Ausgewählte Beiträge aus der ersten Auflage von

# »Agile Projekte mit Scrum, XP und Kanban im Unternehmen durchführen«

Die aktuelle Auflage erhalten Sie hier.


Henning Wolf, Agile Projekte mit Scrum, XP und Kanban , dpunkt.verlag, ISBN 978-3-86490-266-6

# 2 Ein agiles Projekt ist kein Ponyhof

Holger Koschek

*In einem großen mittelständischen Unternehmen bekommen IT-Abteilung und Fachbereich die Chance, auf der grünen Wiese ein neues Anwendungssystem zu errichten. Die externen Architekturberater empfehlen Scrum als agilen Ansatz für das Projektmanagement, und das Unternehmen lässt sich darauf ein. Mit zunehmender Scrum-Erfahrung stellt auch dieses Unternehmen fest, dass ein agiles Projekt kein Ponyhof ist, sondern harte Arbeit – die sich aber lohnt, wie das Projektergebnis und die Motivation der Mitarbeiter belegen.*

## 2.1 Der Weckruf

Alle waren zum Kick-off des neuen Projekts gekommen: Auftraggeber, Business-Analysten, Vertreter des zuständigen Fachbereichs, das designierte Entwicklerteam und deren Linienvorgesetzte. Alle waren gespannt auf die Ideen, mit denen man das ehrgeizige Ziel erreichen wollte, von dem viele glaubten, dass es unmöglich erreicht werden konnte. Und dann hatte auch noch das Gerücht die Runde gemacht, man wolle dieses Projekt mit einem agilen Projektmanagement-Framework namens Scrum durchführen. Was aber war dieses Scrum? Und was sollte daran besser sein? Dies zu beantworten, war unsere Aufgabe.

Mein Kollege Jo und ich hatten unseren Standardfoliensatz zur Scrum-Einführung im Gepäck. Als Präsentationsform wählten wir die mehrfach bewährte agile Version des Klassikers »guter Cop, böser Cop«. Es ist immer eine Freude, die Reaktion der Zuhörer zu erleben, wenn Jo mitten in der Präsentation seine erste Zwischenfrage stellt. Ich präsentierte gerade das Product Backlog und erläuterte, wie dort fachliche Anforderungen beschrieben und nach dem Geschäftswert und anderen Kriterien bewertet werden, als Jo verbal grätschte: »Und wo sind die einzelnen Aufgaben, die zur Implementierung dieser Anforderungen nötig sind? Dahinter stecken schließlich viele verschiedene technische Details. Wenn ich die nicht kenne, dann kann ich doch unmöglich den Aufwand schätzen,

den die Umsetzung einer fachlichen Anforderung verursacht!« »Eine gute Frage!«, entgegnete ich seelenruhig, während in der letzten Reihe einige Zuhörer aufgeschreckt ihre Aufmerksamkeit auf unseren aufkeimenden Diskurs lenkten. »Ich möchte diese Frage mit zwei Anmerkungen beantworten. Erstens: Wir schätzen nicht etwa den Aufwand für die fachlichen Anforderungen, sondern nur die relative Größe. Mehr können wir zu diesem frühen Zeitpunkt auch gar nicht tun. Zweitens: Die einzelnen Aufgaben legt das Team erst zu Beginn des Sprints fest, in dem diese fachliche Anforderung umgesetzt werden soll.« »Aber ist das nicht viel zu spät? Wie soll man denn auf Grundlage solch spärlicher Informationen eine vernünftige Projektplanung machen?«, echauffierte sich Jo. Ich gab daraufhin einen kurzen Überblick über Planung und Controlling agiler Projekte. Spätestens jetzt waren alle Zuhörer hellwach. Wieder einmal hatten wir den Eindruck, dass Jo mit seinen Fragen den Nerv der Skeptiker traf. Sie galt es zu überzeugen – nicht hier und heute, aber im Laufe des Projekts. Deshalb luden wir alle ein, agiles Projektmanagement live in unserem Projekt zu erleben. Gelegenheiten gab es viele: Sprint-Review und (nach Rücksprache mit dem Team) Daily Scrum standen jedem Interessierten offen, und ein transparentes Projektcontrolling sollte schonungslos den echten Status des Projekts offenlegen – jederzeit, für jeden sichtbar. Das war ein Novum – nicht allein in diesem Unternehmen. Wir wussten aus anderen agilen Coachings, dass uns diese Transparenz und Ehrlichkeit neue Skeptiker bescherte. Wir hatten aber auch erfahren, dass meist deren Neugier siegte und sie das Projekt zunächst einmal beobachteten, bevor sie es in der Luft zerreißen wollten (wozu es eigentlich nie kam).

Unser agiler Weckruf hatte wieder einmal funktioniert. Allen war klar geworden, dass dieses Projekt anders sein würde. Wir wollten eine positive Neugier wecken und Ängste abbauen. Die größten Ängste hatte das Entwicklungsteam, das war uns klar. Für diese Gruppe warf unsere Einführungsveranstaltung erfahrungsgemäß mehr Fragen auf, als sie beantwortete. Deshalb hatten wir uns für den nächsten Tag mit dem Team verabredet, um Scrum aus deren Perspektive zu betrachten und gemeinsam einen Fahrplan für den Projektstart zu entwerfen.

Ihre Einführung in Scrum soll einen nachhaltigen Eindruck hinterlassen? Dann spielen Sie doch mit den klassischen Vorurteilen gegen agile Frameworks und Vorgehensmodelle. So nehmen Sie auf eine augenzwinkernde Art und Weise allen Skeptikern den Wind aus den Segeln.

## 2.2 Das Team findet sich

Wir trafen uns in kleiner Runde. Ein bewusst informelles Ambiente sorgte für eine Atmosphäre, in der wir uns ganz offen allen Fragen zu Scrum widmen konnten. Um uns langsam an die schwierigen Themen wie Eigenverantwortung heranzutasten, begannen wir mit der Beschreibung des idealen Projektraums – und

wurden prompt mit dem ersten Problem konfrontiert. Die Entwickler kamen aus verschiedenen Abteilungen, und jede Abteilung hatte ihre eigenen Räume. Das Entwicklungsteam war also räumlich verteilt. Immerhin saßen alle im selben Gebäude, aber das half uns nicht weiter. Was tun? Der Auftraggeber verstand unser Problem und versprach, sich auf die Suche nach einem geeigneten Projektraum zu machen. Das Entwicklerteam gab ihm bei dieser Gelegenheit noch eine weitere Aufgabe mit auf den Weg: Er sollte sicherstellen, dass die alten Arbeitsplätze für die Dauer des Projekts freigehalten wurden. Wir vermuteten dahinter die Angst, den liebgewonnenen Arbeitsplatz nebst »Mitbewohnern« zu verlieren und nach Projektende einen schlechteren Platz zugewiesen zu bekommen. Als wir ein Teammitglied vorsichtig darauf ansprachen, erfuhren wir einen weiteren Grund: Sein Vorgesetzter hatte ihn nur zu 70 Prozent für das Projekt freigestellt und erwartete, dass er die restlichen 30 Prozent seiner Arbeitszeit am alten Arbeitsplatz mit anderen Aufgaben verbringen sollte. Als wir daraufhin die anderen Entwickler befragten, stellte sich heraus, dass mit einer Ausnahme alle nur in Teilzeit für das neue Projekt vorgesehen waren. Angesichts der strategischen Tragweite und des knappen Zeitplans dieses Projekts war das eine ungewöhnliche Entscheidung, die wir aber vorerst akzeptierten. Wir wollten schließlich Sicherheit schaffen, anstatt weitere Unruhe zu stiften. Deshalb fuhren wir mit der Beschreibung des Teamraums fort. Wir erläuterten das Taskboard, an dem für alle sichtbar die Aufgaben des laufenden Sprints verzeichnet waren und der Fortschritt notiert wurde. Wir erklärten noch einmal das Daily Scrum, in dem sich das Team täglich synchronisieren sollte, und hoben die große Bedeutung der Kommunikation im Team hervor.

Das Team verdient zu Beginn der agilen Reise besondere Beachtung. Ein Certified-ScrumMaster-Kurs wirft für die Teammitglieder weitere Fragen auf, weil der Kurs auf ihren ScrumMaster zugeschnitten ist, die Belange des Teams aber nur indirekt betrachtet. Versetzen Sie sich in die Lage eines agilen Teamnovizen und Sie werden feststellen, dass die agilen Werte, das eigenverantwortliche Handeln, planerische Tätigkeiten und die für erfolgreiches Teamwork nötigen Soft Skills am Anfang mehr Last denn Freude sind. Von den ausgeprägten handwerklichen Fähigkeiten, die von agilen Entwicklern erwartet werden, soll hier gar nicht die Rede sein.

Die Mitglieder unseres Teams kannten sich fast alle und schienen gut miteinander auszukommen. Sie freuten sich auf die anstehenden Aufgaben und die spannenden neuen Technologien, die wir in diesem Projekt erproben und verwenden wollten. Als wir ganz nebenbei über diese Technologien und über mögliche Softwarearchitekturen diskutierten, machten wir eine erstaunliche Erfahrung: Die Teammitglieder erwarteten von uns, den Beratern, fertige Architekturskizzen und Empfehlungen für die zu verwendenden Technologien. Ein wenig überrascht blickten wir in die Runde. Diejenigen, die diesen Wunsch geäußert hatten, waren nicht etwa frischgebackene Uniabsolventen, sondern gestandene Softwareinge-

nieure mit mehrjähriger Praxiserfahrung. »Wie kann man sich aus solch spannenden technischen Diskussionen freiwillig heraushalten?«, schoss es mir durch den Kopf. Ich musste diese Frage nicht einmal laut stellen, um eine Antwort zu bekommen. Eigentlich waren es zwei Antworten. »Es ist hier so üblich, dass die Softwareentwickler von einem Architekten angeleitet werden, der die technischen Entwurfsentscheidungen trifft«, sagte einer, und ein zweiter ergänzte: »Und außerdem seid ihr als Softwarearchitekten für unser Projekt angekündigt worden. Stimmt das etwa nicht?« Doch, es stimmte. Und die Idee, das Projekt mit Scrum durchzuführen, war genau genommen erst während der ersten Architekturdiskussionen entstanden. Da stellten wir nämlich fest, dass wir zwei Möglichkeiten hatten: Entweder verloren wir uns in theoretischen Diskussionen, ergänzt durch endlose Software- und Technologieevaluationen. Oder wir legten einfach los, um in kurzen Iterationen erste Ergebnisse zu erzielen. Wir hatten den designierten Projektleiter deshalb vor die Wahl zwischen einem eher klassischen und einem agilen Vorgehen gestellt – obwohl das gar nicht unser ursprünglicher Beratungsauftrag gewesen war.

## 2.3 Der eigentliche Auftrag

Beginnen wir doch einmal ganz am Anfang: Unsere Aufwartung hatten wir tatsächlich als Softwarearchitekten gemacht. Ausgestattet mit dem nötigen Architektur- und Technologiewissen und einer gehörigen Portion Erfahrung, sollten wir gemeinsam mit dem Kunden auf der sprichwörtlichen grünen Wiese eine neue Softwareplattform für die Kernprozesse des Unternehmens entstehen lassen. Ist das nicht der Traum eines jeden Beraters? Fast ohne Rücksicht auf bestehende Infrastruktur und Applikationen sollten wir mit modernen Technologien und frischen Ideen die Softwarearchitektur für die nächsten Jahre definieren und implementieren.

Die Diskussionen mit den Softwarearchitekten des Kunden waren spannend und meistens auch zielführend. Trotzdem beschlich uns das Gefühl, dass wir nicht schnell genug waren. Deutliches Indiz für eine Tendenz zum Theoretisieren war das Verhältnis von Design- zu Codierungszeit: Zwei Wochen Recherche und Dokumentation stand keine einzige Zeile Code gegenüber! Wir waren mittendrin im Wasserfall, Phase 2: Design. Dabei hatten wir noch gar keine Ahnung davon, was die zu entwickelnde Plattform fachlich zu leisten imstande sein sollte. Die Fachkonzeption (Wasserfall, Phase 1: Anforderungen) oblag nämlich einer anderen Abteilung, von der wir bisher weder ein Gesicht noch ein Dokument zu sehen bekommen hatten.

Der Tag, an dem wir die Kollegen aus der Fachabteilung kennenlernten und uns anhand eines Fachkonzepts über die fachlichen Anforderungen austauschten, markierte einen positiven Wendepunkt. Endlich wussten wir, wofür wir eine technische Plattform bauen sollten. Technologie als Selbstzweck war noch nie unser

Ziel gewesen. Deshalb waren wir froh, endlich eine fachliche Vision vermittelt zu bekommen, die auf unsere zart sprießende Softwarearchitektur wie ein Kraftdünger wirkte. Anhand fachlicher Anwendungsfälle fiel es uns viel leichter, technische Alternativen zu bewerten und auszuwählen. Einige Schwerpunkte verschoben sich zwangsläufig und nicht funktionale Anforderungen wie Sicherheit und Performanz konnten konkreter definiert werden. Nichtsdestotrotz waren wir immer noch theoretisch unterwegs. Den Kollegen der Fachabteilung konnten wir unsere Ideen zwar schildern, aber nicht zeigen. Je abstrakter unsere Ideen, desto schwieriger war es, die Fachexperten zu begeistern. Und wer für eine Sache keine Begeisterung verspürt, der setzt sich nicht bedingungslos für sie ein. Wir wollten etwas Vorzeigbares haben, an dem man sich inhaltlich reiben konnte, das Diskussionen auslöste und Ideen entfesselte. Stattdessen hatten wir nur einen Haufen Papier. Das sollte, nein, das musste sich ändern.

Anstatt lange zu analysieren, zu planen und zu entwerfen, sollte ein Projektteam lieber loslegen, sobald es eine erste belastbare Vorstellung von der anstehenden Aufgabe hat. Dabei leistet eine Produktvision gute Dienste. Sie beschreibt das »große Ziel«, ohne jedoch den Weg vorzugeben und ohne zu stark einzuschränken. Schließlich ändert sich die Welt um uns herum tagtäglich. Ideen und Anforderungen kommen und gehen, werden verändert, verworfen und wieder hervorgeholt. All das hat Einfluss auf die Details des Endprodukts. Das Wesen dieses Produkts, ausgedrückt in der Vision, überdauert aber in der Regel die Dynamik des Projektalltags. Je detaillierter und weitreichender im Projekt geplant wird, desto mehr Änderungen müssen später wieder an die neue Projektrealität angepasst werden. Die ursprüngliche Planung war dann eventuell eine Fehlinvestition. Das Geld hätte man besser in einen frühen Erkenntnisgewinn investiert, indem man einfach einmal loslegt.

## 2.4 Guerillagilisierung

Um schnell konkrete Ergebnisse zu erzielen und um das Risiko zu minimieren, in die falsche Richtung zu laufen, beschlossen Jo und ich, den Kunden von den Vorzügen eines agilen Vorgehens zu überzeugen. Dabei gingen wir ganz vorsichtig zu Werke. Wir begannen mit etwas Unverfänglichem, das dem Uneingeweihten nicht sofort als agiles Werkzeug ins Auge sprang: Wir erstellten ein Product Backlog und füllten es mit fachlichen und technischen User Stories. Die Begründung für die Einführung des Product Backlog lieferten wir gleich mit, indem wir auf die große Anzahl an Ideen deuteten, die wir in den vergangenen Wochen kreiert hatten. Um diese Ideen nicht zu verlieren, zugleich aber nicht alle Ideen sofort bewerten zu müssen, sollten sie zunächst ins Product Backlog aufgenommen und geschätzt werden. Unser Counterpart auf Kundenseite war ein klassischer Projektleiter. Er verstand zwar den grundsätzlichen Nutzen eines Product Backlog. Konfrontiert mit der Schätzung relativer Größen anstelle der gewohnten konkreten Aufwände in Entwicklertagen, wurde er jedoch zunehmend unsicherer. Viel-

leicht haben wir ihm den Vorteil des relativen Schätzens und den Unterschied zwischen einer Schätzung und einer Aufwandsplanung nicht verständlich genug erklärt. Jedenfalls fehlten ihm in dem Bild, das unser Product Backlog darstellte, zwei wesentliche Informationselemente für die Projektplanung: Aufgaben und Aufwände. Unser Hinweis, dass diese vom Team beizubringen seien, prallte gegen die Wand des klassischen Projektmanagements: Wie er denn ein Projekt ohne solche Kenngrößen planen solle, fragte uns der verunsicherte Projektleiter. Außerdem sei das Team noch gar nicht zusammengestellt. Die Entwickler seien es ohnehin nicht gewohnt, eigenständig ihre Aufgaben zu definieren – und das müssten sie auch nicht, denn schließlich gebe es ihn, den Projektleiter und Chefarchitekten. Unsere Soft-Skill-Antennen empfingen eine Mischung aus Unverständnis, Furcht vor Neuem und einer unbestimmten Existenzangst, ausgelöst durch das Infragestellen der Projektmanagementrolle. Es wurde höchste Zeit, einen Gang zurückzuschalten.

Nicht nur das Team steht den unorthodox wirkenden Werten, Prinzipien und Praktiken zunächst oft skeptisch gegenüber. Auch der frisch gebackene ScrumMaster braucht einige Zeit, um sich in seiner Rolle zurechtzufinden. Oft wird diese Rolle mit einem Projektleiter oder Chefarchitekten besetzt. Das ist nicht schädlich, aber auch nicht immer hilfreich. Für viele Projektleiter kommt der Rollenwechsel einem Machtverlust gleich. Anstatt eine Gruppe von Entwicklern zu führen, sind sie plötzlich »nur noch« Schiedsrichter für ein selbstorganisierendes Team und dürfen sich darum kümmern, dass das Team störungsfrei arbeiten kann. Das gilt es zu bedenken, wenn man einen unerfahrenen ScrumMaster erfolgreich an seine neue Aufgabe heranführen möchte.

## 2.5 Scrumkomplex

Wir machten Zugeständnisse. So erweiterten wir das Product Backlog um eine Spalte, in der wir zu den User Stories erste Aufgaben für die Realisierung erfassten. Gemeinsam mit dem Projektleiter trafen wir den Kollegen, der das Fachkonzept erstellt hatte, um dieses Dokument nach Stoff für neue, fachliche User Stories zu durchsuchen. Damit schufen wir ein neues Problem, denn plötzlich enthielt das Backlog sowohl technische als auch fachliche User Stories. Wie konnte man diese auseinanderhalten? Und warum gab es überhaupt technische User Stories? Widersprach das nicht der agilen Lehre, nach der nur die Fachlichkeit zählt? Diese Frage eines Kollegen konnten wir mit dem Hinweis entkräften, dass wir zwei verschiedene Projektaufträge bekommen hatten: Zum einen sollte eine technische Plattform für das Management der Geschäftsprozesse entstehen. Auf dieser Plattform sollte dann ein ausgewählter Geschäftsprozess exemplarisch, aber produktiv umgesetzt werden. Dementsprechend hatten wir es mit einer »technischen Fachlichkeit« (die Plattform) und einer »fachlichen Fachlichkeit« (der Geschäftsprozess) zu tun. Deshalb spendierten wir dem Product Backlog eine weitere Spalte, um dort die beiden Projekte zu unterscheiden. Unser

ursprüngliches Ziel, das Product Backlog so einfach wie möglich zu gestalten, um den Scrum-Neulingen die Angst zu nehmen, hatten wir damit verfehlt. Die Tatsache, dass nicht Scrum daran schuld war, sondern das komplexe Projektgebilde, half uns auch nicht weiter. Wir waren ja weiterhin der Meinung, dass solche komplexen Projekte nur agil in den Griff zu bekommen waren.

Es wurde immer deutlicher, dass wir nicht weitermachen konnten, ohne zuvor allen Projektbeteiligten die Scrum-Grundlagen vermittelt zu haben. Es wurde ohnehin Zeit, das Entwicklerteam zusammenzustellen und den Kontakt zur Fachabteilung auszubauen. Da traf es sich gut, dass der Auftraggeber ein halbtägiges Kick-off-Meeting anberaumt hatte. Wir holten unseren Standardfoliensatz zur Scrum-Einführung aus der Schublade und freuten uns auf die Wiederaufführung unseres Klassikers »guter Cop, böser Cop« – womit ich wieder am Anfang dieser Geschichte angekommen wäre. Wie aber ging es weiter?

Nur selten kann man sich sein erstes Scrum-Projekt aussuchen. Sollten Sie tatsächlich die Wahl haben, dann entscheiden Sie sich am besten für ein überschaubares Projekt mit strategischer Bedeutung. Ein komplexes Projekt, wie das hier geschilderte, legt die Einstiegshürde in die agile Welt unnötig hoch. Einem ganz einfachen und unkritischen »Spielprojekt« fehlt auf der anderen Seite die Relevanz. Selbst wenn es später als Erfolg gewertet wird, so werden die Kritiker immer behaupten, dass die agile Vorgehensweise nur deshalb funktioniert hat, weil das Projekt so klein und überschaubar gewesen ist. Im ungünstigsten Fall wird die Erfolgsmeldung von den Nachrichten aus den großen, geschäftskritischen Projekten übertönt.

## 2.6 Siege und Niederlagen

Das Team forderte, wie bereits erwähnt, die Kombination von Besitzstandswahrung und Teamraum. Und der Auftraggeber erfüllte tatsächlich beide Wünsche. Wir waren stolz, als er uns die neuen Büroräume zeigte, die sich in einem Nebengebäude befanden – hatten wir doch einen ersten Punktsieg für die Agilität errungen. Die räumliche Trennung vom Rest der Firma betrachteten wir als Vorteil. Wie der Zufall es so wollte, »wohnten« wir jetzt Tür an Tür mit dem Fachbereich, für den wir eine erste Anwendung auf unserer neuen Plattform entwickeln sollten. Das war praktisch, denn die kurzen Wege sollten – so unser Plan – die Zusammenarbeit intensivieren. Zugegeben: Die Räume waren nicht ganz so modern wie in der Firmenzentrale, aber wir hatten ausreichend Platz und die nötige Ruhe. Allerdings bestand unser kleines Reich aus zwei Räumen (zuzüglich Besprechungsraum), die der Projektleiter sogleich aufteilte: ein Raum für ihn und die Business-Analysten und ein Raum für uns Entwickler (und Architekten). »Halb so wild«, dachten wir uns und begannen mit der agilen Einrichtung unseres neuen Büros: Eine Metaplan-Wand für das Taskboard wurde aufgebaut und für den ersten Sprint präpariert. Das Team und der Projektleiter beobachteten interessiert unser Treiben. Wir erläuterten das Taskboard, wiederholten noch ein-

mal die innere Struktur eines Sprints und luden alle zum Sprint-Planungsmeeting in den Besprechungsraum ein.

## 2.7 Sprint-Planung I

Spätestens bei der Sprint-Planung rächte es sich, dass das Team nicht an der Erstellung des Product Backlog beteiligt gewesen war. Die Backlog Items waren schon priorisiert, wobei fachliche und technische Abhängigkeiten berücksichtigt waren. Dementsprechend konnten wir uns recht zügig an die gemeinsame Größenschätzung der Items machen. Der Projektleiter, Jo und ich waren voll im Thema – kein Wunder, hatten wir doch das Product Backlog gemeinsam aufgebaut. Den fünf übrigen Teammitgliedern mussten wir die Geschichten hinter den Items erläutern. Ein Teammitglied war als interner Architekt in das Projekt geholt worden und sollte unser Sparringspartner für die Softwarearchitektur sein. Er stellte viele gute Fragen, wollte insbesondere die Entscheidungsfindung für die grundlegende Architektur verstehen und brachte zudem viele frische Ideen ein, die manches Backlog Item in einem neuen Licht erscheinen ließen. Leider verzögerte seine Wissbegierde die Sprint-Planung, und außerdem spürten wir, dass die anderen Teammitglieder einigen unserer Diskussionen kaum noch folgen konnten oder wollten (was zu ihrer Forderung nach einem Architekten für das Team passte). Um das Team nicht ganz zu verlieren, beschränkten wir uns darauf, die fachlichen Details zu erläutern. Das allein war schon herausfordernd genug, denn einige Kollegen waren es nicht gewohnt, sich Gedanken über die fachliche Ausgestaltung zu machen. Bisher hatten sie klare technische Vorgaben bekommen und sich im Zweifelsfall an ihren Softwarearchitekten oder Projektleiter gewandt. Jetzt sollten sie auf einmal fachliche Entscheidungen treffen oder zumindest mittragen – und dann auch noch die Größe der fachlichen wie technischen Backlog Items schätzen. »Größe? Warum nicht den Aufwand?«, fragte uns einer der Entwickler. Wir erläuterten erneut das relative Schätzen und zückten dann das Planning-Pokerspiel[1]. Als wir die Karten austeilten, schlug die Skepsis in echtes Interesse um: Kartenspielen im Projekt, das versprach Abwechslung im Projektalltag. Wir erläuterten die Spielregeln und legten sofort los. Schnell war ein Backlog Item als Referenz gefunden. Dieses wurde mit dem Größenwert 2 belegt. Dann sollte das erste Backlog Item im Verhältnis zu diesem Referenzwert geschätzt werden. Wenngleich alle die Spielregeln verstanden hatten, so war es doch schwierig, sich für einen Größenwert zu entscheiden. Wer weiß, wie groß die anderen Backlog Items sein mochten? Auch stellten wir anhand der Nachfragen fest, dass einige Teammitglieder die Backlog Items inhaltlich noch nicht durchdrungen hatten. Die erste Runde Planning Poker förderte viele verschiedene Schätzwerte zutage. Als wir die »Spieler« mit dem höchsten und dem niedrigsten Wert aufforderten,

1. Planning Poker ist ein Warenzeichen von Mountain Goat Software, Inc.

ihre Schätzung zu begründen, bekamen wir Antworten, die unterschiedlicher kaum sein konnten. Den höchsten Wert hatte der interne Architekt gelegt. Er erläuterte seine Wahl mit dem Hinweis auf die technischen Rüstkosten und die Erfüllung der nicht funktionalen Anforderungen Sicherheit, Skalierbarkeit, Erweiterbarkeit und Wiederverwendbarkeit. Der technische Entwurf, den er für die Umsetzung dieses Backlog Item skizzierte, wäre eine Bereicherung für jedes Lehrbuch zum objektorientierten Design. Für unser konkretes Problem war der Entwurf jedoch viel zu komplex. »Schon mal was vom KISS-Prinzip gehört?«, wollte ich von dem Kollegen wissen. Kopfschütteln. »Keep it simple, stupid – was so viel bedeutet wie: Halte es so einfach wie möglich«, erläuterte ich das Akronym. »Unser Entwurf soll so einfach wie möglich und so kompliziert wie nötig sein – aber eben nicht komplizierter.« »Aber wenn sich die Anforderungen verändern und neue Funktionen gewünscht sind, die der einfache Entwurf nicht erfüllen kann?«, warf er ein. »Dann erweitern wir unser einfaches System, indem wir das Design und den Code refaktorisieren. Dank unserer Unit Tests sind wir in der Lage zu überprüfen, ob sich das refaktorisierte System genauso verhält wie sein Vorgänger. Die Tests minimieren das Risiko des Refaktorisierens.« Auf diese Argumentation ließ er sich nur zum Schein ein – vermutlich, weil der Projektleiter drängte. Ich wusste jetzt, dass ich die Entwürfe dieses Kollegen besser etwas im Auge behielt. Tatsächlich tendierte er zu sehr komplexen, generischen Entwürfen, die für alle Eventualitäten gerüstet waren – auch wenn diese vermutlich nie benötigt wurden und deshalb den Code nur unnötig aufblähten und zudem schwerer verständlich machten.

Das Planning-Pokerspiel machte dem gesamten Team viel Spaß. In dieser spielerischen Atmosphäre trauten sich auch die schüchternen Kollegen, ihre Fragen zu stellen. Am besten gefiel allen die Tatsache, dass die Schätzwerte gemeinsam festgelegt wurden. In der Vergangenheit hatten sie ihre Aufwände immer alleine schätzen müssen. Das Risiko von Fehleinschätzungen lag dort deutlich höher, und außerdem wurde man im schlimmsten Fall allein zur Verantwortung gezogen. Mit dem Team im Rücken wurden alle ein wenig zuversichtlicher und mutiger. Und so trafen wir am Ende des Vormittags gemeinsam eine Auswahl an Backlog Items, die wir im ersten Sprint umsetzen wollten. Die Items passten fachlich gut zueinander, sodass wir schnell ein Sprint-Ziel formulieren konnten.

## 2.8 Sprint-Planung II

Am Nachmittag kamen wir dann endlich zu dem Punkt, auf den der Projektleiter schon so lange gewartet hatte: Wir definierten die Aufgaben (Tasks) zu allen für den Sprint ausgewählten Backlog Items. Der Projektleiter überraschte uns mit der Aussage, dass er bereits die passenden Aufgaben parat habe – und zwar für alle Backlog Items! Erstaunt überließen wir ihm das Feld, und wir wurden belohnt mit einem generischen Aufgabenraster, das den einzelnen Backlog Items einen

Mini-Wasserfall überstülpte: »Analyse«, »Architekturdesign«, »Implementierung«, »Integrationstest«, »fachlicher Test« – so (oder zumindest so ähnlich) lauteten die Tasks. Das missfiel nicht nur uns, sondern glücklicherweise auch dem Softwarearchitekten des Teams. Der hatte nämlich tatsächlich schon konkrete Implementierungsideen für die Items. Ihn mussten wir allerdings immer wieder bremsen, damit er nicht über das KISS-Ziel hinausschoss. Jo und ich steuerten die Ideen aus den initialen Architekturdiskussionen bei. So definierten wir im Dreierkreis die Aufgaben zu den anstehenden Backlog Items, während der Rest des Teams (inklusive Projektleiter) zuschaute und versuchte, unseren Gedankengängen zu folgen. Vielleicht hätten wir die anderen Teammitglieder immer wieder aktiv in das Geschehen einbinden müssen, aber uns lief die Zeit davon. So entschieden wir uns für einen pünktlichen Sprint-Beginn und gegen ein Team mit ausgeglichenem Wissensstand, in der Hoffnung, den Know-how-Transfer während des Sprints bewerkstelligen zu können. Das Commitment des Teams zum Erreichen des Sprint-Ziels fiel am Abend entsprechend halbherzig aus, aber darüber sahen Jo und ich großzügig hinweg.

Kurz vor dem wohlverdienten Feierabend verkündeten wir dem Team, dass ab morgen »so richtig programmiert werden soll!« Darüber freuten sich die Kollegen sehr. Endlich durften sie das tun, was ihnen am meisten Spaß bereitete. Da nahmen sie es gern in Kauf, dass wir uns morgens zunächst zum Daily Scrum treffen wollten.

Vielleicht haben Sie sich schon gefragt, warum der Product Owner nirgends aufgetaucht ist. Die Antwort ist so einfach wie unbefriedigend: Diese Rolle war nicht optimal besetzt. Das lag zum einen daran, dass wir mit der Doppelaufgabe (technische Plattform + erster fachlicher Geschäftsprozess) eigentlich zwei Product Owner benötigt hätten. Die Rolle des technischen Wegbereiters wurde traditionell vom Softwarearchitekten übernommen. Diesem fehlte aber das Wissen, um die Plattform auf die fachlichen Anforderungen auszurichten. Die Experten aus dem Fachbereich waren damit beschäftigt, den ersten Geschäftsprozess zu definieren. Dazu betrachteten sie den existierenden Prozess, standardisierten und optimierten diesen und bereiteten ihn so auf, dass er sich technisch unterstützen ließ. Die Unterstützung des Entwicklungsteams durch die Fachexperten war vorbildlich: Wir fanden immer einen Ansprechpartner, der unsere Fragen prompt beantwortete. Technische Entscheidungen wurden hingegen oft aufgeschoben, weil man auf die Aussage schwer erreichbarer Experten warten musste. Das lag zum Teil auch in der Kultur des Unternehmens begründet, Entscheidungen immer »nach oben« weiterzureichen. Rückblickend betrachtet, hätte die parallele Forschungsarbeit in der fachlichen und der technologischen Domäne besser im Vorfeld des Projekts stattfinden sollen. So wartete das Prozessteam immer wieder auf das IT-Team und umgekehrt.

Viele Entwickler müssen zu Beginn ihres ersten agilen Projekts davon entwöhnt werden, die generalistischste aller möglichen Lösungen entwickeln zu wollen. Immer wieder werden Sie Argumente der Art »Wenn wir diese Flexibilität jetzt nicht vorsehen, dann werden wir sie nie einbauen können!« hören. Ihre Gegenfrage könnte lauten: »Bist du dir sicher, dass wir diese Flexibilität wirklich brauchen?« Einer meiner Professoren sagte immer: »Verwendung kommt vor Wiederverwendung.« Wie wahr …

## 2.9 Daily Scrum

Erst um Viertel nach zehn waren alle Teammitglieder im Büro eingetrudelt, sodass ich endlich zum Daily Scrum rufen konnte. Auf die Frage, warum die vereinbarte Zeit von zehn Uhr nicht eingehalten wurde (was aus meiner Sicht schon reichlich spät war), bekam ich verschiedene unbefriedigende Antworten. Verpflichtungen in anderen Projekten, eine S-Bahn zu spät genommen oder schlicht den Termin vergessen. Ich konnte nicht mehr tun, als alle an die Bedeutung dieses Treffens zu erinnern. Dann wiederholte ich die Spielregeln und übergab das Wort an den ersten Kollegen. Der stand recht unschlüssig vor dem Taskboard, las noch einmal alle Backlog Items durch, die wir für diesen Sprint ausgesucht hatten, widmete sich dann den Aufgabenzetteln und verkündete schließlich: »Ich weiß nicht, welche Aufgabe ich mir nehmen soll!« Bevor Unruhe entstehen konnte, übernahm ich die Moderation, lenkte das Augenmerk aller auf das erste Backlog Item und empfahl dem Kollegen eine der Aufgaben, die ich für angemessen hielt. »Und was genau muss ich da machen?«, war die Reaktion. Die Antwort auf diese Frage kam von einem anderen Kollegen. Kurzerhand führte ich den Fragenden und den Wissenden zusammen – eleganter kann man Pair Programming in einem Team kaum einführen. Beide waren zufrieden damit, gemeinsam an dieser Aufgabe zu arbeiten. Das Beispiel machte Schule, und so gab es nach dem Daily Scrum zwei programmierende Paare und drei »Einzelkämpfer«. Nur der Projektleiter war unglücklich, weil er in dieser Runde keine Rolle gespielt hatte. Mit der Aufgabe, für das Team die Hindernisse aus dem Weg zu räumen, wollte er sich nicht abfinden. Ihm war vielmehr daran gelegen, das Projekt und das Team zu lenken und die Architektur mitzubestimmen. Gegen Letzteres wäre grundsätzlich nichts einzuwenden, sagte ich ihm – solange gewährleistet ist, dass das Team ungestört arbeiten kann, sprich: die Hindernisse zügig von ihm beseitigt werden. Dann aber hatte er noch einen wichtigen Einwand: Ihm fehlte eine Fortschrittskontrolle des Projekts. Zu diesem Zweck hatte er das Projekt bereits in der hausintern genutzten Zeiterfassungssoftware einrichten lassen. Anstatt dort einfach die Sprints oder die Backlog Items als Aufgaben anzulegen, hatte er versucht, fachlich oder technisch motivierte Aufgabenpakete zu schnüren. Die Folge: Niemand wusste genau, auf welches Aufgabenpaket er seine Stunden kontieren sollte. Das war aber nicht so dramatisch, weil (auch) in diesem Unternehmen die Zeiterfassung nach dem Motto »egal, worauf die Stunden gebucht werden – Hauptsache, die Summe stimmt« betrieben wurde. Um diesem Controlling um des Controllings

willen etwas Zielgerichtetes entgegenzusetzen, weihte ich den Projektleiter in die geheimnisvolle Welt der Burndown-Charts ein. Die Idee des tagesgenauen Projektstatus gefiel ihm gut. Natürlich musste es eine Excel-Tabelle mit automatisch generiertem Burndown-Diagramm sein, aber diese Vorliebe für ein elektronisches Werkzeug wollte ich ihm nicht ausreden. Aus Rücksicht darauf, dass der Projektleiter endlich eine in seinen Augen angemessene Aufgabe gefunden hatte, erwähnte ich nicht, dass die Burndown-Charts vom Team gepflegt werden können. Das Team war froh, dass jemand anderes diese vermeintlich langweilige Aufgabe übernahm, und konzentrierte sich auf die Umsetzung der Tasks.

In den folgenden Daily Scrums gingen die meisten Teammitglieder zunehmend couragierter mit der Situation um, vor versammelter Mannschaft die Verantwortung für eine Aufgabe zu übernehmen. Zwei Kollegen machte dies jedoch immer noch schwer zu schaffen, obwohl niemand sie drängte oder lächerlich machte. Meist suchten sich diese ruhigen Vertreter einen Partner, den sie bei dessen Aufgabe unterstützten. Gab es keine passende Paaraufgabe, so ließen sie sich auch auf die Übernahme einer eigenen Aufgabe ein. Während einer der beiden diese Aufgaben souverän löste, benötigte der andere Kollege viel fachliche Unterstützung. Nachdem das deutlich geworden war, konnten wir alle viel besser und vor allem zielgerichteter mit der wiederkehrenden Unsicherheit beim Daily Scrum umgehen.

Pair Programming ist gut, wenn die Paare ausgewogen sind. Entweder sind beide Partner in etwa gleich gut qualifiziert – dann können sie viel voneinander lernen. Oder ein Partner übernimmt die Rolle des Coachs für den zweiten Partner, der sich beispielsweise in eine neue Technologie einarbeitet. In beiden Fällen ist es wichtig, dass regelmäßig die aktive (an der Tastatur) und die passive Rolle getauscht werden. Softwareentwicklung beinhaltet bekanntlich einen hohen Anteil an handwerklichen Tätigkeiten, die beim Pair Programming geübt werden.

Pünktlichkeit ist ein agiles Prinzip, das oft zugunsten der Forderung nach einem kreativen Umfeld fallen gelassen wird. Ein Fehler, denn Scrum ist ein disziplinierter Prozess. Die Behauptung, »agil« sei mit »chaotisch« gleichzusetzen, ist falsch – wird aber gerne als Begründung für Unpünktlichkeit angeführt. Es ist respektlos (und teuer obendrein), seine Kolleginnen und Kollegen warten zu lassen. Respekt ist ein wichtiger agiler Wert – wichtiger als Kreativität. Außerdem hilft das strikte Einhalten festgelegter Zeiträume (Timeboxes) bei der Fokussierung auf die anstehenden Aufgaben und erleichtert die Planung der Sprints.

Beim Daily Scrum lernt man die Stärken und Schwächen der einzelnen Teammitglieder sehr gut kennen. Das hilft dem ScrumMaster beim individuellen Fördern und Fordern. Erkennt er bestimmte problematische Verhaltensweisen (beispielsweise die oben beschriebene Neigung, sich lieber einen Programmierpartner als eine eigene Aufgabe zu suchen), so muss sich der ScrumMaster zunächst fragen, was die Ursachen für dieses Verhalten sein können. Erst mit dem Wissen um die Ursachen kann er das Teammitglied besser integrieren und die ungewünschten Verhaltensweisen abstellen.

Mit Burndown-Charts überzeugt man auch klassische Projektleiter und Controller und macht deutlich, dass das verbreitete Bild des chaotischen, unkontrollierten agilen Projekts schlichtweg falsch ist. Das Gegenteil ist der Fall: Agile Projekte bieten ein tagesgenaues Controlling des Projektfortschritts und sind deshalb jederzeit in der Lage, Fehlentwicklungen zu erkennen und Gegenmaßnahmen einzuleiten. Dieser Vorteil gegenüber klassischen Projekten, die mit ihren Projektplänen der Wirklichkeit hinterherrennen, überzeugt viele Skeptiker.

## 2.10 »Sprint« ist eine relative Geschwindigkeit

Die Softwareentwicklung hielt in diesem Projekt große und spannende Herausforderungen für uns bereit. Neben der neuen Architektur musste auch die Entwicklungsumgebung weiträumig neu konzipiert und aufgesetzt werden. Das lag vor allem daran, dass die Entwickler in anderen Projekten mit der bestehenden Standardentwicklungsumgebung im Laufe der Zeit ihre Erfahrungen gemacht haben, die sie nun in diesem neuen Projekt umsetzen wollten. Grundsätzlich eine gute Idee, die dem agilen »Inspect and Adapt«-Gedanken entspricht. Allerdings ist es nicht sinnvoll, die Kosten (finanziell wie zeitlich) dem neuen Projekt aufzubürden – es sei denn, dies ist explizit eingeplant (was in unserem Projekt nicht der Fall war). So trugen wir von Anfang an die Schulden aus anderen Projekten mit uns herum. Es dauerte lange, bis wir eine komplette Entwicklungsumgebung mit Versionskontrolle, Build-Werkzeug und Continuous Integration auf die Beine gestellt hatten, mit der wir gut und zügig arbeiten konnten. Deshalb fühlte sich der erste Sprint aus Sicht der Erreichung des fachlichen Sprint-Ziels eher wie ein Kriechgang an.

Da sich die Build-Infrastruktur und die Projektstruktur gegenseitig beeinflussten, wurden wir in kürzester Zeit zu Refactoring-Experten. Das klappte sehr gut und war für uns alle ein hilfreicher Drill, der sich später beim Refaktorisieren des Anwendungscodes auszahlte.

Eine testgetriebene Entwicklung konnten wir nicht etablieren. Immerhin gelang es Jo und mir, die Absicherung der Backlog Items durch automatisierte Tests in die Definition of Done aufnehmen zu lassen. Wir hofften, über den regelmäßigen Umgang mit Tests bei den Entwicklern so viel Leidenschaft zu entfachen, dass der Weg zur testgetriebenen Entwicklung eine logische Konsequenz und keine von außen vorgegebene Regel war. Das funktionierte leider nicht. Da die Continuous-Integration-Umgebung nur halbherzig genutzt wurde, gab es zudem keinen Gruppendruck, der einen Entwickler dazu veranlasst hätte, seinen Code besser abzusichern und dafür zu sorgen, dass der Build nicht bricht. Moment mal – habe ich gerade »seinen Code« geschrieben? Müsste der Code nicht allen gehören, und jeder im Team übernimmt Verantwortung für alle Teile des Systems? Grundsätzlich ja. In der Praxis sah das aber anders aus. Die Teammitglieder waren es gewohnt, in ihrem Spezialgebiet zu arbeiten. Dank des Pair Programming wurden diese Wissensinseln zwar ein wenig aufgelöst, aber es gab

noch viele Gebiete, zu denen das Wissen in genau einem Kopf gespeichert war. In Anbetracht der großen Bandbreite an technischen Themen, mit denen wir uns auseinandersetzen durften, war die Idee des generalistischen Spezialisten, der alle Bereiche des Softwaresystems mindestens so gut kennt, dass er im Fehlerfall korrigierend eingreifen kann, kaum in die Tat umzusetzen. Deshalb beschränkten wir uns darauf, für jedes Thema mindestens zwei »Wissende« auszubilden.

Eine strenge Definition of Done stellt hohe Anforderungen an die Softwareentwicklungsumgebung – es sei denn, man möchte Builds und Tests immer wieder manuell ausführen. Ist eine geeignete Infrastruktur nicht vorhanden, dann sollte sich das Team gut überlegen, ob es in der Lage ist, eine solche Umgebung in einem Sprint parallel zur Bearbeitung der Backlog Items aufzusetzen. Falls nicht, dann bietet es sich an, solche Rüstarbeiten in einem vorgelagerten Sprint zu erledigen.

In den meisten frisch agilisierten Projekten, die ich kennenlernen durfte, waren die handwerklichen Fähigkeiten vieler Softwareentwickler ungenügend ausgeprägt. Das ist nicht allein den Entwicklern anzulasten, denn diese können sich nur im Rahmen der ihnen gebotenen Möglichkeiten weiterbilden. Wenn Zeit und Budget für den Erwerb solcher Fähigkeiten fehlen, dann kann keine Softwareentwicklungskultur entstehen. Dazu bedarf es sowohl äußerer Impulse (Fachliteratur, Trainings und Konferenzbesuche) als auch der Möglichkeit, sich im Unternehmen zeitliche Freiräume zu schaffen, um das Erlernte anzuwenden und zu verinnerlichen. In meinen Trainings zur testgetriebenen Entwicklung verwende ich Code-Katas, um die erforderlichen Handgriffe einzuüben und zur Gewohnheit werden zu lassen. Viele Teilnehmer sind zunächst genervt von den wiederkehrenden Übungen – bis sie verstehen, dass in der disziplinierten Wiederholung der Schlüssel zur Übernahme in die tägliche Routine liegt. Eine tägliche Code-Kata ist keine vergeudete Zeit, sondern kommt der Qualität der entwickelten Software zugute.

Dieses Handwerkszeug des Softwareentwicklers und die vielen Schritte zu dessen Beherrschung werden in der Literatur oft mit dem Begriff »Software Craftsmanship« beschrieben. Ohne Software Craftsmanship nützt der beste agile Prozess nichts – am Ende wird doch nur maximal mittelprächtige Software herauskommen, wenn die Entwickler ihr Handwerk nicht beherrschen.

## 2.11 Dimensional Planning

Einige Anforderungen im Product Backlog waren zu groß, um sie innerhalb eines Sprints realisieren zu können. Sie ließen sich aber auch nicht so einfach in kleinere Items zerlegen. So wollten wir beispielsweise eigene GUI-Elemente entwickeln, die speziell auf einen Anwendungsfall unserer Fachanwendung zugeschnitten waren. Dazu benötigten wir eine Reihe von grafischen Basiselementen. Erste Implementierungen der Fachanwendung mussten ohne die Spezialbedienelemente auskommen. Wie aber sollten wir das im Product Backlog beschreiben? Ließ sich die Forderung, ein Backlog Item innerhalb eines Sprints realisieren zu können, nicht auch anders lösen als durch eine Zerlegung des Items?

Anstatt ganze Anforderungen von der Projektbildfläche verschwinden zu lassen, haben Koen Van Exem und Walter Hesius eine neue Steuergröße ins Spiel

gebracht: die *Tiefe* eines Backlog Item. Sie definieren für diese Dimension vier Ausbaustufen. Als Metapher verwenden sie den Straßenbau. Um von A nach B zu gelangen, kann man eine Straße bauen. Man erreicht das Ziel unabhängig davon, wie diese Straße ausgestaltet ist (Feldweg oder Autobahn) – allerdings unterschiedlich schnell und komfortabel.

Wir verpassten unseren fachlichen Backlog Items eine Tiefenangabe. Der Feldweg war eine GUI, in der nur Standardbedienelemente verwendet wurden. Die Benutzeroberfläche war zum Teil etwas kompliziert zu bedienen, lieferte aber die geforderte Funktionalität. Mit dem Austausch der Bedienelemente durch unsere Speziallösungen wurde der Feldweg zur asphaltierten Straße ausgebaut. Die Funktionalität war identisch, aber der Bedienkomfort deutlich gestiegen. Weitere Komfortfunktionen hätten dann die Autobahnlösung ergeben.

Dimensional Planning hilft bei der iterativen Ausgestaltung der fachlichen Anforderungen. Durch die Hinzunahme der Tiefe als neue Planungsdimension lässt sich eine fachliche User Story auf mehrere Sprints aufteilen. Trotzdem liefert jeder Sprint eine in sich geschlossene, allerdings mit unterschiedlichem Komfort ausgestattete Lösung.

## 2.12 Review

Gegen Ende des ersten Sprints blickten wir wie jeden Morgen an unser Taskboard. Dort war für jeden sichtbar abzulesen, dass wir viel erreicht hatten – wenngleich nicht alles, was wir uns ursprünglich vorgenommen hatten. Das ist aber durchaus normal in neuen Scrum-Teams, denn die Einschätzung dessen, was innerhalb eines Sprints geleistet werden kann, erfordert viel Erfahrung und variiert von Team zu Team. Jetzt, da die Leistung so sichtbar und nachvollziehbar war, stellte sich im gesamten Team erstmals ein Gefühl von Zufriedenheit und Stolz ein. Das Erreichte wurde von allen als Teamleistung wahrgenommen. »Und genau deshalb sollen die Sprint-Ergebnisse vom gesamten Team vorgestellt werden!«, verkündete ich an einem der letzten Daily Scrums dieses Sprints. Wir reservierten Zeit für die Vorbereitung des Sprint-Reviews. Da wir echte Ergebnisse in Form von lauffähiger Software realisiert hatten, sollten diese live gezeigt werden – wie sonst hätten wir demonstrieren können, dass wir uns an unsere strenge Definition of Done gehalten haben?

Das Ergebnisartefakt eines unserer Backlog Items war ein Konzept anstelle von lauffähigem Code. Wie sollte man das live vorführen? Ich schlug vor, die Tests zu präsentieren, die der Entwickler durchgeführt hatte, um verschiedene Alternativen zu bewerten. Damit sollte den Teilnehmern des Sprint-Reviews sofort klar werden, welche Variante die geeignetste ist. Und genau so funktionierte es dann auch.

