secret Sauce for Hyperproductive Offshored Development Teams* vorgestellt. Aus ihnen wurden Vorschläge zur Unterstützung der agilen Teams der Beispiel-Firma *Dummy Versicherungs Gruppe* (*DVG*) erarbeitet. Die Teams der Firma arbeiten teils an verteilten Standorten.

Die Analyse zeigte deutlich, dass der Prozessablauf größtenteils schon so technisch unterstützt ist, dass er auch von verteilten Teams genutzt werden kann. Häufig wird ein digitales *Workboard* zur Aufgabenverwaltung und ein Wiki zur Dokumentation genutzt. Jedoch sind es besonders die täglichen Aufgaben, bei denen es weitergehende Unterstützung durch Technik bedarf.

Dazu gehörten einfache Aufgaben wie Terminfindung und direkte schnelle Kommunikation die mit einem flexiblen Team-Kalender und einem Chat unterstützt werden. Aber auch Aufgaben die nur mit mehr Aufwand umzusetzen sind, wie beispielsweise das Zusammenarbeiten an neuen Architekturen, wären mit entsprechender technischen Hilfe leichter zu realisieren.

Erst wenn Team-Mitglieder von den unterschiedlichen Standorten zusammen an einem Ticket arbeiten, und das Gefühl aufkommt, sie säßen an einem Schreibtisch bzw. ständen an einem Whiteboard zusammen, kann das Team so agieren, als gäbe es diese Standortgrenze nicht. Die Unterstützung die ein verteiltes Team benötigt, um diesen Zustand zu erreichen wurde in Kapitel 5 beschrieben.

Literaturverzeichnis

[AIG12] Sultan Alyahya, Wendy K. Ivins, and W. A. Gray. A holistic approach to developing a progress tracking system for distributed agile teams. In *Proceedings of the 2012 IEEE/ACIS 11th International Conference on Computer and Information Science*, ICIS '12, pages 503–512, Washington, DC, USA, 2012. IEEE Computer Society.

[Bec99] Kent Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[Coh05] Mike Cohn. *Agile Estimating and Planning*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.

[Flo06] Nick V. Flor. Globally distributed software development and pair programming. *Commun. ACM*, 49(10):57–58, October 2006.

[HAB+02] James D. Herbsleb, David L. Atkins, David G. Boyer, Mark Handel, and Thomas A. Finholt. Introducing instant messaging and chat in the workplace. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '02, pages 171–178, New York, NY, USA, 2002. ACM.

[Hod13] Matt Hodges. Work smarter, not harder. *eclipse Magazin*, Mai 2013.

[Jon00] Capers Jones. *Software Assessments, Benchmarks, and Best Practices*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000.

[KB14] Arie van Bennekum Alistair Cockburn Ward Cunningham Martin Fowler James Grenning Jim Highsmith Andrew Hunt Ron Jeffries Jon Kern Brian Marick Robert C. Martin Steve Mellor Ken Schwaber Jeff Sutherland Dave Thomas Kent Beck,

Mike Beedle. Manifesto for agile software development. http://agilemanifesto.org/, 12 14.

[KMAM10] Mira Kajko-Mattsson, Gayane Azizyan, and Miganoush Katrin Magarian. Classes of distributed agile development problems. In *Proceedings of the 2010 Agile Conference*, AGILE '10, pages 51–58, Washington, DC, USA, 2010. IEEE Computer Society.

[KS13] Jeff Sutherland Ken Schwaber. Scrum guide. http://www.scrumguides.org/, 07 2013.

[RG09] Christian Willems Matthias Quasthoff Lutz Gericke Christoph Meinel Raja Gumienny, Oliver Böckmann. Verteiltes design thinking mit teleboard. *Hasso Plattner Institut für Softwaresystemtechnik GmbH, Potsdam*, 2009.

[Roy87] W. W. Royce. Managing the development of large software systems: Concepts and techniques. In *Proceedings of the 9th International Conference on Software Engineering*, ICSE '87, pages 328–338, Los Alamitos, CA, USA, 1987. IEEE Computer Society Press.

[SB01] Ken Schwaber and Mike Beedle. *Agile Software Development with Scrum*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 2001.

[Som07] Ian Sommerville. *Software Engineering*. PEARSON - Studiom, 2007.

[SPS14] Julia Schenk, Lutz Prechelt, and Stephan Salinger. Distributed-pair programming can work well and is not just distributed pair-programming. In *Companion Proceedings of the 36th International Conference on Software Engineering*, ICSE Companion 2014, pages 74–83, New York, NY, USA, 2014. ACM.

[SSK+09] Jeff Sutherland, Guido Schoonheim, N. Kumar, V. Pandey, and S. Vishal. Fully distributed scrum: Linear scalability of production between san francisco and india. In *Proceedings of the 2009 Agile Conference*, AGILE '09, pages 277–282, Washington, DC, USA, 2009. IEEE Computer Society.

[SSR09] Jeffrey Sutherland, Guido Schoonheim, and Mauritz Rijk. Fully distributed scrum: Replicating local productivity and quality with offshore teams. In *HICSS*, pages

1–8. IEEE Computer Society, 2009.

[SSRR08] Jeff Sutherland, Guido Schoonheim, Eelco Rustenburg, and Maurits Rijk. Fully distributed scrum: The secret sauce for hyperproductive offshored development teams. In *Proceedings of the Agile 2008*, AGILE '08, pages 339–344, Washington, DC, USA, 2008. IEEE Computer Society.

[Wik14a] Wikipedia. Inkrementelles vorgehensmodell — wikipedia, die freie enzyklopädie, 2014. [Online; Stand 17. Januar 2015].

[Wik14b] Wikipedia. Scrum. http://de.wikipedia.org/, 10 2014. Wikipedia Artikel zu Scrum.

[Wik15] Wikipedia. Wasserfallmodell. http://de.wikipedia.org/, 01 2015. Wikipedia Artikel zum Wasserfallmodell.

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

Hamburg, 13. Februar 2015 Matthias Holsten