Beim Sprint-Review übernahmen die meisten, aber nicht alle Teammitglieder eine aktive Rolle. Es war uns wichtiger, eine schlüssige und flüssige Livepräsentation zu bieten, als zwingend jedes Teammitglied ins Rampenlicht zu stellen. Am

meisten überraschte mich ein Entwickler. Er gehörte nicht zu den Vortragenden, weil ihm die Bühnenpräsenz nicht sonderlich lag. Bei der Diskussion mit dem Auftraggeber lieferte er jedoch die entscheidenden Argumente, die zur Akzeptanz eines Backlog Item führten. Das freute uns alle, Auftraggeber inklusive. Der sah, dass das gesamte Team zum Ergebnis beigetragen hatte. Und er war angenehm überrascht, dass wir ihm zu einem bei Sprint-Beginn fest zugesagten Termin tatsächlich etwas präsentieren konnten – ohne Verschiebung, ohne Ausreden, ohne Wenn und Aber. Eine Frage aber konnten wir ihm nicht beantworten: Ob wir es schaffen würden, den ambitionierten Fertigstellungstermin für das Gesamtsystem zu halten. Wir verwiesen auf die Dynamik des Product Backlog, den ständigen Erkenntnisgewinn und die daraus resultierende Anpassung unserer Größenschätzungen für die Backlog Items. »Aber haben Sie nicht zu Beginn des Projekts gesagt, dass der Termin dank der agilen Vorgehensweise gehalten werden kann?« Nein, das hatten wir nicht. Aber offensichtlich hatten wir es versäumt, die Erwartungen der verschiedenen Interessengruppen – allen voran die des Auftraggebers – im Blick zu behalten und bei Bedarf zurechtzurücken.

## 2.13 Pflegeanleitung für Erwartungen

Verlassen wir kurz unser Projekt für ein paar grundlegende Anmerkungen zum Management von Erwartungen.

Menschen, die zum ersten Mal mit agilen Vorgehensweisen in Berührung kommen, verknüpfen damit meiner Erfahrung nach entweder gar keine oder sehr hohe Erwartungen. Beide Extreme verlangen nach einem adäquaten Umgang. Im Folgenden beziehe ich mich konkret auf die Erwartungen, denen ich in dem hier beschriebenen Projekt begegnet bin. Sie werden diesen in anderen Projekten in ähnlicher Form begegnen.

Beginnen wir mit den bereits angedeuteten Erwartungen des Auftraggebers. »Scrum macht alles schneller«, lautet eine immer wieder gehegte Hoffnung. Das mag zutreffen, wenn sich das agile Team aufeinander eingespielt hat. Zu Beginn eines agilen Projekts ist dies jedoch nur selten der Fall. Verantwortlich dafür ist unter anderem die Definition of Done, die sehr hohe Anforderungen an den Fertigstellungsgrad eines Backlog Item stellt. Je höher die Anforderungen, desto größer ist der Aufwand. Dafür ist ein Backlog Item aber anschließend tatsächlich fertig und nicht nur »zu 90 Prozent«, wie man es aus vielen anderen Projekten kennt. Der größte Vorteil agiler Methoden ist nicht die Geschwindigkeit, sondern die große Transparenz des Projektfortschritts. Tagesgenau lässt sich feststellen, wie gut das Team auf dem Weg zum Sprint-Ziel unterwegs ist. Probleme längs des Weges werden frühzeitig erkannt, benannt und – soweit möglich – beseitigt. Scrum-Projekte haben nicht weniger Probleme als andere Projekte, sie gehen nur offen und ehrlich damit um. Damit sind sie aus Sicht des Auftraggebers und der anderen Stakeholder viel besser zu kontrollieren, weil Entscheidungen über das

Projekt auf der Basis harter Fakten anstelle von Vermutungen und frisierten Halbwahrheiten getroffen werden können.

Eine andere, von Auftraggebern und Projektleitern viel gehegte Hoffnung ist, dass sich Scrum in das traditionelle Rollenmodell eines hierarchisch organisierten Unternehmens einfügt. Tatsächlich hat Scrum ein eigenes Rollenmodell, das auf eine Projektorganisation zugeschnitten ist. Scrum lässt sich zwar in eine Linienorganisation integrieren, stellt aber hohe Anforderungen an die Linienvorgesetzten, die ihren Mitarbeitern ein Umfeld schaffen sollen, das Eigenverantwortung und Kreativität fördert. Damit tun sich viele Linienvorgesetzte, aber auch deren Mitarbeiter[2] schwer. Erstere fühlen sich überflüssig (»Mein Mitarbeiter entscheidet ja ohnehin alles alleine«) und ihrer Macht beraubt, Letztere sind oft überfordert (»Warum sagt mir keiner mehr, wo es langgeht?«).

Viele Linienvorgesetzte sind es gewohnt, bei der Aufgabenverteilung an ihre Mitarbeiter in Projekten ein Mitsprache-, wenn nicht gar ein Vetorecht zu besitzen. Dieses Recht geht in agilen Projekten auf das Team über. Das organisiert sich so, dass die anstehenden Backlog Items zügig und qualitativ hochwertig erledigt werden und sich dabei keine Wissensinseln bilden.

Ein weiteres Problem entsteht, wenn Linienvorgesetzte die Disposition ihrer Mitarbeiter übernehmen, ohne dabei Rücksicht auf die Bedürfnisse einzelner Projekte zu nehmen. Insbesondere agile Projekte gehen von einem stabilen Team aus, das möglichst 100 Prozent seiner Arbeitszeit in diesem einen Projekt verbringt (temporär hinzugezogene Experten einmal ausgenommen). Selbst wenn das nicht immer funktioniert, so muss die Zusammensetzung und die Verfügbarkeit eines Teams während eines Sprints stabil sein. Leider habe ich es mehr als einmal erlebt, dass ein Linienvorgesetzter einen seiner Mitarbeiter für Sonderaufgaben spontan aus dem Projekt abgezogen hat. Das kann das Erreichen des Sprint-Ziels gefährden – es sei denn, die anderen Teammitglieder gleichen die Fehlzeit durch Überstunden aus, was aber auch nur bedingt zu empfehlen ist. Natürlich kann eine ähnliche Situation auch dadurch entstehen, dass ein Teammitglied erkrankt. Das ist dann aber eine nicht planbare Situation, wohingegen die übertragene Sonderaufgabe in den meisten Fällen planbar oder verschiebbar ist. Im Wesentlichen geht es darum, die Linienvorgesetzten positiv davon zu überzeugen, dass ein Sprint geschützt ist. Das geht am besten, indem man sie einlädt, die Mechanismen eines agilen Projekts am praktischen Beispiel zu erleben und kennenzulernen. Neben den Vorteilen agiler Vorgehensweisen müssen auch die notwendigen Voraussetzungen und Spielregeln für ein agiles Projekt erläutert werden. Dazu gehört

2. Mit »Mitarbeiter« meine ich jene Menschen, die einem Linienvorgesetzten zugeordnet sind. Der Begriff »Team« ist zu nah am Projektgeschehen angesiedelt und deshalb irreführend; alle organisatorischen Begriffe wie »Abteilung«, »Gruppe« etc. sind zu konkret; in einem Unternehmen, das ich einmal beraten durfte, sprachen die Vorgesetzten immer von ihren »Menschen«, aber das fühlte sich für mich komisch an.

auch der geschützte Sprint. Aus diesen Voraussetzungen kann man dann Erwartungen an die Linienvorgesetzten ableiten und gemeinsam vereinbaren.

Die Konkurrenz von Linie und Projekt müssen in der Regel jene ausbaden, die zwischen diesen beiden Stühlen sitzen: die Mitarbeiter. Es ist gar nicht so einfach, zeitgleich den Ansprüchen von Linienvorgesetzten und Projekt zu genügen. Die Teammitglieder erwarten in diesen Fällen eine Unterstützung in erster Linie vom ScrumMaster. Diese Erwartung muss der ScrumMaster erfüllen, denn das ist seine wichtigste Aufgabe. Er ist es auch, der das Team beim Einhalten und Weiterentwickeln der agilen Werte, Prinzipien und Praktiken unterstützt, indem er diese vorlebt sowie verständlich und der Projektsituation angemessen vermittelt und einfordert. Auch von den anderen Teammitgliedern kann jeder im Team Unterstützung erwarten. Dazu gehört auch der Product Owner, der beispielsweise auf der Führungsebene das Verständnis für die Bedürfnisse agiler Teams wecken kann.

Einige Teammitglieder wünschen sich eine klare Rollenverteilung im Team. Wie so oft im Leben gibt es für diese Forderung keine allgemeingültige Lösung. Wie bereits weiter oben diskutiert, muss aber nicht zwingend jeder alles wissen und können – solange keine Wissensinseln im Team entstehen.

Einfacher lässt sich dem Wunsch der Teammitglieder nach mehr Zeit für das Tagesgeschäft und weitere Projekte nachkommen. Das Tagesgeschäft muss ohnehin mit einer geeigneten Zeitpauschale (zuzüglich der Zeit für im Voraus bekannte Abwesenheiten) in der Sprint-Planung berücksichtigt werden. Ist das Teammitglied zeitgleich in mehreren Projekten beschäftigt, so darf man ihm die Frage stellen, ob es den Erwartungen aller Projekte gleichzeitig gerecht werden kann. Schließlich trägt jedes Teammitglied in allen seinen Projekten einen Teil der Ergebnisverantwortung. Oft ist es besser, die Teams neu zusammenzustellen, sodass weniger Personen auf mehrere Projekte aufgeteilt werden.

Um die Angst der Linienvorgesetzten vor dem gefühlten Machtverlust zu mildern, sollte man sie auf ihre neue Aufgabe gut vorbereiten. Sie sollen ihre Mitarbeiter zum eigenverantwortlichen Handeln anleiten und motivieren. Aus klassischen Vorgesetzten werden somit Gesprächspartner für die Vorbereitung von Entscheidungen (die dann der Mitarbeiter selber trifft), fachliche Berater, Coachs, Mentoren und Sparringspartner für Ideen. Diese Aufgaben sind so anspruchsvoll, dass von einem Machtverlust nicht mehr die Rede sein kann, nein, darf.

Die konsequente Einführung einer agilen Denkweise hat in den meisten Unternehmen fundamentale Auswirkungen: neue Rollen in der Projektorganisation; eine generalistischere Rollenverteilung zur gezielten Vermeidung von Wissensinseln; mehr Verantwortung im Team und entsprechende Auswirkungen auf die Führungsrolle der Linienvorgesetzten; die Abkehr von der Illusion, dass Scrum-Projekte per se schneller und erfolgreicher sind, verbunden mit der Erkenntnis, dass das Mehr an Transparenz und das tagesgenaue Controlling in

agilen Projekten die ideale (und notwendige) Voraussetzung für den »Inspect and Adapt«-Ansatz sind, auch bekannt als Deming-Zyklus (Plan – Do – Check – Act).

Scrum und andere agile Modelle verändern eine Organisation. Darauf müssen sich alle einlassen: vom Manager bis zum Teammitglied, eventuell sogar Kunden, Lieferanten und Partner. Wer diese Veränderungen nicht nur akzeptiert, sondern als Chance begreift, der wird von diesem organisatorischen Wandel profitieren und mit der neuen, agilen Organisation besser für die durch Wandel bestimmte Gegenwart und Zukunft gerüstet sein.

So weit die Anmerkungen zu diesem nicht allein im agilen Umfeld wichtigen Thema. Weiter geht's mit der Geschichte – und zwar mit einem Blick zurück.

## 2.14 Retrospektive

Auf die erste Retrospektive waren Jo und ich sehr gespannt. Es ist immer aufschlussreich, wenn man Scrum-Novizen nach ihren ersten agilen Gehversuchen um eine Bewertung bittet.

Wir begannen die Retrospektive mit Norman Kerths »Prime Directive«:

> Regardless of what we discover, we understand and truly believe that everyone did the best job they could, given what they knew at the time, their skills and abilities, the resources available, and the situation at hand.

Anschließend zeichnete ich einen Zeitstrahl, der den vergangenen Sprint symbolisierte. Dort sollten alle Teammitglieder die Ereignisse auftragen, die sie für beachtenswert hielten. Das konnten fachliche, aber auch persönliche Dinge sein. Nach fünf Minuten (Timebox!) konzentrierter Stille klebten wir reihum unsere Zettel an den Zeitstrahl. Es dominierten jene Ereignisse, die für die Kollegen außergewöhnlich gewesen waren, darunter der Umzug in das neue Teambüro, verschiedene Workshops und das gut verlaufene Sprint-Review. Einige Teammitglieder gaben offen zu, dass sie sich an kein bemerkenswertes Ereignis erinnern konnten – eine Feststellung, die wir unbewertet im Raum stehen ließen. Nachdem alle ihre Ereignisse untergebracht hatten, gab der Zeitstrahl ein recht genaues und in einigen Facetten überraschendes Bild des vergangenen Sprints wieder. Ich hatte dieser Methode bisher skeptisch gegenüber gestanden, musste aber feststellen, dass sie zumindest in diesem Team gut funktionierte.

Auf die Frage »Was hat im vergangenen Sprint gut funktioniert?«, gab es von einigen Teammitgliedern viel Feedback, andere hielten sich zurück. Gelobt wurde die neue, deutlich teamorientiertere Arbeitsweise. Die intensive Nutzung des Wikis war fast allen Teammitgliedern eine Erwähnung wert – wohl auch, weil es die Transparenz des Projekts nach außen verbesserte. Aber auch Errungenschaften hinsichtlich der Softwarearchitektur und der Technologieauswahl zählten zu

den positiv bewerteten Elementen. Selbst der Teamraum, der für alle einen (zumindest temporären) Abschied vom angestammten Arbeitsplatz bedeutete, wurde von einigen Teammitgliedern positiv hervorgehoben. Sie empfanden den Weg zwischen den Büros mittlerweile als hilfreich, um gedanklich zwischen Tagesgeschäft und Projekt umzuschalten. Auch das Sprint-Review fand hier seinen Platz, weil es das Teamgefühl so gut betont hatte und weil der Auftraggeber voll des Lobes ob des erreichten Sprint-Ziels gewesen war.

Nach einer kurzen Pause wagten wir uns an die Beantwortung der Frage, was verbessert werden kann. Diese Liste war deutlich länger als die erste. Die meisten Verbesserungsvorschläge bezogen sich auf Scrum. Zwei Teammitglieder waren der Meinung, dass wir Scrum zu sklavisch lebten. Tatsächlich hatten Jo und ich versucht, mit einer Lehrbuch-Version von Scrum zu starten, um den Teammitgliedern einen Orientierungspunkt zu bieten. Da aber niemand aus dem Team ein Scrum-Buch gelesen hatte, war das offensichtlich in die Hose gegangen. Allein für diesen Kritikpunkt hatte sich die Retrospektive schon bezahlt gemacht. Aber es gab noch viel mehr an unserer Scrum-Implementierung zu verbessern. So saß der Projektleiter und designierte ScrumMaster nicht im selben Raum wie das Team, was vom Team als störend empfunden wurde. Und wenngleich das Daily Scrum im Laufe des Sprints immer kürzer geworden war, so dauerte es insgesamt immer noch zu lange. Interessant war der Einwand, dass im Product Backlog hauptsächlich fachliche Anforderungen berücksichtigt waren, technische Anforderungen und Ziele aber in den Hintergrund gestellt wurden bzw. gar nicht im Backlog auftauchten. Unser Hinweis, dass das Product Backlog, wie der Name schon sagt, der Sammlung aller Produkteigenschaften dient (die nun einmal fachlicher Natur sind), brachte zwar Klarheit, löste aber keine Begeisterung aus. Wir versuchten, für jeden genannten Kritikpunkt einen Verantwortlichen und eine Maßnahme zu definieren. Einer unzureichenden Qualitätssicherung der Task-Ergebnisse wollten wir beispielsweise im kommenden Sprint mit einer »Review«-Spalte am Taskboard und Checklisten für wiederkehrende Aufgaben begegnen. Bei vielen Punkten wollte es uns aber nicht gelingen, einen Verbesserungsvorschlag zu benennen. Konstruktive Kritik will halt gelernt sein.

Die mit Maßnahmen versehenen Verbesserungsvorschläge setzten wir zügig um, was vom Team wohlwollend zur Kenntnis genommen wurde. Andere Entdeckungen wurden nicht mit einer Maßnahme versehen, weil diese nicht umsetzbar war. Beispielsweise wollte (oder konnte) der Projektleiter nicht in den Teamraum umziehen. Mit den ungelösten Problemen konnte das Team erstaunlicherweise gut leben. Deshalb entstand auch nie die Situation, dass wir uns zu viele Maßnahmen für den kommenden Sprint vornahmen und diese nicht komplett umsetzen konnten – eine Entdeckung, die ich in anderen Scrum-Teams wiederholt machen durfte. Auf der Kehrseite dieser Medaille waren die zwar aufgedeckten, aber dauerhaft ungelösten Probleme zu finden.

Die Retrospektive ist Pflicht. Sie ist der Schlüssel zur Prozessverbesserung, gibt die Teamstimmung wieder, deckt Motivationsprobleme auf. Wer auf die Retrospektive verzichtet, der macht kein Scrum, denn er verzichtet auf die Möglichkeit, sich kontinuierlich zu verbessern und sein Projektvorgehen an veränderte Bedingungen anzupassen. Punkt.

## 2.15 Meta-Retrospektive (klassisch: Fazit)

Ein agiles Projekt ist kein Ponyhof. Es gehört schon einiges dazu, um agil erfolgreich zu sein. Agile Projekte sind nicht einfacher oder weniger fordernd als klassische Projekte. Auch lösen sich Probleme in agilen Projekten nicht einfach in Luft auf, wie mancherorts immer noch behauptet wird. Es ist vielmehr so, dass in agilen Projekten die Probleme früher erkannt werden. Dadurch bekommt das Projektteam die Möglichkeit, gemeinsam das Problem aus dem Weg zu räumen (oder vom ScrumMaster aus dem Weg räumen zu lassen), bevor teure Spätfolgen entstehen. Dazu bedarf es eines ausgeprägten Problembewusstseins, das auf einer einfachen Erkenntnis fußt, die da lautet: Es nützt nichts, wenn ich jemanden für ein Problem verantwortlich mache. Das macht das Problem nicht kleiner, sondern bindet nur unnötig Energie – die man lieber auf die Problembeseitigung verwenden sollte.

Ein wichtiger Grund für die Einführung eines agilen Vorgehens war die Minimierung des Projektrisikos. Wer technologisches Neuland betritt, der ist gut beraten, seine (Forschungs-)Arbeit über schnelle Feedbackzyklen abzusichern. Dieses Argument ist sehr managementtauglich, denn der Begriff »Risikominimierung« spielt im klassischen Management-Vokabular eine große Rolle.

Agil zu werden ist schwer genug – agil zu bleiben, aber noch ungleich schwerer. Das Anforderungsprofil für Mitglieder agiler Teams umfasst Mut, Disziplin, ganzheitliches Denken und eine Kombination aus Soft Skills und handwerklichem Können. Es ist gar nicht so einfach, solche Mitarbeiter zu finden.

Um Scrum erfolgreich in einem neuen Team einführen zu können, das vorher noch nicht agil gearbeitet hat, ist es enorm wichtig, dass alle Projektbeteiligten eine solide und zielgruppengerechte agile Grundausbildung bekommen. Sonst verbringt man während des Sprints zu viel Zeit mit dem Erklären der agilen Werte, Prinzipien und Praktiken. Wird nur der ScrumMaster geschult, dann fehlt dem Team (und meist auch dem Product Owner) das methodische Rüstzeug.

Zu Beginn eines Projekts muss sich ein Team erst einmal finden. Menschen mit unterschiedlichem Wissen, unterschiedlichen Interessenschwerpunkten und teilweise sogar unterschiedlichem kulturellem Hintergrund sollen plötzlich gemeinsam ein Stück Software entwickeln und dafür auch noch verantwortlich zeichnen – eine Last, die ihnen früher der Projektleiter abgenommen hatte. Kein Wunder, dass sich viele Teammitglieder zunächst zurückhalten und die Lage sondieren. Agile Praktiken wie das Planning Poker und die Betonung des kommunikativen Aspekts in der Sprint-Planung bedeuten für viele Menschen eine Belas-

tung, weil der Grad an Transparenz enorm hoch ist – und damit auch die Gefahr, sich vor versammelter Mannschaft eine Blöße zu geben. Es ist eine Gratwanderung, die Menschen zur Mitarbeit zu motivieren und sanft zu drängen, ohne sie jedoch zu überfordern, damit sie sich nicht abschotten oder verweigern. Gerade das Planning Poker hilft mit seiner spielerischen Art, Ängste abzubauen und den Teamcharakter zu unterstreichen. Die Tatsache, dass der Einzelne nun nicht mehr für »seine« Aufwandsschätzungen allein verantwortlich ist, hat bisher noch allen Teams gefallen. Und wem Planning Poker zu lange dauert, für den gibt es schnellere Alternativen, z.B. Magic Estimation.

Das Commitment des Teams, d.h. die Verpflichtung auf das Erreichen des Sprint-Ziels, ist schwierig, aber wichtig. Es erinnert das Team an seine gemeinsame Verantwortung und mahnt wohlüberlegte Entscheidungen an.

Ein gutes Erwartungsmanagement ist wichtig – gerade weil viele Menschen immer noch falsche Vorstellungen von den Prinzipien, Möglichkeiten und Grenzen agiler Vorgehensweisen haben. Nur wenn die Erwartungen bekannt sind, kann man ihnen begegnen und sie mit Anforderungen verknüpfen. Auf diese Weise holt man das Management und die Linienvorgesetzten mit ins agile Boot. Gelingt das nicht, dann bleibt das Scrum-Projekt vermutlich ein U-Boot. Das ist okay, aber dem Ziel einer agilen Organisation bringt es Sie nicht näher.

Das Projekt läuft übrigens heute (Juni 2011) immer noch. Fachlich und technologisch ist es ein Erfolg. Auch Scrum ist noch im Einsatz, wenngleich das Team nicht vollends von den Vorteilen einer agilen Vorgehensweise überzeugt werden konnte. Manchmal dauert es eben ein wenig länger, bis der Funke überspringt.

## 2.16 Agile Werte im Projekt

| Individuen und Interaktionen | vor | Prozessen und Tools |
|---|---|---|

▲

Kommunikation im Team hatte auch schon in der Prä-Scrum-Ära einen hohen Stellenwert. Auf die individuellen Bedürfnisse der Teammitglieder wurde viel Rücksicht genommen.

Der Kunde hat keinerlei Prozessvorgaben gemacht. Es gab einen Satz von Tools, die in anderen Projekten verwendet wurden. Das Projektteam hatte aber die Freiheit, eigene Tools zu verwenden.

| Laufende Software | vor | Ausführlicher Dokumentation |
|---|---|---|

▲

Die Anforderungen sind im Product Backlog beschrieben. Die Dokumentation besteht aus Codekommentaren und einem wachsenden Wiki, das aber im Wesentlichen die Konzepte und projektweiten Konventionen enthält. Allerdings war einigen Entwicklern das Konzept wichtiger als die lauffähige Software. Sie wollten die Dinge zunächst theoretisch durchdringen, bevor sie sich an die Implementierung machten. Unsere Idee einer konsequent testgetriebenen Entwicklung konnte sich nicht durchsetzen.

| Zusammenarbeit mit dem Kunden | vor | Vertragsverhandlungen |
|---|---|---|

▲ (1)

▲ (2)

(1) Unser Ziel war (und ist) es, gemeinsam mit dem Kunden eine tragfähige Plattform zu schaffen. Unser Engagement wurde mehrfach verlängert, ohne die Tätigkeit auf eine bestimmte Rolle (z. B. »Architekt«) zu beschränken. Der Einfluss des Projektbudgets auf die Zusammenarbeit war trotzdem gegeben, was in einer Kunde-Dienstleister-Beziehung aber völlig normal ist.

(2) Die Zusammenarbeit mit dem eigentlichen Kunden, sprich: der Fachabteilung, klappte wunderbar.

| Reagieren auf Veränderungen | vor | Befolgen eines Plans |
|---|---|---|

▲

Der Prozess wurde mehrfach adaptiert, und auch das Product Backlog veränderte sich im Laufe der Zeit deutlich, bedingt durch den rasanten Erkenntnisgewinn in diesem Projekt.

Henning Wolf, Agile Projekte mit Scrum, XP und Kanban , dpunkt.verlag, ISBN 978-3-86490-266-6

# 3 Scrum-Einführung bei einem Internet Service Provider – Leben und Werk eines ScrumMasters

Bernd Schiffer

*Eine Menge Erfahrung sammelt ein Unternehmen bei der Veränderung seiner Softwareentwicklungsprozesse. Diese hier stammen von der Scrum-Einführung bei einem Internet Service Provider. Bei Betrachtung dieser Erfahrung sticht eine Erkenntnis heraus: Pragmatismus geht über Dogmatismus.*

## 3.1 Das weitere Umfeld

Der Internet Service Provider (kurz: ISP) ist ein international operierendes Unternehmen. Es ist schon seit Jahrzehnten erfolgreich am Markt tätig. Zu den Hauptaufgaben gehört u.a. das Domain Hosting. Der ISP verwaltet mehrere Millionen Kunden und registrierte Domains weltweit. Mehrere 1000 Mitarbeiter zählt das Unternehmen und ist mit Niederlassungen weltweit vertreten.

## 3.2 Das nähere Umfeld

Seit jeher wird beim ISP auf die Exploration innovativer Methoden Wert gelegt, um frühzeitig wertsteigernde Maßnahmen zu entdecken und zu fördern. So setzte das Unternehmen zunächst in zwei internen Projekten seit Anfang 2008 Scrum als Framework für agile Softwareentwicklung ein.

Das Ziel des einen Projekts war der Entwurf eines Enterprise Service Bus (ESB). Da der Bus für ausgewählte Anwendungen zunächst innerhalb eines Bereichs des Unternehmens bereitgestellt werden sollte, wurde er als Bereichsbus bezeichnet. Langfristiges Ziel war die Ankopplung aller Services des Bereichs an den Bus. Das Projekt bekam den internen Namen »Greyhound«, angelehnt an die größte nordamerikanische Buslinie »Greyhound Lines«. Die Entwickler wurden folgerichtig als das Greyhound-Team bezeichnet.

Das andere, parallel laufende Projekt verfolgte das Ziel, die Domainverwaltung auf eine skalierbarere und flexiblere Architektur zu migrieren. Der ISP war

in der Vergangenheit immer mehr in den internationalen Markt vorgerückt, was zu immer umfangreicheren Anforderungen und größeren Lasten an das System führte. Auf die neuen Umstände wollte das Unternehmen im Vorfeld mit einer Migration reagieren. Dieses Projekt wurde vom nexdom-Team betreut, benannt nach dem internen Namen des Projekts »nexdom«. nexdom ist ein Wortspiel (»next domain«).

Beide Projekte waren nicht mit vertretbarem Aufwand vorab planbar. Die genauen Anforderungen an den ESB waren unklar, wie etwa die Verarbeitung der Menge von Anfragen an das System oder die durchzuschleusende Menge an Daten. Das Ziel der neuen Architektur der Domainverwaltung war es, die Probleme der alten Architektur zu adressieren, wobei die Entwickler erst sicher sein konnten, dass eine Lösung auch tatsächlich half, nachdem sie sie angewendet hatten.

Beide Projekte sahen einem festgelegten Releasetermin entgegen. Dem Domainprojekt stand ein dediziertes Team zur Verfügung, das etwa zur Hälfte jeweils aus frisch eingestellten Entwicklern und aus langjährig Beschäftigten bestand. Für die Erledigung des ESB-Projekts wurden Entwickler unterschiedlicher Teams des Unternehmens bereitgestellt.

## 3.3 Warum Scrum?

Viele Projekte beim ISP sind nicht mit vertretbarem Aufwand vorab bis zum Schluss planbar. Unverhältnismäßig hohen Aufwand verursachen langwierige Evaluationsprojekte, die im Vorfeld zur Planung von Großprojekten durchgeführt werden. Projekte zur Planung von Projekten? Genau, hört sich nicht gut an.

Hinzu kommt, dass sich der ISP im Fluss befindet: Ständig ändert sich der Markt, werden neue Technologien für Endkunden interessant oder es gilt, sich gegen die Konkurrenz zu behaupten. Jede dieser einflussnehmenden Größen lässt den ISP lernen, wie es besser zu machen ist. Das neue Wissen fließt in die Projekte ein. Doch bereits evaluierte und vorab geplante Projekte sind von Natur aus unflexibler und reagieren auf Veränderungen mit stark ansteigenden Kosten.

Selbst die erfolgversprechendste Planung und die beste Evaluation sind kein Garant dafür, dass ein Projekt nicht vorab gestoppt werden könnte. So auch beim ISP. Dieser vom Management vor dem geplanten Projektende durchgeführte Stopp von Projekten erfolgt aus den unterschiedlichsten Gründen, wobei viele gar nichts mit der Softwareentwicklung zu tun haben, sondern oftmals mit einem veränderten Projektkontext. Ein ungeplanter Stopp des Projekts hat zur Folge, dass die bisher erarbeiteten Ergebnisse kaum oder gar nicht verwertbar sind.

Scrum ist ein Framework für agile Softwareentwicklung und verspricht Lösungen für die oben genannten Probleme. Die Planung in Scrum erfolgt nur teilweise vorab, und dieser Teil ist anfangs nur grob ausgearbeitet. Phasenweise – pro Sprint – wird diese Grobplanung verfeinert, Produkt und Architektur wachsen evolutionär. Auf Veränderungen kann so schnell und unter Berücksichtigung

aller Konsequenzen reagiert werden. Die wichtigsten und risikoreichsten Features werden pro Sprint zuerst umgesetzt bis zur vollen Funktion, sodass zu jedem Zeitpunkt der Produktentwicklung das Wertvollste des Produkts dem Unternehmen zur Verfügung steht – auch und gerade bei vorzeitigem Projektstopp.

## 3.4 Ziele der Scrum-Einführung

Der ISP ging konsequent bei der Einführung von Scrum vor: Zuerst wurde das Management mit dem Framework in Schulungen vertraut gemacht. Die meisten der Entwickler in den Abteilungen, in denen Scrum-Projekte starten sollten, wurden auch geschult, ebenso wie alle an den beiden Projekten Beteiligten, etwa Produktmanager, Team-, Abteilungs- und Bereichsleiter, dazu Mitarbeiter aus dem Projekt-Office, Vertrieb und Marketing.

»Wachrütteln durch Provokation« hieß die Strategie bei den Schulungen. Scrum wurde in direktem Gegensatz zum bestehenden und gelebten Wasserfallmodell beim ISP vorgestellt. Dabei wurde strikt dogmatisch gelehrt, d.h., Scrum wurde so unterrichtet, wie es im Buche steht, ohne oder nur mit wenig Adaption an die Umgebung – Scrum-by-the-Book eben. Ziel war unter anderem das Hinterfragen der vorhandenen Prozesse.

Die Strategie ging auf: Die Mitarbeiter waren nun neugierig auf neue Prozesse. Scrum bekam seine Chance in zwei Projekten. Dabei wurde das Teilziel verfolgt, Scrum in den beiden Projekten zu etablieren, parallel zu vielen anderen Wasserfallprojekten. Nach Erreichen dieses Teilziels sollte Scrum adaptiert werden, um auch in einer Multiprojektumgebung erfolgreich zu sein.

## 3.5 Situation beim Coachwechsel

Im Sommer 2008 wechselte der ISP den agilen Coach aus, in Scrum-Terminologie ScrumMaster genannt. Der bisherige ScrumMaster hatte gute Arbeit geleistet durch provozierende Schulungen der Mitarbeiter und bei der Etablierung von Scrum in den beiden Projekten. Den neuen ScrumMaster, Autor dieses Artikels, stellte die Firma it-agile. Der bisherige ScrumMaster verließ den ISP nach einer Übergabezeit von etwa zwei Wochen.

Das nexdom-Team war zum Zeitpunkt des Coachwechsels mit 7 Entwicklern besetzt. Das Team wurde als dediziert bezeichnet, weil alle Entwickler zu einem Großteil ihrer verfügbaren Arbeitszeit an nexdom arbeiteten. Das Team teilte sich zwei Räume, die nah beieinander lagen. Es gab eine klare Vision dessen, was das Produkt später zu leisten hatte. Der Termin für den Einsatz des Produkts war vorgegeben. Die Sprint-Länge betrug drei Wochen.

Das Greyhound-Team war mit 14 Entwicklern bestückt und nicht dediziert: Es gab einige Teammitglieder, die nur 20 Prozent ihrer Arbeitszeit auf das Projekt buchten. Das Team war separiert auf mehrere Räume, teilweise auf unterschied-

lichen Etagen. Es gab eine vage Vorstellung über die Ziele des Produkts und keinen klaren Termin für die Markteinführung. Die Sprint-Länge betrug hier ebenfalls drei Wochen.

Der bisherige Scrum-Coach hatte größtenteils Scrum-by-the-Book eingeführt. Dogmatismus ist gut für den Anfang, führt aber zu üblen Problemen, wenn die Teams nicht auf die nächste Stufe der Adaption gelangen, um einen gesunden Pragmatismus zu erreichen.

Nach dem ScrumMaster-Wechsel wurde die Situation beobachtet, interpretiert und Probleme identifiziert. Die Probleme wurden sukzessive angegangen durch pragmatische Veränderungen an entscheidenden Stellen und erneuter Beobachtung, Interpretation und Problemidentifikation. Im Folgenden erfahren Sie etwas über einige Probleme und deren Lösung.

## 3.6 Die Planung

Der Autor fand folgende Situation vor: Es gab in beiden Projekten kein Product Backlog. In einem Product Backlog sollten zu Beginn eines Projekts die bekannten Anforderungen an das Produkt in Form von Stories und Epics enthalten sein. Stories sind dabei kurze und prägnante Formulierungen von Anforderungen, die innerhalb eines Sprints abgearbeitet werden. Epics beschreiben Gruppen von Stories, wobei Epics auch als Platzhalter für Stories fungieren. Ein initiales Product Backlog sollte genug Stories haben für die ersten zwei Monate der Entwicklung. Die restlichen Anforderungen können in Epics vorliegen.

Es gab ein Excel-Sheet, in dem Posten verzeichnet waren, die bis zu 100 Personentage Aufwand erforderten und damit mehr, als in einem Sprint abgearbeitet werden könnte. Insofern konnten diese Posten nicht als Stories bezeichnet werden. Es hätten Epics sein können, aber an ihnen wurde aktiv gearbeitet. Der Umstand, dass die Posten nicht in einem Sprint abarbeitbar waren, führte dazu, dass diese Posten nach jedem Sprint neu geschätzt wurden. Fast jede neue Schätzung lag dabei über der vorherigen, sodass die Schätzungen einer Inflation ausgesetzt waren. Fortschritt war so nicht erkennbar. Den meisten Projektbeteiligten, einschließlich Product Owner, sagten diese Posten auch nichts, das Ziel war somit unklar. Kaum ein Entwickler konnte feststellen, wann er eine Story fertig implementiert hatte. Außerdem war das Product Backlog nicht vollständig mit allen Stories gefüllt, die zum damaligen aktuellen Zeitpunkt bekannt waren. Das Product Backlog sollte jedoch zu jedem Zeitpunkt den aktuellen Stand der Produktplanung widerspiegeln.

Parallel zur Entwicklung entwarfen nun die Product Owner beider Projekte die Product Backlogs. Dies erfolgte bei nexdom innerhalb von gut drei Monaten, bei Greyhound binnen vier Monaten. Dass Greyhound langsamer war, lag daran, dass es keine Produktvision hatte, keinen Ausblick auf das große Ziel. Dieses

musste zuerst formuliert werden, erst dann konnte mit der Arbeit am Product Backlog begonnen werden.

Die zuerst fehlende und später nachgeschobene, aber unklare Vision bei Greyhound machte sich nicht nur beim Planen, sondern auch während des gesamten Projekts durch eine geringere Entwicklungsgeschwindigkeit bemerkbar.

Dazu möchte der Autor Folgendes anmerken: Wir haben die beiden Projekte nicht aufgrund der reinen Zahlen bei der Geschwindigkeit verglichen. Storypoints sind individuelle Größen, bezogen auf das jeweilige Team. Entsprechend ergibt es keinen Sinn, auf Storypoints basierende Geschwindigkeiten zweier Teams zu vergleichen. Die Teams profitieren viel mehr davon, sich selbst zu fragen, ob sie selbst glauben, dass eine höhere Geschwindigkeit bei der Entwicklung möglich sei.

Bei Greyhound war laut dem Team noch mehr Potenzial vorhanden: Keine Vision zu haben führt zu schwammigen Anforderungen, führt zu schwer definierbaren Stories (»Wann genau ist die Story fertig?«), führt zu schwer umsetzbarem Code (»Was genau soll ich jetzt hier implementieren?«), führt zu unzulänglichen Abnahmen beim Product-Review (»Tja, passt schon irgendwie, oder?!«).

Die Stories, die der Product Owner schrieb und die im Product Backlog landeten, wurden von den Entwicklern beider Teams unterschätzt. Das Sprint-Commitment für den ersten Sprint wurde deutlich überschritten. Das beeinflusste nicht nur die Moral der Teams negativ, sondern verhinderte auch eine langfristige Planung.

Um zu realistischeren Schätzungen zu gelangen, definierte das Team eine »Definition of Done« (DoD). In der DoD stand genau, wann eine Story oder eine Teilaufgabe (Task) fertig war, z.B. musste sie dazu getestet, refaktorisiert und dokumentiert sein. Das Voraugenhalten der einzelnen Schritte zur Erledigung einer Story half beim Schätzen der Stories enorm.

Darüber hinaus wurden Slack-Stories geschaffen. Slack ist die englische Bezeichnung für Schlupf und kommt ursprünglich aus der agilen Methodik eXtreme Programming. Slack steht metaphorisch für den Unterschied zwischen zwei Geschwindigkeiten. Slack-Stories sind Teil des aktuellen Sprints, aber sie sind nicht im Sprint-Commitment enthalten. Sind alle regulären Stories fertig und ist noch Zeit übrig am Ende des Sprints, können die Entwickler ihre Entwicklungsgeschwindigkeit durch Abarbeiten von Slack-Stories steigern. Durch diesen Trick wird ein Übercommitment effektiv verhindert und auch noch eine motivierende Verbesserungsmöglichkeit der Entwicklungsgeschwindigkeit in Aussicht gestellt.

Die Nerven der Planenden wurden durch das Schätzen von Stories im Backlog Grooming zu Beginn eines Sprints und vor der eigentlichen Planung strapaziert. Sehr lange Diskussionen über das Für und Wider vorgeschlagener Schätzungen zogen das Meeting in die Länge. Planning Poker trug seinen Teil zu einem kürzeren Meeting bei. Schließlich verlagerten die Teams das Schätzen zu einem

Einsortieren von Stories. Alle bereits geschätzten Stories eines Product Backlog wurden nach Schätzungen geclustert auf Karteikarten an die Wand gehängt. Neue Stories wurden dann mit den an der Wand hängenden verglichen. Schienen zwei vom Aufwand her zueinander zu passen, so wurde die neue Story in den Cluster der entsprechend geschätzten Story einsortiert und erhielt die gleiche Schätzung. Dies führte zu deutlich weniger Erklärungsaufwand und damit zu kürzeren Meetingzeiten beim Backlog Grooming.

Ein aus Sicht des Autors bis zum Ende des Erkenntnishorizonts dieses Erfahrungsberichts bestehender Mangel in beiden Projekten bestand im Schätzen der Tasks in konkreten Stunden. Dieser Umstand ist, wie sagt man so schön, »historisch gewachsen«: Als zu Beginn der Projekte kein Product Backlog und keine brauchbaren Stories vorlagen, brachen die Teams in den Planungsmeetings mehrere Tasks als konkrete Aufgaben aus den großen Stories heraus. Diese wurden in idealisierten Stunden gemessen, wobei eine idealisierte Stunde einer konkreten Stunde entspricht, wenn in dieser Zeit störungsfrei an einem Task programmiert werden kann. Über diese idealen Stunden erfolgte die tägliche Aufzeichnung des Burndown-Diagramms.

Ideale Stunden haben den Nachteil, dass zum Schätzen eines Tasks in dieser Einheit ein unangemessener Aufwand erforderlich ist. Die Entwickler gaben keine Schätzung für einen Task ab, ohne nicht wieder und wieder alle Arbeitsschritte durchzugehen, um auch ganz sicher zu sein, keine falsche Schätzung abzuliefern. Das kostet Zeit und treibt die Planungsaufwände in die Höhe. Hinzu kommt, dass der Product Owner kein Interesse an fertigen Tasks hat, sondern an fertigen Stories. Sie sind es, die den Mehrwert bringen. Als Folge davon kam es vor, dass am Ende eines Sprints zwar ein schöner Burndown erfolgte, aber keine einzige Story fertig war. Dies empfanden die Entwickler als weniger schlimm, da ja der Burndown immerhin Erfolg zeigen würde. Ein gefährlicher Trugschluss.

Der Autor empfiehlt statt des Ausarbeitens und Schätzens von Tasks die Konzentration auf kleinere Stories und/oder kürzere Sprints. Große Sprints begünstigen große Stories, da viel Zeit in großen Sprints bleibt, um an großen Stories zu arbeiten. Große Stories bedeuten einen großen Feedbackzyklus. Das Team lernt spät, ob die Arbeitsschritte zu einer großen Story erfolgreich waren. Außerdem beinhalten große Stories viele Arbeitsschritte, wodurch das Bedürfnis nach Visualisierung in Taskform steigt. Und wegen des späten Erfolgserlebnisses beim Erledigen einer großen Story verschafft sich das Team durch Tasks kleinere Zwischenerfolgserlebnisse.

Kleinere Stories haben wenige Arbeitsschritte, und wenige Arbeitsschritte müssen nicht unbedingt visualisiert werden. Kürzere Stories sind idealerweise an einem Tag abzuarbeiten, sodass täglich Erfolgserlebnisse zu vermelden sind. Der Feedbackzyklus ist kurz, sodass das Team schneller lernen kann. Kürzere Sprints begünstigen kürzere Stories, da große Stories schlicht nicht in kurze Sprints hineinpassen.

Kürzere Stories bedeuten zwar einerseits weniger Aufwand bei der Erstellung und Schätzung von Tasks, der Product Owner muss darauf aber gut vorbereitet werden. Dies gehört zu den Dingen, auf die der vorige agile Coach mehr hätte eingehen müssen, um dem Problem mit den Tasks Herr zu werden.

## 3.7 Das Design

Die beste Planung hilft nichts, wenn das Geplante sich am Ende als unbrauchbar erweist. In einem typischen Wasserfallmodell wird nach der Spezifikation das Design entwickelt. Module werden geschnitten aufgrund der Anforderungen aus der Spezifikation. Jedes einzelne Modul wird in Submodule und Klassen zerlegt und alles zusammen auf dem digitalen Reißbrett miteinander verbunden. In der eigentlichen Entwicklungsphase werden zuerst die Module samt Innenleben separat für sich entwickelt und erst kurz vor Projektende in das System integriert – um dann festzustellen, dass sie nicht zusammenpassen.

Vor diesem historischen Hintergrund fokussierten die Teams, insbesondere das vom nexdom-Projekt, zunächst auf Stories für Module. Ein Modul sah beispielsweise vor, die Kontaktdaten für eine Domain zu verifizieren. Am Ende eines Sprints war dieses Modul zwar tatsächlich fertig, doch war es nicht benutzbar, da es ein separater Teil des Produkts war und für sich allein stehend für den Product Owner nicht von Wert war. Er konnte schlicht mit dem Modul nichts anfangen, denn Kontakte ohne Domain waren nutzlos, und ohne das Weiterleiten an eine Registrierung bekam niemand etwas davon mit, dass die Kontaktdaten nun verifiziert waren.

Nach diesem Sprint erkannte das nexdom-Team, dass ein vertikaler Schnitt der Stories Vorteile bringt. Vertikal geschnitten zieht sich eine Story vom Eingang des Systems bis zum Ausgang und tangiert dabei nur die unbedingt notwendigen Bestandteile des Systems. Anstatt die Kontakte zu verifizieren, konzentrierte sich das Team nun darauf, die Kontaktdaten im Auftrag entsprechend zu parsen, um sie dann unverifiziert mit der Domain zu verknüpfen und schließlich an die Registrierung weiterzureichen.

Natürlich ist damit das Thema »Kontakte« noch nicht vollends abgehandelt. Kontakte können beispielsweise noch invalide sein, weil sie nicht verifiziert sind. Dennoch hat der oben skizzierte vertikale Schnitt dem Product Owner einen Mehrwert geliefert, kann er doch das System schon mit validen Kontaktdaten testen. Er hat zwar nur einen Teil des Systems, aber dieser Teil ist voll funktionsfähig.

Das Greyhound-Team kämpfte mit ähnlichen Problemen. Hier war der horizontale Schnitt zwischen dem Front- und dem Backend angesetzt worden, an dem »Frontender« bzw. »Backender« arbeiteten. Beide Programmierertypen gerieten sich immer wieder durch Missverständnisse in die Haare, vorzugsweise nach der Integration von beiden Schichten. Sogar die Stories wurden in Front- bzw. Backend-Stories aufgeteilt.

Scrum setzt auf multidisziplinäre Teams. Teams sollen zusammen arbeiten, auch wenn die Entwickler unterschiedlichen Disziplinen angehören. Konkret löste das Greyhound-Team das Problem durch vertikal geschnittene Stories, an denen jeweils sowohl »Frontender« als auch »Backender« eng zusammenarbeiteten. Das Zusammenlegen des Teams in einen Teamraum half ebenfalls.

Dieser ganze Themenkomplex »Design« wird von Scrum bisher nicht bedient. Scrum selbst sieht seine Rolle im agilen Management und ist auch nicht speziell auf die Softwareentwicklung ausgerichtet. Diese Ausrichtung versucht Scrum gerade durch die Etablierung von Certified Scrum Developer nachzukommen. Für nexdom und Greyhound sieht der Autor hier durchaus noch hohes Entwicklungspotenzial durch inkrementelles Design und testgetriebene Entwicklung. Beides verhilft zu der Flexibilität auf Codeebene, die Scrum auf der Managementebene etabliert hat. Oder anders ausgedrückt: Die Managementebene kann nicht flexibel agieren, wenn die Codeebene unflexibel ist.

## 3.8 Der soziale Umgang

Der Autor fand folgende Situation vor: Die Teams setzten anderthalbstündige Meetings nach dem Product-Review und vor den Planungsmeetings ein, also jeweils zwischen zwei Sprints. In diesen vom ScrumMaster moderierten Meetings unterhielten sich die Entwickler und die Product Owner kurz über Ereignisse im vergangenen Sprint. Die Äußerungen wurden vom ScrumMaster am Flipchart festgehalten und man einigte sich am Ende des Meetings darauf, dass es gut war, mal darüber geredet zu haben. Konkrete Maßnahmen wurden nicht vereinbart.

Die Teams waren angespannt. Der Umgangston war ruppig und der Respekt untereinander ließ des Öfteren zu wünschen übrig. Einige Entwickler starteten an vielen Tagen mit einem schlechten Gefühl in ihre Arbeit. Es gab Konflikte zwischen den Entwicklern, und in den zwei Wochen der Übergabe von einem zum anderen ScrumMaster kam es auch zu Konflikten zwischen dem ehemaligen ScrumMaster und den Entwicklern. Es gab kein Zusammengehörigkeitsgefühl innerhalb der Teams. Schließlich fühlten sich die Teammitglieder ausgebrannt und gestresst, was den vielen Überstunden geschuldet war, die zur Erreichung der bisherigen Sprint-Commitments, also vor den Änderungen in der Planung, erforderlich waren.

Die zwischensprintlichen Meetings wurden durch strukturierte Retrospektiven abgelöst. Die Dauer der Retrospektiven betrug bis zu drei Tage, wobei die Regel eher wenige Stunden lange Retrospektiven waren. Die Retrospektive wurde zwischen den Sprint-Wechseln, also nach dem Sprint-Review und vor dem Sprint Planning, gehalten. Innerhalb der Retrospektiven wurden zwischenmenschliche Konflikte aufgedeckt und geklärt. Gute und schlechte Ereignisse innerhalb des letzten Sprints wurden identifiziert. Die Ereignisse wurden untersucht und dabei Maßnahmen überlegt und beschlossen, das Gute zu verstärken

und das Schlechte zu verhindern. Retrospektiven glichen sich dabei niemals in ihrer konkreten Durchführung, sodass das Energie- und Kreativitätslevel hochgehalten werden konnte. Retrospektiven waren das Mittel überhaupt, Einsichten in die aktuelle Situation zu erhalten und den Prozess zu verändern. Das wurde auch dem Team bewusst und es wollte auf keine einzige Retrospektive verzichten.

Bei den ersten Retrospektiven kam in beiden Teams heraus, dass sie nicht beschlussfähig waren. Scrum setzt auf selbstorganisierte Teams. Selbstorganisiert bedeutet, dass sie eigenverantwortlich vorgehen zur Erreichung der vorgegebenen Ziele. Hierbei bedarf es der Beschlussfähigkeit innerhalb des Teams, um Maßnahmen zur Erreichung der Ziele zu bestimmen. Das Nichtvorhandensein der Beschlussfähigkeit innerhalb der Teams äußerte sich in einer gewissen Gleichgültigkeit, einer Mir-doch-egal-was-die-anderen-denken-Haltung einiger Teammitglieder und frustrierte ein ums andere Mal die Motivierten im Team. Diese Haltung kam aus der Erkenntnis heraus, dass egal, was jemals beschlossen wurde, sich ja doch keiner daran gehalten hätte.

Explizite Commitments brachten die Beschlussfähigkeit der Teams zurück. Hatte das Team eine Einigung erzielt, so wurde diese auf ein Flipchart geschrieben und anschließend von den Teammitgliedern einzeln mit Kürzel unterzeichnet. Durch diese Kürzel-Unterschrift erlangte das explizite Commitment einen Grad an Ernsthaftigkeit, der die Teammitglieder stärker auf die Formulierung und auf die Details dessen achten ließ, was sie da eigentlich unterschreiben sollten. Die Qualität der Einigung wurde damit erhöht. Nach dem Aufschreiben und Abzeichnen wurde das explizite Commitment im Teamraum aufgehängt, sodass es später per Fingerzeig einfach referenziert werden konnte. Wichtig dabei ist zu erwähnen, dass die expliziten Commitments nicht in Stein gemeißelt waren: Wann immer ein Teammitglied nicht mehr mit den expliziten Commitments einverstanden war, wurde neu verhandelt mit dem ganzen Team.

»Verstöße« gegen die expliziten Commitments blieben natürlich trotzdem nicht aus. »Verstöße« meint, dass ein Teammitglied sich nicht mehr an das explizite Commitment hielt, ohne entsprechend Änderungsbedarf angemeldet zu haben. Hier ist klar zwischen der Sach- und der Beziehungsebene zu unterscheiden. Auf der Sachebene mochte das Teammitglied seine Einwände gegen das explizite Commitment haben, dennoch war es auf der Beziehungsebene an das explizite Commitment gebunden, hatte das Mitglied es doch selbst mit gezeichnet. Es konnte gerne mit dem Team erneut auf der Sachebene verhandeln, aber es war ihm nicht gestattet, auf der Beziehungsebene das explizite Commitment zu brechen.

Hier kommt nun der ScrumMaster wieder ins Spiel. Er ist nicht für die Aufgaben der Teams zuständig; dafür gibt es die Rolle des Product Owner. Er zeigt auch nicht auf, wie das Team seine Arbeit zu erledigen hat; dafür ist das Team eigenverantwortlich zuständig, weil es selbstorganisiert vorgeht. Er sorgt dafür, dass der vom Team gewählte und angepasste Prozess eingehalten wird. Die expli-

ziten Commitments stellen nichts weiter dar als einen explizit ausformulierten Bestandteil des Prozesses. Wird der Prozess verletzt, z.B. durch das »Verstoßen« gegen ein explizites Commitment, so schreitet der ScrumMaster klärend ein. Er hält das Team bei der Stange, indem er dem Team die Konsequenzen bei einer Prozessabweichung verdeutlicht und entweder ein Festhalten am Prozess oder eine Prozessänderung erwirkt. Beides, Festhalten oder Verändern, soll transparent erfolgen, sodass jedes Teammitglied weiß, was und warum etwas beschlossen wurde.

Ziel der expliziten Commitments ist die Etablierung von Vertrauen und Respekt innerhalb des Teams. Ist dies einmal erreicht, werden keine expliziten Commitments mehr vonnöten sein für die Beschlussfähigkeit der Teams. Das Team vertraut sich und seinen Beschlüssen wieder.

Um solche Prozesse der Teamentwicklung zu unterstützen, hilft es dem ScrumMaster, auf die Sorgen und Bedürfnisse jedes einzelnen Teammitglieds eingehen zu können. Je besser er die Beteiligten kennt, desto besser kann er in brenzligen Situationen auf die Beteiligten einwirken und zur Bewältigung der Situation beitragen.

Zu diesem Zwecke wurden halbstündige One-on-One-Gespräche zwischen Teammitglied und ScrumMaster pro Sprint eingeführt, sogenannte Sprintlies. Der Name ist eine badische Komposition und heißt so viel wie »etwas Kleines während des Sprints«. Sprintlies wurden meist bei einem Kaffee auf der Couch durchgeführt und hatten einzig das Ziel, dass der ScrumMaster mindestens einmal pro Sprint mit jedem Teammitglied unter vier Augen sprechen konnte. Teilweise fand dabei wirklich nur ein nettes Kaffeegespräch übers Wetter statt. Oft genug schüttete das Teammitglied allerdings sein Herz aus. So konnten durch die Sprintlies viele Probleme erfolgreich geklärt werden, bevor sie eskalierten.

Retrospektiven, explizite Commitments und Sprintlies schufen Vertrauen und Respekt innerhalb der Teams und gegenüber Außenstehenden wie dem ScrumMaster. Der Umgang miteinander wurde deutlich harmonischer. Die Teams hatten den Eindruck, »echte« Teams zu sein. Der Zusammenhalt wuchs und half sehr bei der Bewältigung von schwierigen Problemen. Stressige Situationen wurden lockerer und überlegter bewältigt, kopfloses Verhalten meist im Kern erstickt. Eine angenehme und produktive Ruhe breitete sich aus. Die Maßnahmen trugen zu einem Großteil dazu bei, dass weniger bis keine Überstunden mehr erforderlich waren zur Bewältigung der gleichen Arbeit.

## 3.9 Der ScrumMaster

Der Autor fand folgende Situation vor: Der ehemalige ScrumMaster schrieb dem Team teilweise vor, was es wie zu tun hatte. Beispielsweise wurde vom ScrumMaster ein digitales Tool eingeführt zur Pflege des täglichen Burndown-Diagramms. Schreibt der ScrumMaster dem Team vor, wie es etwas tun soll, so

nimmt er dem Team auch die Verantwortung für dessen Tun ab. Misslingt das Vorhaben, so wäre der ScrumMaster schuld, denn er hat den Mist ja angeordnet. Im konkreten Falle des digitalen Tools für die Pflege des Burndown-Diagramms führte das zu einer Ablehnung seitens des Teams aufgrund von Mängeln beim Tool. Da das Team selbst nicht die Macht hatte, diese Mängel zu beseitigen, schob es die Verantwortung auf den ScrumMaster ab. Das Ergebnis war, dass der ScrumMaster regelmäßig auf die Pflege des Burndown-Diagramms hinweisen musste, was zu ziemlichem Verdruss auf beiden Seiten führte: Die Entwickler fühlten sich belehrt und getadelt, der ScrumMaster sich im Stich gelassen und missachtet.

Wie schon erwähnt achtet der ScrumMaster auf die Einhaltung des vom Team gesetzten Prozesses. Er weist nicht an, was (Product Owner) oder wie (Team) etwas zu tun ist. Wer Verantwortung trägt, dem muss auch die Macht gegeben werden, dieser nachzukommen. Wenn das Team die Verantwortung dafür trägt, wie es seine Ziele erreichen soll, so muss es auch die Befugnis haben, dies zu entscheiden. Dem Team diese Macht zu geben und es bei Bedarf an seine Eigenverantwortung zu erinnern ist Teil der Aufgabe eines ScrumMasters.

Der neue ScrumMaster stellte zunächst Mängel in der Pflege des Burndown-Diagramms fest und zeigte sie dem Team auf. Da das Burndown-Diagramm Bestandteil des Prozesses ist, sah sich der ScrumMaster in seiner Pflicht, das Team zur Einhaltung oder Änderung des Prozesses anzuhalten. Dem Team gehört der Prozess, folglich hatte es jetzt die Wahl: den Prozess zu ändern und das Burndown-Diagramm nicht mehr weiterzupflegen oder den Prozess beizubehalten und das Burndown-Diagramm anders zu pflegen.

Der ScrumMaster half dem Team bei dieser Entscheidung. Er zeigte die Konsequenzen auf, die das Abschaffen des Burndown-Diagramms zur Folge hätte, etwa ein Verlust an Transparenz. Das Team fand keine Antwort darauf, wie es diesen Verlust an Transparenz kompensieren könnte, und es konnte auch nicht auf die Transparenz verzichten. Daraufhin schlug der ScrumMaster ein anderes Verfahren für die Pflege des Burndown-Diagramms vor mit dem Hinweis, dass wenn dieses Verfahren auch nicht funktionieren würde, man sich eben erneut zusammensetzen müsste und weitere Möglichkeiten durchspielen könnte. Das Team entschied sich, das neue Verfahren für einen Sprint auszuprobieren. Nach einem Sprint schließlich zeigte sich das neue Verfahren als tauglich und wurde beibehalten.

Hier wird deutlich, dass der ScrumMaster keinerlei Entscheidungsbefugnisse hat. Er kann Optionen vorschlagen, muss aber immer damit rechnen, dass sie abgelehnt werden können. Das Team entscheidet letztendlich, wie es sein Ziel erreichen wird. Es ist hilfreich für den ScrumMaster, ein hohes Maß an Erfahrung zu haben, um dem Team gute Optionen vorzuschlagen, damit es nicht enttäuscht wird und schließlich das Vertrauen zum ScrumMaster verliert.

ScrumMaster zu sein bedeutet auch, dem Team nicht im Wege zu stehen auf dem Pfad zur Selbstorganisation. Das Daily Scrum ist bekanntlich ein kurzes Meeting im Stehen. Die Entwickler berichten sich gegenseitig, was sie seit dem letzten Daily Scrum getan haben, was sie bis zum nächsten vorhaben und was sie gerade blockiert in ihrem Tun. Der vormalige ScrumMaster moderierte dabei so präsent, dass die Entwickler ihm berichteten und nicht mehr sich gegenseitig. Dabei wurde nicht mehr darauf geachtet, dass die Teammitglieder die Ausführungen jedes Einzelnen verstanden, sondern dass der ScrumMaster sie verstand. War das der Fall, war der Report erfolgreich, unabhängig davon, ob vielleicht das ein oder andere Teammitglied den Report nicht verstanden hatte. Das führte so weit, dass, als der neue ScrumMaster einmal zu spät zum Daily Scrum kam, alle Entwickler auf ihn warteten, da sie ihm ja berichten sollten.

Beim Daily Scrum soll das Team sich selbst berichten und über den Fortgang des Projekts in Kenntnis setzen. Der ScrumMaster ist lediglich für die Moderation zuständig. Er achtet darauf, dass alle Fragen beantwortet werden oder dass sich niemand in lange Monologe ergeht oder er veranlasst bei aufkommenden Diskussionen die Entwickler, sich für später zu einem Meeting zu verabreden.

Nachdem der ScrumMaster zu spät kam und feststellte, dass das Daily Scrum noch nicht angefangen hatte, klärte er mit den Entwicklern, dass sie pünktlich anfangen sollten, auch wenn er noch nicht anwesend sei. Nach zwei weiteren »absichtlichen« Verspätungen des Scrum Masters berichteten die Entwickler ab sofort sich gegenseitig und nicht mehr dem Scrum Master.

Schließlich kann ein ScrumMaster auch mal krank werden oder möchte Urlaub machen. Der ISP leistete sich einen ScrumMaster für zwei Projekte, was zu weiteren Engpässen bei sich überschneidenden Terminen in beiden Projekten führen kann. Die Lösung: Der ScrumMaster wurde später zum Scrum-Coach und rekrutierte aus den beiden Projekten je einen Entwickler, der bereit war, die Rolle des stellvertretenden ScrumMasters zu übernehmen, falls der ScrumMaster verhindert sein sollte.

Der ScrumMaster ist nicht Teil des Entwicklerteams. Er bekleidet seine Position außerhalb des Teams, um neutrale Moderationen zu ermöglichen. Er kann neutral moderieren, weil er weder das Handeln des Teams noch die Anforderungen vom Product Owner vertreten muss. Ein Entwickler, der für kurze Zeit den Hut des stellvertretenden ScrumMasters gegenüber seinem eigenen Team auf hat, kann schwer aus seiner alten Haut. Er verzettelt sich in Rollenkonflikten.

Nach dieser Erkenntnis setzte der ScrumMaster seine Stellvertreter kreuzweise ein, d.h., der Stellvertreter aus dem nexdom-Team fungierte aushilfsweise als ScrumMaster im Greyhound-Team und umgekehrt. Fortan gab es keine Rollenkonflikte mehr und auch die stellvertretenden ScrumMaster konnten eine neutrale Position bei ihren Moderationstätigkeiten einnehmen.

Die Stellvertreter wurden übrigens im Laufe der Projekte zu echten ScrumMastern ausgebildet, inklusive Zertifizierung. Der ScrumMaster von it-agile fun-

gierte dabei als Scrum-Coach und führte die ScrumMaster des ISP in der Praxis ins Moderieren von Daily Scrums, Planungsmeetings und Retrospektiven ein. Heute führen die ScrumMaster des ISP die Teams.

## 3.10 Die Werkzeuge

Über die Verwendung des digitalen Tools zur Pflege des Burndown-Diagramms wurde bereits weiter oben geschrieben. Dieses Tool war eine Excel-Tabelle nebst generiertem Diagramm. Die Tabelle lag auf einem Dateiserver. Excel ist leider nicht Multiuser-fähig, d.h., es kann nur ein Entwickler gleichzeitig die Tabelle öffnen. Vergisst er, sie wieder zu schließen, kann sie kein anderer öffnen. Das nexdom-Team hat deshalb ein webbasiertes Tool im Intranet aufgesetzt. Aber auch dieses hatte den großen Nachteil, ebenso wie die Excel-Tabelle, dass es in den Teams nicht präsent war. Jeden Abend galt es daran zu denken, die Tabelle oder das Tool auszufüllen bzw. zu bedienen, und jeden Abend vergaßen es wieder Entwickler. Es gab immer wieder Unstimmigkeiten.

Außerdem war das Burndown-Diagramm nicht sofort präsent, wenn das Team es brauchte. Entweder musste man erst an einen Rechner, um die Tabelle oder das Tool zu öffnen, oder das Burndown-Diagramm wurde regelmäßig großflächig ausgedruckt, wobei es mit der Regelmäßigkeit aus unterschiedlichsten Gründen oft haperte, sodass es nicht tagesaktuell war.

Ähnliche Probleme betrafen auch die Pflege der Stories und der Tasks. Auch sie wurden digital via Excel oder webbasiertem Tool gepflegt, und auch bei ihnen entstanden oft Unstimmigkeiten. Sie waren ebenso wenig transparent wie das Burndown-Diagramm und quasi per Mausklick ausgeblendet aus dem täglichen Geschehen. Und auch sie waren nicht präsent bei den Entwicklern: Sie gehörten nicht zur täglichen Arbeit dazu, sondern waren leidiges Beiwerk.

Die Einführung eines Taskboards samt gezeichnetem Burndown-Diagramm half. Beides wurde großflächig an den Wänden der Teamräume aufgehängt. Das Taskboard bestand aus Reihen und Spalten. Die Reihen wurden mit Stories und Tasks gefüllt, die auf Karteikarten geschrieben wurden. Auf jeder Karteikarte stand die Schätzung der Story oder des Tasks. Die Spalten kennzeichneten Zustände der Stories und Tasks, etwa »geplant«, »in Arbeit« oder »fertig«. Karten, die in der Spalte »fertig« hingen, wurden zum Burndown-Diagramm hinzugezählt.

Wer auch immer in den Raum kam, konnte auf einen Blick erkennen, wie das Projekt vorankam. So auch beim Burndown-Diagramm, das auf ein quer an die Wand geklebtes Flipchart-Papier gezeichnet wurde. Vor jedem Daily Scrum zählten die Entwickler die Schätzungen in der Spalte »fertig« des Taskboards und trugen das Ergebnis ins Burndown-Diagramm ein. Unstimmigkeiten in Taskboard und/oder Burndown-Diagramm wurden sofort offensichtlich und konnten direkt angegangen werden.

Die Teams fanden die Idee des Taskboards so gut und bauten sie aus. Jedes Team führte ein Impediment Backlog an der Wand, in dem es jeglichen Blocker festhielt und nach jedem Daily Scrum zur Blocker-Jagd blies, d.h. den Fortschritt der Blockerbeseitigung feststellte und den Stand aktualisierte.

Überhaupt die Nutzung der Wände! Der Ausbau von informationsstiftenden Quellen im Entwicklerraum nennt sich Information Radiator. Jeder Entwickler hat diese Informationen ständig vor Augen und kann sie nutzen. Dazu zählten bei den Teams das Taskboard, das Impediment Backlog, das Burndown-Diagramm und die expliziten Commitments. Temporär in den Zeiten des Backlog Grooming und der Planungsmeetings wurde auch das komplette Product Backlog an die Wand übertragen – auf Karteikarten, je eine Karteikarte pro Story. All diese Informationen sind stets präsent – und werden genutzt! Bei der Planung werden Karteikarten an den Wänden gruppiert, angeordnet, sortiert. Der Projektfortschritt wird kontinuierlich bei Bedarf und mehrmals am Tag am Taskboard durch Verschiebung der Karteikarten aktualisiert. Burndown-Diagramm und Impediment Backlog werden vor und nach dem Daily Scrum aktualisiert. Und es kommt nicht selten vor, dass Entwickler in Diskussionen an bestimmte Stellen der Wand im Teamraum deuten und dieser Fingerzeig mehr erklärt als sehr viele Worte.

Ein kleines, aber hilfreiches Werkzeug soll nicht unerwähnt bleiben: der Daily-Scrum-Ball. Im Daily Scrum werden drei Fragen von jedem Entwickler reihum beantwortet. Nun kann der moderierende ScrumMaster bei jedem Auslassen einer Frage entsprechend auf die Frage hinweisen. Das ist recht mühsam und wirkt auch ein wenig oberlehrerhaft. Alternativ dazu kann der Daily-Scrum-Ball, auf dem die drei Fragen aufgedruckt sind, als Gesprächs-Token die Runde machen. Wer den Ball hat, darf sprechen, der Rest nicht. Wer sprechen darf, der beantwortet die Fragen auf dem Ball. Fertig.

## 3.11 Multiprojektmanagement

Eine der größten Herausforderungen aus Sicht von Scrum ist das Multiprojektmanagement. Scrum gibt ein multidisziplinäres Team von Entwicklern vor, das genau ein Produkt entwickelt. Es bezieht seine Anforderungen von genau einem Product Owner. In Multiprojektumgebungen wie beim ISP werden mehrere Projekte von einem Team parallel erledigt, oder es besteht erst gar kein Team und die an einem Projekt beteiligten Entwickler werden je nach Bedarf zusammengebracht. Beide Szenarien bedeuten, dass ein Entwickler in mehreren Projekten gleichzeitig werkelt.

Tatsächlich lohnt es sich nicht, parallel mehrere Projekte zu fahren. Durch den Kontextwechsel erfahren die Entwickler sehr viel Ablenkung und müssen sich immer wieder neu einarbeiten, was zeitaufwendig und damit teuer ist. Darüber hinaus werden die Projekte im Schnitt deutlich langsamer fertig. Wenn beispielsweise ein Entwickler zwei Projekte erledigen soll, und er würde für je eines

der Projekte 10 Entwicklungstage benötigen, so würde er parallel mit beiden nach genau 20 Entwicklungstagen fertig werden. Das gilt natürlich nur idealerweise, also wenn man den Produktivitätsverlust durch Kontextwechsel außer Acht lässt. Erst nach 20 Entwicklungstagen könnten beide Produkte aus den Projekten am Markt gewinnbringend eingesetzt werden. Würde der gleiche Entwickler zuerst das eine Projekt und dann das andere entwickeln, so könnte ein Produkt bereits nach 10 Tagen an den Start gehen und Gewinn einfahren, das andere nach insgesamt 20 Tagen. In diesem Beispiel würde die sukzessive Entwicklung einen Geschwindigkeitsvorteil von 25 Prozent bringen – nicht die schlechteste Rendite.

Fragt man die Scrum-Community nach einer Möglichkeit, Scrum in Multiprojektumgebungen einzusetzen, wird geantwortet, dass ein Unternehmen besser kein Multiprojektmanagement betreiben sollte, eben aus der obigen Begründung heraus. Unternehmen wie der ISP können diesen Ausführungen allerdings nur bedingt folgen oder halten dennoch an Multiprojektmanagement fest. Für sie ist es schlicht eine enorme Herausforderung, ihre Prozesse so umzustellen, dass Projekte ab sofort sukzessive angegangen werden. Und ob sich dieser Aufwand tatsächlich lohnt, ist für diese Unternehmen nicht gewiss.

Einige Unternehmen möchten erst einmal im Kleinen wissen, ob sich agile Softwareentwicklung am Beispiel von Scrum lohnt, und dann kann man ja immer noch über weitgreifendere Umstellungen sprechen. Nirgendwo sonst hilft der Dogmatismus so wenig und Pragmatismus so viel. Scrum nützt es nichts, Multiprojektmanagement zu ignorieren, denn das Problem besteht weiterhin. Außerdem sollte ein agiler Coach nicht gleich die Flinte ins Multiprojekt-Korn werfen, denn er sollte in der Lage sein, auch in diesem Umfeld Veränderungen zu bewirken.

Das nexdom-Team kämpfte ebenfalls mit vielen Projekten, die es gleichzeitig zu erledigen hatte. Das Projekt nexdom war nur eines ihrer Projekte, und auch nur dieses wurde mit Scrum umgesetzt. Hinzu kamen andere Projekte, die auch nicht immer von allen im Team erledigt werden konnten. Gerade an Altsystemen arbeiteten nur die alten Hasen des Teams, da sie diesen Code meist noch von früher her kannten. Dazu kam dann noch das sogenannte Tagesgeschäft, was größtenteils Wartungsarbeiten an bestehenden Systemen umfasste.

Wenn die Sprint-Planung für das nexdom-Projekt anstand, gab jedes Teammitglied an, wie viel es für den kommenden Sprint in Tagen zur Verfügung stehen würde. Der Rest war für Arbeiten an anderen Projekten oder für das Tagesgeschäft reserviert. Nach mehreren Sprints wurde immer deutlicher, dass das Sprint-Commitment nur durch enorme Schmerzen erreicht werden konnte – wenn überhaupt!

In einer Retrospektive entdeckte das Team, dass seine Schätzungen für das nexdom-Projekt eigentlich recht gut waren. Die Schätzungen für die übrigen Projekte und für das Tagesgeschäft liefen allerdings ständig aus dem Ruder und ver-

schlangen mehr Arbeitszeit als eigentlich gedacht. Diese Arbeitszeit wiederum wurde größtenteils mit Überstunden kompensiert, was wiederum die Arbeitsleistung insgesamt negativ beeinflusste, so auch die Arbeit an nexdom. Daher also kamen die Schmerzen beim Erreichen des Sprint-Commitments.

Die Arbeiten an nexdom waren durch Scrum transparent, nur nicht die übrigen Arbeiten an anderen Projekten und am Tagesgeschäft. Der Teamleiter von nexdom war für die Koordination der anderen Projekte und des Tagesgeschäfts zuständig, hatte aber keine Übersicht darüber, wie viel Aufwand das Team in welches Projekt steckte. Ohne konkrete Begründung für eine Überlastung des Teams konnte er kaum eine Anfrage nach Erledigung von Arbeiten ablehnen. Eine konkrete Begründung könnte ihm dagegen Scrum durch seine Transparenz liefern ...

Die Lösung war dann recht einfach: Alle Anforderungen aus Nicht-nexdom-Projekten wurden vom Teamleiter in Story-Form gegossen, der fortan als Product Owner für Nicht-nexdom-Projekte fungierte. Diese Stories wurden vom Team geschätzt. Im Planungsmeeting für den kommenden Sprint saßen ab sofort zwei Product Owner, einer für nexdom und einer für den Rest. Die Entscheidung, ob die Arbeit an nexdom oder einem anderen Projekt mehr Priorität hatte, fällte ab sofort nicht mehr das Team, sondern die Product Owner. Stories beider Product Owner füllten ein einziges Product Backlog, innerhalb dessen sich die Arbeit priorisieren ließ. Ein kleines Projekt sollte vor nexdom durchgeführt werden? Gerne, aber dann würde nexdom einen Monat später fertig werden.

Mit dieser Transparenz lassen sich nicht unbedingt angenehme Konsequenzen aufzeigen, aber die Intransparenz geht nun nicht mehr zulasten der Entwickler. Der Teamleiter kann besser denn je belegen, wie stark und womit seine Leute ausgelastet sind. Die Konsequenzen von parallelen Projektabläufen werden unmittelbar sichtbar, und zwar schon in der Planung und nicht erst beim Nichterreichen utopischer Ziele.

Kern dieser Lösung fürs Multiprojektmanagement mit Scrum ist das dedizierte Team. Dieses Team ist eingespielt und kennt sich und seinen Prozess sehr gut und hat dadurch viel weniger Aufwände bei der Abstimmung. Es ist selbstorganisiert und kann damit selbst entscheiden, wie es die ihm gestellten Aufgaben löst. Es arbeitet Neulinge durch alte Hasen in Bereiche ein, auf denen vormals Kopfmonopole bestanden, und organisiert sich so selbst zu einem multidisziplinären Team um, das deutlich flexibler einsetzbar ist.

## 3.12 Fazit

Dogmatismus mag anfangs ein gangbarer Weg bei der Vorstellung neuer Methodiken sein und per Provokation und Konfrontation wachrütteln. Spätestens nach den ersten Schritten mit der neuen Methode muss eine schrittweise Adaption an die Umgebung und vor allem an das Gelernte erfolgen. Seien es anfängliche Schwierigkeiten bei der Planung, die Berücksichtigung des Designs, der angemessene soziale Umgang, adäquates Verhalten des ScrumMasters oder eine gute Werkzeugauswahl: Scrum lebt durch ständige Anpassung an Veränderungen. Im Falle des ISP hat dieses pragmatische Vorgehen sogar zu einem guten ersten Schritt in Richtung einer Lösung fürs Multiprojektmanagement geführt.

## 3.13 Agile Werte im Projekt

| **Individuen und Interaktionen** | **vor** | **Prozessen und Tools** |
|---|---|---|
| ▲<br>Das Unternehmen machte keine Vorgaben, *wie* Scrum gelebt werden sollte. Die Teams konnten sich mithilfe des agilen Coachs auf konkrete Lösungen für ihre Probleme konzentrieren. | | |
| **Laufende Software** | **vor** | **Ausführlicher Dokumentation** |
| ▲<br>Die Notwendigkeit von Dokumentation konnten die Teams frei verhandeln:<br>Der ScrumMaster stimmte die Dokumentation mit dem Auftraggeber ab; der Product Owner stimmte die Dokumentation mit den Stakeholdern ab; die Entwickler stimmten die Dokumentation mit den Usern ab. Für alle zusammen war der Fokus auf die lauffähige Software gerichtet. | | |
| **Zusammenarbeit mit dem Kunden** | **vor** | **Vertragsverhandlungen** |
| ▲<br>Das agile Vorgehen hat maßgeblich in beiden Projekten beigetragen, dass sehr zeitnah auf Kundenwünsche eingegangen werden konnte. | | |
| **Reagieren auf Veränderungen** | **vor** | **Befolgen eines Plans** |
| ▲<br>Probleme wurden erkannt und effizient angegangen. Durch die gegebenen Strukturen war die Bereitschaft zu Veränderungen jedoch nicht überall für notwendig befunden. | | |

Henning Wolf, Agile Projekte mit Scrum, XP und Kanban , dpunkt.verlag, ISBN 978-3-86490-266-6

# 4 Scrum-Einführung bei ImmobilienScout24

André Neubauer

*ImmobilienScout24 ist der größte deutsche Internetmarktplatz für Immobilien. Mit über fünf Millionen Nutzern pro Monat ist die Webseite auch das mit Abstand meistbesuchte Immobilienportal im deutschsprachigen Internet. Das Unternehmen sitzt in Berlin und beschäftigt über 500 Mitarbeiter.*

## 4.1 Die Situation

Vor der Einführung von Scrum (als agile Methode der Softwareentwicklung) basierte der Produktentwicklungsprozess auf dem Wasserfallmodell, wobei die einzelnen Abteilungen klassisch nacheinander in die Projekte involviert wurden. Die Entwicklung von Features für das Internetportal war damit träge und langwierig. Projekte mit einer Dauer von deutlich über einem halben Jahr waren normal und ließen sich im Vorfeld sowohl aufwandstechnisch als auch inhaltlich nur schwer abschätzen. Dieser Umstand führte im Projektverlauf zu Änderungen an den Fachanforderungen und zu Zeitdruck. Als Ergebnis verspäteten sich Projekte bzw. konnten Endtermine nur durch zusätzlichen Einsatz eingehalten werden. Ein anderes Problem war, dass Projekte nur in ihrer Gänze funktionsfähig waren, die Livestellung von Teillösungen war nicht möglich. Damit war die Organisation dazu gezwungen, Projekte bis zum Ende durchzuführen oder abzubrechen, dann jedoch auch keinen Ertrag für den investierten Aufwand zu erhalten. Zusammengefasst trafen die typischen Probleme von wasserfallbasierten Projekten auch auf ImmobilienScout24 zu.

Die mangelnde Flexibilität resultierte in der Unzufriedenheit bei den Mitarbeitern. Jede Änderung an der Plattform erschien langwierig. Dazu trug auch die Plattform aus technischer Sicht bei. In die Pflege der Technik wurde wenig investiert. Jedes Projekt betrachtete nur fachliche Aspekte, sodass nach 10 Jahren Produktentwicklung die Komplexität der Gesamtanwendung enorm war.

Alles in allem war die Situation zwar nicht aussichtslos, es gab jedoch genug Ansatzpunkte für Verbesserungen.

## 4.2 Die Einführung

Im Frühjahr 2008 besuchten zwei Produktmanager und zwei IT-Teamleiter ein Scrum-Training. Das Ziel war es zu bewerten, inwiefern eine agile Methode Vorteile für den Produktentwicklungsprozess bei ImmobilienScout24 bietet. Die Einschätzung war eindeutig: Sowohl das iterative Vorgehen, gepaart mit der pragmatischen Haltung sowie dem Fokus auf das Team und dessen Selbstbestimmung, als auch die Verantwortlichkeit stellten eindeutige Vorteile dar. Vor diesem Hintergrund wurde die Entscheidung getroffen, im Rahmen eines abgeschlossenen Projekts für ein Softwareentwicklungsteam und das zugehörige Produktmanagement Scrum auszuprobieren.

Scrum-Einführungen können grundsätzlich aus zwei Richtungen initiiert werden, vom Management oder von den Teams. Beide Ansätze weisen dabei spezifische Vor- und Nachteile auf. Ein wesentlicher Punkt für den Erfolg einer entsprechenden Initiative ist die beidseitige Akzeptanz des Vorgehensmodells. Teams dürfen die Einführung einer agilen Methode nicht als eine weitere Produktivitätsmaßnahme aus dem Management verstehen, sondern als Chance, sich stärker in den Produktentwicklungsprozess zu involvieren und erfolgreich in kurzen Zyklen einen Mehrwert zu schaffen. Aus Sicht des Managements darf mit dem Einsatz von agilen Methoden und der damit einhergehenden Selbstbestimmung und Verantwortlichkeit der Teams nicht ein Machtverlust assoziiert werden. Beide Aspekte zielen in erster Linie auf operative Themen ab und sind in diesem Zusammenhang durchaus sinnvoll. Eine konkrete Verantwortung sollte dort benannt werden, wo selbige auch wahrgenommen werden kann. Erst wenn ein Team Verantwortung für ein (Teil-)Produkt übernimmt und sich damit identifiziert, wird es für eine nachhaltige Entwicklung sorgen.

ImmobilienScout24 war in der glücklichen Situation, dass sowohl die Teams als auch das Management (der CTO war zum damaligen Zeitpunkt bereits Certified ScrumMaster) den agilen Weg beschreiten wollten. Im September 2008 wurde es konkret, Scrum wurde für das Team, das bei ImmobilienScout24 die Suche betreut, eingeführt. Im Vorfeld wurden dafür ScrumMaster und Product Owner geschult. Für das Team erfolgte die Scrum-Einführung nach einem eintägigen Workshop als Training-on-the-Job (Lernen durch Tun). In den ersten Wochen war der Umbruch deutlich zu beobachten. Das Team erarbeitete zusammen mit dem Product Owner das Product Backlog und setzte in einem 3-Wochen-Sprint die ersten User Stories um. Auch wenn das Commitment des ersten Sprints nicht eingehalten werden konnte, erfüllten einige Stories am Ende des Sprints das Level of Done und gingen live! Aber noch etwas anderes war passiert: Das Team hatte durch die klar definierten und abgegrenzten Stories einen Fokus. Auf der anderen Seite erhielt der Product Owner mit dem Burndown-Chart, das den Fortschritt im Sprint visualisierte, Planungssicherheit.

Neben den ersten Erfolgen zeigte das agile Vorgehen jedoch auch deutliche Probleme und Schwächen in den umgebenden Prozessen auf. Das Team begann

eingeübte Prozesse zu hinterfragen und Probleme, die schon lange existierten und den Arbeitsablauf störten, zu benennen. Aus Sicht des ScrumMasters war die Zeit geprägt durch Impediments (Hindernisse). Gerade von außen auf das Team einwirkende Hindernisse waren schwierig zu lösen, da die Organisation um das Team herum nicht agil war. Auch wenn nicht alle Impediments gelöst werden konnten, war es wichtig, selbige anzusprechen, um die Selbstbestimmung des Teams zu fördern.

Trotz der Hindernisse waren die erhofften Vorteile offensichtlich, sodass auch andere Teams »scrummen« wollten. Das Teststadium wurde verlassen und Scrum schrittweise in den einzelnen Teams eingeführt. Als wesentlicher Erfolgsfaktor der Scrum-Einführung hat sich in diesem Zusammenhang der Einsatz externer Scrum-Coachs herausgestellt. Diese waren nicht mit den Prozessen bei ImmobilienScout24 vertraut und konnten so unvoreingenommen agieren. Ein Vorteil dieser Objektivität ist, dass nicht der Drang existiert, Gegebenheiten zu akzeptieren und Altlasten schönzureden. Scrum wurde vor diesem Hintergrund bei ImmobilienScout24 weitgehend ohne Anpassungen eingeführt. Lediglich die hohen fachlichen und technischen Abhängigkeiten zwischen den Teams wurden dediziert berücksichtigt.

Um die notwendige Unterstützungsleistung zwischen den Teams zu gewährleisten, gleichzeitig jedoch nicht die Stabilität im Sprint zu gefährden, waren die Sprints über alle Teams synchron. »Support Requests« wurden im Sprint Planning unter den Teams ausgetauscht und konnten im Commitment berücksichtigt werden. Darüber hinaus wurde ein zusätzliches Meeting eingeführt: Das tägliche »Scrum of Scrums« hatte das Ziel, dass sich die internen und externen Teams über den aktuellen Stand im Sprint, aber auch zu Impediments austauschen und gegebenenfalls unterstützen.

Über einen Zeitraum von gut 9 Monaten wurde Scrum für die gesamte Softwareentwicklung inklusive der externen Teams eingeführt. Das Vorgehen war dabei immer identisch. Product Owner, ScrumMaster und Team erhielten vorbereitend eine Schulung und wurden dann für die ersten Sprints intensiv durch einen Scrum-Coach betreut. Das Produktmanagement stellte die Product Owner, die Rolle des ScrumMasters wurde durch die Teamleiter der Softwareentwicklungsteams übernommen. Die Entscheidung, dass Teamleiter und ScrumMaster ein und dieselbe Person waren, wurde vor dem Hintergrund der unterschiedlichen Aufgaben und Ziele, die mit den beiden Rollen einhergehen, anfangs häufig diskutiert. Der vor allem argumentierte Intrarollenkonflikt trat nicht ein, sehr wohl jedoch ein Interrollenkonflikt durch die zusätzliche Aufgabe. Auch wenn dies in erster Linie eine zeitliche Herausforderung bedeutete, ist dieser Aspekt als kritisch zu bezeichnen, da gerade in der Einführungsphase ein höheres Engagement notwendig ist, um die Akzeptanz für das neue Vorgehen zu entwickeln.

Neben internen Softwareentwicklungsteams verfügt ImmobilienScout24 über vier externe Teams an den Standorten Lünen und Breslau. Mit diesem Engagement wird kein klassisches Projekt-Outsourcing verfolgt, sondern das Ziel,

zusätzliche Softwareentwicklungsteams aufzubauen, die sich in die bestehende Organisation integrieren – im Sinne einer »verlängerten Werkbank«.

Als wesentliche Herausforderung stellten sich die Integration der externen Teams und deren Informationsaustausch mit dem Product Owner bzw. der internen Entwicklung heraus. Um eine gute Kommunikation sicherzustellen, musste ein hoher Aufwand betrieben werden – und es kam dennoch zu Missverständnissen. Die Implikationen einer fremdsprachlichen Kommunikation sowie der verstärkte, notwendige Einsatz digitaler Kommunikationsmedien zur Überbrückung der Distanz zwischen den Standorten sind nicht zu unterschätzen. Ein Meeting, das stark darunter litt, war das tägliche »Scrum of Scrums«. Die Notwendigkeit, Englisch mit nicht anwesenden Teams über teilweise schlechte Audioverbindungen zu sprechen, führte dazu, dass die Teams die Kommunikation auf ein Minimum begrenzten und Informationen damit nicht fließen konnten. Zur Verbesserung der Verständigung wurden Englischkurse angeboten und verschiedenste Kommunikationsmedien ausprobiert. Zudem reisen Teams und Product Owner auch heute noch häufig zwischen den Standorten hin und her, um einen regelmäßigen und direkten Austausch zu realisieren. Auch wenn sich die kommunikativen Einschränkungen nicht vollständig kompensieren lassen, so ist der Austausch mittlerweile eingespielt.

Angefangen mit einfachen Kommunikationsmitteln (z.B. Skype) ist ImmobilienScout24 mittlerweile bei professionellen Konferenzsystemen angekommen, die für Meetings mit den externen Teams eingesetzt werden.

Eine zentrale Eigenschaft in vielerlei Hinsicht ist die mit Scrum einhergehende Transparenz. Neben dem Fortschritt im Sprint werden auch Defizite bei der Umsetzung sichtbar. Gerade in der Phase der Einführung scheint jedoch alles, was vorher (halbwegs) funktioniert hat, infrage gestellt zu werden. Scrum deckt Missstände schonungslos auf, das Lösen obliegt dann der Organisation. Das war bei ImmobilienScout24 nicht anders. Diese Sichtbarkeit mag teilweise auch unangenehm sein, ist aber notwendig. Erst wenn Probleme offensichtlich sind, existiert die Bereitschaft, diese anzugehen! Eine Herausforderung stellt dabei der schmale Grad zwischen situativer und perfektionistischer Lösung dar. Das Ziel sollte es sein, Probleme nicht mehrfach, sondern nachhaltig zu lösen. Sinnvoll ist es dabei, nicht alle möglichen Eventualitäten zu berücksichtigen. In jedem Fall ist es wichtig, das eigentliche Problem zu betrachten und nicht zu überlegen, wie der Scrum-Prozess anzupassen ist, um das Hindernis erneut zu verschleiern.

Die »Missstände«, die im Rahmen der Scrum-Einführung bei ImmobilienScout24 sichtbar wurden, waren vielfältig. Neben nachgelagerten Test- und umständlichen Releaseprozessen galt vor allem die Entwicklungsinfrastruktur als Hindernis. Um den Zustand bei zuerst genannten Punkten zu verbessern, wurden Mitarbeiter der Testabteilung in die Teams integriert, sodass die Qualitätssicherung bereits durch das Team im Sprint abgedeckt werden konnte. Der Releaseprozess wurde entschlackt und wird bis heute kontinuierlich vereinfacht mit dem Ergebnis, dass der Releasezyklus von vier auf eine Woche gesenkt wurde.

Die Schwierigkeit der Abwägung einer Lösungsalternative zeigte sich bei der Entwicklungsinfrastruktur. Die Einschränkungen in diesem Umfeld waren so gravierend und durch Scrum transparent geworden, dass das Team, das diesen Bereich bis dahin dediziert verwaltete, keine Chance hatte, die notwendigen Anpassungen vorzunehmen. Neben der Verantwortung, die zwar klar benannt war, fehlte dem Team jedoch die Möglichkeit, Einfluss auf Entscheidungen in diesem Umfeld zu nehmen und damit für eine nachhaltige Entwicklung zu sorgen. Die resultierende, undankbare Rolle des Diensterfüllers führte im Endeffekt dazu, dass das Team aufgelöst und die Verantwortung explizit an alle Entwicklungsteams übergeben wurde. In der Folge wurde das Thema stiefmütterlich behandelt und nur kleinere Änderungen durch die Scrum-Teams vorgenommen, die jedoch lediglich den Betrieb sicherstellten – das Resultat: ein Treten auf der Stelle.

Das, was bei ImmobilienScout24 nicht funktionierte, häufig jedoch in der Scrum-Literatur als Maß für eine erfolgreiche Scrum-Implementierung propagiert wird, ist die kontinuierliche Verbesserung der Velocity (umgesetzte Storypoints pro Sprint). Streng genommen sollte selbige bis zu einem bestimmten Level steigen. Die Gründe für das Nichteintreten waren vielfältig. Schwankungen in der Velocity wurden vor allem durch sich stetig verändernde Rahmenbedingungen (z.B. die Entwicklungsinfrastruktur) sowie unterschiedliche Teamgrößen (geschuldet durch z.B. Urlaub oder Krankheit) ausgelöst. Darüber hinaus veränderten Teams ihr Urmeter für die Schätzungen und bewerteten Stories geringer, wenn sie sich in einem Thema besser auskannten. Mittlerweile wird die Velocity nur noch für den Burndown-Chart verwendet, um eine ungefähre Releaseplanung durchzuführen. Das Management konzentriert sich auf andere Metriken zum Beispiel das Function-Point-Verfahren bzw. einfachere Statistiken wie Features pro Release oder die Buganzahl.

## 4.3 Das Review

Ein Jahr nach der Einführung war Scrum in der Softwareentwicklung fest verankert. Die Teams agierten selbstständig und sprachen auch Probleme an. Neben der gewonnenen Transparenz stellten sich erste Erfolge ein, unter anderem hatte sich die Entwicklungsleistung (gemessen an den Features pro Release) nahezu verdoppelt und die Anzahl der Fehler halbiert!

Vor diesem Hintergrund war nicht die Frage, ob Scrum das richtige Vorgehen ist, sondern wie das Vorgehen sinnvoll weiterentwickelt werden kann. Im Frühjahr 2010 lud ImmobilienScout24 Alan Atlas[1] zu einem Scrum-Review ein. Im Fokus stand neben der Bewertung der gegenwärtigen Scrum-Implementierung die Identifikation weiterer Potenziale. Alan Atlas begleitete den Scrum-Prozess eine

1. Alan Atlas ist ein Scrum-Guru, hat unter anderem Scrum bei Amazon eingeführt und ist einer von wenigen Certified Scrum Trainer.

Woche lang und führte eine Reihe von Gesprächen mit Teams, Product Ownern und dem Management.

Das Feedback war insgesamt positiv und umfasste auch die erhofften Verbesserungsvorschläge. Als wichtigsten Punkt merkte Alan Atlas an, dass der agile Gedanke außerhalb der Scrum-Teams nicht existiert. Vor allem vorgelagerte Phasen wie das grafische Design waren nicht Bestandteil des agilen Vorgehens und erfolgten in ihrem Ablauf sequenziell. Andere zentrale Punkte seines Resümees betrafen die Abhängigkeiten zwischen den Scrum-Teams und die Entwicklungsinfrastruktur. Ein explizites Lob entfiel auf die Integration externer Teams: »Outsourced Teams are in surprisingly good shape.«

Zur Überwindung der identifizierten Probleme führte Alan Atlas ein Paradigma an, das für die nachfolgende Entwicklung von Scrum bei ImmobilienScout24 bezeichnend sein sollte: »Inspect and Adapt«. Der Ansatz bezeichnet die Denkweise, dass alle Aspekte des agilen Vorgehens kontinuierlich hinterfragt und den Bedürfnissen angepasst werden sollten. Dabei geht es nicht darum abzukürzen, also den Anspruch zu senken, sondern sinnvolle Optimierungen zu identifizieren und Dogmatismus vorzubeugen.

## 4.4 Scrum 2.0

Motiviert durch das Review mit Alan Atlas war das Unternehmen bereit, die Scrum-Implementierung nachhaltig zu verbessern. Vonseiten des Managements wurde dafür der Begriff Scrum 2.0 geprägt. Die Einstufung zeigt, dass es nicht um kleinere Anpassungen ging, sondern darum, Scrum nach der Einführung nun individuell für ImmobilienScout24 zu adaptieren. Im Fokus standen zwei zentrale Ziele. Neben einer tieferen Integration von Scrum im Unternehmen sollte mehr Verantwortung in die Teams transferiert werden.

Um das agile Vorgehen im Unternehmen voranzutreiben, mussten dafür die bisher nicht berücksichtigten Abteilungen und deren Aufgaben in der Produktentwicklung in den Scrum-Prozess einbezogen werden. Die Scrum-Teams wurden deshalb um entsprechende Experten aus diesen Bereichen (u.a. User Experience (UX) und IT-Production) ergänzt. Das Ziel war es, Cross-funktionale Teams zu schaffen, die alle Aspekte der Produktentwicklung beherrschen und Verantwortung für ein (Teil-)Produkt übernehmen. Das entsprechende Team-Setup umfasst bei ImmobilienScout24

- Software Developer,
- Software Architect,
- Product Owner,
- QA (Quality Assurance) Engineer,
- UX Developer,
- Application Manager sowie
- ScrumMaster,

wobei die drei zuletzt genannten Rollen häufig durch Personen, die noch in einem weiteren Teams tätig sind, ausgeübt werden. Diese Situation ist, wie man sich vorstellen kann, nicht optimal, hilft jedoch die Aufgaben, die früher nicht Bestandteil des Sprints waren, innerhalb der Iterationen im Team umzusetzen.

Da mit Veränderungen häufig Ängste einhergehen, war die Transformation von fachlich- hin zu produktorientierten Teams nicht einfach und bedurfte viel Überzeugungsarbeit (bei den neuen Teammitgliedern und deren Management). Auch wenn nur mit Veränderungen die Chance besteht, etwas zu verbessern, wurde schnell klar, dass die Umstellung hin zu wirklichen Cross-Teams einen längeren Zeitraum in Anspruch nehmen wird. (In der Konsequenz existieren erst seit Anfang 2011 erste Cross-Teams bei ImmobilienScout24.) Parallel zur Etablierung dieses Team-Setups erfolgte ebenfalls die Trennung zwischen Teamleiter und ScrumMaster. Die Teamleiter, die im Rahmen der Scrum-Einführung die Rolle des ScrumMasters übernommen hatten, wurden sukzessive durch »Vollzeit-ScrumMaster« ersetzt und konnten sich wieder verstärkt anderen Themen zuwenden.

Ein anderer Aspekt von Scrum 2.0 war die Verantwortungsübernahme durch die Teams. Dieses Thema darf dabei nicht einseitig betrachtet werden. Damit Teams Verantwortung übernehmen können, müssen die Rahmenbedingungen durch das Management geschaffen werden. Eine wichtige Voraussetzung ist in diesem Zusammenhang die Selbstbestimmung. Damit Verantwortung sinnvoll übernommen werden kann, muss selbige im Vorfeld übergeben werden. Gerade bei der Einführung agiler Methoden stellen die Themen Selbstbestimmung und Verantwortung eine große Herausforderung dar. Da das Wasserfallmodell beide Aspekte einschränkt, war es nicht verwunderlich, dass die Teams ob der veränderten Rahmenbedingungen zögerlich reagierten.

Im ersten Schritt wurden das tägliche »Scrum of Scrums« und das gemeinsame Sprint Planning abgeschafft. Die Teams erhielten damit die Freiheit, die Sprint-Dauer zu variieren. Gleichzeitig ging die Koordination der »Support Requests« in deren Verantwortung über. Das eigentliche Ziel war jedoch, die Abhängigkeit zwischen den Teams zu reduzieren. Durch die stark strukturierte Planungsphase (sowohl auf Tages- wie auch Sprint-Basis) existierte bis dahin nicht die Notwendigkeit, das Problem der Abhängigkeiten anzugehen. Die Hoffnung war, dass die Teams unabhängiger agieren und Aufgaben, die früher die Unterstützung eines anderen Teams erforderlich machten, selbstständig lösen. Die notwendigen Unterstützungsleistungen reduzierten sich bereits nach kurzer Zeit deutlich und die Teams begannen auch dafür Verantwortung zu übernehmen.

Ein gutes Beispiel für die Reduzierung von Abhängigkeiten unter den Teams stellt das Thema »Multimedia« dar. Auch wenn es sich dabei um einen Querschnittsaspekt auf dem Internetportal von ImmobilienScout24 handelt, existierte nur bei einem Team das Wissen über die technische Konstruktion. Im Rahmen der Überarbeitung des Multimedia-Bereichs im ScoutManager, dem gewerblichen Anbieten-Bereich, wurde deshalb eine Taskforce aus den bestehenden Teams

gebildet, die gemeinsam die Änderungen vornahmen. Weitere Anpassungen konnten in der Folge dann selbstständig umgesetzt werden. Neben der Reduzierung der Abhängigkeiten fördert das Vorgehen auch den Abbau von Wissenssilos, die unter anderem der Grund für Ressourcenengpässe sind.

Neben der Verantwortung für fachliche Themen war auch eine übergreifende technische Verantwortung Ziel von Scrum 2.0. Die technische Plattform des Internetportals wurde über viele Jahre vernachlässigt, der Fokus lag auf der Produktentwicklung. Um die vorhandenen Mängel zu beheben, existierten zwei Szenarien, ein iteratives Vorgehen oder ein Stopp der Produktentwicklung. ImmobilienScout24 entschied sich für die erste Variante. Über einen Zeitraum von zwei Jahren (2009 – 2010) investierte das Unternehmen ein Drittel der IT-Ressourcen in die Restrukturierung der IT-Plattform parallel zur fortlaufenden Produktentwicklung. Im ersten Jahr waren dazu dedizierte User Stories mit rein technischen Anforderungen, die sich jedoch an den fachlichen Themen der Produktentwicklung orientierten, notwendig, um mit der Behebung der Altlasten zu starten. Beispielhaft wurden mit der fachlichen Überarbeitung des ScoutManager ein Standard-Webframework sowie ein neuer Persistenzmechanismus eingeführt. Die Art der Umsetzung oblag dabei dem Team und wurde nicht durch die Architekturabteilung vorgegeben, sondern nur durch eine Sammlung von Grundsätzen und Leitlinien flankiert. Im zweiten Jahr übernahmen die Teams selbstständig schrittweise die Verantwortung. In diesem Zusammenhang wurde das namentlich zum Unternehmen passende Scouty-Prinzip etabliert. Dieses besagt (in Anlehnung an das Pfadfinderprinzip von Robert Baden-Powell), dass man die Umgebung immer ein bisschen besser zurücklässt, als man sie vorfindet, und adressiert eine nachhaltige Entwicklung.

Auch die Probleme mit der Entwicklungsinfrastruktur wurden angegangen. Dabei ist jedoch anzumerken, dass Maßnahmen erst durch Initiative des Managements gestartet wurden. Trotz aller Bestrebungen, Teams mehr Verantwortung zu übergeben, wurde deutlich, dass die Klärung übergreifender Themen, die keinem Team explizit zugeschrieben sind, nach wie vor des Managements bedürfen.

## 4.5 Kanban & Co.

Neben Scrum hat ImmobilienScout24 auch erste Erfahrungen mit anderen agilen Vorgehensmodellen gesammelt. Fast unbemerkt hatte das Datenbankteam im Herbst 2009 Kanban eingeführt.

Auch wenn viele Datenbankaspekte in den Scrum-Teams gelöst werden, existiert zusätzlich ein kleines Team von Datenbankspezialisten, die das Thema übergreifend bearbeiten und für die Anbindung zum Data Warehouse und anderen Datenbanksystemen verantwortlich zeichnen.

Da die Arbeit des Teams vor allem durch operative Aufgaben geprägt ist, war ein auf mehrere Wochen festgelegtes Commitment selten verbindlich. Die Anfragen aus den anderen Teams waren schwer im Vorfeld abzuschätzen und hatten dann meistens eine hohe Dringlichkeit. Scrum funktionierte vor diesem Hintergrund nur bedingt. Auf eine Initiative des Teamleiters und ScrumMasters probierte das Team Kanban aus. Vor allem die Limitierung der Tasks, die gleichzeitig bearbeitet werden (»Work in Progress«), und die Abbildung der verschiedenen Systeme auf die Spalten am Storyboard halfen dem Team, sich zu organisieren und flexibel auf neue Anforderungen zu reagieren.

Im Sommer und Herbst 2010 folgten zwei weitere Teams, unter anderem jenes, mit dem die Scrum-Einführung gestartet hatte. Auch wenn die Initiative aus den Teams kam, waren die Gründe für den Wechsel unterschiedlich. Bei dem Team, das die Suche betreut, bestand die Intention darin, den Abstimmungsaufwand untereinander zu reduzieren. Aufgrund der Teamgröße (5 Personen) und der Tatsache, dass keine Abhängigkeiten zu anderen Teams bestanden, erschienen die Scrum-Rituale überdimensioniert. Bei dem anderen Team existierte ein hoher Anteil schwer planbarer Aufgaben, da mit der API ein Produkt betreut wurde, auf das zahlreiche andere Produkte aufsetzen. Anfragen von internen Teams und externen Softwaredienstleistern haben häufig einen hohen Stellenwert und müssen zeitnah bearbeitet werden. Trotz des Versuchs, mit Timeboxes die Zeit für entsprechende Aufgaben einzukalkulieren, gefährdete die Volatilität der Anfragen die Stabilität im Sprint.

Auch wenn vor allem aus Managementsicht mit Kanban häufig die Flucht vor dem Commitment assoziiert wird, lässt sich selbiges für ImmobilienScout24 nicht bestätigen. Bei den Teams, die von Scrum auf Kanban gewechselt sind, konnte keine Veränderung der Velocity (umgesetzte Storypoints in einem bestimmten Zeitraum) festgestellt werden. Vielmehr sticht die gewonnene Flexibilität positiv hervor. Konkret arbeiten die Teams an ein bis zwei User Stories gleichzeitig, aus denen Work-Items abgeleitet werden. Das Sprint Planning wurde gegen ein Story Planning ersetzt, das nach dem Review der vorherigen Story stattfindet. Dabei erhält der Product Owner eine ungefähre Angabe, wie lange die Umsetzung dauern wird. Eine Retrospektive wird alle 6–8 Wochen durchgeführt. Um zukünftige Themen durchzusprechen, wurde ein Backlog Grooming eingeführt. Im Endeffekt hat jedes Scrum-Meeting ein Pendant in der »Kanban-Welt«

erhalten, jedoch mit dem entscheidenden Vorteil der Flexibilität. Meetings finden nur statt wenn notwendig.

## 4.6 Retrospektive

Zurückblickend lässt sich aus vielerlei Hinsicht die Einführung von Scrum bei ImmobilienScout24 als Erfolg bezeichnen. Im Vergleich zum vorherigen Wasserfallmodell wurden zahlreiche Verbesserungen erzielt.

Die Produktentwicklung erfolgt in kurzen Iterationen und reduziert damit die Time-to-Market. Mitarbeiter sind zufriedener und motiviert, da sie die Möglichkeit haben, selbstbestimmt Verantwortung zu übernehmen und so für eine nachhaltige Produktentwicklung aus fachlicher wie auch technischer Sicht zu sorgen. Diese Übernahme im operativen Geschäft entlastet das Management und ermöglicht die Konzentration auf andere Themen. Probleme, wie die hohen Abhängigkeiten unter den Teams, wurden nachhaltig reduziert und haben ein skalierbares Entwicklungsumfeld geschaffen, in dem es ImmobilienScout24 möglich ist, neue Teams aufzubauen.

Die Zahl der Scrum-Teams hat sich mit 17 seit der Scrum-Einführung nahezu verdoppelt.

Nicht zuletzt hat sich auch die Produktivität verbessert. Es existieren verschiedene Auswertungen, die der Scrum-Einführung eine Produktivitätssteigerung bis um den Faktor vier bescheinigen. Entsprechende Aussagen sind natürlich dahingehend zu relativieren, dass nicht alle Aspekte berücksichtigt werden können. So hat ImmobilienScout24 parallel zur Scrum-Einführung enorme Anstrengungen unternommen, um die technische Plattform zu überarbeiten, die sicherlich ebenfalls in einer höheren Produktivität resultieren. Wichtig ist die Ableitung einer Tendenz, die im Falle von ImmobilienScout24 deutlich positiv ausfällt.

Der Weg zum Erfolg, sprich die Scrum-Einführung, war nicht immer einfach. Grundsätzlich darf Scrum nicht als Problemlöser verstanden werden. Es bezeichnet »lediglich« ein Vorgehen auf Basis klarer Regeln und Paradigmen, die Denk- und Verhaltensmuster verändern und den Pragmatismus fördern. Dies macht auch deutlich, dass Scrum nicht von heute auf morgen eingeführt werden kann, sondern ein langwieriger Transformationsprozess ist. Scrum stellt vor diesem Hintergrund eine tägliche Herausforderung dar, die Disziplin und den Willen, Dinge nachhaltig zu verbessern, erfordert. Wird das Vorgehen nicht richtig angewendet, wird sich auch keine Transparenz einstellen und es wird die Chance vergeben, Probleme anzugehen.

Zurückblickend hat es sich als richtig erwiesen, Scrum ohne größere Anpassungen einzuführen. Auch wenn die ersten Monate nach Lehrbuch teilweise etwas dogmatisch gewirkt haben, war es wichtig, das Vorgehen zu verinnerlichen, bevor individuelle Anpassungen vorgenommen wurden. Mit der Scrum-Einführung wurden viele Probleme sichtbar. Zweifelsohne wurden auch nicht alle

Impediments gelöst. Wichtiger ist es jedoch, die zentralen Probleme anzugehen und nachhaltig zu lösen. Unter anderem wurden die »Missstände« in der Entwicklungsinfrastruktur durch eine Initiative adressiert, die durch alle Scrum-Teams getragen wird. Über einen Zeitraum von einem Jahr erfolgte in kleinen Schritten die nahezu komplette Migration der Entwicklungsinfrastruktur.

## 4.7 Agile Werte im Projekt

| **Individuen und Interaktionen** | **vor** | **Prozessen und Tools** |
|---|---|---|
| ▲<br>Auch wenn Rahmenbedingungen z. B. zum Management-Reporting oder zur Projektzeiterfassung vorgegeben waren, wurde deren Einfluss auf die Teams minimiert. Vorhandene Prozesse wurden über die Zeit obsolet bzw. auf Initiative der Teams vereinfacht. Gleichzeitig zeichnen die Teams nun für die Entwicklungsumgebung und die dabei eingesetzten Tools verantwortlich.<br>Die Weiterentwicklung der Fähigkeiten in den Teams ist nach wie vor eine wesentliche Aufgabe des Managements. Neben klassischen Schulungen wurden auch neuere Formate wie Coders Dojo etabliert. | | |
| **Laufende Software** | **vor** | **Ausführlicher Dokumentation** |
| ▲<br>Ausgehend von veralteten Prozessdokumenten und Handlungsanweisungen ist der Entwicklungsprozess nun mit Praktiken wie Continuous Integration und Anstrengungen hin zu einer Continuous Delivery Chain klar auf laufende Software ausgerichtet. Die Dokumentation ist dabei minimal auf einen Issue-Tracker sowie ein Wiki begrenzt. | | |
| **Zusammenarbeit mit dem Kunden** | **vor** | **Vertragsverhandlungen** |
| ▲<br>Die Einführung von Scrum hat die Zusammenarbeit zwischen der IT und dem Produktmanagement deutlich verbessert. Cross-funktionale Teams, kurze Iterationen und damit schnelleres Feedback sind geschätzte Bestandteile der Produktentwicklung. Mit der Abstimmung von Abhängigkeiten sowie der Priorisierung von Themen existieren jedoch immer noch Herausforderungen. | | |

Henning Wolf, Agile Projekte mit Scrum, XP und Kanban , dpunkt.verlag, ISBN 978-3-86490-266-6

# 5 (Fast) agil in einem Großunternehmen

Alex Bepple · Sven Günther · Henning Wolf

*Gleich vorweggenommen: Wir sind nicht richtig zufrieden mit der Einführung von Agilität in dieser Organisation geworden, auch wenn sich ein paar Dinge zum Guten gewandelt haben. Wir wissen heute aber, was wir wohl falsch gemacht haben und was wir heute tun würden, damit es besser läuft.*

*Wir haben uns trotzdem entschieden, eine solche nicht ganz so erfolgreiche Einführung zu beschreiben, weil sich bekanntlich gerade aus Fehlern besonders gut lernen lässt.*

## 5.1 Der Kunde

Unser Kunde war ein großes Unternehmen mit über 100 Einzelfirmen und mehr als 50.000 Mitarbeitern weltweit. Das Produkt, das agil entwickelt werden sollte, war ein Backend-Logistiksystem, das mehr als eine Million Aufträge pro Tag abwickelte. Die Weiterentwicklung des Produkts zielte nicht nur darauf ab, neue fachliche Anforderungen umzusetzen, sondern auch darauf, die noch auf dem Host laufenden Teile des Systems auf Java zu migrieren.

Wir waren am deutschen Hauptstandort tätig, wo auch der größte Teil der Entwicklungsmannschaft saß. Insgesamt arbeiteten an dem Produkt rund 100 Entwickler an drei Standorten: ungefähr 60 am Hauptstandort und jeweils 15 bis 25 an zwei weiteren deutschen Standorten.

Die Fachlichkeit des Produkts war in vier Themen eingeteilt. Jeweils eine Abteilung des IT-Bereichs war für ein Thema zuständig. Jede Abteilung hatte mehrere Entwicklungsteams. Die disziplinarische Abteilungsleitung, die Business-Analysten sowie ein Team jeder Abteilung arbeiteten am Hauptstandort. Jede Abteilung hatte mindestens ein Team an einem Standort in einer anderen Stadt.

Die Fach- und die IT-Abteilungen am Hauptstandort waren in verschiedenen Gebäuden in Gehreichweite voneinander untergebracht. Die Business-Analysten und die Entwicklungsteams vor Ort teilten sich je Abteilung ein Großraumbüro.

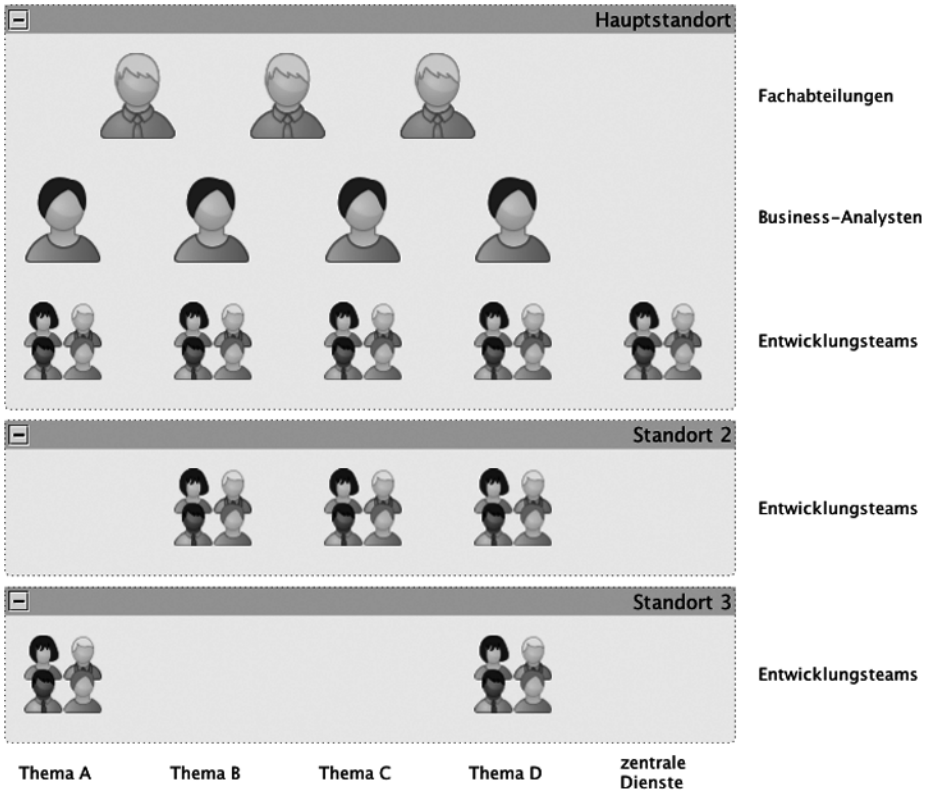


***Abb. 5–1*** *Organisation der Entwicklungsabteilung*

Neben den Fachteams gab es ein Team für sogenannte zentrale Dienste wie Datenbankadministration, Build-Automatisierung und Continuous-Integration-Server.

## 5.2 Ausgangssituation

Unser Kunde hatte sich entschlossen, den Entwicklungsprozess agiler zu gestalten. Nach eigenen Vorarbeiten holte der Verantwortliche uns als Berater mit ins Boot, um ihn bei der Definition und Einführung des neuen Entwicklungsprozesses zu unterstützen.

Folgende Probleme wurden identifiziert:

- Seltene Releases:
  Statt der geplanten zwei bis drei großen Releases pro Jahr gab es nur ein bis zwei. Das letzte aktuelle Release hatte 13 Monate gedauert.
- Lange Time-to-Market:
  Von der Abgabe des Fachbereichskonzepts an die IT bis zum Rollout verging mehr als ein Jahr.

- Geringe Flexibilität:
  Späte und spätere Änderungen an den Konzepten durch die Fachbereiche waren sehr teuer und führten auf IT- und Fachseite zu hoher Unzufriedenheit.
- Abhängigkeiten:
  Hohe technische und fachliche Abhängigkeiten der Projekte untereinander waren schwer zu handhaben und führten zu häufigen Verzögerungen und Blockaden von Projekten.
- Big-Bang-Integration:
  Die Integration der einzelnen (Sub-)Projekte ins Release erfolgte erst ganz am Ende und bedurfte eines sehr hohen Aufwands.
- Späte Abnahmetests:
  Ein Gesamttest am Ende benötigte sehr hohen manuellen Aufwand und deckte für gewöhnlich viele Probleme auf, die parallel zum Testen (und der Entwicklung am nächsten Release) noch gefixt werden mussten.

## 5.3 Das neue Liefermodell

In Anlehnung an andere agile Vorgehensweisen (insbesondere Scrum) und auf der Grundlage der Idee, vertikale Schnitte durchs System zu bauen (also letztlich Features), entstand ein neues Prozessmodell für die Softwareentwicklung. Unser Kunde nannte es Liefermodell.

Im ersten Schritt schien es damals nötig, dass es sich nicht um ein Standardverfahren handeln durfte, sondern zwingend eine firmenspezifische Anpassung. Wir sind uns heute nicht mehr ganz sicher, auf Basis welcher Hypothese diese Entscheidung getroffen wurde, unsere Hauptfavoriten sind:

- Die Vermutung war, dass nur so möglichst viele der vorhandenen Probleme gelöst werden konnten.
- Der Wunsch war, dass so mit möglichst wenig späteren Veränderungen gleich ein passender Prozess mit möglichst geringem Einführungsaufwand erstellt werden könnte.

Im Folgenden ist der Kern des neuen Liefermodells beschrieben.

### 5.3.1 Vorphase eines Projekts

Seine Entwicklungsvorhaben bereitete unser Kunde nach dem neuen Liefermodell genau so wie früher vor.

Die Anforderungen des Fachbereichs wurden in einem Fachbereichskonzept beschrieben. Das Fachbereichskonzept diente dazu, grob abzuschätzen, wie die Anforderungen umgesetzt werden sollten, welche Systeme beteiligt waren und welche fachlichen Abhängigkeiten es zu anderen Projekten gab. Die Business-

Analysten der IT erstellten dann ein IT-Konzept. Dieses beschrieb in Form von Features die Anforderungen, die einzeln grob geschätzt wurden. Auf dieser Basis unterbreitete die IT der Fachabteilung ein Festpreisangebot.

Ein weiteres Ergebnis des IT-Konzepts war ein grober Architekturentwurf. Dieser war notwendig, um die Einbettung des Projekts in die Systemlandschaft des Unternehmens sicherzustellen.

### 5.3.2 Projektablauf

Die nach fachlichem Nutzen priorisierten Features wurden in Iterationen umgesetzt. In einer »Exploration« genannten Iteration 0 wurden die Stories für die Umsetzung in der ersten Iteration von den Business-Analysten tiefer gehender analysiert. Weiterhin wurden Testfälle spezifiziert, die am Ende der Iteration zur Abnahme der Stories dienten.

Die Entwickler schätzten die Stories nach Zerlegung in Tasks und planten diese in die Iterationen ein. Die Entwicklung erfolgte in zweiwöchigen Iterationen. Hier wurde das technische Design im Detail festgelegt und die Stories umgesetzt.

Zur Definition of Done gehörte die Umsetzung der Stories inklusive entsprechender Unit Tests und automatisierter Akzeptanztests. Innerhalb eines Produkts sollten die verschiedenen Projekte früh integriert werden.

Alle acht Wochen wurde aus dem Entwicklungsbranch ein Releasebranch erzeugt. Dieser Releasebranch eines Produkts sollte mit den jeweils aktuellen Versionen der beteiligten externen Produkte integriert und übergreifend getestet werden. Diese Tests prüften die fachliche und technische Integration in die aktuelle Systemlandschaft sowie die Performanz und Stabilität unter Betriebsbedingungen. Aufgrund dieser Testergebnisse erfolgte die Abnahme des Release durch die Fachbereiche und durch den Betrieb. Dieser Testlauf umfasste weitere acht Wochen.

## 5.4 Vorgehen bei der Einführung

Um das neue Liefermodell einzuführen, wurde ein Mitarbeiter des Unternehmens bestimmt. Dieser entwarf gemeinsam mit einem von uns den Prozess, der sich an verschiedene agile Prozesse aus der Literatur anlehnte (z.B. Scrum, Feature Driven Development – FDD). Ziel dieses Entwurfs war es, diesen Prozess langsam einzuführen und die Projektentwicklung nicht zu gefährden. So wurden viele Begriffe, die im bisherigen Vorgehensmodell verwendet wurden, weiter benutzt und die bestehenden Rollen nur wenig angepasst.

Bei der Umsetzung wurde nach dem Zwiebelmodell vorgegangen: viele kleine Schritte von innen nach außen. Innen entsprach in dem Fall der Kernmannschaft der Entwickler und Business-Analysten. Die Schicht ganz außen entsprach dem

Fachbereich, den jeweiligen Kunden. Damit sollte erreicht werden, dass die Mannschaft selbst mit dem neuen Prozess umgehen kann und erste Erfolge verbucht hat, bevor dies nach außen umgestellt werden sollte. Insbesondere wollten wir so verlorenes Vertrauen wiedergewinnen und nichts versprechen, was später nicht gehalten werden konnte. Historisch hatte es in den letzten Jahren den Fachbereichen gegenüber schon einige Versprechen gegeben, dass neue Technologien und Vorgehensweisen alle Probleme lösen würden. Eingehalten wurden diese Versprechen nicht.

Die Mitarbeiter wurden in mehreren Trainings auf ihre neue Arbeitsweise vorbereitet. Dabei sind wir auch hier wieder schrittweise vorgegangen, indem zuerst die Lead Developer und Meinungsführer der einzelnen Teams geschult wurden. Diese sollten dann ihr Wissen in den jeweiligen Teams verbreiten. In nachfolgenden Phasen wurden dann die weiteren Mitarbeiter geschult.

Die Fachbereiche wurden von dem für die Einführung verantwortlichen Mitarbeiter über die Umstellung des Entwicklungsprozesses früh informiert, ohne dass sie jedoch schon ihre Arbeitsweise umstellen mussten.

Der neue Entwicklungsprozess wurde nicht sofort auf alle Projekte ausgerollt. Stattdessen wurde ein kleines Pilotprojekt ausgewählt, in dem ein Team das neue Vorgehen erprobte. Dies zeigte u. a., dass die gewählte und von vielen vorab mit Skepsis bedachte Sprint-Länge von zwei Wochen ausreichend war. Es gab aber noch viele Schwierigkeiten, kleine unabhängige Stories zu schreiben. Problematisch war auch, dass alle Mitarbeiter keine Vollzeitmitglieder des Pilotteams waren.

Richtig ernst wurde es erst mit dem nächsten Projekt. Dieses war zum einen ein sehr wichtiges Projekt, bei dem der Einführungstermin aus firmenpolitischen Gründen nicht gefährdet werden durfte, und zum anderen eines, an dem mehr als ein Fachteam beteiligt war. Hier konnten viele Erfahrungen aus dem Pilotprojekt einfließen. So wurde erstmals ein Projektteam zusammengestellt aus Mitarbeitern der verschiedenen beteiligten Entwicklungsabteilungen. Weiterhin wurde festgelegt, dass sehr früh eine Minimalfunktionalität ausgeliefert werden sollte, mit der der Kunde seine angepassten Prozesse testen konnte. Dieses Projekt konnte pünktlich fertiggestellt werden.

Nach diesem Projekt wurden die anderen beteiligten Entwicklungsabteilungen auf das neue Entwicklungsvorgehen umgestellt. Dabei wurde den laufenden Projekten freigestellt, ob sie auf das neue Vorgehen umstellen wollten. Neue Projekte sollten aber in jedem Fall nach dem neuen Entwicklungsprozess umgesetzt werden.

Zum abteilungsübergreifenden Erfahrungsaustausch trafen sich die ScrumMaster der einzelnen Teams regelmäßig. Sie einigten sich z. B. nach kurzer Zeit darauf, die ScrumMaster über Teamgrenzen hinweg zu tauschen. Damit sollten die ScrumMaster mehr Objektivität gewinnen und die Teams so besser unterstützen können.

Eine weitere Baustelle waren die fachlichen und betrieblichen Abnahmetests. Diese dauerten im bisherigen Vorgehen zwischen drei und sechs Monaten. Das war nicht mehr angemessen, weil wir ja alle acht Wochen ein neues Release produktiv stellen wollten. Um dieses Testvorgehen neu aufzusetzen, wurde ein Team zusammengestellt. Dieses Team bestand aus verschiedenen am Test beteiligten Stakeholdern.

## 5.5 Verbesserungen

Die beiden sichtbarsten Veränderungen unter unserer Begleitung waren die Erhöhung der Releasefrequenz sowie eine Verbesserung der Zusammenarbeit in den Teams.

Allen Schwierigkeiten zum Trotz nahm unser Kunde alle acht Wochen eine neue Version der Software in Betrieb – und machte die wohlbekannten Erfahrungen: Die mit einem einzelnen Release verbundenen Schmerzen wurden kleiner. Die Integration der Einzelentwicklungen, bevor das Produkt in die Testphase übergeben wurde, fiel leichter. Die Abnahmetests gingen schneller von der Hand und deckten weniger Fehler auf. Die Fehler waren zudem leichter zu beheben. Schließlich war die Inbetriebnahme selbst zügiger und mit weniger Problemen erledigt. Kurzum: Ein Release war kein Großereignis mehr. Das bedeutete zwar auch, dass es weniger zu feiern gab nach einer durchgemachten Nacht. Vor allem bedeutete es aber weniger Stress.

Zweitens verbesserte sich die Zusammenarbeit in den Teams am Hauptstandort erheblich. Während alle Teams zu Beginn unserer Beratung dazu neigten, die Stories eines Sprints im Voraus einzelnen Teammitgliedern zuzuweisen, veränderte sich dies nach und nach. So wies sich ein Teammitglied eine Story später nur zu, wenn die anderen bereits in Arbeit waren. Besser noch: Teammitglieder arbeiteten später häufig zusammen an Stories und halfen bei Stories, die bereits in Arbeit waren, wenn sie mit ihrer Aufgabe fertig waren. Zwei Teams haben häufig, obwohl unregelmäßig, in Paaren gearbeitet.

In einem Team führte die enge Zusammenarbeit zu einem bemerkenswert gleichmäßigen Flow auf dem Scrum-Board und einer geringen Anzahl von Stories in Arbeit. Insgesamt verbesserte sich die Zusammenarbeit im Team bei drei von vier Teams am Hauptstandort erheblich. In diesen Teams nivellierte sich das Wissensgefälle sowohl fachlich als auch technisch. Diese Teams beobachteten in Folge weniger Engpässe bei der Umsetzung von Stories und eine geringere individuelle Belastung. Die Teammitglieder unterstützten einander effektiv. Insgesamt haben wir bei diesen Teams auch eine bessere Stimmung sowie eine höhere Arbeitsmotivation wahrgenommen. Das Team mit dem Coach verbesserte seine Zusammenarbeit am meisten.


***Abb. 5–2*** *Gute Teamarbeit: Karten fließen gleichmäßig über das Board*

Indem die Teammitglieder eng zusammenarbeiteten, »commiteten« sie häufiger und stellten so die Integration des Codes innerhalb der jeweiligen Komponente sicher. Sie reduzierten so die Integrationsrisiken für das Gesamtprodukt.

Neben diesen beiden wichtigsten haben wir auch andere Verbesserungen gesehen.

Alle Teams am Hauptstandort haben im Sprint-Takt Retrospektiven durchgeführt und sie genutzt, um sich kontinuierlich zu verbessern. Insbesondere ein Team nutzte die Retrospektiven sehr konsequent, um Störungen im Arbeitsfluss zu beseitigen. Zum Beispiel war der Architekt für die vorgeschriebenen Codereviews oft nicht verfügbar. Das Team einigte sich mit ihm darauf, dass die erfahrenen Entwickler innerhalb des Teams die Codereviews durchführen würden. Ganz nebenbei unterstützten die internen Reviews die Bildung gemeinsamer Arbeitsstandards, die durch die engere Teamarbeit notwendig wurden. Früh schlug dieses Team auch vor, die Rechte für Änderungen am Datenbankschema aus dem Team »zentrale Dienste« in die anderen Teams zu verlagern und so zu dezentralisieren, um Wartezeiten bei Schemaänderungen zu vermeiden. Später unterstützten andere Teams den Vorschlag. Mit einiger Verzögerung wurde er umgesetzt. Diese Beispiele machen deutlich, dass die Teams sich kontinuierlich verbesserten und damit eine der wichtigsten agilen Feedbackschleifen nutzten.

Alle Teams am Hauptstandort interessierten sich stärker als zuvor für automatisierte Tests, insbesondere Mikrotests. In einem Team kam dieses Interesse besonders stark zum Ausdruck. Mit Unterstützung des Coachs übte das Team testgetriebene Entwicklung im Alltag und in Programmier-Dojos. Es behielt diese Praktiken auch bei, nachdem der Coach das Team verlassen hatte.

Auf unsere Empfehlung hin wurden die automatisierten Tests in die Continuous-Integration-(CI-)Läufe der Einzelkomponenten integriert. Der CI-Lauf war nur erfolgreich, wenn auch alle Tests liefen. Die Tests dauerhaft grün zu halten und jeden Fehlschlag als Ausnahme anzusehen, war eine zunächst ungewohnte, aber schließlich befriedigende Erfahrung für die Teams.

Parallel dazu etablierten wir Build-Ampeln, um die Sichtbarkeit der Ergebnisse der Continuous-Integration-Läufe zu erhöhen. In allen Teams haben wir in Folge ein höheres Bewusstsein für Qualität und für die Konsequenzen des eigenen Handelns wahrgenommen.

Einige Teams haben schließlich positive Erfahrungen mit vertikalen, fachlichen Schnitten als Entwicklungseinheiten gesammelt. Sie haben vor allem die Möglichkeit geschätzt, die Grundfunktionalität bereits umsetzen zu können, während viele Details noch unbekannt waren. Einige wenige Product Owner haben User Stories zudem als Möglichkeit gesehen, Termintreue zu gewährleisten, indem sie die unverzichtbaren Bestandteile zuerst und weniger wichtige erst später umsetzen lassen. Es war uns damit gelungen, einige zarte Pflänzchen agiler Produktentwicklung hochzuziehen.

## 5.6 Schwierigkeiten mit dem neuen Liefermodell

Bei der Umsetzung dieses Vorgehens gab es aber auch etliche Schwierigkeiten.

Es gab eine Unmenge an Entwicklungsvorhaben, die auf der gleichen Codebasis von den verschiedenen Entwicklungsteams umgesetzt werden sollten. Diese Projekte beschränkten sich dabei nicht auf die Codebasis, die ein Team jeweils für sich betreute, sondern zogen sich quer durch das gesamte Produkt. Die Projekte waren jeweils dem Team zugeordnet, das die Hauptarbeit zuliefern konnte, benötigten aber auch Zuarbeiten von anderen Teams. Da es keine übergreifende Abstimmung und Priorisierung der Entwicklungsvorhaben gab, fiel es den Business-Analysten oft schwer, die Anforderungen an ihre Teams zu priorisieren. In Folge warteten die Teams mehrfach auf Zulieferungen.

Des Weiteren gab es Qualitätsprobleme, da die verschiedenen Projekte nicht das Gesamtprodukt im Fokus hatten. Widersprüchliche Anforderungen aus verschiedenen Projekten wurden meist sehr spät erkannt. Die Projekte wurden vorwiegend erst in der Endphase in das Gesamtprodukt integriert. Dadurch gab es häufig Probleme beim Build und bei der übergreifenden Qualität.

Die Systemlandschaft des Kunden war sehr komplex. Um die Anforderungen End-to-End testen zu können, wurde diese Systemlandschaft für den Test nachgebaut. Aufgrund des geringen Automatisierungsgrads der Tests war der Testablauf sehr aufwendig durchzuführen und somit fehleranfällig.

Das Team zur Verkürzung der Testzeiten hatte viele hohe Erwartungen bei gleichzeitig unklarer Zielvorgabe zu erfüllen. Es war vielen schwer zu vermitteln, dass die Qualität des Systems nicht hinterher hineingetestet werden konnte, sondern die Tests und Qualitätsansprüche schon sehr früh in der Entwicklung erfüllt werden müssen. So konnte man sich lediglich auf einen Kompromiss einigen, der eine Testphase von acht Wochen vorsah.

## 5.7 Lernerfahrungen

Im Folgenden geben wir einige aus unserer Sicht markante Lernerfahrungen aus dieser agilen Transition wieder.

### 5.7.1 Lernerfahrung: Zwischenziele mit dem Management vereinbaren

Die Ziele für das neue Liefermodell waren laut verschiedener Präsentationsstände in etwa die folgenden:

- Häufigere Releases, kürzerer Releasetakt
- Bessere Qualität liefern
- Tests effektiver durchführen
- Entwicklung kostengünstiger gestalten
- Schnellere Reaktion auf Änderungen ermöglichen

- Geplante (und zugesagte) Termine zuverlässig einhalten
- Personal gleichmäßiger auslasten

Tatsächlich ist es uns nicht gelungen, sie mit dem Prozessverantwortlichen und dem Management in eine Reihenfolge zu bringen, die das Management mitgetragen hätte. Gerade dies wäre wichtig gewesen, um Rückendeckung für Veränderungen über einen langen Zeitraum zu bekommen. Stattdessen gab es von Anfang an eine sehr hohe Erwartung an das neue Liefermodell, die dieses nicht sofort erfüllen konnte. Folgerichtig hat der Prozessverantwortliche im Management die Unterstützung verloren, die agile Transition voranzutreiben.

### 5.7.2 Lernerfahrung: Product Owner ermächtigen

Wir haben die Business-Analysten in den Abteilungen zu Product Owner gemacht, ohne sie mit dem Recht auszustatten, Produktentscheidungen zu treffen. In der Tendenz sind die Product Owner damit Anforderungsverwalter geblieben. Sie hatten in den meisten Fällen nicht das Recht, den funktionalen Umfang des Systems zu ändern. Das erstellte IT-Konzept musste wie früher auch vollständig umgesetzt werden. Durch diese Orientierung an der Spezifikation verloren die Product Owner den Anreiz, Anforderungen nach Geschäftswert zu priorisieren, neue hinzuzunehmen und weniger wertvolle auch wieder aus dem Product Backlog zu entfernen. Am Ende musste die Bestellung der Fachabteilung eben doch vollständig erfüllt werden.

Wir haben auch an der Zusammenarbeit der Product Owner mit den Fachabteilungen, den internen Kunden, nichts geändert. Sie war also weiterhin von umfangreichen, weit vor der Umsetzung erstellten IT-Konzepten geprägt, die in erster Linie Lösungen beschrieben, aber die Kundenbedürfnisse nicht in den Vordergrund stellten. Diese Art der Zusammenarbeit verstärkte die Bestellmentalität in den Teams: »Wir liefern, was bestellt ist.« Gewünscht hätten wir uns: »Wir lösen eure Probleme.«

Diese Krankheitsbilder haben wir zunächst bewusst in Kauf genommen, weil wir unseren Schwerpunkt zu Beginn auf die Arbeitsorganisation in den Teams gelegt hatten. Wir haben uns allerdings dabei verschätzt, wie sehr wir damit den agilen Veränderungsbestrebungen die Triebkraft entziehen würden. Agile Methoden und Praktiken speisen sich aus dem Bestreben, erfolgreiche Produkte zu entwickeln. Indem wir die Product Owner in den Teams über lange Zeit als Spezifikationsverwalter gewähren ließen, versiegte nach und nach die wichtigste Quelle für Veränderungsbemühungen: die Motivation, das Produkt immer besser zu machen. Denn je länger die Business-Analysten die Rolle des Product Owner als die eines Spezifikationsverwalters in der Linie lebten, während die Etiketten »agil« und »Scrum« schon darauf klebten, desto schwieriger wurde es, hier eine grundsätzliche Veränderung der Mentalität zu erreichen. So ging verloren, dass

ein Product Owner deutlich mehr ist als ein Fachexperte, der die Anforderungen im User-Story-Format »Als X möchte ich Y, um Z zu erreichen« aufschreibt.

Immerhin gab es einige Business-Analysten, die sich die Freiheit des Empowerments herausgenommen haben und mit ihren Teams positive Erfahrungen sammeln konnten, dass dies auch möglich ist. Dieses Vorgehen hat sich aber nicht durchgesetzt.

### 5.7.3 Lernerfahrung: Schlechte Qualität macht langsam

Das entwickelte System litt zu Beginn der agilen Einführung an einer geringen Qualität. Das wurde sowohl an der generellen Bewertung aller Beteiligten deutlich als auch an dem Stellenwert, den die Analyse und Behebung von Problemen in der täglichen Arbeit der Teammitglieder einnahm. Zu beinahe jedem Zeitpunkt war mindestens ein Teammitglied damit beschäftigt. Solche Arbeiten häuften sich, wenn die separate Qualitätssicherungsabteilung gerade dabei war, den zuletzt in den Test übergebenen Softwarestand zu überprüfen. Dann gingen die Teams aus Erfahrung ein geringeres Commitment ein.

Bereits vor dem Test durch die Qualitätssicherungsabteilung machte sich eine zu schwache Definition of Done der Teams dadurch bemerkbar, dass es schwer fiel, einen Gesamtstand des Systems auch nur zu kompilieren, um ihn testen zu können. Ursächlich dafür war die Kombination aus der gewählten Versionsverwaltung für den Code, den langen Build-Laufzeiten und dem mangelnden Verantwortungsbewusstsein für den integrierten Code-Stand.

In der Entwicklung kam eine Toolsuite von Telelogic zum Einsatz. Sie war als organisationsweiter Standard gesetzt. Diese Suite bestand aus Telelogic Synergy für die Versionskontrolle und Telelogic Change, einem Tracking Tool für Change Requests (CRs). Die Telelogic-Suite wurde so konfiguriert, dass es möglich war, eine bestimmte Version der Software zu bauen, die aus ausgewählten CRs besteht. Das aufgesetzte Continuous Integration hat dann immer verschiedene Builds gebaut, die den verschiedenen Zuständen aller CRs entsprach (all resolved CR, all approved CR, all tested CR usw.). Daraus ergaben sich eine Vielzahl von Builds, deren Erfolgsquote nur mäßig war. Bei Buildbreakern ergab sich dann unter vielen Entwicklern recht schnell eine »Egal-Haltung«, weil diese Builds schwer nachvollziehbar waren und durch Setzen eines Status am CR sich wieder fixen ließen. Um diese Builds dann zu testen, war der Aufwand nochmals deutlich größer, sodass es nur bei Unit Tests und rudimentären Integrationstests für eines der vielen verschiedenen Builds blieb.

Die Laufzeit eines kompletten Builds des gesamten Produkts war selbst ohne Tests länger als eine Stunde, sodass die Entwickler sich lokal mit einem Build ihrer aktuellen Komponente zufriedengaben. Da Änderungen an den Komponentenschnittstellen relativ einfach einzuführen waren, gab es dadurch regelmäßig

Buildbreaker in den abhängigen Komponenten und die Bereitschaft zum Fixen dieser Buildbreaker war eher gering.

Da es im alten Prozess immer sehr aufwendig war, alle Softwarezulieferungen zu integrieren, entschied man sich damals, diese Integration sehr spät vorzunehmen. Aufgrund der bis dahin angesammelten Änderungen in den einzelnen Komponenten konnte diese Integrationsphase durchaus auch zwei Wochen dauern. Die Einsicht, dass eine frühe und häufige Integration bei diesen Problemen Abhilfe schafft, kam auch nach der Umstellung auf agile Entwicklung nur sehr langsam bei allen Entwicklern an und behinderte die qualitätsgerechte Lieferung der Software.

### 5.7.4 Lernerfahrung: Radikale versus inkrementelle Prozessinnovation

Das neue Liefermodell unseres Kunden war eine Mischung aus alt und neu. Elemente agiler Vorgehensmodelle waren bunt gemischt mit bisherigen Rollen und Praktiken. So hat unser Kunde die Zuordnung von Teams zu Abteilungen nicht angetastet. Er hat auch nichts daran geändert, dass die Releasekonfiguration erst im Nachhinein, nach Ablauf mehrerer Sprints erfolgte. Beibehalten hat er auch die Trennung zwischen Entwicklungs- und Testphase.

Viele etablierte Rollen und Praktiken beizubehalten, erschwerte es allen Beteiligten, sich auf die Prinzipien und Werte agiler Softwareentwicklung einzulassen. So blieben unserem Kunden viele vertraute, aber mit agilem Arbeiten inkompatible Denkweisen und Handlungsabläufe erhalten. Wir vermuten, dass es uns eher gelungen wäre, einen nachhalten Wandel zu erreichen, wenn wir zügig einen agilen Prozessrahmen, zum Beispiel Scrum, nach Lehrbuch eingeführt hätten. Das hätte uns erlaubt, die große Veränderungsenergie zu Beginn der Transition zu nutzen und die erhebliche Belastung für die Organisation durch viele Veränderungen über einen langen Zeitraum einzudämmen. Indem wir als Berater wiederholt darauf hingewiesen haben, die Organisation wäre noch nicht agil genug, haben wir ihr nicht nur immer weitere Veränderungskosten aufgebürdet, sondern auch das jeweils Erreichte entwertet.

Die konsequente Einführung eines Prozessrahmens, der wie Scrum durchgängig auf kurze Feedbackschleifen setzt, hätte auch viele strukturelle Probleme früher transparent gemacht. Aufgrund der langen Feedbackschleifen bei unserem Kunden blieben sie verborgen.

## 5.8 Fazit

Wir haben zwar schon etwas verändert, aber unserem eigenen Anspruch in diesem Projekt nicht entsprochen.

Wir haben die Mitarbeit engagierter Entwickler erlebt, vieler kompetenter Analysten und aufgeschlossener Abteilungs- und Bereichsleiter. Trotzdem ist uns bis zum Zeitpunkt der Entscheidung, SAP einzuführen, also in ungefähr einem Jahr der Beratung beim Kunden nicht mehr gelungen. Das liegt vor allem an uns als den Beratern und vielleicht auch daran, dass wir nicht mehr Zeit hatten. Sicherlich lag es auch an der vom Kunden so gewählten Beratungsintensität und der Vorstellung, man könnte die Veränderung komplett »störungsfrei« nebenbei bewerkstelligen (auf die wir uns nicht erneut einlassen würden).

Es könnte aber auch sein, dass mehr gar nicht möglich war und man für bestimmte Kunden und Konstellationen tatsächlich den langen Atem braucht (wenn man nicht alternativ ausreichend Druck hat, um bestimmte Veränderungen schneller durchzuführen).

Wir hoffen, dass nicht nur wir in diesem Projekt viel gelernt haben, sondern trotz unserer Unzufriedenheit auch der Kunde und seine Mitarbeiter einiges mitnehmen konnten. Nach letzten Informationen aus dem Projekt sind doch einige der von uns eingeführten Vorgehensweisen auch heute noch fester Bestandteil des Arbeitens.

## 5.9 Agile Werte in der Produktorganisation

| **Individuen und Interaktionen** | **vor** | **Prozessen und Tools** |
|---|---|---|
| ▲<br>Das Umfeld erwartet strikt zu befolgende Prozesse und Prozesshandbücher – zumindest formal. Tatsächlich wurden auch bisher Projekte pragmatisch gehandhabt (solange alles gut lief). Es gab aber durchaus eine Bereitschaft, an diesem Wertpaar zu arbeiten. | | |
| **Laufende Software** | **vor** | **Ausführlicher Dokumentation** |
| ▲<br>Aufgrund der Größe des Systems und der vielen Beteiligten wurde durchaus vernünftig auch an Dokumentation gedacht. Manches davon wurde allerdings im Vorwege so erstellt, als wäre es ein nur noch von Entwicklern auszuführender Plan, der oftmals so nicht hinkam. Zudem fiel es mit der Erstellung von Dokumenten manchmal leichter, eine lokale Fertigstellung zu argumentieren, statt durch den oft fehlschlagenden Gesamtbuild tatsächlich laufende Software zu erstellen.<br>Auf der anderen Seite wurde die Dokumentation nur oberflächlich überprüft und die Wertschätzung erfolgt für tatsächlich laufende Software. | | |
| **Zusammenarbeit mit dem Kunden** | **vor** | **Vertragsverhandlungen** |
| ▲<br>Auch wenn das Management der IT sehr bemüht war, einen formalen Vertrag und Verbindlichkeit im Vorhinein mit den Fachbereichen zu erlangen, wurde letztlich in den intensiven und ausgedehnten Testphasen das entwickelt, was der Kunde braucht (und nicht das, was in einem Konzept steht). | | |

| Reagieren auf Veränderungen | vor | Befolgen eines Plans |
|---|---|---|

▲

Vielleicht wären wir hier weiter gekommen, wenn wir die Einbindung der Fachbereiche erreicht hätten. Tatsächlich war aber das Herausbringen von Releases mit versprochenem, zugesagtem Inhalt nach Plan das Hauptbestreben der IT-Organisation. Durch den Druck in den Testphasen wurde dann auf den »Reagieren auf Veränderungen«-Modus geschaltet. Da dieser aber mit hohem Druck einherging, träumte man weiter von sich erfüllenden Plänen.

# 6 Zurück auf die agile Spur

## Vom Micromanagement zu Scrum

Sven Röpstorff

*Ein Unternehmen aus der Telekommunikationsbranche sucht die Unterstützung eines Coachs, weil sich der durch die Einführung agiler Vorgehensweisen erhoffte Projekterfolg nicht wie erwartet eingestellt hat. Obwohl viele Voraussetzungen geschaffen wurden, arbeitet kein Team wirklich agil. Dieser Beitrag beschreibt ein konkretes Projekt, dessen Ziel es ist, ein Webportal für eine neue Mobilfunkmarke zu entwickeln.*

## 6.1 Die Ausgangssituation

Ein großer, sehr hierarchisch aufgebauter Konzern aus der Telekommunikationsbranche hatte eine eigene Forschungs- und Entwicklungsabteilung mit ca. 200 Mitarbeitern. Seitens des obersten Managements gab es ein klares Commitment zu agilen Vorgehensweisen. Es wurde ein Team aufgesetzt, das die Agilisierung steuern und begleiten sollte. Dieses Team bestand aus fünf Personen, die allerdings alle aus einem traditionellen Projektumfeld kamen und bisher Projektaudits durchgeführt hatten. Dieses Quintett war hinsichtlich agiler Vorgehensweisen durch Schulungen externer Koryphäen hervorragend ausgebildet und hatte sehr gutes internes Lehr- und Trainingsmaterial erstellt, mit dem sie nach und nach alle Kollegen schulten. Die Voraussetzungen klangen also auf den ersten Blick gar nicht schlecht. Beim näheren Hinsehen ergaben sich aber an mehreren Stellen Handlungsbedarf:

- Das Team war zu klein, um sowohl Ausbildung, Fortbildung, Coaching und operative Projektarbeit zu leisten.
- Nur zwei der fünf Teammitglieder arbeiteten vor Ort, drei weitere hatten ihren Arbeitsplatz an einem entfernten Standort.
- Es fehlte allen Teammitgliedern an praktischer Erfahrung in agilen Projekten. Der theoretische Hintergrund war zwar ausgezeichnet, aber sie konnten keine eigenen Erfahrungen in die Projektteams tragen. Und obwohl alle sich sehr bemühten, den Projekten als Coach und Berater zur Verfügung zu stehen,

wurden sie von vielen Kollegen weiterhin als Projektauditoren wahrgenommen.
- Niemand aus dem Team hatte vorher als Entwickler gearbeitet, sodass Wissen über Methoden wie eXtreme Programming (XP) nur in der Theorie vorhanden war. Dies machte es schwierig, Entwickler von den Vorteilen des Pair Programming oder Test Driven Development zu überzeugen.

Das Unternehmen hatte diese Handlungsfelder erkannt und sich dazu entschlossen, für einen Zeitraum von drei Monaten externe Experten zurate zu ziehen. Das Expertenteam bestand aus drei Personen:

- ein XP-Coach, den man als Hardliner bezeichnen konnte und der XP am liebsten in seiner Reinform benutzte,
- ein »Hybride«, der jahrelange Erfahrung sowohl als Entwickler als auch als agiler Coach mitbrachte,
- ich – ein agiler Coach, ausgestattet mit Erfahrung sowohl bei der Einführung von Scrum als auch in der operativen Arbeit als ScrumMaster und Product Owner sowie mit Trainings- und Workshop-Erfahrung.

## 6.2 Die Mission

Unser Kernauftrag umfasste folgende Tätigkeiten:

- Verbesserung der Qualifikation der fachlichen und technischen Teams durch Hands-on-Training und -Coaching,
- bei Bedarf Übernahme der notwendigen operativen Rollen (ScrumMaster, Product Owner, Softwarearchitekt, Entwickler ...),
- Schulung der agilen Produktentwicklung und -lieferung,
- Nachweis, dass agile Vorgehensweisen auch in diesem Haus funktionieren,
- Unterstützung bei der Einführung von XP-Praktiken,
- Unterstützung bei der Einführung und beim Coaching von Scrum,
- nachweisbare Verbesserung der Softwarequalität,
- Schaffen einer agilen Start-up-Kultur.

Uns war natürlich sofort klar, dass diese Aufgabenstellung zu dritt für eine Abteilung von mehr als 200 Personen in einem traditionell und hierarchisch geführten Unternehmen nicht innerhalb von drei Monaten zu erfüllen war. Glücklicherweise sah unser Kunde das genauso und wir vereinbarten drei laufende Projekte, mit denen wir intensiv an den oben genannten Zielen arbeiten wollten.

Im Folgenden werde ich mich in meiner Darstellung auf das Projekt beschränken, für das ich die Verantwortung übernommen habe.

## 6.3 Das konkrete Projekt

An meinem Einsatzort gab es ein Team, das im Rahmen eines Einführungsprojekts für eine neue Mobilfunkmarke ein Webportal erstellen sollte. Das Projekt war vor 14 Monaten gestartet, die Entwicklung begann vier Monate nach Projektstart und nach sieben Monaten Entwicklungszeit (also elf Monate nach Projektstart) fand der erste große Launch statt.

Der Kunde des Teams war ein Spin-off einer Tochterfirma und hatte seinen Sitz in London. Das Webportal war das wichtigste von drei Zugangswegen für die Endkunden. Hinter diesen Zugängen verbarg sich eine hochkomplexe technische Infrastruktur, die bis in die Systemtiefen eines Mobilfunkanbieters reichte und von mehreren Drittanbietern ergänzt wurde.

Das Team bestand zum aktuellen Zeitpunkt aus 15 Mitarbeitern und einem Projektmanager; ein Teammitglied arbeitete an einem anderen Standort.

## 6.4 Los geht's

Um mir einen Überblick über das Projekt zu verschaffen, habe ich mich zunächst mit dem lokalen Projektleiter, dem Systemarchitekten und zwei Kollegen aus dem internen Coaching-Team zusammengesetzt. Zur Vorbereitung des Gesprächs hatte ich einen Fragenkatalog erstellt und an die beiden Kollegen verschickt, damit wir fokussiert an den wichtigsten Fragen arbeiten konnten.

---

### Fragenkatalog für das Überblicksgespräch

#### Eckdaten

- Wer ist der Auftraggeber?
- Wer ist der Endkunde/Benutzer?
- Wer sind die Stakeholder?
- Könnt ihr mir bitte einen kurzen Überblick über die Historie des Projekts geben?

#### Aktueller Prozess

- Gibt es einen Product Owner? Wenn ja, wer ist es?
- Gibt es einen ScrumMaster? Wenn ja, wer ist es?
- Wer sind die Teammitglieder? Welche Rollen haben sie (Entwickler, Business-Analyst, Designer, Architekt, Qualitätssicherung ...)?
- Sitzen die Teammitglieder zusammen oder sind sie räumlich verteilt?
- Ist den einzelnen Teammitgliedern ihre jeweilige Rolle bewusst?
- Werden die Rollen hinreichend gelebt?

- Gibt es ein Product Backlog?
- Wurde das Product Backlog geschätzt? Wenn ja, wer hat es geschätzt?
- Wie lang ist ein Sprint?
- Wie lange dauern die Meetings (Estimation, Sprint Planning, Daily Scrum, Review, Retrospektive)?
- Finden die genannten Meetings regelmäßig statt?
- Gibt es einen Releaseplan?
- Gibt es Rahmenbedingungen wie z.B. fixe Meilensteine?
- Welches sind eurer Meinung nach die größten Probleme?
- Habt ihr Vorschläge, wie man diese Probleme lösen könnte?

#### Erfahrung mit agilen Vorgehensweisen

- Kennen sich alle beteiligten Personen zumindest grundlegend mit Scrum aus?
- Kennen alle beteiligten Personen die Werte, Rollen, Meetings und Artefakte von Scrum?

#### Sonstiges

- Gibt es für unsere Zusammenarbeit MUSTs oder MUSTNOTs?

## 6.5 Erste Erkenntnisse

Das Team vor Ort war zu Entwicklungsbeginn agil (bzw. als das, was man darunter verstanden hat) gestartet, dann aber unter dem Druck des anstehenden Release zwei Monate vor der Markteinführung wieder zu einer klassischen Command-and-Control-Struktur zurückgekehrt. Der Product Owner hatte wieder die Rolle eines traditionellen Projektmanagers eingenommen und den Teammitgliedern dedizierte Aufgaben zugeordnet. Später stellte sich heraus, dass der vermeintliche Product Owner eher ein Product Owner Proxy (Stellvertreter) war, der Anforderungen des Kunden aus London in konkrete Arbeitsanweisungen für das Team umsetzte. Im Folgenden spreche ich vom »lokalen« Product Owner, wenn ich den Product Owner Proxy vor Ort meine.

Nach Aussage des lokalen Product Owner ist die Situation ins Chaos abgeglitten. Das Release wurde zwar mit Ach und Krach geschafft, aber die letzten Wochen vor der Markteinführung waren geprägt von Überstunden und Wochenendarbeit. Erstaunlicherweise sind nach dem Launch keine nennenswerten Probleme aufgetreten. Offensichtlich hatte das Team zu Recht den Ruf, aus den besten Entwicklern des Unternehmens zu bestehen. Nach dem sehr anstrengenden Start mit hoher Belastung aller Beteiligten hatte das Management Angst, dass die Entwickler aufgrund der stressigen Situation das Unternehmen verlassen würden.

Um nicht immer mit allen 15 Teammitgliedern direkt kommunizieren zu müssen, ernannte der lokale Product Owner inoffizielle Leiter für die Teilbereiche Frontend-Programmierung, Backend-Programmierung, Qualitätssicherung und Integration. Diese vier Personen waren seine einzigen Ansprechpartner und verantwortlich dafür, Informationen und Aufgaben weiterzuverteilen.

Das vorhandene »Product Backlog« war ein Sammelsurium aus einer Excel-Liste und PowerPoint-Dokumenten. Es gab keine einheitliche Quelle mit verständlichen Anforderungen in Form von User Stories, keine Akzeptanzkriterien, keine Schätzungen und erst recht keine Priorisierung.

Der Kunde in London hatte kaum Kontakt zum Team und konnte mit dem Begriff »Agilität« auch wenig anfangen. Neue Anforderungen wurden unkontrolliert und spontan von unterschiedlichen Personen gestellt, häufig auch am lokalen Product Owner vorbei direkt bei den Entwicklern. Zudem kam es vor, dass Anforderungen doppelt eingereicht wurden oder sich sogar widersprachen, weil die Mitarbeiter des Auftraggebers sich nicht abstimmten. Der extremste Fall war eine neue Anforderung, die morgens telefonisch gestellt wurde und sich im Laufe des Tages viermal komplett änderte, sodass der bereits erstellte Code jedes Mal verworfen wurde.

Etwa zwei Monate vor Beginn meines Einsatzes ist der bisherige Hauptansprechpartner (der eigentliche Product Owner) in London auf den Posten des CTO gewechselt und es wurde ein Program Manager eingestellt, der seine Aufgaben als Product Owner übernehmen sollte. Allerdings sah sich der Program Manager eher als technischer Releasemanager, sodass weiterhin der CTO als Ansprechpartner fungierte. Aufgrund der neu hinzugekommenen Aufgaben konnte er dies aber nicht mehr mit der gewohnten Intensität und Qualität erledigen.

Die technische Infrastruktur des Gesamtsystems lag komplett außerhalb des Entwicklungsteams. Da das entwickelte Webportal aber den Hauptzugangsweg für Endkunden darstellte, tauchten sämtliche Fehler aus den Tiefen des Gesamtsystems letztendlich an dieser Oberfläche auf. Dies führte oft zu aufwendigen Recherchen und Analysen im Team, in deren Folge immer wieder festgestellt wurde, dass die Ursache in einem Fremdsystem lag.

Iterationen in Form von Sprints gab es schon lange nicht mehr. Es wurde immer auf das nächste Release hingearbeitet, dessen Termin und Umfang vorgegeben waren. Der Umfang war allerdings dynamisch, sodass immer mehr wichtige Dinge hinzukamen, ohne weniger wichtige Features zu streichen oder auf spätere Releases zu verschieben. Keines der Standard-Scrum-Meetings fand statt.

Obwohl es im Team offiziell drei Tester gab, beschränkte sich deren Aufgabe hauptsächlich darauf, Protokolle aus dem Produktivsystem auszuwerten, entsprechende Fehlerreports zu generieren und diese an die Entwickler zur Bearbeitung weiterzuleiten. Auch hier gab es keine Priorisierung, es sollte einfach alles erledigt werden. Kurz vor einem Release wurde begonnen, hektisch zu testen. Ein echter funktionaler Test fand aber erst beim Kunden in London statt, da dort die Koor-

dination mit den Betreibern der Drittsysteme erfolgte. Von echtem »Shippable Code« im Sinne von Scrum konnte also keine Rede sein.

Es gab früher einen ScrumMaster, der in einer Doppelrolle auch als Architekt arbeitete und vor einiger Zeit frustriert das Handtuch geworfen hatte. Aktuell arbeitete das Team ohne einen ScrumMaster. Einen guten Einblick in das Scrum-Verständnis der Beteiligten gab an diesem Punkt die folgende Nachfrage: »Was genau ist eigentlich die Aufgabe des ScrumMasters? Koordiniert er nur die Arbeitspakete oder macht er auch Codereviews?« Ich habe an dieser Stelle nachgehakt und festgestellt, dass trotz der im Unternehmen vorhandenen hervorragenden theoretischen Grundlagen und Schulungen das Verständnis von Agilität im Allgemeinen und Scrum im Besonderen nicht besonders ausgeprägt war.

Eines der Drittsysteme brauchte für die Umsetzung neuer Anforderungen eine Vorlaufzeit von mindestens sechs Wochen, tendenziell eher mehr. Änderungswünsche innerhalb dieser Zeit führten zu einem Neubeginn des 6-Wochen-Zeitraums.

Meine Empfehlung, den Entwicklungsprozess des Gesamtsystems zu analysieren und zu verbessern, wurde mit Hinweis auf die Verantwortlichkeit des Kunden und den vertraglich eingeschränkten Umfang des zu liefernden Webportals abgelehnt. Einerseits sicher nachvollziehbar, andererseits muss man sich dann aber darüber im Klaren sein, dass man in diesem Fall auch nur einen Teilnutzen aus einer agilen Vorgehensweise ziehen kann.

So viel also zur Ausgangssituation. Harter Stoff. Meine Aufgabe als agiler Coach war es nun also, unter Berücksichtigung des oben formulierten Kernauftrags aus dieser Gemengelage eine agile Einheit zu formen und mich baldmöglichst wieder entbehrlich zu machen.

## 6.6 Analyse

Gemeinsam mit dem lokalen Product Owner habe ich beschlossen, zunächst persönliche Gespräche mit den Teammitgliedern zu führen, um deren Perspektive kennenzulernen und mein Bild von der Situation abzurunden. Für diese Gespräche habe ich mir ein paar Fragen überlegt und bin diesen Fragebogen mit den Kollegen durchgegangen, um eine Vergleichbarkeit der Gespräche zu gewährleisten.

### Fragebogen für die Kennenlerngespräche mit dem Team

- Wer bist du und was ist dein fachlicher Hintergrund?
- Was tust du bzw. dein Subteam im Projekt? Wie interagierst du mit den anderen Mitgliedern des Projektteams?
- Kannst du mir bitte einen typischen Arbeitstag beschreiben?
- Bitte beschreibe mir in einem kurzen Satz, was Agilität für dich bedeutet. Was bedeutet Scrum für dich? Was bedeutet XP für dich?
- Hast du Interesse daran, agil zu arbeiten? Wenn ja, warum? Wenn nein, warum nicht?
- Bis du ausreichend geschult worden, sodass du die Vorgehensweise von Scrum verstehst?
- Fühlst du dich in diesem Projekt wohl?
- Lass uns ein Daily Scrum durchführen:
  - Was hast du seit gestern erreicht (gemacht)?
  - Was willst du bis morgen erreichen (machen)?
  - Gibt es irgendetwas, was dich daran hindert, deine Arbeit so effektiv wie möglich zu erledigen?
- Weißt du, was die anderen Teammitglieder heute machen? Oder was sie gestern gemacht haben?
- Woran erkennst du, dass ihr vorankommt?
- Woran erkennst du, ob etwas fertig ist?
- Was sind im Moment die größten Hindernisse für deine Arbeit?
- Was sind deiner Meinung nach die drei größten Hindernisse für das Projekt?
- Wenn du bis zu drei Dinge ändern könntest, was wäre das?
- Gibt es sonst noch etwas, was du ansprechen oder wissen möchtest?

Bei der Auswertung der Gespräche machte ich ein paar interessante Feststellungen:

- Niemand aus dem Team hatte die internen Schulungen besucht.
- Niemand aus dem Team konnte die Begriffe »XP«, »Scrum« und »Agilität« auseinanderhalten oder gar erklären.
- Trotz der angespannten Situation vor der Markteinführung fühlten sich alle Teammitglieder wohl im Team. Dies lag an dem guten Zusammenhalt und dem sehr angesehenen und kollegial agierenden lokalen Product Owner.
- Keines der Teammitglieder konnte genau sagen, woran die anderen gerade arbeiteten. Sogar innerhalb der Subteams war die Ahnung nur vage.
- Ein Taskboard gab es nicht, kommuniziert wurde über das Bugtracking-System Jira.
- Akzeptanzkriterien oder eine Definition of Done gab es nicht.

Mit diesen Ergebnissen habe ich mich wieder mit den Projektverantwortlichen zusammengesetzt. Normalerweise hätte man das Projekt stoppen und nach einem Grundlagentraining für alle Beteiligten (inklusive der Kundenvertreter) komplett neu aufsetzen müssen. Da es sich aber um ein Projekt für einen externen Kunden handelte, das nächste große Release schon wieder vor der Tür stand und das Management große Erwartungen hatte, wurde diese Option von vornherein ausgeschlossen. Um trotzdem handeln zu können, haben wir gemeinsam einige Maßnahmen beschlossen.

Zunächst sollte eine Retrospektive mit dem Team über die letzten Monate seit der Markteinführung durchgeführt werden, damit sich die Teammitglieder miteinander über das Projekt und ihre Zusammenarbeit unterhalten und Verbesserungsvorschläge einbringen können, statt auf das Management zu warten.

Als Nächstes wollte ich einen ganztägigen Scrum-Grundlagen-Workshop (zunächst mit allen Beteiligten vor Ort) durchführen, um sicherzustellen, dass alle das gleiche Verständnis von den Begrifflichkeiten und der agilen Vorgehensweise haben.

Weiterhin sahen wir es als dringend notwendig an, dass ich schnellstmöglich Kontakt zum Kunden des Teams aufnahm, um auch hier die Grundlagen zu vermitteln und die Vorteile der gewählten Vorgehensweise aufzuzeigen. Besonders wichtig würde es sein, den Kunden davon zu überzeugen, dem Team planerisch immer ein bis zwei Sprints voraus zu sein und das Team während eines Sprints in Ruhe arbeiten zu lassen.

Das Teammitglied, das an einem anderen Standort arbeitete, sollte aus dem Team genommen werden, da der Kontakt zum Team bisher eher sporadisch war und der erheblich höhere Kommunikationsaufwand in keinem Verhältnis zu dem ins Team eingebrachten Nutzen stand.

Ein Kick-off für den Start der neuen Vorgehensweise sollte schnellstmöglich durchgeführt werden. Dazu wurden die Teilteams (Frontend, Backend, Qualitätssicherung, Integration) aufgelöst und ein einziges Scrum-Team gebildet. Die Sprint-Länge sollte gemeinsam mit dem Team festgelegt, die Definition of Done bestimmt und die Scrum-Rollen geklärt werden. Um sicherzugehen, dass das Team die neu erlernte Vorgehensweise auch korrekt umsetzt, sollte ich für die nächsten zwei bis drei Sprints als ScrumMaster fungieren. Dabei würde ich ein Teammitglied als ScrumMaster ausbilden, sodass er die Rolle nach dieser Übergangszeit übernehmen kann.

Der lokale Product Owner sollte wieder als solcher denn als Projektmanager fungieren. Außerdem wurde er aufgefordert, die bestehende Excel-Liste und die PowerPoint-Dokumente schnellstmöglich in einen Zustand zu bringen, den man guten Gewissens als Product Backlog bezeichnen kann, sodass basierend darauf ein Estimation Meeting durchgeführt werden konnte.

Die oben beschriebenen Schritte haben wir innerhalb einer Woche durchgeführt (Ausnahme: Einbeziehung des Kunden in London), sodass wir schnell mit

dem ersten offiziellen Sprint starten konnten. Den Kunden haben wir darüber informiert, dass wir unsere Arbeitsweise zu seinem Nutzen ändern werden. Des Weiteren haben wir ihn – mit Hinweis auf zeitnahe Einbindung in die Vorgehensweise – gebeten, uns eine Chance zu geben und das Team für die Dauer des Sprints in Ruhe arbeiten zu lassen.

## 6.7 Der erste Sprint

Nach dem Scrum-Grundlagen-Workshop war das Team zunächst noch sehr skeptisch. Die meisten Teammitglieder hatten noch den Misserfolg vom ersten Versuch im Kopf. Jetzt kam jemand von außerhalb und wollte ihnen zeigen, wie sie es besser machen konnten. Immerhin konnte ich sie überzeugen, der vorgeschlagenen Vorgehensweise eine ernsthafte Chance zu geben.

## 6.8 Estimation

Nachdem der lokale Product Owner aus den Anforderungsfragmenten ein einziges Product Backlog erstellt hatte, haben wir ein Estimation Meeting angesetzt. Doch bevor wir starten konnten, sind wir noch über ein anderes Problem gestolpert: Pünktlichkeit. Ich hatte das Thema Timeboxing im Scrum-Grundlagen-Workshop hinreichend strapaziert und jetzt ergab sich die Gelegenheit zu zeigen, dass es wirklich ernst gemeint war.

Fünf Minuten nach dem offiziellen Beginn des Meetings fehlten leider immer noch zwei Kollegen, während alle anderen mich erwartungsvoll ansahen. Ich habe die Gelegenheit genutzt, sie darauf hinzuweisen, dass Verspätungen immer zulasten der Gruppe gehen und dass wir nicht eher anfangen werden, bis auch der letzte Kollege erschienen ist. Nachdem dann tatsächlich alle Kollegen da waren, hat der Product Owner sich ziemlich deutlich dahingehend geäußert, dass er diese Verspätungen ziemlich respektlos gegenüber den pünktlichen Kollegen findet und dass er erwartet, dass zukünftig alle Kollegen pünktlich erscheinen. Diese Ansage muss tatsächlich etwas bewirkt haben, denn die latente Komm-ich-heute-nicht-komm-ich-morgen-Mentalität hat sich in den folgenden Wochen signifikant verbessert.

Als wir dann endlich loslegen konnten, haben wir ungefähr 60 Backlog Items in 54 Minuten geschätzt. Das Team hatte viel Spaß dabei und wir haben drei oder vier Backlog Items identifiziert, bei denen sich das Team nicht einigen konnte. Die Tatsache, dass sie diese Items als »ungenügend vorbereitet« zurückgeben durften, hat die Teammitglieder sehr beeindruckt, denn im bisherigen Gehorsam-Schema gab es diese Form der Beteiligung nicht.

## 6.9 Sprint Planning

Am nächsten Tag haben wir das erste Sprint Planning durchgeführt. Auch hier kam es sehr positiv bei den Teammitgliedern an, dass sie nach ihrer Einschätzung gefragt wurden und dass sie entscheiden durften, was in den nächsten zwei Wochen umgesetzt werden sollte. Der Product Owner, der es gewohnt war, dem Team Arbeit zuzuweisen (und das nicht zu knapp), musste sich merklich zurückhalten, aber er hat das Meeting bravourös überstanden. Im zweiten Teil ging es dann an den Aufbau des Taskboards. Die Arbeit bestand im Wesentlich darin, die ausgewählten Backlog Items in Aufgaben mit einer vernünftigen Granularität zu zerlegen, was ohne größere Herausforderungen gelang.

## 6.10 Daily Scrum

Richtig spannend wurde es noch einmal bei den Daily Scrums in den nächsten Tagen. Es wurde »Ich bin ein Scrum-Zombie« gespielt. Was das heißt? Das Teammitglied, das gerade an der Reihe war, hat die drei Fragen des Daily Scrum folgendermaßen beantwortet: »Gestern habe ich *nuschelnuschelnuschel*, heute werde ich *nuschelnuschelnuschel*, Impediments habe ich keine.« Um dies zu vermeiden, haben wir festgelegt, dass jedes Teammitglied nach vorn an das Taskboard gehen, auf die gerade angesprochene Taskkarte zeigen und in Richtung des Teams sprechen soll. Wenn man die Aufforderung alle zwei Tage wiederholte, klappte das auch.

Die nächste Herausforderung bestand darin, das Daily Scrum nicht zu einem Reporting für den Product Owner werden zu lassen, sondern zu einer Synchronisierung für das Team. Bisher waren die Teammitglieder es gewöhnt, immer an den lokalen Product Owner zu berichten. Da es diesem sowieso schon schwerfiel, aus seinen gewohnten Command-and-Control-Strukturen auszubrechen, nahm er das Daily Scrum dankend zum Anlass, Detailfragen zu stellen, Lösungen vorzuschlagen und Aufgaben zu verteilen – ein absolutes No-Go. Gleich nach dem ersten Daily Scrum habe ich ihn gebeten, sich zukünftig zusammen mit mir im Hintergrund zu halten und jede Form des Reportings an ihn sofort mit dem Hinweis auf das Team als Adressaten zu unterbinden. Er war sehr überrascht, weil ihm seine Dominanz überhaupt nicht bewusst war. Er gelobte Besserung, was ihm in den nächsten Wochen auch recht gut gelang. Nur selten musste ich ihn freundschaftlich am Hemdzipfel zurück aus dem Teamkreis zerren.

Entgegen meiner Bitte, die Backlog Items von oben nach unten abzuarbeiten und immer gemeinsam an genau einem Item zu arbeiten, suchten sich die Teammitglieder immer Aufgaben verschiedener Backlog Items aus und bearbeiteten diese parallel. Die Begründungen dafür waren vielfältig. Es gab beispielsweise einen Python-Spezialisten, der nichts anderes als Python-Skripte bearbeitete. Wenn nun die ersten sechs Items nichts mit Python zu tun hatten, hat er eben am siebten Item gearbeitet. Allerdings hat das Team seine Sprint-Ziele trotz dieser

Vorgehensweise geschafft, weil es die Backlog Items in den Sprint Plannings unter diesen Gesichtspunkten ausgewählt hat.

## 6.11 Sprint-Verlauf

Es fiel dem Team sichtlich schwer, aus dem Denkmuster der Spezialisierung auszubrechen und gemeinsam an Themen zu arbeiten. Die Teamgröße von 14 Teammitgliedern erschwerte diese Arbeitsweise zusätzlich. Allerdings war zu beobachten, dass der gute Wille vorhanden war und beispielsweise Frontend- und Backend-Entwickler zunehmend gemeinsam an ihrem Code saßen, um Parameterübergaben abzustimmen oder früh zu testen, ob der Durchstich tatsächlich funktionierte. Vorher hatte es dies nicht gegeben, man unterhielt sich über die Kommentarfunktion im Ticketsystem. Dank der täglichen Synchronisation in den Daily Scrums und der konsequenten Verwendung von Scrum hat das Team die Spur halten können und das Sprint-Ziel des ersten Sprints erreicht. Gegen Ende wurde es doch noch etwas hektisch, als das Team realisierte, dass es im Review tatsächlich etwas Lauffähiges zeigen sollte, anstatt nur darüber zu reden, aber auch das hat letztendlich geklappt.

## 6.12 Review

Das erste Review war für alle Beteiligten eine Offenbarung. Der Kunde aus London wurde über Screen-Sharing und Telefonkonferenz eingebunden und vorher informiert, was ihn erwarten würde. Das Team war unendlich stolz, dass es das Commitment aus dem Sprint Planning gehalten hatte und tatsächlich lauffähige Software zeigen konnte. Der Kunde war beeindruckt, dass in so kurzer Zeit tatsächlich so viel Funktionierendes geschaffen wurde. Der lokale Product Owner war glücklich, weil sein bisheriger Arbeitsaufwand deutlich zurückgegangen war, und ich habe mich gefreut, dass alles so gut geklappt hat. Kurzum, alle waren zufrieden.

## 6.13 Retrospektive

Die erste Retrospektive haben wir nach dem klassischen Muster durchgeführt:

- Storytelling
- Was lief gut?
- Was sollte verbessert werden?
- Was können wir als Team dafür tun?
- Was soll die Organisation für uns tun?

Es kamen bis auf eine Ausnahme keine besonderen Punkte zur Sprache, die man nicht aus anderen Erst-Retrospektiven kennt. Viele Teams kreisen anfangs um

sich selbst und versuchen, ihre internen Abläufe zu verbessern. Dieses Team hat sich jedoch bereits zu diesem Zeitpunkt Gedanken darüber gemacht, dass eine bessere Vorbereitung und ein besseres Prozessverständnis des Kunden in London die Arbeit des Teams immens erleichtern würde. Daher hat das Team eine Aufgabe für die Organisation definiert: »Workshop mit dem Kunden durchführen, um ihm unsere Arbeitsweise nahezubringen und ihm die Vorteile für ihn und uns aufzuzeigen, wenn er uns darin durch entsprechende Vorbereitung unterstützt.« Dies ist genau der Punkt, den wir vor dem Start des Sprints als dringend angesehen hatten. Nun wurde die Dringlichkeit vom Team noch untermauert. Ich bin in der folgenden Woche für ein paar Tage nach London geflogen, um vor Ort mit den Mitarbeitern des Kunden zu arbeiten.

## 6.14 Ein paar Wochen später ...

Nachdem ich in den ersten beiden Sprints als ScrumMaster im Projekt mitgewirkt hatte, habe ich die Rolle für den dritten Sprint an einen Kollegen aus dem Team übergeben, den ich bereits seit dem ersten Sprint als zukünftigen ScrumMaster aufgebaut hatte. Damit habe ich mich operativ aus dem Projekt zurückgezogen und mich nur noch auf das Coaching und gelegentliches Korrigieren beschränkt.

Gemeinsam mit dem lokalen Product Owner und dem Kunden in London haben wir das anfangs recht rudimentäre Product Backlog immer weiter verbessert. Für eine größtmögliche Sichtbarkeit für alle Beteiligten haben wir es in Google Docs eingepflegt. Außerdem haben wir sukzessive Akzeptanzkriterien eingeführt, die das gemeinsame Verständnis verbesserten und für bessere Schätzungen sorgten. Zunächst haben wir dies jeweils für die im nächsten Sprint vorgesehenen Backlog Items getan, später auch für alle neu erstellten Backlog Items.

Als die organisatorischen Voraussetzungen geschaffen und der Entwicklungsprozess optimiert waren, konnten wir uns auch dem XP-Teil aus unserer obigen Aufgabenbeschreibung widmen. Nach dem dritten Sprint ist der XP-Spezialist aus unserem Expertenteam als Entwickler mit ins Team gekommen und hat aktiv mit dem Team XP-Praktiken geübt, allen voran Test Driven Development.

Mein Engagement in diesem Projekt hat etwa neun Wochen gedauert. Erstaunlicherweise habe ich wenig direktes Feedback aus dem Team bekommen, obwohl ich explizit danach gefragt habe. Ich vermute, dass ich in diesem doch sehr hierarchisch organisierten Unternehmen als eine Art Führungskraft angesehen wurde, die man nicht kritisiert (und schon gar nicht lobt). Ich habe jedoch über Umwege erfahren, dass das Team aufgrund des misslungenen ersten Versuchs vor einigen Monaten anfangs sehr skeptisch war, inzwischen die neue Vorgehensweise aber völlig begeistert adaptiert hat. Dies deckte sich mit den Schwingungen, die ich auch während der täglichen Arbeit mit dem Team und insbesondere während der Retrospektiven wahrgenommen habe. Das schönste Feedback habe ich gegen Ende unseres Auftrags direkt bekommen, und zwar vom lokalen Product Owner.

Er sagte zu mir: »*Wenn ich mein nächstes Projekt starte, dann werde ich es genauso aufsetzen und durchführen, wie du es uns beigebracht hast.*«

## 6.15 Ein paar Monate später ...

Ich habe ein halbes Jahr später noch einmal nachgefragt, wie das Projekt läuft und ob das Team weiterhin nach Scrum arbeitet. Erfreulicherweise wurden – auch ohne meinen direkten Einfluss – immer noch Daily Scrums und Retrospektiven durchgeführt. Außerdem arbeitete das Team nach wie vor mit einem physischen Taskboard und arbeitete die Backlog Items von oben nach unten ab. Aufseiten des Kunden haben sich einige Veränderungen ergeben, wichtige Know-how-Träger haben das Unternehmen verlassen. Dadurch geraten die Anforderungen wieder durcheinander und zusätzlich gibt es wohl Probleme mit den Betreibern der Drittsysteme.

## 6.16 Fazit

Die Arbeit mit dem Team war auch für mich sehr interessant und lehrreich. Stünde ich heute wieder vor der gleichen Ausgangssituation, würde ich das meiste vermutlich genauso machen, ein paar Dinge aber doch anders gewichten und angehen.

Es war zum Beispiel genau richtig, sich am Anfang genügend Zeit für eine Analyse der Situation und für persönliche Gespräche mit allen Beteiligten zu nehmen. Als externer Coach ist man oft dem Druck ausgesetzt, schnell Ergebnisse liefern zu müssen, um die Kosten für den eigenen Einsatz zu rechtfertigen. Ich bezweifle, dass das Ergebnis so gut gewesen wäre, wenn ich die Analyse nicht so gründlich durchgeführt hätte.

Ein Punkt, der sich auch in diesem Projekt wieder bewährt hat, ist die konsequente Einhaltung der Rollen. Der Product Owner darf dem ScrumMaster genauso wenig dessen Kompetenzen und Verantwortlichkeiten streitig machen wie umgekehrt. Das Team soll sich um die Umsetzung der Backlog Items kümmern und die Beseitigung von Impediments dem ScrumMaster überlassen.

Wichtig und richtig war es auch, das Team nicht gleichzeitig mit Scrum und XP zu konfrontieren. Wir haben zuerst eine geschützte Atmosphäre geschaffen, bevor wir in die Details der technischen Umsetzung gegangen sind. Wir haben uns diese Vorgehensweise als das Aufspannen eines Regenschirms (»Scrum-Umbrella«) vorgestellt, unter dem das Team geschützt arbeiten konnte und von dem alle störenden »Geschosse« abprallten.

Ändern würde ich beim nächsten Mal die Teamgröße. 14 Teammitglieder sind meines Erachtens zu viel, insbesondere dann, wenn nicht alle wertschöpfend für das Sprint-Ziel arbeiten. Für dieses Projekt wären 8-9 Teammitglieder angemessen gewesen, wenn man das oben erwähnte Analysieren der Produktionspro-

tokolle konsequent aus dem Team verbannt hätte und die Verantwortung für die vom Team durchzuführenden Tests und für die funktionalen Abnahmetests beim Kunden klarer geregelt hätte.

Im Nachhinein denke ich manchmal darüber nach, ob nicht auch Kanban eine Option gewesen wäre. Der Kunde hatte explizit Scrum gefordert und auch die Trainings der Organisation darauf ausgerichtet, sodass der Vorschlag vermutlich gescheitert wäre. Heute würde ich es vielleicht doch noch einmal ernsthafter prüfen, da Kanban besonders dann erfolgreich ist, wenn bestehende Prozesse langsam und kontinuierlich geändert werden sollen.

## 6.17 Agile Werte im Projekt

| **Individuen und Interaktionen** | **vor** | **Prozessen und Tools** |
|---|---|---|
| ▲<br>Wir sind auf die Bedürfnisse einzelner Teammitglieder eingegangen, anstatt auf die perfekte Einhaltung von Prozessen zu bestehen (z. B. durfte der Python-Entwickler früh mit weniger hoch priorisierten Backlog Items anfangen). | | |
| **Laufende Software** | **vor** | **Ausführlicher Dokumentation** |
| ▲<br>Das Team hat lediglich das dokumentiert, was es momentan und für die spätere Wartung für notwendig hielt. Der Code wurde sowieso inline dokumentiert und der Kunde hatte keinen Bedarf an Dokumenten. | | |
| **Zusammenarbeit mit dem Kunden** | **vor** | **Vertragsverhandlungen** |
| ▲<br>Obwohl es Verträge mit dem Endkunden gab, haben wir gemeinsam mit dem Kunden am Produkt gearbeitet, statt uns mit irgendwelchen Klauseln zu befassen. Während meiner Zeit vor Ort ist nicht einmal mit irgendwelchen Fristen, Vereinbarungen oder Verträgen gedroht worden, wir waren alle auf derselben Seite. | | |
| **Reagieren auf Veränderungen** | **vor** | **Befolgen eines Plans** |
| ▲<br>Der Erfolg dieses Projekts war nur möglich, weil alle Beteiligten sich auf die vorgeschlagene Vorgehensweise eingelassen haben und sowohl größere Anpassungen (einheitliches Product Backlog, ab sofort Akzeptanzkriterien benennen) als auch kleinere Tuningmaßnahmen verstanden und mitgemacht haben. | | |

# 7 Agile Softwareentwicklung als fundamentaler Bestandteil einer Unternehmensgründung

Andreas Leidig

*Im folgenden Beitrag wird geschildert, wie die Etengo (Deutschland) AG – kurz: Etengo – mithilfe der andrena objects ag – kurz: andrena – in engster Zusammenarbeit in weniger als einem Jahr den erfolgreichen Start in den Markt realisiert hat. Der Autor begleitete das Projekt in beratender Tätigkeit als agiler Coach im Namen der andrena. Das Beispiel zeigt, wie dicht ein unternehmerischer Geist und ein agiles Arbeitsethos beieinanderliegen und sich sogar ergänzen.*

## 7.1 Die Player

- Etengo:
  Ein Unternehmen soll gegründet werden. Der Gründer hat tief gehende Branchenerfahrung und eine Vision, wie er mit hervorragender Softwareunterstützung ein neuartiges Geschäftsmodell in seiner Branche etablieren will. Die dazu notwendige Software existiert nicht. Erfahrung in der Erstellung von Software ist ebenfalls nicht vorhanden.
- andrena:
  Ein Unternehmen mit mehr als zehn Jahren Erfahrung in der Erstellung von betriebswirtschaftlicher Individualsoftware auf Basis agiler Methoden.

## 7.2 Der Zug formiert sich

Etengo vereinbart ein Meeting mit uns, der andrena, um uns als möglichen Softwarepartner ins Boot zu holen. Sie kommen zu dritt: der Gründer Nikolaus Reuter, ein Berater, ein Jurist. Mein erster Eindruck: Diese drei wissen genau, *wohin* sie wollen.

Ist das etwas Besonderes? – Für mich schon. Ich arbeite überwiegend für Inhouse-IT-Projekte in großen Organisationen, wo ich oft schon froh bin, einen Ansprechpartner zu haben, der weiß, *was* er will. Dies dann meistens in Form

schriftlicher Anforderungen. Auf die Frage »Wohin?«, erhalte ich üblicherweise die Antwort: »Diese Anforderungen müssen umgesetzt werden.«

Die drei Etengo-Vertreter erläutern uns also, was sie vorhaben.

### 7.2.1 Die Vision: Wohin geht die Reise?

Die Etengo etabliert sich als Personaldienstleister, der einfach, schnell, transparent und preiswert IT-Freelancer vermittelt:

- **Einfach:**
  Nach Möglichkeit sollen die Freelancer selbst ihre Stammdaten (Skills, Verfügbarkeit etc.) verwalten.
- **Schnell:**
  Projektangebote sollen in möglichst kurzer Zeit mit passenden Freelancern in Verbindung gebracht werden, am besten softwareunterstützt (»Match«).
- **Transparent:**
  Die Grundlage eines Matches muss für einen Etengo-Mitarbeiter nachvollziehbar sein. Transparent ist auch das Abrechnungsmodell: Sämtliche Provisionen sind für die Beteiligten (Freelancer und Projektanbieter) offen auszuweisen.
- **Preiswert:**
  Die Provisionen sollen deutlich unter dem liegen, was die Konkurrenz verlangt.

All dies ist nur mit einer guten, integrierten Softwarelösung erreichbar, die alle Aspekte des Geschäftsmodells abdeckt. Die Vision ist schon so weit gereift, dass Folgendes klar ist:

1. Es soll einen öffentlichen Internetauftritt geben. Ein Freelancer soll eine zielgruppengerechte Website präsentiert bekommen, damit er auf einfache Weise seine Profile und Abrechnungen verwalten kann. Zusätzlich ist die Website natürlich Marketing- und Vertriebsinstrument.
2. Das Backoffice benötigt eine angepasste integrierte Software, um mit der Angebotsseite und den »Matches« zu arbeiten. Insbesondere ist gewünscht, sämtliche Interaktionen mit den Kunden (Freelancern und Projektanbietern) lückenlos verfügbar zu halten.
3. Es bestehen schon Vorstellungen hinsichtlich einiger technischer Details wie zum Beispiel Tabellenstrukturen oder möglicher technischer Ansätze.

Es muss ein Partner gefunden werden, der in der Lage ist, innerhalb von neun Monaten aus diesen Ideen eine langfristig tragfähige Software zu entwickeln. Für ein explizites Konzept, vergleichbar mit der Analyse- und Designphase in Wasserfallprojekten, ist keine Zeit.

### 7.2.2 Ergebnisse des Treffens

Nach dieser Vorstellung möchten die drei nun wissen:

1. Ist das in neun Monaten machbar?
2. Wie teuer wird das?
3. Sind die technischen Ideen tragfähig?

Wir, die Anwesenden der andrena, schauen uns verwundert an. Die Anforderungen sind selbst für Etengo noch nicht wirklich greifbar. Für uns ist darüber hinaus die Domäne völlig fremd. Natürlich verstehen wir, dass für sie die Antwort auf diese drei Fragen enorm wichtig ist. Schließlich hängt ihre Existenz daran. Wir erklären ihnen das Dilemma:

> *Sogar mit weitaus mehr und detaillierteren Informationen könnten wir ihnen keine Zahlen nennen, die ihnen einen nennenswerten Grad von Verlässlichkeit geben. Die Erstellung von Individualsoftware ist kein Autokauf.*

Allerdings können wir ihnen eine Perspektive aufzeigen, die es ihnen ermöglicht, die Projektverantwortung wahrzunehmen und kontinuierliches Risikomanagement zu betreiben: Scrum. Wir erklären ihnen in wenigen Minuten die Grundprinzipien iterativ-inkrementeller Entwicklung und schnell wird klar, dass die Durchführung des Projekts auf dieser Basis denkbar ist. Insbesondere leuchtet ihnen ein, dass sie die Verantwortung nicht delegieren können; ein Scheitern der Softwareentwicklung ist gleichbedeutend mit einem Scheitern des Unternehmensstarts.

Ganz nebenbei stellt sich heraus, dass Etengo noch ein Büro braucht. Also vermietet andrena einen Büroraum an Etengo, und der Gründer zieht ein. In demselben Büro wird das Entwicklerteam sitzen: Eine engere Zusammenarbeit mit dem Kunden kann es wohl kaum geben.

## Zwischenton: Interview mit Nikolaus Reuter

**Was waren damals die Gründe, sich für Scrum und andrena zu entscheiden?**

Unser Mentor kannte andrena bereits und hat den Kontakt hergestellt. Scrum war für mich und meine Berater völlig neu, wir hörten zum ersten Mal bei unserem Besuch davon. Die Argumente leuchteten uns unmittelbar ein: Risikominimierung durch regelmäßige Lieferung funktionierender Software.

**Wie konnten Sie beurteilen, ob die geschätzten Kosten nicht zu hoch waren, speziell im Vergleich zu anderen Softwarehäusern?**

Mir war früh klar, dass die Leistungen eines Dienstleisters nicht standardisiert sind, sodass Preisvergleiche nicht möglich sind. Auch dass ich hier einfach absolut das Ergebnis beurteilen muss und dann den Preis dafür akzeptiere. Hätte sich im Laufe des Projekts herausgestellt, dass die Entwicklung das dafür geschätzte Budget drastisch übersteigen würde, hätte die gesamte Unternehmensgründung infrage gestellt werden müssen.

**Hatten Sie damals keine Angst, ohne Detailplanung das ambitionierte Ziel insbesondere hinsichtlich des Termins zu verfehlen?**

Angst nicht, aber mit einer Unternehmensgründung ist natürlich immer eine hohe Unsicherheit verbunden. Letztendlich hat mir andrena gezeigt, dass ich das Risiko für den Erfolg meiner Idee nicht delegieren kann. Allerdings boten sie mir mit Scrum ein angemessenes Konzept, bei dem sowohl Verantwortung als auch Erfolgskontrolle bei mir bleiben. Der Preis ist natürlich, dass ich hier einhundert Prozent geben muss. Andererseits bietet das Vorgehen auch eine Risikosenkung in der Hinsicht, dass ich nur für bereits geleistete Arbeit zahlen muss und anhand der Ergebnisse nach jedem Sprint sofort beurteilen kann, ob das sein Geld wert ist. Zwischen andrena und Etengo bestand eine Vereinbarung, dass ich das Projekt jederzeit beenden kann und alle bis dahin erbrachten Leistungen behalte.

## 7.3 Vorbereitung

Noch ist kein Team gebildet, es gibt erst einen Entwickler für das Projekt. Er untersucht zunächst den Markt vorhandener Standardsoftware nach geeigneten Produkten, auf die man aufbauen könnte. Parallel beginnt Nikolaus Reuter ein initiales Product Backlog zu erstellen. Dabei unterstützen wir ihn, lassen ihm aber seine bewährten Arbeitsmethoden. Für den Großteil seiner Arbeit genügt ihm ein Tabellenkalkulationsprogramm. Es zeichnet sich schnell ab, dass mit den am Markt verfügbaren Standardprodukten keine tragfähige Basis errichtet werden kann, und das Backlog ist weit genug gediehen, um damit die ersten Sprints zu füttern.

## 7.4 Die ersten Wochen

Das Team startet mit zwei erfahrenen Entwicklern; zusätzlich wird ein sowohl in Architektur als auch in agilen Prozessen erfahrener Coach gestellt, der punktuell insbesondere den in Softwareentwicklung noch unerfahrenen Product Owner berät. Er erstellt für jeden Sprint aus seinen Tabellen User Stories, die er auf Karteikarten schreibt, damit die Planning Meetings ohne Computerunterstützung über die Bühne gehen können. Das ist zunächst ungewohnt, doch die Ergebnisse sprechen für sich: Die Meetings verlaufen extrem produktiv und fokussiert.

### 7.4.1 Es geht los: Aufstellung des Teams

In den ersten Tagen geht es hauptsächlich darum, dem Team eine grundsätzliche Vorstellung der Vision und der Domäne zu geben. Auf der Basis dieser Informationen soll das Team eine geeignete Plattform wählen. Die Wahl der Technologien liegt dabei allein in der Hand von andrena. Kern der von andrena zu entwickelnden Software sind das Backoffice sowie die dynamischen auf der Anwendung basierenden Teile des Internetauftritts.

Dem engen Kostenrahmen wird Rechnung getragen, indem die Teamgröße zunächst sehr klein gehalten und das endgültige Team aus einer Mischung von sehr erfahrenen und noch projektunerfahrenen Kollegen zusammengesetzt wird. Letztere werden zu einem deutlich geringeren Stundensatz abgerechnet, dafür erhalten sie ein exquisites Training »on-the-job«. Mit dem dritten Sprint ist dann die Teamgröße auf vier Entwickler angewachsen.

Die Entwickler entscheiden, die Aufgaben nach technologischen Schwerpunkten unter sich aufzuteilen. Sie sind jedoch durchaus in der Lage, in allen Bereichen der Software mitzuarbeiten. Auf dogmatisches Pair Programming wird verzichtet, stattdessen wird ein Vieraugenprinzip für das Einchecken von Code ins Repository vereinbart. Diese Entscheidungen sind der Kosteneffizienz geschuldet und werden trotz der möglichen nachteiligen Aspekte wie der Bildung riskanter »Wissensinseln« oder verringerter Code- und Designqualität getroffen.

Kostenrahmen, Termindruck und Konzentration auf Kernkompetenzen sowie Kooperationen mit anderen Firmen führen zu folgender technischen Ausgangsbasis:

- Die Entwicklung beschränkt sich zunächst auf rein technische Aspekte des Web-Frontends. Das endgültige Design und Verhalten der Benutzeroberfläche des Internetauftritts werden in einer späteren Projektphase mit einer Zulieferung integriert.
- Für das Web-Frontend und das Backend wird ein gemeinsames Objektmodell entwickelt.
- Eine realitätsnahe Laufzeitumgebung kann frühestens aufgebaut werden, wenn die wesentlichen Betriebsaspekte geklärt sind. Diese Fragen werden auf einen späteren Zeitpunkt verschoben. Dies betrifft insbesondere die Ausfallsicherheit durch Cluster sowie die Dimensionierung.
- Integrationstests erfolgen also bis auf Weiteres auf einer Umgebung, die in einigen Bereichen nicht mit der endgültig erwarteten übereinstimmt.
- Die Anforderung an ein ontologiebasiertes automatisiertes Matching soll ein Spezialist übernehmen.

Diese Einschränkungen sind dem Team bewusst, allerdings kann niemand die Konsequenzen abschätzen. Alle gehen davon aus, dass die späte Integration des Web-Frontends in die Betriebsumgebung gelingen wird.

Die Länge der Sprints wird auf zwei Wochen festgelegt. Dieser Rhythmus wird bis zum Ende des Projekts durchgehalten. Die ersten Sprints bringen schon sichtbare Ergebnisse, enttäuschen aber zunächst die Erwartungen des in Softwareprojekten noch unerfahrenen Product Owner. Dabei werden aber auch wesentliche Fundamente für die weitere Arbeit geschaffen, sodass die pro Sprint realisierten sichtbaren Funktionalitäten nach dem dritten Sprint auch den Product Owner zufriedenstellen.

Entsprechend den Prioritäten werden sowohl im Web- als auch im Backoffice-Bereich Stories umgesetzt.

### 7.4.2 Reviews und Retrospektiven

Die ersten Reviews bringen für den Product Owner und das Team enorme Lerneffekte:

- Beschreibungen, die für ihn völlig klar waren, sind nicht in seinem Sinne umgesetzt. Dem Team waren sie auch klar, allerdings hatte es die Beschreibung anders aufgefasst.
- Stories, die gemäß seinen Vorstellungen geliefert sind, gefallen ihm jetzt nicht so, wie sie sind.
- Der Product Owner legt großen Wert auf die Benutzerschnittstelle. Das Team ist eher auf das technische Funktionieren fokussiert.

Das heißt: Die Umsetzung im Sprint-Ergebnis muss nicht gefallen. Ein solches Nichtgefallen muss aber nicht an einer fehlerhaften oder missverstandenen Umsetzung liegen, sondern es kann auch daran liegen, dass der Product Owner seine Meinung geändert hat.

Jetzt wird das ganze Ausmaß des iterativ-inkrementellen Vorgehens klar. Änderungen und Anpassungen sind relativ preiswert, da sie frühzeitig erkennbar sind. Allerdings reicht das zweiwöchige Inspizieren durch den Product Owner nicht aus. Er und das Team einigen sich, auch während der Entwicklung öfter als bislang die Fortschritte zu besprechen. Dies fällt natürlich angesichts des gemeinsamen Büros sehr leicht.

### 7.4.3 Sprint-Ergebnisse und Velocity

Die Sprint-Ziele werden erreicht, Stories geliefert. Natürlich kommt es vor, dass eine Story in einem Sprint mal nicht komplett geliefert wird. Dies bleibt aber die Ausnahme.

Auch nach vier Sprints steigt die Velocity nicht auf ein Niveau, das zum Erreichen des planmäßigen Launch-Termins notwendig wäre, sondern bleibt konstant[1]. Das Team ist schlicht zu klein. Alle Beteiligten kommunizieren diese Tatsache mit großer Offenheit und Vertrauen. Zunächst entscheidet Nikolaus Reuter jedoch, den eingeschlagenen Kurs beizubehalten.

Bemerkenswert ist, dass die Zufriedenheit über die konkrete Arbeitsleistung des Teams jederzeit ausnehmend hoch ist. Es wird zu keiner Zeit Druck auf das Team ausgeübt, schneller zu arbeiten.

1. Als Velocity bezeichnet man die Menge an Funktionalität, die ein Team pro Sprint erledigt. Die Velocity bleibt hier konstant, da das Team immer neu auf Basis des Gelernten schätzt.

## Zwischenton: Interview mit Nikolaus Reuter

**Wie erlebten Sie die ersten Tage als Product Owner eines Entwicklungsteams?**

Es war neues Terrain, unbekanntes Gefilde und gerade deshalb auch verdammt spannend. Bis zum Projektstart hatte ich ehrlich gesagt zuvor noch nie etwas von einem Product Owner oder einer agilen Projektsteuerung mit der Managementmethode Scrum gehört. Rückblickend wusste ich auch nur theoretisch oder besser gesagt vage, auf was ich mich da einlasse, was da auf mich zukommt und schon gar nicht, wie sehr mich das fordern und in Zukunft weiterbringen wird. Nun kam Tag 1 und ich befand mich inmitten des Entwicklerteams, das gespannt auf Input wartete. Also los geht es: Aber was zuerst? Was sind überhaupt wichtige Infos, die so ein Team braucht? Intuitiv habe ich mich dann entschieden, zunächst meine Vision darzulegen. Also das große Bild und die Antwort auf die Frage, wo denn die Fahrt überhaupt hingehen soll. Denn eines wusste ich aus meiner bisherigen Berufserfahrung ganz sicher: Ein Kapitän, der den Hafen nicht kennt, den er ansteuern möchte, wird niemals, oder höchstens durch Zufall, die Segel richtig setzen können. Das Ergebnis entspricht dann entweder nicht dem gewünschten, oder die notwendigen Korrekturen auf der Wegstrecke kosten verdammt viel Geld. So, dachte ich, wird es auch einem Team gehen. Insbesondere dann, wenn sie noch nie mit der Branche Personaldienstleistung geschweige denn mit deren Prozessen und Software zu tun hatten. Also startete ich mit meiner Vision, was Etengo anders machen möchte, wie das Geschäftsmodell funktionieren soll, erklärte die zentralen Prozesse, mir bekannte Stolperfallen, die involvierten Abteilungen und deren Aufgaben im Einzelnen. Von mir fast unbemerkt sprang wohl ein Funke über. Das Team erkannte wohl meine Begeisterung für die Geschäftsidee und merkte, wie sehr ich dieses Unternehmen lebte und verkörperte. Kurz dachte ich mal darüber nach, nicht so viel zu erzählen und lieber das Team mit dem Programmieren starten zu lassen. Aber wirklich nur kurz. Die ersten Tage glichen so mehr einem Workshop und an vielen Stellen einem Power-Shot an Wissen über die Personaldienstleistung im Allgemeinen und das IT-Freelancergeschäft im Besonderen. Rückblickend bin ich mir ganz sicher, dass dies der entscheidende Punkt war: das Team auf einen soliden Wissenstand zu bringen, es zu involvieren und offen alle Themen anzusprechen. Die ersten Tage waren somit ein wertvoller Invest in Wissen und Gemeinschaft, auch im Sinne eines gemeinsamen Verständnisses. Eben nicht die Hammermethode: Hier ist mein Pflichtenheft, lest es und baut mir das genau so und nicht anders.

### 7.4.4 Routine und Flow

Nach den ersten Sprints kehrt eine positive Routine ein. Das Team hat sich gefunden; die Zusammenarbeit mit dem Product Owner hat sich eingespielt. Nikolaus Reuter hat Zeit, sich mehr auf andere Aspekte zu konzentrieren. Etengo selbst stellt fachliche Mitarbeiter ein, Marketing und Organisation nehmen ihre Arbeit auf. Dies mündet darin, dass sich Etengo ein eigenes Büro im selben Gebäude mietet. Nikolaus Reuter zieht dorthin zu seinen fachlichen Mitarbeitern. Er kann sich fortan nicht mehr so intensiv mit Softwarefragen befassen, da ihn andere Aufgaben fordern. Auf die Zusammenarbeit mit dem Entwicklungsteam hat dies keinen negativen Einfluss. Dank der räumlichen Nähe hat das Team ein sehr tiefes fachliches Verständnis der Domäne erlangt und kann so nach ausführlichen

Planning Meetings sogar fachlich anspruchsvolle Stories weitgehend selbstständig und mit nur wenigen Rückfragen umsetzen.

Die endgültige Aufhebung der anfangs fast symbiotischen Beziehung zwischen Etengo und dem Entwicklungsteam geschieht durch die Verlegung des Etengo-Firmensitzes von Karlsruhe nach Mannheim. Dieser erfolgt kurz vor dem Launch der Site und damit kurz vor Abschluss des Projekts.

### 7.4.5 Die Umgebung

Räumliche Nähe ist für ein agiles Team, speziell für ein solch kleines, unabdingbar. Daher stellen wir bei andrena einen eigenen Raum für das Team zur Verfügung. Wie bereits erwähnt setzt sich Nikolaus Reuter zu dem Team in denselben Raum. Dies ist auch für das Scrum-erprobte Team eine neue Erfahrung: ein Product Owner zum Greifen nah. Schnell bildet sich Vertrauen. Für einen Außenstehenden, der einen Blick in das Zimmer wirft, ist nicht zu erkennen, dass es sich um eine Kunden-Lieferanten-Beziehung handelt. Der Umgang zwischen allen Beteiligten erfolgt auf Augenhöhe. Die Fähigkeiten und Verantwortlichkeiten der Einzelnen sind allen klar und werden respektiert. Das Team wird vom Product Owner sehr eng integriert, wenn es darum geht, die Anforderungen im Detail auszuarbeiten. Selbst bei der Priorisierung wird der Rat des Teams gesucht und gefunden.

Das gemeinsame Büro hat aber auch auf anderer Ebene Auswirkungen:

#### »No Pain, no Gain!«

Dieser bewusst übertriebene Leitspruch hängt hinter Nikolaus Reuters Schreibtisch. Und dieser Fokus auf die Zielerreichung ist immer spürbar. Nikolaus Reuter schafft es, den existenziellen Funken auf das Team überspringen zu lassen: Hier geht es um etwas! – Es geht nicht um die Umsetzung isolierter Features, sondern darum, ein funktionierendes Ganzes zu erschaffen. Er hat seine Vision erfolgreich auf das Team übertragen. Zusätzlich motivierend ist die Vereinigung von inhaltlicher und finanzieller Verantwortung in einer Person. Auch ist allen transparent, dass Nikolaus Reuter mehr als jeder andere an dem Projekt arbeitet und ein Scheitern von ihm alleine getragen werden müsste.

Schon früh (ca. ein halbes Jahr vorher) ist klar, wann die Site live geht und dass dieser Termin nicht verschiebbar ist. Was jedoch bis zum Schluss verhandelbar bleibt, ist der Umfang. So wird das Ganze nicht zum berüchtigten Todesmarsch.

#### »Alle Tage sind gleich lang, jedoch verschieden breit …«

Mit diesen Worten aus einem Likörell Udo Lindenbergs beginnt eine Mail, mit der Nikolaus Reuter mitten im Projekt nach Feierabend zu einem spontanen Abendumtrunk einlädt, um die Erfolge der vorangegangenen Tage zu feiern. Gemeinsame Abende sind regelmäßiger Ausdruck über die Freude, zusammen etwas Wertvolles geschaffen zu haben, und ein weiteres Zeichen der gegenseitigen Wertschätzung. Hierbei nehmen selbstverständlich auch die neuen Mitarbeiter Etengos teil, die in Bereichen wie Marketing, Verwaltung und Vertrieb ihre Arbeit aufnehmen.

## 7.5 Schwierigkeiten

Ein Projekt, das ohne nennenswerte Probleme fertiggestellt wird, ist keinen Artikel wert. Was also stößt diesem Projekt während seiner Laufzeit zu?

#### Wir sind zu langsam!

Nach den ersten Sprints wird anhand der Velocity und der noch umzusetzenden Stories klar: In diesem Tempo sind die ambitionierten Ziele nicht zu erreichen. Massive Einschnitte an der Qualität stehen nicht zur Debatte. Allen ist klar, dass dadurch die Fertigstellung nicht beschleunigt, sondern verlangsamt wird. Der Zieltermin ist auch nicht mehr verhandelbar. Als variable Aspekte bleiben: Umfang und Kosten. Zunächst entscheidet sich Nikolaus Reuter für den Umfang. Er ändert sein Backlog, sodass große Funktionsblöcke wie die Rechnungserstellung für das initiale Release komplett wegfallen. Doch auch diese Reduktion des Umfangs bringt noch nicht den Durchbruch.

#### Das Team wird vergrößert

Ein Kardinalfehler vieler Softwareentwicklungsprojekte: Wenn die Zeit knapp wird, bringe neue »Ressourcen« ins Team. Und im agilen Umfeld gilt die Faustregel: »Zusätzliche Entwickler senken die Velocity.« Wir sehen jedoch keine Alternativen. Schweren Herzens beschließen wir – im Bewusstsein aller Risiken –, das Team aufzustocken. Den zu erwartenden Nachteilen – Turbulenz im Team und zunächst zurückgehende Velocity – können wir ein paar starke Fakten entgegensetzen:

1. Durch das konsequente Vieraugenprinzip ist die gesamte Codebasis homogen und konsistent. Dadurch kann der vorhandene Code sehr gut als Blaupause für neue ähnliche Funktionalitäten verwendet werden.

2. Unsere Kollegen sind mit der Arbeitsweise und der Kultur vertraut. Wir müssen keine »Fremden« in das Team integrieren, sondern können aus der Stammbelegschaft schöpfen.
3. Anstehende Tasks sind in weiten Teilen vom Bestehenden unabhängig, und es wird nicht sehr viel Domänenwissen benötigt. Dadurch wird es den Neuankömmlingen leicht gemacht, in die Fachlichkeit einzudringen.

Was wir dann erleben, übertrifft unsere kühnsten Vorstellungen: Die Velocity steigt linear! – Erklären können wir das nicht. Wir werden uns in zukünftigen Projekten auch nicht darauf verlassen.

#### Wir haben Features, die wir für den Launch nicht brauchen

Gegen Ende des Projekts stellen wir rückblickend fest, dass Features entwickelt wurden, deren Bewertung jetzt nicht mehr nachvollziehbar ist. Hätten wir doch statt X besser Y entwickelt! Doch anstatt sich dem hinzugeben und Motivation zu verlieren, blicken wir in die Zukunft und konzentrieren uns darauf, den Wert von Stories noch kritischer zu betrachten.

#### Der Pre-Live-Test kann erst spät gestartet werden

Auch nach dem Aufstocken des Teams bleibt keine Zeit zum Durchatmen. Es wird bis kurz vor Ende an neuen Stories entwickelt. Schlussendlich ist erst vier Wochen vor dem Launch (und damit eine Woche vor Weihnachten) Zeit für einen umfassenden Integrationstest unter produktiven Bedingungen. Jetzt zahlt sich die sorgfältige Arbeit der vergangenen Monate aus: Es tauchen nur kleine Bugs auf, die allesamt schnell behoben werden können. Die hohe Fehlerzahl jedoch überrascht das Team; sie haben mit weniger gerechnet. Viele dieser Bugs beruhen auch auf der extrem späten Integration der Anwendung mit der gelieferten Website. Eine frühere Integration hätte dem Projekt in dieser Hinsicht gut getan.

#### Die Datenmenge in den Tests war zu gering

Die ersten Tests unter realitätsnahen Bedingungen bringen eine böse Überraschung: Die Performanz der Anwendung ist außerhalb der Toleranz. Ursache hierfür ist das Versäumnis, rechtzeitig mit realistischen Datenmengen zu testen. Der Fokus in den vorangegangenen Monaten lag definitiv auf der Erstellung von Features zulasten einer frühzeitigen realitätsnahen Testumgebung.

### Zwischenton: Interview mit Nikolaus Reuter

**Warum haben Sie erst spät im Projektverlauf auf die sich früh abzeichnende Ressourcenknappheit reagiert und das Team vergrößert?**

Die Antwort schießt mir heute noch intuitiv in den Kopf. Weil ich versucht habe, dem Prinzip Hoffnung noch die Stange zu halten. Warum? Weil ich natürlich im klassischen Zwiespalt zwischen Budgetrestriktionen und einer finiten Timeline bzw. einem Go-Live-Termin feststeckte. Und da macht man dann gerne mal die Augen zu bzw. wird zum Strohhalm-Taktiker. Es könnte ja auch noch passieren, dass wir es schaffen, die Velocity schlagartig nach oben zu schrauben, oder einige Tasks oder User Stories doch weniger Zeit als geschätzt in Anspruch nehmen. Leider trat sowohl das erste als auch das zweite Szenario nicht ein, zumindest nicht im notwendigen Maße. Das Prinzip Hoffnung scheiterte wie so oft. Wenn man als Unternehmer oder Projektverantwortlicher dann endlich aufhört, sich selbst in die Tasche zu lügen, fällt es einem wie Schuppen von den Augen. Und dann handelt man auch sofort und ergreift Maßnahmen. Zum Glück kam meine Erkenntnis nicht zu spät. Das war aber auf die Hartnäckigkeit und das echte Wahrnehmen der Aufgabe des Coachs zurückzuführen.

## 7.6 Über den Tellerrand hinaus

Wie bei jedem Projekt sind einige nicht funktionale Anforderungen und externe Abhängigkeiten zu berücksichtigen.

### 7.6.1 Nicht funktionale Anforderungen

Nicht funktionale Anforderungen sind nichts Seltenes. So auch in diesem Projekt. Folgende wesentliche Querschnittsanforderungen sind erwähnenswert:

#### Innere Codequalität

Vor dem Hintergrund, die zentrale IT-Basis eines Unternehmens zu erstellen, sind Aspekte wie Wartbarkeit, Nachhaltigkeit sowie Änder- und Erweiterbarkeit der erstellten Software von zentraler Bedeutung für den Gründer. andrena hat tief gehende Erfahrung darin, Indikatoren für diese Aspekte durch statische Codeanalyse kombiniert mit der zusätzlichen Erfassung einiger anderer Kennzahlen zu erstellen. Die Basis dafür bietet ein selbst erstelltes Tool (ISIS). Auswertungen mithilfe dieses Werkzeugs bieten auch Laien nach kurzer Einführung eine Möglichkeit, die innere Qualität von Software im Blick zu behalten. Jede Sprint-Retrospektive wird mit der Besprechung eines solchen Messpunkts eröffnet. Insbesondere der zeitliche Trend der Ergebnisse über den Verlauf mehrerer Sprints zeigt oft Potenziale. Durch die Diskussion dieser Trends bleibt die Entscheidungskompetenz über zu ergreifende Maßnahmen in den Händen des Product Owner. Dem Team kommt hier eine beratende Rolle zu.

#### Performanz und Stabilität

Dass die Software »rock solid« werden muss, steht nie zur Debatte. Dieser Aspekt ist von Anfang an im Blickpunkt. Bezüglich der Performanz können wir auf Erfahrungen in technologisch ähnlichen Projekten bauen. Zusätzlich gilt die Maxime, die gesamte Webseite der Anwendung zustandslos zu implementieren, um so bei auftretenden Eventualitäten handlungsfähig zu bleiben, indem beispielsweise zusätzliche Serverinstanzen integriert werden. Im Zuge der Vorbereitungen für den zukünftigen Betrieb werden diese Aspekte frühzeitig mit den zu erwartenden Zugriffsszenarien getestet. Das bereits erwähnte Problem mit den Datenmengen wird dennoch übersehen.

#### Usability

Im Laufe der ersten Sprints stellt sich heraus, dass der Backoffice-Teil der Anwendung weit mehr leisten muss als ein Datenbank-Frontend. Um den in der Vision formulierten Zielen – speziell Einfachheit und Schnelligkeit – gerecht werden zu können, muss die Benutzerschnittstelle komfortabel und konsistent sein. Deshalb wird in den ersten Sprints im Rahmen der ersten Features ein UI-Framework mitentwickelt, um dem gerecht zu werden.

### 7.6.2 Externe Abhängigkeiten und Kooperationen

Die wesentliche Kernkompetenz von andrena liegt in der agilen Entwicklung von betriebswirtschaftlicher Individualsoftware. Etengo will aber auch

- ein ontologiebasiertes Matching von Freelancern und Projektangeboten,
- einen grafisch ansprechenden Internetauftritt,
- einen zuverlässigen Betriebsdienstleister.

Schnell wird klar, dass hier Kooperationen notwendig werden. andrena hilft bei der Auswahl von Partnern, belässt aber die übergeordnete Koordination bei Etengo.

Als Partner für die Ontologie fällt die Wahl schnell auf einen lokalen Spezialisten. Die Zusammenarbeit erweist sich als schwierig hinsichtlich der Arbeitsweise, weil die gewohnte agile Arbeitsweise nicht auf den Partner übertragen werden kann. Wir hätten uns eine sehr enge Kooperation und Synchronisation mit unserem Rhythmus gewünscht. Der Lieferant besteht auf konventionellem Vorgehen mit durch ihn bezüglich Umfang und Termin festgelegten Lieferungen. Diese Abhängigkeit beeinflusst die Priorisierung auf unserer Seite; wir müssen unsere Integrationen an dessen Lieferungen anpassen, die teilweise spät kommen.

Bezüglich Grafikdesign und Internetagentur bringt Etengo zwei Firmen ihrer Wahl mit ein. Die notwendige Integration erfolgt partnerschaftlich. In der End-

phase bezieht ein Mitarbeiter der Agentur einen Arbeitsplatz im zentralen Teamraum bei andrena.

Bei beiden Partnern wird Zeit für die Integrationen eingeplant. Dies ist notwendig, um deren nicht agiles Vorgehen zu kompensieren. Jede Lieferung muss qualitätsgesichert werden und erfordert teilweise unvorhergesehene Anpassungen.

Für die Auswahl eines Betriebsdienstleisters greifen wir auf die Hilfe eines spezialisierten Beraters zurück, mit dem andrena schon in der Vergangenheit gute Erfahrungen gesammelt hat. Zusammen mit ihm werden auch die Themen Stabilität und Performanz näher beleuchtet und getestet.

Die Behandlung von Querschnittsaspekten sowie die Kooperation mit Partnern wird frühzeitig betrachtet und immer aus dem Projektbüro koordiniert. Dadurch wird erreicht, dass auftretende Probleme schnell erkannt werden und darauf reagiert werden kann. Die externen Lieferungen werden als Risiko betrachtet und diesem wird durch entsprechende Puffer Rechnung getragen.

Viele der Integrationen, die leider nicht immer synchron mit den Sprints stattfinden, laufen nicht so rund, wie wir es uns wünschen. Allerdings laufen sie auch nicht schlimmer als befürchtet.

## 7.7 Rückblickende Betrachtung und Zusammenfassung

#### Prozess

Der zugrundeliegende Prozess war Scrum. Allerdings haben wir den Prozess an einigen Punkten verändert:

- Es gab keinen dedizierten Vollzeit-ScrumMaster.
- Reviews werden um Aspekte der inneren Qualität durch Analysen mit ISIS erweitert.
- Als das Team in einen guten Arbeitsfluss gekommen war, haben wir die Frequenz der Retrospektiven gesenkt sowie die Dauern von Review- und Planning Meetings bewusst auf ein Minimum reduziert.
- Das Gesamtsystem (mit Ontologie und Weboberfläche) wurde nicht von Anfang an integriert entwickelt; auch wurden diese externen Teile nicht mit Scrum entwickelt.

#### Fazit für den Product Owner

Der Unternehmensgründer Nikolaus Reuter wäre mit seinem betriebswirtschaftlichen Hintergrund zunächst nicht darauf gekommen, eine Software iterativ-inkrementell zu entwickeln. Die Idee leuchtete ihm aber schnell ein. In der Anwendung von Scrum erkannte er viele ihm bekannte Konzepte:

- Risikomanagement:
  Entdecke Risiken und behandle sie offen. In Scrum erfolgt dies durch die Priorisierung sowie die regelmäßigen Inkremente.
- Konzentration auf das absolut Notwendige:
  In einer Unternehmensgründung ist ein Tag für den Gründer mit 24 Stunden eigentlich zu kurz, um alles zu erledigen, was erledigt werden sollte. In Scrum kann er sich auf Details im aktuellen und nächsten Sprint konzentrieren, ohne dabei das Gesamtrelease aus den Augen zu verlieren.
- Ziel- und Ergebnisorientierung:
  Anstatt Tage oder gar Wochen damit zu verbringen, Konzepte zu erstellen, wird laufende Software geschaffen.
- Kooperation und Teamarbeit:
  Ein junges Unternehmen kann nur erfolgreich wachsen, wenn sich alle Mitarbeiter blind vertrauen können. Genauso ist es auch in einem Softwareprojekt. Diese Mentalität brachte er mit und fand sie auch im Team wieder.
- Umarme das Unbekannte, Unsichere und sich Ändernde:
  Die Gründung eines Unternehmens ist in viel höherem Maß mit diesem Aspekt verbunden als die Entwicklung von Software.
- Steuerbarkeit durch Transparenz:
  Anstatt sich auf Berichte zu verlassen, werden echte Arbeitsergebnisse inspiziert.

#### Fazit für andrena

Die Rolle des Product Owner ist ein zentraler Erfolgsfaktor von Projekten. Dieses Projekt führte uns vor Augen, wie motivierend ein gesamtverantwortlicher Product Owner ist, der sich hundertprozentig engagiert. Dadurch

- wurden notwendige Entscheidungen schnell getroffen,
- wurde die Entwicklung in die erfolgreiche Richtung getrieben,
- waren Anforderungen klar,
- stieg die Motivation des Entwicklungsteams.

Bis heute betreut die andrena die Weiterentwicklung der Software. Persönliche Treffen erinnern an Begegnungen alter Freunde.

## 7.8 Agile Werte im Projekt

| Individuen und Interaktionen | vor | Prozessen und Tools |
|---|---|---|
| ▲<br>Der Kunde war völlig frei, wie die Umsetzung zu gestalten ist. Die Hoheit über die Umsetzung lag beim Team. Es wurde lediglich sehr kurz »Scrum« als Rahmen definiert. | | |
| **Laufende Software** | **vor** | **Ausführlicher Dokumentation** |
| ▲<br>Die Dokumentation hat der Kunde selbst erstellt, um damit die Entwicklung im Auge zu behalten. Es wurde nur dokumentiert, was notwendig war. Die vorherrschende Kommunikation war mündlich und flüchtig. | | |
| **Zusammenarbeit mit dem Kunden** | **vor** | **Vertragsverhandlungen** |
| ▲<br>Der Vertrag beschränkte sich auf die Aushandlung eines Stundensatzes und das Recht, zu jedem Sprint-Ende den Vertrag zu beenden. Das Risiko lag voll auf Kundenseite. Der Kunde saß mit den Entwicklern im gleichen Raum. | | |
| **Reagieren auf Veränderungen** | **vor** | **Befolgen eines Plans** |
| ▲<br>Es fanden regelmäßige Retrospektiven statt und der Entwicklungsprozess wurde kontinuierlich feingetunt. Das Backlog wurde kontinuierlich angepasst, um das Ziel des Livegangs nicht aus den Augen zu verlieren. | | |

Henning Wolf, Agile Projekte mit Scrum, XP und Kanban , dpunkt.verlag, ISBN 978-3-86490-266-6

# 8 Erfolgreich im Festpreis

Stefan Roock

*In großen bürokratischen Organisationen dauert alles ewig. Festpreisprojekte und agile Softwareentwicklung passen nicht zusammen. Überstunden sind Teufelszeug und funktionieren nicht. Die Entwicklungsgeschwindigkeit des Teams ist von höchster Wichtigkeit. Oder vielleicht doch nicht?*

## 8.1 Vor dem Projekt

Es ist bereits einige Jahre her, da erhielten wir Anfang August die Ausschreibung einer großen Versicherung. Es sollte eine Internetanwendung zur Berechnung und Angebotserstellung für Kfz-Versicherungen entwickelt werden. Der fixe Fertigstellungstermin war der 31. Oktober und der Kunde meinte es ernst. Würde das System nicht rechtzeitig verfügbar sein, würde ein wichtiger Vertrag mit einem Kooperationspartner platzen. Druck war also mehr als ausreichend vorhanden. Wir nahmen die Herausforderung an. Wir hatten ein ähnliches System bereits für eine andere Versicherung entwickelt und hielten das Projekt zwar für anspruchsvoll, aber machbar. Das Team bestand anfänglich aus drei Entwicklern und sollte über die kurze Projektlaufzeit von 10 Wochen auf fünf erweitert werden.

Folgende Überlegungen führten zu der Entscheidung, den Auftrag anzunehmen:

- Es gab bereits mehrere vergleichbare Beitragsrechner im Internet von anderen Versicherungsunternehmen. Dadurch war klar, welche Features notwendig sein würden, denn die Differenzierung zum Markt würde nicht durch die Anwendung, sondern durch die geplante Zusammenarbeit mit dem Kooperationspartner erfolgen.
- Der Kunde hatte das Lastenheft zwar zum aktuellen Zeitpunkt noch nicht komplett fertiggestellt, aber es schien auf den Marktdurchschnitt hinauszulaufen. Die Anforderungen schienen klar.

- Da wir ein ähnliches System bereits für einen anderen Kunden entwickelt hatten, fühlten wir uns bei der Bewertung des Aufwands recht sicher: ca. 120 Personentage.
- Das System sollte neben Pkw auch andere Fahrzeugtypen unterstützen, z. B. Motorräder, Anhänger und Wohnmobile. Damit schien es möglich und sinnvoll, die Entwicklung in mehrere Releases aufzuteilen. So könnten wir bei Problemen oder zu niedriger Aufwandsschätzung zumindest für das erste Release sicherstellen, dass der Termin eingehalten wird.
- Wir entschieden uns, das System im Wesentlichen mit Technologien zu entwickeln, die wir bereits aus früheren Projekten kannten.[1] Damit hatten wir auf technologischer Ebene nur geringe Risiken.
- Als Unsicherheit identifizierten wir die Tatsache, dass das Team in der geplanten Konstellation noch nicht zusammengearbeitet hatte und einer der später hinzukommenden Entwickler sogar von einem anderen Unternehmen kommen würde. Die Zusammenarbeit im Team konnte uns also noch ein Bein stellen.

## 8.2 Der Projektbeginn

Für uns war klar, dass wir agil arbeiten wollten, und zwar orientiert an Scrum und eXtreme Programming. Dem Kunden war das letztlich egal. Für ihn zählte, dass der Termin gehalten würde. Um möglichst schnell und direkt mit dem Kunden zusammenarbeiten zu können, bezogen wir einen Büroraum beim Kunden. Wir sichteten die Anforderungen des Kunden, überlegten uns erste fachliche Entwürfe und vereinbarten im Team, wie wir arbeiten wollten.

### 8.2.1 Rollen

Aufseiten des Kunden gab es einen Projektleiter und einen fachlichen Ansprechpartner. Der fachliche Ansprechpartner war für fachliche Entscheidungen und die Beantwortung fachlicher Fragen zuständig. Der Projektleiter sollte bei der Beseitigung von Hindernissen helfen und auf die Vertragseinhaltung achten. Der fachliche Ansprechpartner war also dem Product Owner aus Scrum nicht unähnlich und der Projektleiter übernahm ScrumMaster-Tätigkeiten. Zu behaupten, damit hätten wir Scrum eingesetzt, würde aber sicher zu weit gehen. Der Projektleiter hat sich weder um die Einhaltung des Scrum-Frameworks gekümmert noch um die Teambildung. Der fachliche Ansprechpartner war vom Kunden aus nicht bevollmächtigt. Im Zweifel hätte der Projektleiter jede seiner Entscheidungen überstimmen können.

1. Dem Kunden war klar, dass der Zeitrahmen sehr eng war, und er hat uns vor diesem Hintergrund weitgehend freie Wahl bei der Auswahl der Technologien gelassen.

Der Kunde wünschte sich auch auf unserer Seite einen Projektleiter, um einen dedizierten Ansprechpartner zu haben. Ich übernahm diese Rolle zusätzlich zu meiner Rolle als Entwickler und entschied mich dafür, sie nur gegenüber dem Kunden auszufüllen, nicht aber in Richtung des Teams. Gegenüber dem Team war ich ein gleichberechtigtes Teammitglied.

### 8.2.2 Arbeit im Team

Zumindest die Startbesetzung des Teams hatte mehrjährige Erfahrung in agilen Teams. Wir hatten eine klare Vorstellung davon, wie die agile Softwareentwicklung ablaufen sollte. Uns war aber auch klar, dass einige wünschenswerte Eigenschaften in der kurzen Projektlaufzeit nicht sinnvoll herstellbar sein würden. So vereinbarten wir pragmatisch im Team eine Mischung aus Scrum, eXtreme Programming und Feature Driven Development.

#### Organisatorische Festlegungen

- Terminerreichung ist das wichtigste Ziel. Wir opfern dafür aber nicht die Qualität des Systems.
- Wir führen jeden Morgen um 9:00 Uhr ein Standup-Meeting zur Einsatzplanung für den Tag durch.
- Wir arbeiten mit wöchentlichen Iterationen. Montags führen wir Iterationsreview und -planung durch, jeden Freitag stellen wir dem Kunden eine neue Testversion des Systems zur Verfügung.
- Durch den Festpreisvertrag mit Lastenheft sind die Anforderungen fixiert. Wir nehmen Änderungswünsche des Kunden auf und bilden daraus Change Requests, die dieser dann zusätzlich beauftragen kann.
- Wir splitten das zu liefernde System in drei Releases auf und führen nach jedem Release eine Retrospektive zur Prozessverbesserung durch.
- Der Raum gehört uns. Wir hängen an die Wände, was wir für richtig halten. Das ruiniert wahrscheinlich die Tapeten, sodass wir dem Kunden nach Projektabschluss einen Gutschein für die Renovierung des Büros schicken würden.

#### Festlegungen zu Entwicklungspraktiken

- Wir entwickeln testgetrieben mit Unit Tests für den kompletten Code mit Ausnahme der Benutzungsoberfläche. Diese muss so »dumm« sein, dass dadurch kein nennenswertes Problem entsteht.
- Wir arbeiten mit einem Weak-Ownership-Modell. Wir teilen das System grob in Zuständigkeitsbereiche je Entwickler auf und jeder Entwickler arbeitet hauptsächlich in seinem Zuständigkeitsbereich. Bei Bedarf darf und soll jeder aber auch Code aus anderen Zuständigkeitsbereichen ändern.

- Dazu passend benutzen wir Pair Programming nach Bedarf. Wenn ein Entwickler der Meinung ist, dass er mit einem Partner arbeiten möchte, wird er einen Partner bekommen.

### 8.2.3 Überstunden

Vor dem Hintergrund der engen Deadline war uns klar, dass Überstunden früher oder später ein Thema werden könnten. Wir entschieden uns für »früher«: Späte Überstunden sind meist von jeder Menge Stress begleitet und stehen in der Regel unter dem Vorzeichen, das Schlimmste zu verhindern. Wir vereinbarten stattdessen, ab Tag 1 Überstunden zu arbeiten. Jeder arbeitet so viel, wie er sich persönlich zumuten kann. Damit standen die Überstunden unter einem positiven Vorzeichen: *Play to win*.

Der Umgang mit Überstunden wurde von allen Teammitgliedern als sehr positiv bewertet. Alle waren auf das Projekt committet und haben sich daher gerne über die reguläre Arbeitszeit hinaus engagiert, und das, obwohl eine große Unsicherheit bezüglich der Abgeltung der Überstunden herrschte. In der Firma hatten wir die Regel, dass maximal 24 Überstunden in den Folgemonat übernommen werden können und der Rest verfällt. Uns war bewusst, dass wir diese Grenze schnell überschreiten würden, und wir vertrauten darauf, dass unsere Geschäftsführung eine vernünftige Regelung finden würde. Das tat sie auch. Die Überstunden verfielen nicht und konnten nach dem Projekt abgebummelt werden.

## 8.3 Projektdurchführung

### 8.3.1 Problematische Releaseplanung

Schließlich legten wir los. Zunächst sprachen wir mit dem Kunden über die einzelnen Releases. Das System legte eine Unterteilung in Fahrzeugtypen sehr nahe und wegen des hohen Zeitdrucks würden Einzelreleases deutlich zur Risikominimierung beitragen. Überrascht stellten wir fest, dass unser Kunde dafür nicht offen war. Aus seiner Sicht gab es nur ein Release: alle Features bis zum Fertigstellungstermin.

Immerhin konnten wir mit dem Kunden darüber sprechen, dass eine Organisation der Entwicklung entlang der Fahrzeugtypen sinnvoll wäre und welche Fahrzeugtypen den größten Mehrwert generieren würden. So kamen wir zu einer Priorisierung nach Fahrzeugtypen: Pkw, Motorräder, Wohnmobile, Anhänger …

Wir beschlossen, für die Strukturierung der Entwicklung zwei Fake-Releases und das finale Release zu planen: Pkw, Motorräder, restliche Fahrzeugtypen.

Ich habe die Mitarbeiter beim Kunden als fähig und intelligent erlebt. Daher glaube ich nicht, dass die Verweigerung der Releases an Betonköpfigkeit lag. Ich gehe davon aus, dass unsere Darstellung des Konzepts und seines Nutzens nicht angemessen war. Der Kunde hat schlicht nicht verstanden, worauf wir hinauswollten und was das für ihn bedeuten würde.

### 8.3.2 Der Raum

Wir saßen mit dem gesamten Team gemeinsam in einem Raum, den wir für uns allein hatten. Eine Wand war komplett mit Flipcharts vollgehängt, auf denen die Features des nächsten Release standen. Die Features haben wir aus dem Lastenheft ermittelt. Jedes fertiggestellte Feature wurde sofort durchgestrichen.

An einer zweiten Wand hing eine Flipchart-Seite mit den aktuellen Hindernissen im Projekt. Wir etablierten die Regel, dass Hindernisse so schnell beseitigt werden müssen, dass immer alle noch nicht beseitigten Hindernisse Platz auf dieser einen Flipchart-Seite hätten.

Daneben hingen Burndown-Charts für den Restaufwand (gemessen in Anzahl Features) sowie die investierten Personentage. Letzteres benutzten wir, um im eigenen Unternehmen früh darauf hinweisen zu können, falls die geplanten Aufwände deutlich überschritten werden würden.

### 8.3.3 Umgang mit Hindernissen

Wir gingen sehr offen mit Hindernissen um und klassifizierten Hindernisse grob in folgende Kategorien:

a) unter unserer Kontrolle,
b) in unserem Einflussbereich,
c) außerhalb unseres Einflussbereichs.

Für Hindernisse der Kategorie a) und b) trafen wir sofort Maßnahmen zur Beseitigung. Für Hindernisse der Kategorie c) beschlossen wir, wie wir mit ihnen umgehen würden, und jammerten nicht weiter darüber.

Die Hindernisbeseitigung erfolgte immer zeitnah und binnen weniger Tage. So wurden die Hindernisse immer kleiner und wir hatten am Ende tatsächlich Dinge auf der Hindernisliste wie »Ich habe gestern 30 Minuten gebraucht, um die Telefonnummer eines Ansprechpartners herauszufinden« oder »1 Minute Build-Zeit ist zu lang«. Auch diese Mikrohindernisse haben wir beseitigt und dies führte gefühlt zu einer höheren Entwicklungsgeschwindigkeit.

### 8.3.4 Die Tests

Die abschließenden Abnahmetests sollten durch die Fachabteilungen durchgeführt werden und es gab keine Chance, die Mitarbeiter der Fachabteilungen für die Tests in das Team zu integrieren. Also wählten wir einen Kompromiss: Wir würden selbst während des Projekts testen und hoffen, dass die Fachabteilungen am Ende nur noch wenige Fehler finden würden. Das würde möglicherweise die Aufwände auf unserer Seite vergrößern, aber die Wahrscheinlichkeit erhöhen, den Termin zu halten.

Es stellte sich schnell heraus, dass das Testen der Beitragsberechnung sehr zeitraubend sein würde. Für jeden Fall die Daten passend in die Oberfläche einzutragen und dann die Ergebnisse mit den erwarteten Ergebnissen (die wir mit einem Sachbearbeiterprogramm ermittelten) zu vergleichen, würde jeweils mehrere Minuten dauern. Wir führten daher für die Beitragsberechnung automatisierte Akzeptanztests mit FIT (siehe *http://fit.c2.com*) ein. Wir füllten die Tests mit unseren Testdaten. Die Fachabteilungen begannen mit ihren Tests, als wir das erste Fake-Release für Pkw fertiggestellt hatten. Die Rückläufer bezüglich der Beitragsberechnung überführten wir in FIT-Testfälle und hatten so eine kontinuierlich wachsende Menge automatisierter Akzeptanztests.

Die Fachabteilungen fanden anfänglich tatsächlich ein paar Fehler in der Beitragsberechnung. Danach waren die meisten gemeldeten Fehler aber keine Fehler im System, sondern wurden durch die Vorgehensweise beim Testen verursacht: Die Tester suchten sich bestimmte Fälle heraus und berechneten diese mit dem existierenden Hostsystem, um den erwarteten Wert zu bekommen (»der Host hat immer Recht«). Anschließend gaben sie dieselben Werte in unser System ein und verglichen die Werte. Das war allerdings eine nicht triviale Aufgabe. Konnte man z.B. in der Webanwendung »Hauseigentümer« aus einer Drop-down-Liste auswählen, musste man im Host z.B. »02« in ein Feld namens »HET« eintragen. Das führte zu häufigen Eingabefehlern, wodurch Hostwerte und die Werte der Webanwendung logischerweise voneinander abwichen.

Diese Fehler schlugen natürlich trotzdem zunächst einmal bei uns auf. Wir wollten aber das System fertigstellen und kein Ticket-Ping-Pong spielen, wenn wir ein Ticket mal nicht auf Anhieb nachvollziehen konnten. Wir etablierten daher ein Verfahren, mit dem die Fehler schnell eingekreist werden konnten. Wir vereinbarten mit den Testern, dass sie als Anlage zu den Tests Screenshots der Hostmasken und der Webanwendung beifügten, aus denen wir erkennen konnten, was sie konkret eingegeben hatten. Wurde ein neuer Fehler gemeldet, haben wir zunächst die Werte von den Screenshots der Webanwendung in unsere FIT-Tests übertragen. Die Ausführung der FIT-Tests haben wir um umfangreiche Log-Funktionen erweitert, sodass wir aus den Protokollen direkt die Werte ablesen konnten, die in das Hostsystem hätten eingegeben werden müssen. So mussten wir nur noch die Logs mit den Host-Screenshots vergleichen, um herauszufinden, ob fehlerhafte Eingaben zu dem Problem geführt haben.

Natürlich wäre alles viel einfacher gewesen, wenn die Fachabteilungen direkt mit FIT gearbeitet hätten. Wir hatten allerdings leider nicht die Zeit, die Fachabteilungen in FIT zu schulen und ihnen die benötigte Infrastruktur aufzusetzen.

### 8.3.5 Die Releases

Auch wenn es nur Fake-Releases waren, sorgten diese für eine große Sicherheit im Team. Es war bereits nach kurzer Zeit am Burndown-Chart ablesbar, dass wir mindestens das erste Release problemlos bis zur Deadline fertigstellen könnten. Wir machten unsere interne Releaseaufteilung sowie den Fortschritt gegenüber dem Kunden transparent, und kontinuierlich wurde er der Idee gegenüber aufgeschlossener. Ihm wurde klar, dass auch so für ihn Sicherheit generiert wurde. Möglicherweise würde er nicht den kompletten Funktionsumfang zur Deadline erhalten, aber er würde ganz sicher nicht ohne Hosen dastehen.

Möglicherweise hatte der Kunde ursprünglich die Befürchtung, dass das Zulassen von Releases den Druck auf das Team so weit reduzieren würde, dass es nicht mehr mit Hochdruck an den Folgereleases arbeiten würde. Das wäre eine Fehlannahme gewesen. Wir haben die ganze Zeit über mit Hochdruck gearbeitet und letztlich alle Releases bis zur Deadline geliefert.

### 8.3.6 Zwei neue Entwickler

Die Integration neuer Entwickler ist immer eine anspruchsvolle Aufgabe. Werden sie spät integriert, kann der Effekt sogar negativ sein (siehe »The Mythical Man-Month« von F. Brooks). Auch vor diesem Hintergrund hatten wir das Weak-Ownership-Modell für Quellcode gewählt. Die neuen Entwickler mussten sich nicht in den kompletten existierenden Code einarbeiten und konnten sich innerhalb ihres Zuständigkeitsbereichs »ihre Welt« schaffen.

Wir erkauften uns die leichte Integrierbarkeit der neuen Entwickler dadurch, dass die verschiedenen Zuständigkeitsbereiche nach unterschiedlichen Philosophien entwickelt wurden.

Rückblickend erscheint mir diese Strategie als sehr sinnvoll, auch wenn sie dazu geführt hat, dass letztlich der Code eines Zuständigkeitsbereichs komplett neu geschrieben wurde. Das passierte allerdings erst nach der Deadline.

## 8.4 Der Projektabschluss

Wir haben den gewünschten Funktionsumfang zum gewünschten Termin geliefert und einige kleinere Änderungswünsche während der Projektlaufzeit bereits integriert. Dabei haben wir den initial geschätzten Aufwand sogar ungefähr eingehalten. Vor diesem Hintergrund war das Projekt ein großer Erfolg.

Allerdings ist der Vertrag mit dem Kooperationspartner nicht zustande gekommen. Der Kunde beschloss zunächst, das System trotzdem in Betrieb zu nehmen. Vorher schienen allerdings noch einige Änderungen notwendig, die durch mehrere Change Requests umgesetzt wurden. Wegen interner Unstimmigkeiten beim Kunden wird das System allerdings bis heute nicht eingesetzt. Das Produkt war damit leider kein Erfolg.

Nach dem Projekt haben wir im Team eine Abschluss-Retrospektive durchgeführt. Einige der Ergebnisse sind im Folgenden aufgelistet:

- Festpreisprojekte mit agilen Vorgehensweisen können funktionieren, wenn die verwendeten Technologien gut verstanden sind, sich Anforderungen nur moderat ändern und das Team sich zumindest in Teilen kennt.
- Die Überstunden an den Projektanfang zu stellen statt ans Projektende wurde von allen Entwicklern als sehr positiv bewertet.
- Strikte testgetriebene Entwicklung führte zu einem sauberen Design mit sehr hoher Testabdeckung, das uns mehrfach größere Refactorings sehr leicht machte.
- Weak-Code-Ownership führte zu einigen Unschönheiten bezüglich der Homogenität des Codes, schien aber ein sinnvoller Kompromiss, um die Einarbeitungszeit der neuen Entwickler zu minimieren.

Interessant ist eine Betrachtung der Entwicklungsgeschwindigkeit. Nach einer internen Analyse war in dem Projekt die Entwicklungsgeschwindigkeit höher als in allen anderen Projekten, die wir vorher durchgeführt hatten. Allerdings kann man an dem Projekt sehr schön sehen, dass dies für den Gesamtdurchsatz im Grunde irrelevant ist: Zu Beginn des Projekts stand der Kooperationsdeal über allem. Das Projekt hatte »Vorstandsattention« und damit war die Priorität gegenüber den anderen Projekten im Unternehmen klar. Das haben wir jeden Tag immer wieder erlebt. Wenn wir eine Auskunft brauchten, hatten wir die nach wenigen Minuten. Wenn wir eine Änderung an einem Austauschdatenformat benötigten, war das eine Sache von wenigen Tagen. Wenn wir morgens den Bedarf für einen neuen Rechner angemeldet haben, stand der nachmittags in unserem Büro. Der Kunde hat sich sehr reaktionsfähig gezeigt.

Als der Kooperationsdeal geplatzt war und wir an den Change Requests arbeiteten, fiel das Projekt in der Priorisierung zurück in die Stufe »undefiniert«, die alle anderen Projekte im Haus auch hatten. Und auf einmal mussten wir Wochen auf Informationen oder Zulieferungen warten und Unmengen Formulare ausfüllen, um einen defekten Monitor zu ersetzen. Das drosselte die Entwicklungsgeschwindigkeit enorm und es zeigt, dass es nicht sinnvoll ist, agile Entwicklung isoliert für ein Entwicklungsteam oder die IT zu betrachten. Die Abhängigkeiten zum Rest des Unternehmens sind so groß, dass sich nur begrenzt eine Steigerung der Entwicklungsgeschwindigkeit feststellen lässt.

## 8.5 Agile Werte im Projekt

| Individuen und Interaktionen | vor | Prozessen und Tools |
|---|---|---|
| ▲<br>Es gab von Kundenseite kaum Vorgaben für Prozesse und Tools – lediglich die Verwendung von Lotus Notes für E-Mails war verpflichtend. Das Team hat selbst über Prozesse und Tools entschieden und die Verwendung angepasst, wenn das sinnvoll erschien. | | |
| **Laufende Software** | **vor** | **Ausführlicher Dokumentation** |
| ▲<br>Es wurde nur das dokumentiert, was notwendig war, um das Produkt zu entwickeln. Es gab insbesondere keine Prozessdokumentation, mit der jemand bei einem Fehlschlag hätte versuchen können, alle Schuld von sich zu weisen. Wir sind auch nur sehr selten irgendwelchen Antragsformularen begegnet. | | |
| **Zusammenarbeit mit dem Kunden** | **vor** | **Vertragsverhandlungen** |
| ▲<br>Beide Vertragspartner haben leidenschaftlich daran gearbeitet, gemeinsam ein nützliches Produkt zu entwickeln. An einigen Stellen hat die Festpreiskonstellation aber seinen Tribut verlangt, als es z. B. um Änderungswünsche/Featurewünsche ging, die der Kunde während des Projektverlaufs äußerte. | | |
| **Reagieren auf Veränderungen** | **vor** | **Befolgen eines Plans** |
| ▲<br>Der Prozess wurde mehrfach adaptiert, das Featureset war aber aufgrund der Festpreis-konstellation nur bedingt anpassbar. | | |

Henning Wolf, Agile Projekte mit Scrum, XP und Kanban , dpunkt.verlag, ISBN 978-3-86490-266-6

# 9 Kanban – so starten Systemadministratoren

Susanne Reppin

*Die XING AG setzt ihre Karten auf Scrum und das nicht erst seit Kurzem. Aufgrund der extrem guten Erfahrungen mit Scrum entschieden wir uns, auch Scrum für das Maintenance Team (kümmert sich um xing.com und dessen Lauffähigkeit) einzusetzen, anfänglich. Ralf Wirdemann, freiberuflicher agiler Coach, brachte die Idee auf, Kanban auszuprobieren. Das Team ist damit heute so erfolgreich, dass weitere Teams folgten und folgen. Das Team »Internal System Administration« traut sich als erstes »Nicht-Software-Team« an das Thema Kanban heran. Wie dieses Team beginnt darüber nachzudenken, die Entscheidung trifft und dann mit Kanban zu arbeiten anfängt, das wird in diesem Beitrag geschildert.*

## 9.1 Der Samen »Kanban« wird gesät

Kai Lippok, der Teamleiter des Teams »Internal System Administration« sprach mich an: »Was hältst du davon, wenn wir mit dem Team Kanban machen?« Ich entgegnete: »Wow, mutig. Bisher haben wir nur Erfahrungen mit Teams in der Softwareentwicklung und nur mit einem Team mit Kanban. Aber warum nicht, lass uns ein paar Fragen klären und dann sehen wir weiter, ob wir es beide als sinnvoll erachten.« Kai stimmte zu.

Angeregt war das ganze Thema durch Johannes Mainusch, Vice President Operations bei der XING AG, der ebenfalls überzeugt ist, dass Kanban für viele Teams – mit den unterschiedlichsten Strukturen und Kulturen im IT-Bereich – funktionieren kann und der das Thema Kanban inzwischen hilft voranzubringen.

***Abb. 9–1*** *Das Team »Internal System Administration« im Daily Standup. V.l.n.r.: Patrick Pampe (verdeckt), Bjarne Sander, Carlo Hohenbrink, Kai Lippok, Christopher Maak, Oliver Stoldt.*

Für dieses spannende Thema fanden Kai und ich noch am selben Tag Zeit und setzten uns zusammen. Ich, im folgenden Text »Susanne«, stelle meine Fragen und Kai antwortet.

*Susanne*: »Warum Kanban?«

*Kai*: »Die Kollegen im Maintenance Team sprechen immer wieder davon, dass Kanban so toll ist und sie sich damit wohlfühlen und so unglaublich viel schaffen und so gut zusammenarbeiten. Da dachten wir (Johannes und Kai), dass Kanban vielleicht auch für uns was wäre, auch wenn wir andere Aufgaben haben und kein Softwareteam sind.«

*Susanne*: »Wo drückt dich oder euch der Schuh? Was versprichst du dir für dich und dein Team von Kanban für eure tägliche Arbeit im Team?«

*Kai*: »Oft wissen wir nicht so recht, was genau ansteht, was heute wichtig wäre. Strategische Themen werden auf die lange Bank geschoben und bekommen keinen Raum, weil die tägliche Arbeit Vorrang bekommt. Manchmal wissen

wir nur, dass etwas nicht gut läuft, wissen aber nicht so genau warum? Und wir wissen manchmal gar nicht, was wir geschafft haben, obwohl wir dauernd im Stress sind. Wir brauchen mehr Transparenz: Wer macht was? Was steht an? Wo hängt es? Schaffen wir die Arbeit? Wie können wir strategische Themen und deren Umsetzung mit Alltagsthemen kombinieren?

Zudem haben wir lauter Spezialisten im Team und wenn einer mal ausfällt, haben wir ein Problem. Wir wollen voneinander lernen, um unser Wissen zu teilen, um auch mal für den anderen einspringen zu können. Und Wissen bringt uns schließlich alle weiter und macht unseren Job interessant.«

*Susanne*: »Kanban könnte helfen. Zumindest bei der Visualisierung, also bei der Transparenz all der genannten Dinge. Ein Ziel von Kanban ist tatsächlich, möglichst alles sichtbar zu machen. Ein anderes besteht darin, die Arbeit zu limitieren, also so wenig wie möglich parallel zu machen. Wenn wir zusätzlich das Thema »Agilität« fördern, die Selbstverantwortung des Teams, gemeinsames Erledigen und Lernen, um nur einige Aspekte zu nennen, dann könnte auch die Wissensverteilung im Team gefördert werden.«

Kai und ich sprachen noch eine ganze Weile über Details, über das Team, über seine Arbeit als Teamleiter, der gleichzeitig kräftig mit anpackt, und über »Agilität« und mögliche praktische Ansätze. Am Ende schien alles zu passen und Kanban ein Versuch wert zu sein.

Unsere Zusammenfassung fürs Erste: Kanban visualisiert und die Kombination »Agiles Kanban-Team« könnte das Zusammenarbeiten und die Wissensverteilung stärken, die Fokussierung auf die wichtigsten Dinge fördern und die strategischen Themen könnten geplant einfließen. Kais Vorgesetzter befürwortet bereits »Agiles Vorgehen«, »Agile Teams« und »Kanban«. »Agile Softwareentwicklung« ist seit einigen Jahren ein aktives Thema im Unternehmen, was zusätzlich hilft. Die Frage also, ob das Vorhaben im Management Befürwortung finden würde, war bereits geklärt, ein Glücksfall.

## 9.2 Die Vorbereitung des Vorschlags für das Team

Doch bevor es losgehen sollte, wollten wir noch die Rollen und ein anfängliches Szenario klären und festlegen. So hofften wir, dem Team einen runden Vorschlag zu präsentieren, über den wir diskutieren können.

### 9.2.1 Um dieses Team geht es hier ...

**Team »Internal System Administration«**

Kanban in der Praxis: seit Januar 2010 bis heute

Thema im Team: Die Aufgabe der IT-Administratoren ist klar und komplex zugleich. Neue Kollegen und neue Teams bei der XING AG verlangen zusätz-

lich nach neuen Lösungen, beispielsweise umfangreiche Testumgebungen und Server für zweckgebundene Software. Die Aufgabe des sechsköpfigen IT-Admin-Teams besteht darin, die Kollegen bestmöglich zu unterstützen und das Know-how im Team gleichmäßig zu verteilen. Konkret reicht die Palette vom »Helpdesk« über »ein neuer Rechner für einen neuen Mitarbeiter« bis hin zu technischen infrastrukturellen Themen wie »Mailserver«. Ad-hoc-Anfragen stehen auf der Tagesordnung. Die Administratoren springen von Aufgabe zu Aufgabe und löschen viele Herde gleichzeitig. Der dadurch bedingte Alltagsstress macht das Team oft unzufrieden. Sie selbst fragen sich, ob unter diesen Bedingungen nicht die Qualität leidet.

Teamgröße 2011: 4 Administratoren, 1 Auszubildender, 1 Teamleiter

### 9.2.2 Die Rollen zur Einführung von Kanban

Das Team bearbeitet die Aufgaben und organisiert sich selbst.

Das Team: 4 Administratoren plus ein Teamleiter (Beim Start mit Kanban hatte der Auszubildende noch nicht bei der XING AG begonnen. Heute ist er ebenfalls Teammitglied im Kanban-Team.)

Der Kanban-Coach: So nennen wir den agilen Coach speziell für ein Kanban-Team im Unternehmen. Er ist das Pendant zum ScrumMaster in Scrum-Teams. Der Kanban-Coach ist für dieses Team der Teamleiter Kai, er priorisiert die Aufgaben und arbeitet teilweise an Aufgaben mit.

Dass Kai, der Teamleiter, die Rollen Kanban-Coach und Teammitglied gleichermaßen zu bewältigen hat, stellt eine besondere Herausforderung dar, die später im Text besprochen wird.

Agiler Coach: Da kein Teammitglied, inklusive Kai, Erfahrung mit Kanban hat, begleitet das Team und auch Kai ein agiler Coach, Susanne, jedoch nur so lange, bis im Team genug Erfahrung vorhanden ist.

### 9.2.3 Die Werkzeugkiste

Details zu den Werkzeugen wollen wir mit dem Team erarbeiten, diese werden im folgenden Abschnitt näher beschrieben. Zur Vorbereitung erfolgt die Festlegung, welche Werkzeuge wir benutzen wollen:

- Ein Kanban-Board, oder auch Teamboard genannt, soll möglichst viel sichtbar machen.
- Daily Standups vor dem Kanban-Board sollen helfen, gemeinsam den Tag zu organisieren.
- Retrospektiven sollen das ständige Lernen und eine stetige Verbesserung ermöglichen.

### 9.2.4 Der Plan für das Treffen mit dem Team

- Kai
  - Beschreibung der Motivation
  - Ziel für das Team
- Susanne
  - Veränderungen mit Kanban für den Alltag
  - Vorgehensweise »ständiges Lernen« und »Finden des passenden Wegs«
  - Grundzüge Kanban
  - Philosophie agiles Team
  - Philosophie Lean Management
  - Werkzeugkiste
- Alle
  - Entscheidung, ob alle dafür sind, es mit Kanban für mindestens drei Monate zu versuchen.

## 9.3 Der Vorschlag wird dem Team vorgestellt

Kai und ich haben unseren Vorschlag gut vorbereitet und wollen ihn nun mit dem Team besprechen.

Wir geben uns allen eine Stunde Zeit, um die Idee dem Team vorzustellen und um darüber zu entscheiden, ob wir den mutigen Schritt »Kanban« wagen.

Alle vier Administratoren sind gekommen und gespannt, was wir ihnen erzählen würden und was es für Veränderungen geben würde. Wie es zur Idee kam, ist schnell erzählt. Kai erklärt nun, was er sich als Ziel vorstellt, seine Hoffnungen und seine Erwartungen. Was soll uns das Ganze bringen? Das Team hört aufmerksam zu. Am Ende fasst Kai das Ganze noch einmal zusammen. Das könnte unser gemeinsames Ziel sein:

Wir wollen alle Aufgaben zeitnah und mit hoher Qualität zur besten Zufriedenheit erledigen … »Do it!«

Danach komme ich an die Reihe. Warum bin ich eigentlich hier? Es geht um Kanban. Da aber keiner im Team Erfahrung mit Kanban hat, würde ich das Team und Kai begleiten. Ich erläutere, bevor ich genauer auf das Thema Kanban eingehe, was es für konkrete Veränderungen geben würde, und konzentriere mich auf die Veränderungen im täglichen Ablauf. Es wird zukünftig ein Teamboard geben, vor dem wir uns täglich kurz am Morgen versammeln und den Tag organisieren. Alle zwei bis drei Wochen wird eine Retrospektive (1 Stunde) durchgeführt, in der wir gemeinsam schauen, was gut lief und was nicht so gut lief und wie wir Letzteres verbessern können. Ansonsten wird der Alltag erst einmal nicht verändert sein. Die Anwesenden nicken. Dennoch spüre ich Skepsis, was mich nicht beunruhigt. Zu viele offene Fragen schweben noch im Raum.

Da keiner im Team Erfahrung mit Kanban hat, schicke ich ein paar Worte voraus zur Art und Weise, wie wir vorgehen werden. Wir werden nicht den Anspruch haben, von Beginn an perfekt zu sein, darum geht es auch nicht, sondern um ständiges Lernen und stetige Verbesserung. Wir schauen uns also regelmäßig an, was wirklich gut lief, um es beizubehalten, und wir betrachten, was nicht so gut lief, um es zu verbessern. Das wird in den Retrospektiven geschehen. So nähern wir uns immer mehr dem genau passenden, effektiven und sinnvollen Weg an für genau dieses Team und dessen Aufgaben. Zu Beginn werden wir ein Setting, eine Vorgehensweise, vorschlagen, auf das wir uns einigen. Mit der Zeit werden wir die Anpassungen vornehmen, und zwar gemeinsam. Nichts wird geschehen, auf was sich das Team nicht geeinigt hat.

Folgendes soll also unser Motto sein:

> *Heute sehen wir diese Lösung als beste Lösung an, so lange, bis wir Erfahrung damit gesammelt haben und es eventuell besser wissen und dann verändern werden.*

Das gefällt allen, auch wenn es wenig konkret ist. Immer noch stochern wir ja ein bisschen im Nebel. Ein klein wenig Geduld noch, wir haben noch 40 Minuten, um eine Entscheidung zu fällen.

#### Die Philosophie in Grundzügen

Wir tauchen kurz in die Philosophien ein, nur so viel, um ein kleines Bild zu schaffen und einen Samen zu säen. Was bedeutet es, ein agiles Team zu sein? Was bedeutet »Lean Management«? Was sind die Grundgedanken bei »Kanban«?

Es gibt ein paar wenige wichtige Komponenten, wenn man Kanban einsetzen möchte. Ohne die Umsetzung dieser vier Komponenten wäre es kein Kanban:

1. Visualisierung:
   Die Aufgaben werden visualisiert, sodass jeder im Team weiß, an was gerade gearbeitet wird, was bald zu arbeiten ist, was gerade »brennt« und wer sich darum kümmert.
   Konkret:
   Wir werden mit einem Teamboard arbeiten, an dem die Aufgaben als Karten aufgehängt werden.
2. Limitierung:
   Genauer »Limitierung der Aufgaben, die in Bearbeitung sind«.
   Fachvokabel:
   limit your WIP – limitiere dein »Work in Progress«.
   Konkret:
   Wir werden also gleich zu Beginn eine Anzahl dafür festlegen, wie viele Aufgaben maximal parallel »in Bearbeitung« sein sollen. Auch diese Zahl werden wir am Teamboard zeigen.

3. Durchlaufzeit:
   Die Beobachtung der Durchlaufzeiten der Aufgaben ermöglicht es, sich zu verbessern, und kann ein Steuerungs- sowie Planungselement sein.
   Konkret:
   Sobald die Arbeit für eine Aufgabenkarte begonnen wird, schreibt der Verantwortliche seinen Namen darauf und wann er begonnen hat. Der Name bedeutet nur »Ich kümmere mich darum«. Wer alles an der Aufgabe arbeitet, wird gemeinsam festgelegt und/oder ergibt sich während der Bearbeitung. Ist die Aufgabe erledigt, wird das Enddatum auf die Karte geschrieben.
4. David J. Anderson beschreibt in seinem Buch »Kanban – Evolutionäres Change Management für IT-Organisationen« des Weiteren »Mache die Regeln für den Prozess explizit« und »Verwende Modelle, um Chancen für Verbesserungen zu erkennen«. Diese zwei Punkte sollen durch die Retrospektiven und durch das Begleiten des agilen Coachs umgesetzt werden. Wir definieren den anfänglichen Prozess, an den sich das Team halten möchte, und implementieren geeignete Vorgehensweisen, die uns die Chancen erkennen lassen. Sowohl der Prozess als auch die obengenannten »Modelle« werden sich mit der Zeit ändern im Sinne des ständigen Lernens und der Verbesserung.

Wir beschließen noch kurz, über das Thema »Agiles Team« zu sprechen, um dann gemeinsam das Teamboard zu skizzieren oder, wenn noch Zeit ist, das Teamboard gleich aufzubauen. Ich schaue auf die Uhr, die Zeit rennt, wir haben noch 20 Minuten.

Weitere wichtige Komponenten zum agilen Team:

5. Das Team soll und darf eigenverantwortlich arbeiten und das »Wie?« selbst bestimmen.
6. »Pull« sollte konsequent umgesetzt werden:
   Das Team organisiert sich die Arbeit selbst. Aufgaben werden priorisiert, nach Wichtigkeit sortiert und dann vom Team von oben vom Stapel genommen. Aufgaben werden also nicht mehr an einzelne Mitarbeiter zugewiesen.
7. Das Team bekommt Zeit, um sich zu besinnen, um zu lernen – in Form von Retrospektiven.
8. Das Management vertraut dem Team.

Das Team ist sich einig: Kai lässt uns bereits mitentscheiden, diskutiert mit uns Themen und wir denken jetzt schon immer nach, was wir besser machen könnten.

Kai wirft ein, dass oft gute Vorschläge im Alltag untergehen oder doch wieder in Vergessenheit geraten, und er möchte dem Team noch mehr Handlungsspielraum lassen. Ich bin begeistert und teile das allen mit. Das sind hervorragende Voraussetzungen für ein Gelingen.

Es geht auch um »Flow« und darum, Wartezeiten zu vermeiden.

Was ist das und warum ist das so wichtig?

Unter Flow versteht man den Fluss, während man eine Aufgabe erledigt. Wir alle fühlen uns wohler, wenn wir nicht unterbrochen werden, nicht warten müssen, wenn wir eine Aufgabe erledigen, bis wir sie beenden können. Warten ist langweilig, ärgerlich und belastet, weil wir weiterhin dran denken müssen. Arbeiten wir fließend, so arbeiten wir meist effizienter und die Qualität erhöht sich. Zudem macht es uns glücklich, eins nach dem anderen zu erledigen und Dinge fertig zu machen. Es geht also um den »Flow«.

Deshalb möchten wir Wartezeiten erkennen und die Ursachen dafür beheben. Das wird also auch unser Ziel im Team der Administratoren sein.

Meist deuten Aussagen, die im Daily Standup getroffen werden, darauf hin, dass hier gewartet werden muss: »Ich muss nun warten, bis Herr X mir Bescheid gibt« oder »Ich muss warten, bis die Bestellung geliefert wird«. Der erste Schritt ist, die Wartezeiten sichtbar zu machen.

Konkret: Wir werden Wartezeiten erst einmal durch sogenannte »Blockkarten« am Teamboard sichtbar machen. Ein kleiner Post-it-Zettel wird auf die Karte geklebt, worauf steht, wann begonnen wurde zu warten und der Grund, warum gewartet wird. Die Ursache und dann die Lösung können vielfältig sein. Die Ursachenforschung und Lösungsfindung werden wir in den Retrospektiven betreiben.

Die Anwesenden stellen erstaunlicherweise nicht viele Fragen, sondern hören aufmerksam zu. Meine Ausführungen sind sehr kurz und auf einige wesentliche Aspekte begrenzt. Wir werden noch viel Zeit in der täglichen Arbeit miteinander haben, um tiefer in die Themen einzusteigen, dann wenn sie auftreten und passen. Ich habe die Erfahrung gemacht, dass ein Grundstock genügt, damit der Rahmen klar ist und die Richtung aufgezeigt werden kann. Mehr erschreckt manchmal und so werden wir in der Zukunft immer wieder kurz auf Themen eingehen und vielleicht später, in Abstimmung mit dem Team, noch einmal das eine oder andere Thema vertiefen.

Also beantworten wir alle auftretenden Fragen gleich und möglichst kurz. Oft heißt die Antwort: »Wir werden sehen und das gemeinsam entscheiden, das Team entscheidet.«

Skepsis kommt auf: »Werden wir nicht mehr diskutieren als arbeiten?«

Ich bin ehrlich. Anfänglich könnte es schnell so scheinen. Wir werden allerdings die Zeiten der Diskussionen begrenzen. Wir werden Themen/Probleme/Verbesserungen/Fragen direkt in den Daily Standups oder gleich anschließend daran besprechen, aber nicht länger als 10 Minuten täglich. Und wir versuchen die restlichen Themen in die Retrospektiven zu verlegen.

Genug der Philosophien und der Theorie. Der Zeitpunkt der Entscheidung ist gekommen. Es ist ein Versuch, ein Wagnis. Die Frage ist konkret: Haben alle Anwesenden Lust, es auszuprobieren? Wir sagen aber auch, dass ein Versuch

über ca. 3 Monate laufen soll, weil wir glauben, dass man erst dann erkennt, ob es sinnvoll sein kann weiterzumachen.

Wir haben noch 8 Minuten und ich biete an, dass Kai und ich kurz den Raum verlassen, damit sich das Team besprechen kann. Das allerdings möchte das Team nicht, wozu auch, wir reden offen. Ein Teammitglied sagt: »Ich mache mit.« Danach folgen alle anderen.

Nun blieb keine Zeit mehr, das Teamboard gleich aufzubauen. Aber wir haben uns entschlossen, das Wagnis »Kanban für das Team Internal System Administration« einzugehen und sind allesamt froh und gespannt.

Wir vereinbaren, noch heute ein leeres Teamboard im Teamraum aufzubauen. Gleich am nächsten Morgen um 10 Uhr soll das erste Daily Standup stattfinden. Kai bereitet bis dahin einige Aufgaben vor, die seiner Ansicht nach anliegen. Den Rest machen wir dann gemeinsam.

## 9.4 Das neue Kanban-Team startet

Zu Beginn standen also der Wunsch nach Veränderung, der Wunsch nach Verbesserung und die Hoffnung, dass Kanban eine Lösung sein könnte.

Das Aufbauen des leeren Teamboards ist schnell erledigt.

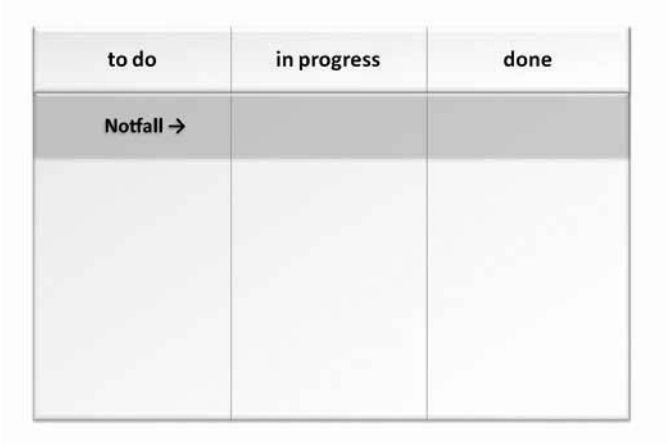


***Abb. 9–2*** *Das leere Teamboard*

#### Das Teamboard

Das Teamboard besteht aus bunten Post-its auf einer freien Wand und den drei Spalten »to do«, »in progress« und »done«.

Die Aufgaben sollen auf Karten geschrieben werden, sodass man weiß, worum es geht. Woher kommen die Aufgaben, fragt sich das Team: Der Kanban-Coach und Teamleiter Kai macht einfach das, was er bisher auch gemacht hat,

nur visualisiert am Teamboard. Er schreibt die Aufgaben auf Karten und entscheidet täglich neu, welche Aufgaben als Nächstes bearbeitet werden sollen.

Am nächsten Morgen treffen wir uns alle wieder um 10 Uhr. Kai hat ein paar Karten mitgebracht, auf denen er Aufgaben notiert hat. Doch zuerst möchte ich gerne von jedem wissen, woran er gerade arbeitet. Jeder erzählt kurz, was das ist, schreibt eine Karte mit dem Datum, wann er ca. damit begonnen hat, und notiert seinen Namen darauf. Alle Karten werden in die Spalte »in progress« gehängt, da die Bearbeitung bereits begonnen hat. Dann erklärt Kai seine mitgebrachten Aufgabenkarten und hängt sie sortiert in die Spalte »to do«.

Die erste Übersicht ist fertig und zeigt, was in Bearbeitung ist und was als Nächstes ansteht.

Alle anderen Aufgaben, inklusive der strategischen Themen, die heute noch nicht in der »to do«-Spalte landen, hat Kai an die Schranktüren neben dem Teamboard gehängt. Wir nennen das »Backlog«, was sicherlich von den vielen Scrum-Teams im Unternehmen kommt. Aus diesem Backlog wird Kai Karten auswählen und sie in die »to do«-Spalte hängen, sobald dort wieder Platz ist. Das ist bei diesem Team möglich, da das »Backlog« aus einer übersichtlichen Menge besteht.

Nun werden die Spalten limitiert, im Kanban-Kontext wird das »WIP – Work in Progress« genannt. Das bedeutet, das Team definiert zum Beispiel für die Spalte »in progress« eine maximale Anzahl von Karten, die in dieser Spalte hängen dürfen. So wird die Anzahl der Themen, die parallel in Arbeit sind, limitiert. Wie viele das sind, ist zu Beginn reines Bauchgefühl.

Für die »to do«-Spalte entscheidet sich das Team für so viele Karten, wie das Team selbst glaubt, höchstens bis zum nächsten Daily Standup bearbeiten zu können.

Für die »in progress«-Spalte wählt das Team die Anzahl fünf, also die Anzahl der Teammitglieder, inklusive Kai, der auch Karten bearbeiten möchte.

Die Spalte »done« wird nicht limitiert. Sie wird ab und zu geleert. Das Team entscheidet, dies jeweils zur Retrospektive zu tun.

Eine Notfallzeile wird eingerichtet, im Kanban-Kontext meist »Expedite-Lane« genannt. Sie ist für Notfälle bestimmt, die wichtiger und dringender sind als alles andere, was am Teamboard hängt. Sie wird also über allen anderen Karten als oberste Zeile platziert. Wir vereinbaren, dass Kai oder Einzelne im Team entscheiden können, was ein Notfall ist. Tritt ein Notfall ein, sollen alle im Raum Anwesenden kurz darüber informiert werden. Eine Karte soll geschrieben und aufgehängt und es soll direkt mit der Bearbeitung begonnen werden (später mehr zu Notfällen).

Uns ist allen jetzt schon klar, dass Kai mit seinen vielen Aufgaben und zusätzlichen Rollen bei Kanban sehr beschäftigt sein wird. Deshalb soll das Team ihn unterstützen beim Thema Durchlaufzeit und Statistik. Damit dies keine Belastung für das Team wird, versuchen wir einen neuen Weg zu gehen. Die Karten, die in der Spalte »done« landen, werden wie folgt vorsortiert. Ich frage das Team:

»Was meint ihr, wann beginnt ihr, euch über Karten zu ärgern, oder wann werdet ihr ungeduldig, wenn eine Karte lange in der Spalte ›in progress‹ hängt? Nach wie vielen Tagen?« Erst unschlüssig und dann doch recht zügig entwickelt sich ein gemeinsames Gefühl. Wenn etwas länger als 7 Tage dauert, dann nervt die Aufgabe. Die Spalte »done« wird in zwei Unterspalten aufgeteilt: »≤7 Tage« und »>7 Tage«. Besprochen werden in Zukunft nur die Karten, die in der Spalte »>7« landen, denn nur diese nerven. Für diese werden wir versuchen, eine Lösung zu finden, wie wir zukünftig eine kürzere Durchlaufzeit für diese Art von Karten erreichen können. So braucht keiner Statistik zu führen.

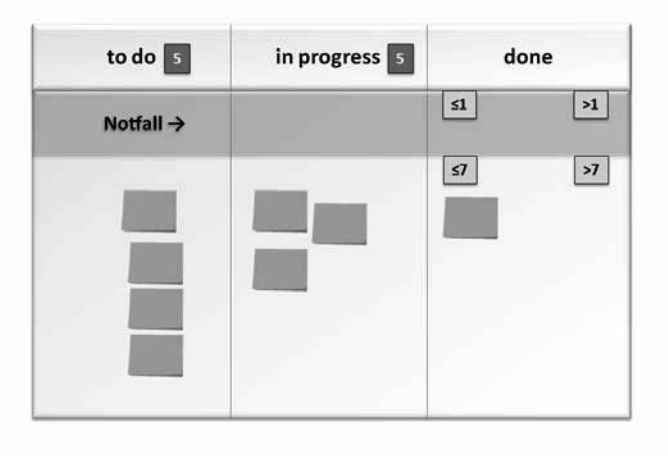


***Abb. 9–3*** *Erstbestückung des Teamboards*

Für heute sind wir fertig. Keiner hat mehr Fragen, sondern möchte mit der Arbeit fortfahren.

### Daily Standups

Wir treffen uns nun jeden Tag zur selben Uhrzeit vor dem Teamboard, besprechen die Aspekte, die bei der Arbeit seit dem letzten Standup anfielen, und was für den heutigen Tag geplant ist. Wer arbeitet mit wem zusammen? Wer kann wem helfen? Hat jemand eine Idee zur Umsetzung und/oder Problemlösung? Neue Aufgaben, von denen nur immer fünf Stück auf dem Board sein dürfen, werden von Kai aufgehängt, der auch die Aufgaben priorisiert, erläutert und alle offenen Fragen dazu geklärt. So ist sichergestellt, dass die Aufgabe klar ist, wenn sie vom Team von der Spalte »to do« in die Spalte »in progress« gehängt wird, also wenn begonnen wird, die Aufgabe zu bearbeiten.

## 9.5 Der Start des Teams – Zusammenfassung

Nach ein paar Gesprächen im Vorfeld von Kai und mir, einer einstündigen Besprechung, in der sich das Team für Kanban entscheidet, und einem verlängerten morgendlichen Standup, ist das Thema für das Team installiert und gestartet.

Das bedeutet: Ein Anfangsprozess ist installiert, die Rollen sind klar, alle im Team haben sich darauf verständigt, und das Teamboard visualisiert den Fluss der Arbeit, wenn auch die Spalten noch nicht haargenau den Arbeitsprozess darstellen. Die Menge an begonnener Arbeit (WIP) ist begrenzt und das Messen der Durchlaufzeit ermöglicht. Es ist ein Anfang. Durch die Retrospektiven und die täglichen Erfahrungen veränderte sich dieses Anfangsszenario, das Teamboard und sogar die Rollendefinition.

## 9.6 Unsere Erfahrungen

Zu diesem Zeitpunkt beginnen auch die Erfahrungen für das Team und für mich. Diese möchte ich anhand von Themen mit Ihnen teilen.

**Weniger ist manchmal mehr zu Beginn. Es ist ein Balanceakt: Wie detailliert sollten die Spalten des Teamboards den Arbeitsprozess widerspiegeln?**

Das Team startete mit einer Spalte »in progress« und versuchte, durch Blockkarten erst einmal die schlimmsten Wartezeiten zu bekämpfen. Mit der Zeit entstanden weitere Spalten wie »Test« und ein »Parkplatz«. Eine extra Zeile »New Employees«, direkt unter der Notfallzeile (Expedite), wurde für alle Aufgaben für neue Mitarbeiter geschaffen. Neue Mitarbeiter beginnen an einem festgelegten Tag und benötigen dann sofort alle technischen Arbeitsmittel, wie Macbook, iPhone usw. Bis zu einem festgelegten Datum muss das Administratorenteam also alles bereitstellen, was an Technik benötigt wird. So werden diese Aufgaben mit festem Termin vorrangig bearbeitet, pünktlich fertig, aber gelten nicht als Notfall. Ergebnis: Gute Transparenz für diese Aufgaben mit Terminen.


***Abb. 9–4*** *Das heutige Teamboard des Teams »Internal System Administration«*

**Karteninhalte sollen klar sein und die richtige Größe des Umfangs muss gefunden werden.**

Alle Aufgaben unter 10 Minuten Arbeit werden einfach gemacht. Das betrifft hauptsächlich die Helpdesk-Aufgaben. Doch wenn Planungen nötig werden oder konzentrierte Arbeit an einem Konzept erforderlich ist, werden Karten geschrieben, damit die Aufgaben geplant und geschützt umgesetzt werden können. Es ist wichtig, sich bei den Karten/Aufgaben zu überlegen, wann sie fertig sind, also wann man sie auf »done« hängen kann, auch bekannt als »Akzeptanzkriterium«. Als Beispiel sei hier die Erstellung eines Plans angeführt, und zwar dann, wenn der Plan steht und er zur Umsetzung oder Entscheidung bereit ist. Macht man die Karte zu groß, sodass zum Beispiel die Planung und Umsetzung in mehreren Schritten den Inhalt einer einzigen Karte bildet, dann wird schnell klar: Unübersichtlichkeit ist der Preis. Der Flow wird unterbrochen durch andere wichtige Tätigkeiten und es wird unmöglich, mit einem Blick und einem Satz zu erkennen, wie weit man in der Arbeit vorangeschritten ist. Die Durchlaufzeit erhöht sich in eine Dimension, die das Team nicht wünscht. Sobald also die Planungskarte durchgelaufen ist und in der Spalte »done« hängt, ist es die Aufgabe des Priorisie-

rers, die Umsetzungskarten in die Spalte »to do« zu hängen. So kann ein Umsetzungsschritt nach dem anderen durchlaufen werden.

**Der Helpdesk, immer zeitnah zu bearbeitende Aufgaben, können in das Kanban-Team integriert werden.**

Zu Beginn führten wir ein rotierendes System ein, sodass jedes Teammitglied reihum jeweils für einen Tag die Anfragen, die über den Helpdesk kommen, bearbeitet. Aufgaben, die länger als 10 Minuten dauern, sollten als Kanban-Karte priorisiert am Teamboard einfließen. Zwischendurch gab es auch eine mehrwöchige Erprobungsphase, dass ein Teammitglied vollständig den Helpdesk übernimmt und auch nicht weiter im Kanban-Team integriert sein sollte. Dieses Modell bewährte sich jedoch gar nicht, da zu wenig Informationen bezüglich der Helpdesk-Themen in das Kanban-Team flossen und eine Person mit den Helpdesk-Themen überlastet war. Das Team kehrte wieder zum anfänglichen Modell zurück und integrierte die Helpdesk-Aufgaben wieder in das Kanban-Team. Heute steht auf dem Tisch des momentan für den Helpdesk verantwortlichen Teammitglieds ein Fähnchen, das halbtägig getauscht wird. Die Aufgaben landen ab 10 Minuten Aufwand und mehr wieder am Board für alle.

**Pair Doing: gemeinsames Bearbeiten von Aufgaben ermöglicht fließendes Arbeiten – Flow.**

Sobald ein Teammitglied seine Arbeit unterbricht, weil es zum Beispiel in einem Meeting ist, Urlaub hat oder ein Teilzeit-Teammitglied ist, wartet die Aufgabe auf weitere Bearbeitung. Das hat zur Folge, dass die Aufgabe – bei häufigeren Wartezeiten – nervt und letztendlich langsamer vorankommt und später fertig wird. Zufriedenes Arbeiten und Erreichen einer hohen Qualität sieht anders aus. Die Lösung war schnell geboren: Es arbeiten möglichst zwei Teammitglieder an einer Aufgabe oder ein anderes Teammitglied wird zumindest so mit einbezogen, dass die Aufgabe vom Kollegen weiter bearbeitet werden kann, sobald einer ausfällt.

Bei der schwierigen Anfangskonstellation war sofort aufgefallen, dass Kai als Teamleiter, Kanban-Coach und Priorisierer der Aufgaben auch gleichzeitig bei den Karten mitarbeitete. So sehr sich Kai auch bemühte, er hatte einfach zu wenig Zeit, um seine Karten »fließen« zu lassen. Nun nimmt er sich keine Karten mehr selbst, sondern klinkt sich in die Bearbeitung von Karten ein. So gestaltet er mit, er gibt seine Erfahrung weiter und er stört nicht in dem Sinne, dass Aufgaben warten müssen, bis er wieder Zeit hat.

Die anderen Teammitglieder schauen, bevor sie eine neue Karte nehmen, ob sie helfen oder etwas lernen können, indem sie bei anderen bereits angefangenen Aufgaben mitmachen. Sich und den anderen die Frage zu stellen, ob man helfen

bzw. mitmachen kann, ist wichtig, um entsprechende Möglichkeiten zu erkennen. Auch wenn es dann nicht möglich oder sinnvoll ist, so versäumt man nicht die Chance zu helfen und/oder zu lernen.

Wissensverteilung durch »Pair Doing« (Pendant zum Pair Programming in der agilen Softwareentwicklung) war ein Ziel, das schon durch den Wunsch der Wissensverteilung des Teamleiters im Raume stand. Die Einsicht und der Wunsch, es umzusetzen, war schnell da. Doch wie sollte das geschehen? Zu Beginn scheint alles langsam zu gehen und es ist ungewohnt, mit anderen derart sein Wissen zu teilen. Manchmal ist es sehr einseitig, einer scheint nur zu lernen, der andere nur zu geben, was den Gebenden unter Umständen unzufrieden macht. Die Vorteile sind erst nach einiger Zeit der Umsetzung zu bestaunen. Die Lösung des Teams: Im Team Backlog hing eine ganze Weile eine Erinnerungskarte »mehr Pair Doing«. So konnte jeder sich selbst und die anderen erinnern: »Wir haben uns vorgenommen ...«

Das Teilen von Wissen ist eine Stärke eines Teams und es sind hier viele Aspekte gemeint. Äußere Einflüsse, wie Urlaub, Krankheit, temporäre Abwesenheit, fallen oft mit dringend zu erledigenden Aufgaben zusammen.

Es geht nicht darum, »Spezialistentum« zu vermeiden, im Gegenteil. Spezialisten sind wichtig, um die Qualität der Ergebnisse zu sichern. Fällt der Spezialist allerdings aus, so sollte ein Team in der Lage sein, die Aufgabe, wenn sie dringlich ist, zu erledigen. Ist die Aufgabe nicht dringlich und ist noch nicht begonnen worden, sie zu bearbeiten, dann priorisiert man andere Aufgaben nach oben. Sie kann warten, bis der Spezialist wieder mitarbeitet. Dieses Abwägen ist eine wichtige Aufgabe. Starke Teams tauschen sich auch fachlich aus und erhöhen per se dadurch ihr Wissen und die Genialität ihrer Lösungen. Außerdem dürfen die Faktoren wie Wertschätzung und Spaß nicht vergessen werden. Es macht einfach mehr Spaß, wenn man mehr weiß, mehr kann und das durch »Visualisierung« auch noch vor Augen geführt bekommt.

Deshalb entschloss sich das Team bei größeren Themen, für die eine Einführung ins Thema oder im Nachlauf eine Information für die anderen Teammitglieder notwendig ist, dies in gesonderten Wissensstunden vorzutragen. Teilnehmer sind die anderen Teammitglieder, die dann zukünftig im Pair Doing zu diesen Themen mitarbeiten.

Mit der Zeit tauchten in der Retrospektive immer häufiger Karten auf, wie »Pair Doing mit Kollege machte richtig Spaß und war effektiv«. Die Erfolgserlebnisse halfen dem Team, die Geduld und Kraft aufzubringen, ihren Vorsatz zu »mehr Pair Doing« häufiger umzusetzen.

**Einen agilen Coach für das Team und den frisch gebackenen Kanban-Coach einzusetzen, hat sich bewährt.**

Keiner der Teammitglieder und auch der Kanban-Coach selbst hatten vorher eine praktische Begegnung mit dem Thema Kanban. Deshalb hatten wir uns entschieden, dass ich als Coach das Team begleite und dabei helfe, das Thema und die Philosophie ins Leben zu rufen.

Das fanden wir sehr gut im Nachblick. Ich nahm als Coach an den Daily Standups teil, sonst war ich nur auf Zuruf erreichbar und ich moderierte anfänglich die Retrospektive. Das Team und auch Kai, alle konnten so jegliche Art von Fragen jederzeit stellen und bekamen sie beantwortet. So wuchsen das Team und Kai langsam in das Thema hinein und konnten sich gleichzeitig nach wie vor auf ihre wesentliche Aufgabe als Team »Internal System Administration« konzentrieren.

**Die Rollenverteilung sollte klar sein, auch wenn sie noch so kompliziert und unterschiedlich ist.**

Gleichzeitig Teamleiter und Kanban-Coach zu sein ist aus folgendem Grund schwierig, aber nicht unmöglich. Man stelle sich die Situation in einer Retrospektive vor. Ein Teammitglied spricht etwas an, was nicht gut lief. Im Idealfall ist es auch Selbsterkenntnis, weil Fehler per se nicht schlecht sind, sondern einem dazu verhelfen, es zukünftig besser zu machen. Es ist aber wesentlich einfacher, eigene Fehler oder auch Fehler von anderen anzusprechen, wenn der jeweilige Teamleiter nicht anwesend ist. Denn Fehler könnten sich in Jahresgesprächen und Beurteilungsgesprächen negativ auswirken, so die Befürchtung. In unserem Fall war das auch zu spüren, wurde aber durch das Vertrauen der Teammitglieder zu ihrem Teamleiter und umgekehrt kompensiert. Kai hingegen entschloss sich sogar, von den Jahreszielen, bezogen auf einzelne Mitarbeiter, abzukommen. Stattdessen definiert er heute mit seinen Teammitgliedern den Erfolg zu Themen in der Art, dass gemessen wird, inwiefern das Teammitglied dazu die Lösungen mit den anderen zusammen erarbeitet hat. Es ist dabei unwichtig, wer die besseren Ideen hatte oder wer die Aufgabe umgesetzt hat. Wichtig ist es, das Thema verantwortlich in die Hand zu nehmen und voranzutreiben. Dieses Team, mitsamt Teamleiter, zeigt mit Bravour, dass scheinbar unumstößliche Tatsachen genauso gut auch anders gehandhabt werden können, in diesem Fall das Thema »Jahresziele für Teammitglieder eines agilen Teams«.

**Strategie und Planung der Aufgaben in naher Zukunft sind auch für dieses Administratorenteam wichtig – Backlog.**

Backlog ist ein Begriff aus der Scrum-Welt, den wir einfach übernommen haben, um die Aufgaben zu sammeln, die erledigt werden sollen. Warum ist gerade für ein Administratorenteam das Backlog so wichtig? Täglich tauchen so viele Aufgaben auf, Wünsche und gute Ideen zur Umsetzung, dass ein Administratorenteam sich leicht leiten lässt von diesen gerade aufgetauchten Aufgaben und Ideen. Ein Administratorenteam möchte und muss allerdings auch strategisch denken, die Zukunft planen und größere Themen angehen. Diese Themen werden im Backlog registriert, manchmal nur als Thema. Sobald dieses Thema angegangen werden soll, kann es immer noch ausformuliert und auf mehrere Karten verteilt werden. Im Vorfeld gibt es oft eine Planungskarte fürs Team, aus der dann verschiedene Umsetzungskarten entstehen, die wieder, wie alle anderen Karten, priorisiert und sortiert am Board erscheinen. So können auch größere Aufgaben, in mehrere bis viele Karten aufgeteilt, geplant bearbeitet werden und vor allem gehen sie so nicht unter.

**Die Priorisierung/Sortierung von Aufgaben ist eine sehr verantwortungsvolle Aufgabe.**

Priorisiert werden Aufgaben in der »to do«-Spalte. Wir sagen heute lieber »sortiert«, denn es wird die Reihenfolge bestimmt: Welche Karte soll vor einer anderen Karte bearbeitet werden. Es reicht völlig, wenn so viele Aufgaben sortiert werden, wie bis zum nächsten Daily Standup höchstens angefangen werden oder erledigt werden können. Das Team arbeitet bis heute mit fünf Karten in der »to do«-Spalte. Kai versucht also, die »to do«-Spalte möglichst mit fünf Karten zu bestücken und diese von oben nach unten zu sortieren.

**Die Priorisierung/Sortierung der Aufgaben kann auch das Team übernehmen.**

Das ist die Erfahrung für uns heute. Zu Beginn ist es allerdings sehr hilfreich, wenn dies eine bestimmte Person macht, die von allen als Sortierer akzeptiert wird. In unserem Fall war es Kai selbst. Das Team priorisiert heute gerne, denn so können die Teammitglieder gemeinsam festlegen, an was sie arbeiten, und entscheiden, was wichtig ist – eigenverantwortliches Arbeiten. Die Konsequenz ist jedoch, dass dafür Zeit benötigt wird. Das Team erledigt dies direkt nach dem Standup, meist gemeinsam mit dem Kanban-Coach Kai. Mit einiger Übung sind das täglich 5–10 Minuten für dieses Team.

**Das Team ist für Aufgaben geschützt, die bereits begonnen wurden. Das bringt jedem einzelnen Teammitglied Entspanntheit.**

Wenn eine Aufgabe bereits angefangen wurde, so ist sie geschützt davor, dass jemand sie einfach vom Board nimmt, sprich sagt: »Bitte arbeitet an etwas anderem, hört mit dieser Aufgabe auf, diese ist nun nicht mehr wichtig.« Durch diesen Schutz soll vermieden werden, dass investierte Arbeit weggeworfen wird. In Ausnahmefällen gibt es sicher Gründe, warum eine angefangene Aufgabe abgebrochen wird. Ein solcher Ausnahmefall ist zum Beispiel, wenn das ganze Team sich einig ist, dass es nicht sinnvoll ist weiterzumachen, weil sich eine äußere Begebenheit geändert hat. Alle im Team empfinden diesen Schutz als sehr hilfreich, weil bei neuen, scheinbar wichtigeren Themen routinemäßig die Frage gestellt wird: »Ist es wirklich wichtiger als das, was ich gerade bearbeite, erkennbar durch die Karte am Teamboard ›in progress‹.« Das Abwägen lastet nicht mehr auf einer Person, die für sich priorisieren muss. Es wird gemeinsam thematisiert und entschieden, meist zugunsten der bereits begonnenen Aufgabe.

**»Notfälle lassen alle anderen Aufgaben warten« als Kriterium bei der Entscheidung, ob eine spontan auftretende Aufgabe vorrangig bearbeitet wird, ist hilfreich.**

Hier kann das Team von der Erfahrung der Teammitglieder aus dem bestehenden Kanban-Team »Maintenance« profitieren. Notfälle gibt es dort auch und wie wir alle wissen, bestehen sehr viele Meinungen in einer großen Bandbreite darüber, was ein Notfall ist. Die Erfahrung zeigt, dass es hilfreich ist, sich stets zu fragen: Ist dieser »Notfall« wichtiger und dringender als alles andere am Teamboard? Meist genügt ein Blick aufs Board, um diese Frage zu beantworten, und das Team ist sich einig.

Was nun doch kein Notfall ist, wird wie alle anderen Karten über die »to do«-Spalte priorisiert. Jedes Team, so auch das Team »Internal System Administration«, benötigt eine gewisse Zeit, um für Notfälle ihr eigenes Gefühl zu entwickeln.

Das Ziel ist es, so wenig wie möglich Notfälle zu haben, denn diese unterbrechen die laufende Arbeit, was eine Unterbrechung des Flusses bedeutet.

Taucht also ein scheinbarer Notfall auf, so werden alle dann verfügbaren Teammitglieder zusammengerufen, das Problem schnell und kurz erläutert und entschieden, ob es sich tatsächlich um einen Notfall handelt. Unserer Erfahrung nach kann das Team das entscheiden und benötigt nicht den Teamleiter, der allerdings ein Veto einlegen kann.

Entscheidet das Team, dass ein Notfall vorliegt, so wird eine Karte geschrieben. Das Team hat für Notfälle die Kartenfarbe Pink gewählt. Diese Karte wird in die Notfallzeile gehängt, und zwar sofort in die »in progress«-Spalte. Es wird geklärt, wer sich darum kümmert, am besten zwei Personen, und es wird sofort

mit der Bearbeitung begonnen. Die anderen Teammitglieder setzen ihre Arbeit an den bereits begonnenen Karten fort. Sobald der Notfall erledigt ist und die Karte auf »done« gehängt werden kann, werden alle anderen und die Personen, die die Auswirkungen des Notfalls gespürt haben, informiert.

**Wartezeiten erkennen und sichtbar machen ist das »A« und »O«. Blockkarten sind ein geeignetes Werkzeug, um sie zu visualisieren.**

Schon von Beginn an wurden die Blockkarten in diesem Team eingesetzt. In der Praxis sind dies bei uns quadratische Post-its. Sie werden direkt auf die Kanban-Karte geklebt, sobald kein Teammitglied mehr an dieser Karte weiterarbeiten kann, die Karte aber noch nicht abgeschlossen ist.

Das wichtige Merkmal einer Blockkarte ist, dass sie eine Karte sofort und klar erkennbar als blockiert kennzeichnet. Wir vermerken darauf die Anfangszeit und den Grund. Ist die Blockade aufgelöst, so notieren wir die Dauer der Blockade und befestigen die Blockkarte auf der Rückseite der Kanban-Karte. Wird diese Karte später besprochen, kann man die Informationen auf der Blockkarte in die Analyse einbeziehen.

Wann ist eine Kanban-Karte blockiert?

Sobald keiner im Team an dieser Kanban-Karte weiterarbeiten und/oder helfen kann. Der Grund sind meist Abhängigkeiten von außen, hier einige Beispiele:

- Warten auf eine Antwort von Herrn X
- Warten, bis Bestellung eintrifft
- Warten, bis Teammitglied Y aus dem Urlaub zurück ist
- Warten, bis der Vorstand eine strategische Entscheidung gefällt hat
- Warten, bis Team Z einen Teil entwickelt hat

Als erste Maßnahme versucht das Team gemeinsam zu klären, ob es eine Möglichkeit sieht oder Idee hat, wie die Blockade aufgehoben werden könnte. Das kann auch bedeuten, dass der Kanban-Coach gebeten wird, etwas zu tun und zu helfen, zum Beispiel bei organisatorischen Angelegenheiten.

Wenn die Blockkarte nicht zu entfernen ist, bleibt sie vorerst hängen und »nervt« – denn auch blockiert zählt die Karte zum limitierten »Work in Progress« (WIP).

Im allerbesten Fall wird jede Kanban-Karte mit Blockkarte kurz besprochen, sobald sie abgeschlossen ist. Das Team stellt sich die Frage: »Was können wir tun, damit solch eine Blockade nicht wieder passiert?«

Die einfache Antwort kann sein: »Das war eine absolute Ausnahme, wir brauchen nicht darüber nachzudenken, weil dieser Fall zu selten auftritt.« Das Team entscheidet in jedem Fall, ob es über Lösungen nachdenken möchte. Bei häufigem Auftreten desselben Typs von Blockkarten wird das Team im Normalfall automatisch nach einer Lösung suchen, z.B. bei »Warten, bis Bestellung eintrifft«.

»Ich warte, bis die Bestellung kommt« war in diesem Team eine häufige Aussage. Es werden vorbereitende Maßnahmen getroffen, dann wird bestellt und dann das Ganze installiert und/oder konfiguriert und ausgeliefert (»done«). Eine mögliche Lösung wurde von diesem Team aufgegriffen: Die Karte wird geteilt, es werden also zwei Karten geschrieben: eine Karte für alle Aufgaben einschließlich der Bestellung und eine Karte für die fortführenden Aufgaben bis zur Auslieferung. Die zweite Karte kommt zunächst ins Backlog und wird dann an die Wand gehängt, sobald die Lieferung da ist. Was dem Team dabei fehlte, war der Überblick, was bestellt wurde und was bald geliefert werden sollte, um vielleicht auch mal nachzufragen, falls dies nicht geschah. Das Team entschied sich, eine Art Parkplatz einzurichten, wo die Karten »bis zur Bestellung« so lange hängen, bis die Lieferung da ist. Dann wird die erste Karte abgehängt und die zweite Karte fließt sortiert in die Bearbeitung.

Die Blockkarten bewirken, dass das Team konsequent nach Lösungen sucht bzw. Gegenmaßnahmen trifft, damit Blockkarten erst überhaupt nicht entstehen. Letztendlich werden es im Laufe der Zeit immer weniger und der Flow erhöht sich, und damit auch die Zufriedenheit des Teams.

Durch weniger Blockkarten entstehen weniger Unterbrechungen. Wenn eine Aufgabe ohne Unterbrechung bearbeitet werden kann, dann erhöht sich meist auch die Qualität.

Verbesserungen, Gespräche darüber und die Entscheidung, ob im Einzelfall darüber nachgedacht wird, brauchen Raum und ein wenig Zeit. Retrospektiven sind sehr geeignet dafür. Eine zusätzliche Möglichkeit nutzen wir direkt nach den Daily Standups. Das ist eine günstige Zeit, weil keiner seine Arbeit unterbrechen muss und so die kontinuierliche Verbesserung täglich stattfinden kann.

### Tools – was das richtige ist, bestimmt das Team.

Das Teamboard ist ein Tool. Blockkarten sind ein Tool. Die Notfallzeile am Teamboard ist ein Tool.

Welches Tool sinnvoll für das Team ist, bestimmt das Team. Ob ein Teamboard aus Post-it-Karten besteht oder ob es ein digitales Teamboard ist, deren es zahlreiche am Markt gibt, ist Geschmacksache. Was einem schmeckt, benutzt man gerne, also ist es am besten, hier das Team bestimmen zu lassen. Und wenn es sich nicht als geeignetes Tool herausstellt, dann entscheidet das Team neu. Mit Retrospektiven kann man Tools in Benutzung beleuchten, bewerten und sich bei Bedarf und Lust neu entscheiden.

**Durchlaufzeiten zu messen ist unbedingt wichtig und unerlässlich.**

Puh, das klingt nach Statistik und dafür war in diesem Team keine Zeit vorhanden, da der Kanban-Coach auch der Teamleiter ist. Dennoch ist dieses Thema wichtig für die Themen Planbarkeit und Zufriedenheit, Erhöhung des Flows usw.

Betrachtet man die Durchlaufzeiten der Karten stellt, man gerade zu Beginn fest, dass sie sehr unterschiedlich sind. Hängt aber eine Karte wochenlang in der Spalte »in progress«, kann sie schnell nerven, die Aufgabe wird unübersichtlich und zäh. Visualisierung ist das Stichwort und bietet als Konsequenz Zufriedenheit und Erhöhung der Qualität. Unsere Empfehlung ist es, mit dem Team gemeinsam herauszufinden, wann sie sich wohlfühlen würden. Wie lange darf eine Karte »in progress« hängen, bis sich ein Unwohlsein breitmacht? Das Team entschied sich für 7 Tage. Dauert eine Karte länger als 7 Tage, wurde vereinbart, die Aufgabe in den Retrospektiven genauer anzusehen. Dort versuchen wir, die Ursache zu klären und eine Lösung zu finden. Ziel dabei ist es, dass ähnliche Karten nicht mehr so lange am Board hängen.

Dennoch sollte keine aufwendige Statistik betrieben werden. Wir entschieden uns für folgende Betrachtung der Durchlaufzeiten:

Wird eine Karte von der »to do«-Spalte in die »in progress«-Spalte gehängt und mit der Arbeit daran begonnen, schreibt derjenige, der dies tut, seinen Namen darauf und die Anfangszeit mit Datum. Die »done«-Spalte wird unterteilt in »≤1 Tag« und »>1 Tag bis ≤7 Tage« und »>7 Tage«. Die fertigen Aufgaben werden innerhalb der »done«-Spalte in die jeweilige Spalte gehängt. Dadurch wurde schnell sichtbar, wie viele Aufgaben länger als 7 Tage dauern. Zukünftig sollten solche Karten im Vorfeld in mehrere Karten unterteilt werden, sofern sie keine Wartezeiten beinhalteten, was die Durchlaufzeit wesentlich erhöhen kann. Auch so können zusätzlich Wartezeiten noch einmal angesprochen werden. Dies waren alles Ergebnisse aus den Retrospektiven dieses Teams und keiner musste sich mit Statistiken beschäftigen. Das Beschriften der Karten und das Aufhängen in der richtigen Spalte passierte quasi nebenher.

Dieses Vorgehen ist für dieses Team so erfolgreich, dass wir auch einen Blogbeitrag geschrieben haben für das Projektmanagement-Blog *www.projekt-log.de* von Robert Wiechmann. Der Titel ist »Statistiken für Kanban-Teams – Verbesserungen aufdecken«.

**Soft Skills – ein ewig unterschätztes Thema.**

Die veränderte Vorgehensweise und die Teamphilosophie für agile Teams, das Miteinander, stellt die Teammitglieder oft vor große Herausforderungen. Teilen von Wissen, voneinander lernen, geduldig sein mit den Kollegen, zuhören, klare Sprache, wertschätzender Umgang miteinander, Feedback geben, Konfliktlösung und all diese scheinbar selbstverständlichen Themen bringen agile Teams in Wal-

lung, besonders zu Beginn, aber auch immer wieder. Unsere Empfehlung ist, diese Begriffe zu thematisieren und darüber zu sprechen, Vereinbarungen im Team zu treffen, auf die sich alle Teammitglieder verständigen, sie umsetzen zu wollen. Das macht es einfacher, sich darauf zu beziehen: »Wir haben uns vorgenommen, uns zuzuhören, ich fühle mich nicht gehört und würde mir wünschen, dass ihr mir jetzt zuhört.« Das klingt wie aus einem Buch und ist ungewohnt, hilft aber und mit der Zeit klingen die Sätze natürlicher.

Voraussetzung dazu ist natürlich der absolute Wille, wirklich in einem agilen Team arbeiten zu wollen. Umso besser ist es, wenn die Teammitglieder von Hause aus kommunikativ sind und gerne mit anderen Menschen zusammenarbeiten. Das alles war in diesem Team gegeben und kein Problem. Ich war sehr positiv beeindruckt, von Anfang an, wie dieses Team miteinander spricht, wertschätzend auch Dinge ansprichst, die nicht so gut laufen oder wo sich jemand geärgert hat. Dies konnte auch live erlebt werden in der »Live-Retrospektive« auf der Seacon in Hamburg, 2010, die dieses Team durchführte. Fazit: ein erstaunliches Team!

**Retrospektiven ermöglichen die Verbesserung und das Lernen.**

Anfänglich wird alle 2 Wochen eine Retrospektive durchgeführt. Der Abstand wird verkürzt oder verlängert (heute 3 Wochen), wenn das Team merkt, dass es mehr oder weniger Zeit zur Besinnung benötigt. In der Retrospektive wird die Zeit seit der letzten Retrospektive betrachtet. Was gut lief wird beibehalten und alle freuen sich über den Erfolg. Was nicht so gut lief, wird thematisiert, und alle gemeinsam versuchen, eine gute Lösung dafür zu finden in der Hoffnung und Erwartung, dass es zukünftig besser läuft. Das, was das Team sich vornimmt, wird auf ein Post-it geschrieben und offen im Teamraum in einem Team Backlog (Liste der Dinge, die sich das Team vornimmt) aufgehängt. So kann jeder sich selbst und auch andere mal dran erinnern. Hindernisse, die organisatorische Lösungen brauchen oder einen Sprecher, werden vom Kanban-Coach Kai angenommen und sichtbar im Teamraum aufgehängt. Wir nennen das Impediment Backlog. Ob die Lösungen Erfolg hatten, wird in der nächsten Retrospektive besprochen. Dann fällt wieder die Entscheidung für jede einzelne Lösung: Weiter so oder verändern und wie könnte dann eine neue Lösung aussehen. Die Retrospektiven klar strukturiert und zeitlich begrenzt in einem bestimmten Rhythmus stattfinden zu lassen, hat sich auch für dieses Team bewährt. Routine schafft Vertrauen und ermöglicht große Offenheit. Dieses Team erstaunt wieder einmal durch eine ungewöhnlich schnelle Adaption.

*Ein ganz besonderes Erlebnis – das Team ist live mit einer Retrospektive auf der Bühne.*

*Nachdem das Team gut ein halbes Jahr mit Kanban und Retrospektiven arbeitete, trat Claudia Schröder von der oose gmbh an uns heran und fragte uns, ob wir bei einem Experiment mitmachen würden. Wir kannten uns durch Konferenzen und den daraus folgenden Austausch. Sie schlug uns vor, eine Retrospektive live auf der Bühne auf einer Konferenz abzuhalten mit einem echten Team. Anwesend sollte auch ein Reflektionsteam sein, das dem Team nach der Retrospektive seine Beobachtungen schildert und Feedback gibt. Das Ganze sollte ca. zwei Stunden dauern.*

*Zu unserer großen Überraschung zauderte keiner im Team einen Moment, das wollten alle ausprobieren. Das Ergebnis war ein großer Erfolg, der zudem riesigen Spaß machte und per Videoaufzeichnung festgehalten ist. Sie als Leser können diese Aufzeichnung und den Bericht darüber auf folgenden Blogs nachlesen und ansehen:*

*Artikel und Video zur Live-Retrospektive auf dem XING-Blog:*

*http://blog.xing.com/2011/01/retrospektive-%E2%80%93-live-auf-der-buhne*

*Artikel und Video zur Live-Retrospektive auf dem oose-Blog:*

*http://www.oose.de/teamblog/team/filme-der-live-retrospektive-auf-der-seacon-jetzt-live*

**Auch die Daily Standups eignen sich für schnelle Veränderungen, Klärung sowie Lernen.**

Schon nach wenigen Tagen wird klar. Anfänglich reichen die Retrospektiven nicht, um den besten Weg zu finden. Fragen wie »Schreiben wir denn jede Miniaufgabe des Helpdesks auch auf eine Karte?« müssen sofort besprochen werden. So fallen die Daily Standups etwas länger aus, ca. 20 Minuten. Wir einigen uns darauf, dass wir zuerst eine schnelle Runde im Daily Standup machen und die neuen Aufgaben besprechen, um einen gemeinsamen Überblick zu bekommen. Danach werden sofort störende Faktoren und offene Fragen geklärt und Vorgehensweisen (»So machen wir's ab jetzt«) definiert. Unserer Erfahrung nach sollte man Dinge/Themen/Vorgehensweisen, für die sich das Team entschieden hat, mindestens bis zur nächsten Retrospektive tatsächlich ausprobieren. Besser ist es sogar, diese Dinge so lange auszuprobieren, bis sich das Team ganz sicher ist, ob es funktioniert oder nicht, also eher Wochen als Tage. Schon nach wenigen Wochen werden die Daily Standups wieder kürzer, die Fragen weniger und die Lösungsfindung der übrigen Themen verlagert sich wieder hauptsächlich in die Retrospektiven. Ein Lerneffekt, der sich für dieses Team eingestellt hat: Wir lösen einfach die Fragestellungen gleich und so macht es das Team bis heute. Die Retrospektiven finden trotzdem statt.

**Die ersten Schritte, bevor ein Team beginnt, sind manchmal entscheidend für den Erfolg.**

- Das Ziel für das Team definieren.
- Definition der Rollen
- Wo soll sich das Teamboard befinden?
- Wie soll das Teamboard aussehen und strukturiert sein?
- Soll es ein Onlinetool oder eine Zettelwand sein?
- Klärung bei Management und jedem Teammitglied, ob die notwendige Offenheit und Unterstützung vorhanden ist. Skepsis ist gesund und hilfreich. Die Bereitschaft sollte klar vorhanden sein, es auszuprobieren.
- Rahmenbedingungen, wie Daily Standups und Retrospektiven, zeitlich planen und organisieren.

**Sowohl Scrum als auch Kanban ist hervorragend. Das Passende für das jeweilige Team ist entscheidend.**

Ich selbst arbeite als ScrumMaster, Kanban-Coach und agiler Coach und habe das genau so erlebt. Erfolgreiche Teams haben das Passende für sich gefunden.

Für das Team »Internal System Administration« überlegten wir nicht lange, weil wir bereits Erfahrung hatten durch das bestehende Team »Maintenance«. Von vornherein befürworteten wir in diesem Fall Kanban.

Für welches Team Scrum und für welches Team Kanban besser funktioniert, entscheidet das Team und die Umstände, unter denen es arbeitet. Auch die Aufgaben selbst bieten Hinweise, was besser funktionieren könnte.

#### Indizien für Kanban

Nachfolgend finden Sie ein paar Indizien, die darauf hinweisen könnten, dass Kanban ein guter Weg sein könnte.

1. Sind die Aufgaben oft erst kurzfristig bekannt oder tauchen häufig sehr dringliche Aufgaben auf, die sofort behandelt werden müssen, ist dies ein Indiz für Kanban. Diese Aufgaben dulden keinen Aufschub und können nicht warten, bis eine Scrum-Iteration (Sprint) vorbei ist, auch wenn die Iteration nur eine Woche dauert. Auch Scrum bietet die Möglichkeit, Notfälle zu bearbeiten. Dann bleibt anderes so lange liegen. Allerdings geht bei häufigerem Auftreten der positive Aspekt des Geschütztseins des Teams für die Länge einer Iteration verloren. Auch das »Committment«, das Versprechen, das ein Scrum-Team dem Produktverantwortlichen gibt, kann dann nur noch schwerlich eingehalten werden. Diese Unruhe stört. Kanban bietet dem Team auch einen geschützten Raum für alle Aufgaben, die »in progress«, d.h. in Bearbeitung, sind. Die Aufgaben, die noch nicht begonnen wurden, können

jedoch jederzeit umpriorisiert werden. So ist es möglich, strikt nur die allerdringlichsten Notfälle sofort zu bearbeiten und damit die begonnenen Aufgaben warten zu lassen. Meist genügt es, die dringliche Aufgabe hoch zu priorisieren und die nächste Karte, die vom Stapel genommen wird, ist dann genau diese.

2. Sind die vom Team zu erledigenden Aufgaben strukturell so beschaffen, dass sie nicht zwei Wochen im Voraus geplant werden können, bzw. ist der Planungsprozess (aktuell) nicht so beschaffen, tritt die gleiche Situation wie eben beschrieben ein und Kanban wäre zu empfehlen.
3. Manche Teams lieben es, nur mit einem täglichen Daily Standup und einer Retrospektive, z.B. alle 2 Wochen, auszukommen. Keinerlei zusätzlichen Besprechungstermine sind standardisiert. Selbst wenn die Aufgaben nicht in die Indizienkategorie 1) und 2) fallen, sondern gut im Vorfeld geplant und vorbereitet werden können, ist Kanban eine Alternative. Dann allerdings sollte sich das Team gut überlegen, ob es nicht vielmehr den länger andauernden geschützten Raum einer Scrum-Iteration (Sprint) als Gewinn sieht, und sich für Scrum entscheidet.
4. Das Team hat das Gefühl, dass eigentlich alles gut läuft. Aber immer wieder schleicht sich auch das Gefühl ein »Wir könnten noch besser sein, noch schneller«. Egal wie dieses Team arbeitet, so sollte dieses Gefühl ernst genommen werden und sowohl Scrum also auch Kanban könnte eine Lösung darstellen. In Retrospektiven können Ursachen aufgespürt und Lösungen gefunden werden. Kanban eignet sich auf alle Fälle dazu, es einfach mal auszuprobieren, sofern es jemanden gibt, der Erfahrung damit hat und das Team begleitet.

## 9.7 Die Administratoren machen weiter als Kanban-Team

Dieses Team liebt das Arbeiten im agilen Team und mit Kanban, also macht es genau so weiter. Inzwischen werden Retrospektiven nur noch alle 3 Wochen durchgeführt, manchmal auch erst nach 4 Wochen. Das genügt bei einem derart eingespielten Team. Hat das Team eine Frage oder ein Thema, für das sich einfach keine Lösung finden lässt, so kann es den Coach jederzeit einladen, um darüber zu sprechen. Das ist natürlich eine Luxussituation, die nicht jedem Team gegeben wird. Gut, dass wir so viel Erfahrung im Hause haben und ein gesondertes Team von agilen Coachs.

## 9.8 Erkenntnis für uns

Nun wissen wir es, dass Kanban sich nicht nur für Teams eignet, die Softwareentwicklung als Aufgabe haben, sondern auch Administratoren lieben die Kombination von Kanban und agilem Team.

Das ist der springende Punkt: die Kombination von Kanban und der Philosophie der »agilen Teams«. Der Prozess bringt genug Struktur ins Spiel und die agilen Teams – durch ihr eigenverantwortliches Handeln und den Wunsch nach ständiger Verbesserung – verfolgen den besten Weg für ihren Erfolg – Best Practise.

Kanban lässt genug Spielraum für die Eigenarten jedes Teams. Die Aufgabe besteht darin, seinen eigenen Weg zu finden. Nur wie das genau geht, das ist die Frage. Dieser Bericht beschreibt eine der vielen Alternativen.

## 9.9 Das Fazit

Die Alltagsthemen haben sich für das Team nicht geändert, aber die Alltagssituation und das Gefühl, organisiert zu sein, macht das Arbeiten ruhiger. Noch offener als früher werden Themen besprochen und Lösungen gesucht und häufiger als früher werden sie schnell umgesetzt. Die Wissensverteilung im Team nahm deutlich zu, die Spezialisten sind nach wie vor gefragt. Das Team erhöhte ihre gefühlte Qualität, da es erkennt, wie wenig wirkliche Notfälle es bearbeiten muss. Dadurch, dass strategische Themen nun Themen des Teams sind, achten alle gemeinsam darauf, dass sie Raum bekommen. Das Team weiß nun, wie viel es schaffen kann, und das erhöht deutlich die Planungssicherheit und das Wohlbefinden. Dieses Team hatte schon immer viel Spaß an der Arbeit und jetzt wissen alle auch noch warum.

## 9.10 Das Umfeld

**XING AG, *www.xing.com***

Die Hamburger XING AG, das führende soziale Netzwerk für berufliche Kontakte im deutschsprachigen Raum, beschäftigt im Frühjahr 2011 rund 300 Mitarbeiter. Über 10 Millionen Mitglieder weltweit nutzen die Plattform für Geschäft, Job und Karriere (Stand: September 2010).

Scrum und Kanban sowie agile Teams und Lean Management sind etabliert und werden auch vom Management unterstützt und gelebt.

**Dieses Team gab den Anstoß, Kanban auch für das Team »Internal System Administration« einzusetzen:**

**Team »Maintenance«**

Kanban: seit Mai 2009 bis heute

Thema im Team: Software Development, *www.xing.com*

Das Maintenance-Team kümmert sich um alle Themen, die nicht in den Scrum-Teams bearbeitet werden; also um alle »live bugs«, Notfälle auf der Live-Plattform von *www.xing.com* und alle Wünsche, Aufgaben und Anforderungen, z.B.

Marketingthemen, neue Themen, kleine Themen, für die es sich nicht lohnt, ein gesondertes Team aufzusetzen. Es hat sich inzwischen etabliert, dass das Team in Ausnahmesituationen auch den Scrum-Teams hilft, wenn ihnen für eine Aufgabe mal ein Skill fehlt.

Teamgröße anfänglich:

7 Teammitglieder, 1 Produktverantwortlicher, 1 Kanban-Coach

Teamgröße heute:

11 Teammitglieder, 1 Produktverantwortlicher, 1 Kanban-Coach

## 9.11 Agile Werte im Projekt

| **Individuen und Interaktionen** | **vor** | **Prozessen und Tools** |
|---|---|---|
| ▲<br>Der Anstoß, dass Kanban vielleicht »das Richtige« sein könnte, kam vom Chef. Das Team entschied sich bewusst dafür. Die Prozesse und Tools in der praktischen Anwendung wählte das Team selbst und passt sie bis heute an, wenn es sinnvoll erscheint. | | |
| **Laufende Software** | **vor** | **Ausführlicher Dokumentation** |
| ▲<br>Das Team bestimmt selbst, was und wie es dokumentieren möchte. Nur das Nötigste wird dokumentiert, hauptsächlich um in Vertretungssituationen Wissen zu teilen. | | |
| **Zusammenarbeit mit dem Kunden** | **vor** | **Vertragsverhandlungen** |
| ▲<br>Die klassische Kundensituation ist als internes Administratorenteam nicht vorhanden. | | |
| **Reagieren auf Veränderungen** | **vor** | **Befolgen eines Plans** |
| ▲<br>Der Prozess wurde immer dann vom Team selbst angepasst, wenn es sinnvoll erschien. Die äußeren Veränderungen hatten immer zur Folge, dass auch der Plan überdacht und entsprechend angepasst wurde. | | |

Henning Wolf, Agile Projekte mit Scrum, XP und Kanban , dpunkt.verlag, ISBN 978-3-86490-266-6

# 13 Agil bei it-agile: Pull in Vertrieb und Verwaltung

Arne Roock

*Als Firma, die sich schon seit etlichen Jahren intensiv mit verschiedenen agilen Methoden auseinandersetzt und Kunden bei der Einführung von Scrum & Co. unterstützt, ist uns das Pull-Prinzip natürlich nicht nur vertraut, sondern auch lieb und teuer. Trotzdem mussten wir feststellen, dass auch in der agilen Welt gilt: »Der Schuster hat die schlechtesten Schuhe!« Denn sowohl in unserem Vertrieb als auch in der Verwaltung haben wir erst vor einiger Zeit angefangen, auf Pull umzustellen. Dieser Beitrag beschreibt, was die Gründe dafür waren, welche Hürden bisher zu nehmen waren und was noch vor uns liegt.*

## 13.1 Changeban

Getreu dem Kaizen-Gedanken legen wir seit Gründung der Firma großes Gewicht darauf, uns intern immer weiter zu verbessern – sei es unsere interne Kommunikation und der Umgang miteinander, unser Dienstleistungsportfolio, unsere Weiterbildung, unsere Außendarstellung usw. Der wichtigste Hebel dafür waren von Anfang an sogenannte Tuning-Tage, also firmeninterne Workshops mit allen Kollegen, die wir zwei Mal jährlich durchführten, um genau solche Themen systematisch zu besprechen und Änderungen zu beschließen. Gute Ideen hatten wir stets genug. Aber mit der Zeit dämmerte uns immer mehr, dass wir ein Umsetzungsdefizit hatten: Wir begannen immer wieder mit der Umsetzung neuer Ideen, ohne darauf zu achten, dass die alten Beschlüsse überhaupt zu Ende umgesetzt waren. Das war nicht nur die reinste Verschwendung von Geld und Arbeitskraft, sondern wir nahmen uns damit auch die Chance, Dinge wenigstens so weit zu treiben, dass wir uns über die Ergebnisse Feedback einholen konnten, um daraus etwas zu lernen.

Als wir wieder einmal bei so einem Tuning-Tag zusammensaßen (sie nahmen immer öfter die Gestalt von Open Spaces an), bot ein Kollege die Session »Wie können wir die Lead Time für interne Veränderungen verkürzen?« an und

brachte so einen ziemlich großen Stein ins Rollen (nicht nur, weil er in dieser Session die fast schon legendäre »Lead-Time-Schnecke« erfand). In dieser Open-Space-Session wurden erste Ideen besprochen und später dem Plenum vorgestellt. Abends beim Essen diskutierten kleine Grüppchen weiter, und schon am nächsten Tag beschlossen wir gemeinsam, unser Changeban einzuführen – also eine Art Kanban-Board, um unsere internen Veränderungsprojekte zu organisieren. Zuerst sammelten wir, welche Changes wir bereits begonnen hatten und welche bereits beschlossen waren, ohne dass schon jemand daran arbeitete. Dann priorisierten wir diese Changes gemeinsam in der großen Gruppe, was erstaunlich schnell ging. Nun kam allerdings der schwierige Teil: Wenn wir dafür sorgen wollten, dass Changes so schnell wie möglich durchgeführt werden sollten (und zwar so lange, bis sie wirklich »donedone« sind), dann müssten wir unseren ChIP (Change in Progress) limitieren. Sich auf eine Anzahl an Changes zu verständigen, die wir als Firma gleichzeitig stemmen können, war noch einfach: Nicht mehr als vier hielten wir für sinnvoll. Aktuell hatten wir aber mit mindestens zehn Changes begonnen! Was sollten wir also tun, um unter das selbst gesetzte Limit zu kommen? Die erste Reaktion (die wir von unseren Kunden nur allzu gut kannten): »Das geht ja gar nicht!« Dann kamen aber doch konstruktive Vorschläge: »Wir lassen nur die vier am höchsten priorisierten Changes im System und schicken alle anderen zurück ins Backlog.« Dieser Vorschlag sorgte sofort für großen Unmut bei den Entwicklern, denn das hätte bedeutet, dass ein Thema, das ihnen sehr wichtig war, mehrere Monate lang auf Eis gelegt worden wäre. Ich war damals der Meinung, wir hätten diesen Schritt gehen sollen, um den Druck auf uns selbst zu erhöhen. Denn das hätte geheißen, auch die Entwickler wären in der Pflicht gewesen, aktiv an anderen Changes mitzuarbeiten (selbst wenn es keine »Entwicklerthemen« waren), um so möglichst schnell Platz für ihren »Change« auf dem Board zu machen. So wäre drastisch deutlich geworden, dass es die beste Lösung ist, wenn wir alle gemeinsam als Firma an unseren Changes arbeiten und wegkommen von einer Art Arbeitsteilung, bei der jeder nur auf einen kleinen Ausschnitt schaut, der ihn direkt betrifft. Heute bin ich allerdings froh darüber, dass wir uns für einen anderen Weg entschieden haben. Wir sind übereingekommen, erst einmal mit einer deutlichen Überschreitung unseres Limits zu leben, aber auf keinen Fall noch mehr ChIP ins System zu ziehen. So haben wir verhindert, dass unser neues Changeban gleich zu Beginn mit Akzeptanzproblemen zu kämpfen hatte.

Nun ging es daran, das initiale Board zu designen. Obwohl wir große Fans von Low-Tech-Tools sind (also Karteikarten oder Haftnotizen an einem Whiteboard o. Ä.), entschieden wir uns doch für eine elektronische Lösung. Schließlich sind wir tendenziell über 30 verschiedene Standorte verteilt, und der Sinn des Boards sollte ja gerade darin bestehen, dass möglichst alle Kollegen regelmäßig draufblicken – oder zumindest die Möglichkeit dazu haben.

Als Spalten legten wir fest: Idee, Backlog (priorisiert), Change vorbereiten, Change begleiten, Done.

In die Spalte »Idee« darf jeder Mitarbeiter neue Tickets einstellen für Changes, die er für sinnvoll hält.

Ins Backlog darf ein Ticket erst dann wandern, wenn feststeht, worin der Wert dieser Veränderung für die Firma besteht (dieser muss nicht direkt monetär sein, aber er muss sich benennen lassen), und wenn festgelegt ist, was die Definition of Done für dieses Ticket ist. Dass es keine einheitliche Definition of Done für alle Tickets gibt, war (und ist) verwirrend für uns. Aber trotz mehrerer Anläufe ist es uns bis heute nicht gelungen, allgemeingültige Definitionen für alle Tickets zu finden. Priorisiert wird das Backlog ca. vier Mal im Jahr, wenn wir uns mit allen Kollegen treffen. Bei solchen Firmentreffen gehört die Priorisierung inzwischen zur ritualisierten Agenda.

Sobald aktiv an einem Change gearbeitet wird, gelangt das entsprechende Ticket in die Spalte »Change vorbereiten«. Hierzu ist es nötig, dass es einen Change Owner gibt – also jemanden, der sich für den Change verantwortlich fühlt und den alle anderen Kollegen ansprechen können, wenn sie eine Frage haben. Der Change Owner muss nicht zwangsläufig alle Arbeiten selbst durchführen. Aber er muss dafür sorgen, dass die Aufgaben erledigt werden, und sich dafür die nötige Hilfe von den Kollegen oder der Geschäftsführung holen.

In die Spalte »Change begleiten« kommt ein Ticket dann, wenn die aktive Arbeit hieran abgeschlossen ist, aber die Veränderung noch nicht vollständig institutionalisiert ist. In diesem Stadium geht es in erster Linie darum zu beobachten, ob die Veränderung die gewünschte Wirkung hat, und kleinere Nachbesserungen vorzunehmen. Wenn der Change beispielsweise darin besteht, mit einer neuen Schulung Geld zu verdienen, dann sind die Vorbereitungen abgeschlossen, wenn wir die Schulung ein erstes Mal erfolgreich beim Kunden durchgeführt haben. Danach ist es aber sinnvoll, diese Schulung »unter besondere Beobachtung« zu stellen, so lange, bis sie z.B. drei weitere Male durchgeführt wurde und wir uns sicher sind, dass wir sie ohne großen Stress immer wieder anbieten können.

Sobald ein Change institutionalisiert ist, also zum Tagesgeschäft gehört, wandert er in die Spalte »Done«.

## 13.2 Lernerfahrungen

Wie zu erwarten, lief unser Changeban nicht von Anfang an rund. Zu Beginn hatten wir ziemlich stark damit zu kämpfen zu entscheiden, was genau wir eigentlich als Change definieren wollen. Gehört die Einstellung eines neuen Kollegen beispielsweise auf das Changeban-Board? Oder eine große Akquise? Nach etlichen Diskussionen sind wir inzwischen zu einer Definition gelangt, die zwar nicht wirklich erschöpfend ist, für uns aber gut genug funktioniert.

Eine interessante Lernerfahrung bestand darin, dass wir die ChIP-Limits pro Spalte wieder abgeschafft haben! Nach einer Weile wurde uns nämlich deutlich, dass an fast jedem Ticket ein Mitglied der Geschäftsleitung beteiligt war, was auch nicht weiter verwunderlich ist, wenn man bedenkt, dass es hier um strategische Veränderungen unserer Firma geht. Also erschien es uns sinnvoller, ein Personenlimit einzuführen. Wir definierten also die folgende Regel: Niemand darf an mehr als zwei Changes gleichzeitig arbeiten! Dieses Limit wurde zu Beginn durch die Geschäftsführer deutlich überschritten. Aber inzwischen haben wir uns so weit diszipliniert, dass die Limits eingehalten werden. Die Personenlimits geben uns die Flexibilität, mehr Changes durchzuführen, wenn wir eine Möglichkeit finden, dass sie nicht das »Bottleneck Geschäftsleitung« passieren müssen. Das hat zu weniger Anspruchsdenken gegenüber der Geschäftsleitung geführt: Wenn jemand etwas in der Firma verändern möchte, dann ist er in erster Linie selbst gefragt, hier aktiv zu werden.

Ein anderer Punkt, den wir lernen mussten, betrifft die Art, das Board zu betrachten und über die Tickets zu sprechen. Beim wöchentlichen Jour fixe der Geschäftsleitung gehört das Changeban-Board zur ritualisierten Agenda. Zu Beginn verlief das Gespräch meistens nach dem Muster: »Gibt es zu diesem Ticket etwas Neues« –»Nein« – »Okay, wie sieht es beim nächsten Ticket aus?« Inzwischen sind wir dazu übergegangen, die unangenehme Frage zu stellen: »Was können wir tun, damit sich das Ticket demnächst weiterbewegt?« Immer wieder den Finger in die Wunden zu legen, ist anstrengend, und wir müssen nach wie vor ständig mit der Versuchung kämpfen, einfach abzuwarten, dass sich Dinge von selbst verändern. Aber durch Kanban sollen wir ja gerade immer wieder dazu gezwungen werden, ernsthaft nach Lösungen zu suchen, wie es eben doch schneller vorangeht und wie man Blockaden sofort beseitigen kann. Und natürlich gilt bei unserem Changeban dasselbe wie im Projektalltag: Hindernisse lassen sich fast immer beseitigen, wenn man wirklich will.

Ein anderes Problem, mit dem wir zu kämpfen hatten, bestand darin, dass die Arbeit an Changes immer wieder verzögert wurde, weil wir Kundenanfragen (ohne dafür eine explizite Regel zu haben) immer wieder Vorrang einräumten. Auch dieses Phänomen kennen wir aus unserer Beratungspraxis: Strategische Aufgaben (also Aufgaben, die wichtig, aber nicht dringend sind) werden ständig von dringenden Aufgaben verdrängt, die einen sofortigen Nutzen bringen. Das ist natürlich allzu verständlich. Aber wenn dies permanent geschieht, bedeutet das den sicheren Tod für jede Organisation! Also haben wir etliche Diskussionen darüber geführt, wie fakturierbare Arbeitszeit im Verhältnis zur Arbeit am Changeban zu gewichten sei. Wir haben dafür keine allgemeingültige Regel definiert, aber ganz deutlich gemacht: Im Zweifelsfall ist es legitim (und auch gewünscht), Kunden zu vertrösten (was natürlich auch zu Absagen führen kann), um dadurch die Arbeit am Changeban nicht zu weit zu verzögern. Hieran wird deutlich (und ist uns selbst deutlich geworden): Unser Changeban ist kein Spielkram, sondern

unser zentrales Koordinierungs- und Planungsinstrument, mit dem wir langfristig wettbewerbsfähig bleiben!

Interessant zu beobachten war (und ist) es, wie schwer es uns selbst manchmal fällt, Dinge umzusetzen, die wir in der Beratung ganz selbstverständlich unseren Kunden empfehlen. Beispielsweise mussten wir erst lernen, unsere Geschäftsleitung als Bottleneck anzusehen – mit den bekannten Konsequenzen und Linderungsstrategien. Auch hier sah es zuerst so aus, als müsste man damit leben, dass Changes ewig dauern, weil sie diesen Engpass passieren müssen. Aber wenn dies wirklich so ist, warum arbeiten dann nicht alle anderen Kollegen daran, die Geschäftsleitung von allen anderen Aufgaben zu entlasten, sodass sie sich auf ihre Tätigkeit am Bottleneck konzentrieren können? Warum überlegen wir nicht, wie wir unsere Changes so schneiden können, dass sie nicht durch den Engpass müssen? Und wenn die Geschäftsleitung in so vielen Dingen ein Engpass ist, warum arbeiten wir dann nicht daran, die nötigen Kompetenzen auch an andere Kollegen weiterzugeben, sodass sich zumindest langfristig das Bottleneck etwas weitet? All diese Fragen sind extrem wichtig und helfen uns dabei, unser Changeban immer weiter zu verbessern.

Für mich persönlich besteht der größte Wert unseres Changeban in der Fokussierung auf die aktuell wichtigsten Veränderungen. Wie weitreichend die Konsequenzen hieraus sind, lässt sich gut an einer kleinen Anekdote zeigen: Zusammen mit einem Kollegen traf ich mich vor einigen Wochen zu einem ganztägigen Termin, den wir vor über sechs Monaten vereinbart hatten. In dem Termin ging es darum, die Grundzüge unseres Marketings zu überdenken. Das Meeting war nach 15 Minuten zu Ende! Denn wir waren uns einig, dass dieses Thema auf das Changeban-Board gehört und wir beide aktuell unsere Personenlimits ausgeschöpft hatten. Also erstellten wir ein neues Ticket für die Spalte »Idee« und verwendeten den Tag dafür, an anderen Dingen zu arbeiten.

## 13.3 Akquise-Kanban

Alles begann mit einer E-Mail. Nachdem ich einen Anruf erhalten hatte, in dem ein Interessent nach einer Scrum-Schulung fragte, schrieb ich eine Mail an alle Berater bei it-agile, in der ich die Anfrage kurz schilderte und fragte, wer die Schulung durchführen kann. Das war ungewöhnlich, denn bis zu diesem Zeitpunkt wurden (bis auf einige Ausnahmen) alle Aufgaben, die im weitesten Sinne mit Vertrieb zu tun haben, ausschließlich von der Geschäftsleitung erledigt. Dazu gehörte nicht nur der gesamte »Pre-Sales-Prozess«, also Klärung des genauen Auftrags, Preisverhandlungen, Angebotserstellung usw. Die Anfragen wurden auch von der Geschäftsleitung besetzt, d.h., es wurde entschieden, welcher Berater den Auftrag ausführen sollte (natürlich nach Rücksprache mit dem jeweiligen Kollegen). Das hatte in der Vergangenheit auch ganz gut funktioniert. Aber in letzter Zeit waren genau diese drei Personen, die für den Großteil des Vertriebs

zuständig waren, permanent überlastet, was nicht nur zu persönlichem Stress führte, sondern auch zulasten der Kundenzufriedenheit ging. Die Durchlaufzeiten für Antworten an Kunden und Angebote wurden immer länger, sodass wir nicht nur immer öfter in die Verlegenheit kamen, uns entschuldigen zu müssen, sondern auch Gefahr liefen, potenzielle Kunden zu verlieren. Kurz vorher war es sogar vorgekommen, dass wir eine vielversprechende Anfrage gar nicht beantwortet hatten, weil sie schlicht übersehen wurde! Den Auftrag hatten wir also (natürlich ohne es zu wollen) unserer Konkurrenz überlassen.

Im Nachhinein betrachtet, ist die Ursache für diesen Schlamassel gar nicht ungewöhnlich: Es handelte sich um ein typisches Skalierungsproblem, wie es viele Unternehmen kennen, die auf eine bestimmte Größe wachsen. Als wir 2005 mit zehn Kollegen starteten, waren die Vertriebs- und Verwaltungsaufgaben überschaubar. Damals waren ein Geschäftsführer, der die meiste Zeit selbst als Berater beim Kunden unterwegs war, und ein Assistent ausreichend. Sechs Jahre und 20 neue Kollegen später reichten selbst zwei Geschäftsführer, ein Prokurist und eine Assistentin nicht mehr, um diese Aufgaben zuverlässig und in vernünftiger Qualität zu erledigen. Denn bekanntlich steigen Kommunikationsaufwände ja nicht linear, sondern exponentiell mit zunehmender Zahl an beteiligten Personen (in unserem Fall Mitarbeiter und Kunden). In dieser Situation liegt es erst einmal nahe, einen Vertriebler einzustellen, um die Geschäftsleitung zu entlasten. Diese Option hatten wir schon mehrmals diskutiert und auch in Ansätzen ausprobiert, uns dann aber aus verschiedenen Gründen dagegen entschieden.

Zurück zur E-Mail. Der erste »Pull-Request« funktionierte gut: Es meldete sich ein Kollege, der freie Kapazitäten hatte und die Schulung übernehmen wollte. So kam es, dass bald immer mehr solcher E-Mails kursierten. Oft klappte es, die Arbeit so besser zu verteilen. Aber es kam, wie es kommen musste, und nach einer Weile blieb eine Mail unbeantwortet. Was nun? Als auch eine wiederholte E-Mail unbeantwortet blieb, kribbelte es natürlich in den Fingern, wieder in alte Push-Manier zu verfallen und einfach einen Kollegen dazu zu »verdonnern«, diese Schulung zu übernehmen. Aber das widersprach allem, was wir unseren Kunden beinahe täglich rieten: Pull funktioniert nur bei ausreichendem Vertrauen. Wir gehen davon aus, dass alle Kollegen hoch motiviert sind und ihr Bestes geben. Wenn sich also niemand auf einen Pull-Request hin meldet, dann hat auch niemand freie Kapazitäten. Die notwendige Konsequenz: Wir sagten dem potenziellen Kunden ab. Das verursachte fast physische Schmerzen, aber es schien das Richtige zu sein (und im Nachhinein bin ich sicher, dass es das einzig Richtige war). Zum Glück blieben solche Absagen die Ausnahme, sodass unsere ersten Pull-Versuche schon recht erfolgreich verliefen.

Aber es zeigten sich sofort zwei neue Herausforderungen:

Zum einen war die Entlastung der Geschäftsleitung noch nicht groß genug. Denn nun waren sie zwar die Aufgabe los, ständig mit Auslastungs-Sheets zu jonglieren und herumzutelefonieren, um herauszufinden, welcher Kollege wann verfügbar ist. Aber die eigentliche Vertriebsarbeit (Angebote schreiben, Nachfragen beantworten, beim Kunden nachhaken usw.) blieb, wo sie war.

Zum anderen fehlte die Transparenz darüber, welche Aufgaben noch offen waren und welche schon gepullt wurden. Dies führte dazu, dass es zu einer Zunahme und Redundanz in der E-Mail-Kommunikation kam, weil regelmäßig E-Mails an alle Berater geschrieben wurden, die in etwa den Inhalt hatten: »Vor einiger Zeit gab es diesen Pull-Request. Jetzt hätte ich Zeit dafür. Wo kann ich mehr Informationen dazu finden? Oder hat das jetzt schon jemand anders übernommen?« Außerdem kam die Frage auf, ob das denn überhaupt Pull wäre, wenn man eine E-Mail schreibt mit der (impliziten) Aufforderung, dass da doch bitte eine Aufgabe erledigt werden soll.

Das erste Problem lösten wir dadurch, dass nicht mehr nur die Schulungen/Coachings selbst gepullt werden sollten, sondern auch die gesamten zugehörigen Vertriebsaufgaben. Die Pull-Requests lauteten jetzt eher: »Wir haben eine Anfrage zu einer Scrum-Schulung in Hamburg von Herrn Meier. Wer mag dort anrufen und alles Weitere klären?« Um dorthin zu kommen, war es allerdings nötig, den Beratern Basiswissen in Vertriebsarbeit angedeihen zu lassen. Also wurde eine kleine Weiterbildungsoffensive ins Leben gerufen, inklusive ganztägigem Pair-Vertrieb und direktem Feedback durch die Geschäftsleitung nach Kundentelefonaten. Das nahm alles in allem mehrere Monate in Anspruch, aber der Gewinn ist beachtlich: Wenn eine Anfrage jetzt gepullt ist, dann ist sie komplett abgegeben – und kommt normalerweise auch nicht mehr zurück.

Die Lösung für fehlende Transparenz ist in agilen Methoden sozusagen schon eingebaut: ein Board mit Tickets für jede Aufgabe darauf. Also führten wir so ein Board auch für unseren Vertrieb ein. Zuerst wurde die Wertschöpfungskette für unseren Vertriebsprozess visualisiert, was uns schwerer fiel als gedacht. Nach einer Testphase und einigen Korrekturen war es dann so weit: Das Board wurde auf die gesamte Firma ausgerollt. Wenn eine neue Anfrage hereinkommt – egal ob per E-Mail, Telefon oder durch persönliche Kontakte –, wird ein Ticket mit allen nötigen Informationen darauf erstellt, das in einer Input Queue landet und darauf wartet, gepullt zu werden. Der Ersteller des Tickets behält im Auge, ob das Ticket innerhalb der vereinbarten Reaktionszeit von einem Kollegen gezogen wird. Falls nicht, soll er dem Kunden absagen oder mit der Geschäftsleitung besprechen, wie in diesem Fall weiter zu verfahren ist. Interessanterweise ist es seit der Einführung des Boards nur zwei Mal vorgekommen, dass ein Ticket nicht gezogen wurde (beide Male haben wir dem Kunden abgesagt)!


***Abb. 13–1*** *Akquise-Board. Die Spalten sind »Anfrage«, »Anfrage besetzen«, »Angebot erstellen«, »Deal anschließen« und »Leistung abrechenbar«. Jedes Ticket stellt eine Kundenanfrage dar. Visualisiert sind außerdem die jeweiligen Bearbeiter, Blocker und Deadlines.*

Insgesamt ist unser Vertrieb jetzt so transparent geworden, wie wir es uns erhofft hatten. Und dies nicht nur auf der Ebene einzelner Aufgaben. Immer mehr ändert sich die allgemeine Einstellung weg von »Das ist nicht meine Akquise, sondern deine« hin zu »Wir als Firma haben eine bestimmte Anzahl von Akquise-Aufgaben, die wir erledigen möchten. Lass uns überlegen, wie wir das gemeinsam hinbekommen«.

Und durch Pull hat sich noch ein weiterer positiver Effekt eingestellt: Auf Pull-Requests meldeten sich nämlich auch Kollegen, die wir vorher für bestimmte Aufträge gar nicht gefragt hätten. Sie erhielten jetzt die Chance, sich allmählich an für sie neue Themenbereiche heranzutrauen – und sich dafür ggf. auch die nötige Unterstützung zu holen.

Und die Geschäftsleitung? Sie hat sich natürlich nicht komplett von allen Vertriebsaufgaben verabschiedet. Zum einen ziehen sich selbstredend auch die Ge-

schäftsführer regelmäßig Tickets aus der Input Queue, zum anderen betrachten sie einmal in der Woche systematisch das Akquise-Board, um Probleme zu erkennen, abgeschlossene Tickets auszuwerten (»Warum hat uns Kunde XY abgesagt?«) und weitere Verbesserungspotenziale zu erkennen.

Das Board hat übrigens als Nebeneffekt noch ein anderes Problem gelöst: In der Vergangenheit gab es nämlich immer wieder Verwirrung, welche Leistungen zu welchem Zeitpunkt abgerechnet werden können. Die Rechnungen wurden also oft später geschrieben als nötig. Und es entstand unnötiger Kommunikations-Overhead, um jede einzelne Abrechnung zu klären. Heute heißt die letzte Spalte auf unserem Akquise-Board: »Leistung abrechenbar«. Für jedes Ticket, das sich in dieser Spalte befindet, kann die Rechnung geschrieben werden – alle nötigen Informationen finden sich auf dem Ticket. Die Kollegen aus der Verwaltung (sie heißen bei uns »Moneypennys«) übernehmen dann diese Tickets auf ihr Rechnungs-Board (das es schon seit Langem gibt) und beginnen mit der Rechnungslegung, sobald sie freie Kapazitäten dafür haben. Dieses Rechnungs-Board ist übrigens kein elektronisches Tool, sondern eine Wand voller Haftnotizen. Ohne großes Zutun haben wir also auch die Erstellung der Rechnungen auf Pull umgestellt.

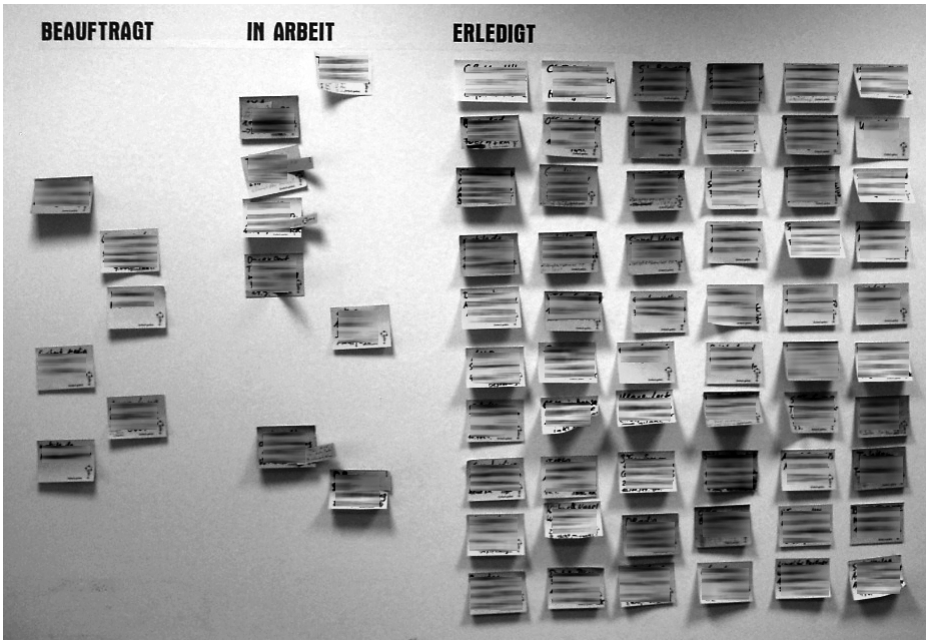


***Abb. 13–2*** *Rechnungs-Board. Jedes Ticket stellt eine Rechnung dar. Verschiedene Arten von Rechnungen sind durch unterschiedliche Farben gekennzeichnet: Grüne Tickets sind einzelne Dienstleistungen, blaue Tickets monatlich wiederkehrende Leistungen und weiße Tickets stellen Sammlungen von kleinen Rechnungen dar. Blockierte Tickets sind durch rote Sticker gekennzeichnet.*

Die Ergebnisse, die wir bisher erreicht haben, sind beachtlich, aber wir wollen noch viel besser werden. In Zukunft wollen wir systematisch die Entwicklung der Durchlaufzeiten und andere Metriken auswerten, um uns auf dieser Basis noch weiter zu verbessern. Einen weiteren Ansatzpunkt bietet uns die Engpasstheorie. Ein Blick auf unser Board offenbart uns sofort, wo unser Engpass liegt: beim Kunden! Die meisten Tickets stauen sich dort, wo wir auf die Antwort von Interessenten warten. Diesen Engpass zu beseitigen, wird eine der größten Herausforderungen der nächsten Zeit werden.

Und natürlich stellt sich die Frage nach den WIP-Limits. Wir haben erst einmal ohne explizite Limits angefangen, aber es wird immer wieder deutlich, dass auch hier eine Limitierung sinnvoll ist – wie auch immer diese genau aussehen wird.

Etwas schwer tun wir uns auch noch damit, eine Kaizen-Kultur für unsere Vertriebsprozesse zu etablieren, um uns kontinuierlich zu verbessern. Dies ist ein weiter Weg, aber ich bin zuversichtlich, dass wir ihn meistern werden.